Xpert.press

Die Reihe **Xpert.press** vermittelt Professionals in den Bereichen Softwareentwicklung, Internettechnologie und IT-Management aktuell und kompetent relevantes Fachwissen über Technologien und Produkte zur Entwicklung und Anwendung moderner Informationstechnologien.

Christian Benjamin Ries

BOINC

Hochleistungsrechnen mit
Berkeley Open Infrastructure for Network Computing

Springer Vieweg

Christian Benjamin Ries
Bielefeld
Deutschland

ISSN 1439-5428
ISBN 978-3-642-23382-1 978-3-642-23383-8 (eBook)
DOI 10.1007/978-3-642-23383-8

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Springer Vieweg

Gedruckt auf säurefreiem und chlorfrei gebleichtem Papier

Springer Vieweg ist eine Marke von Springer DE. Springer DE ist Teil der Fachverlagsgruppe Springer Science+Business Media www.springer-vieweg.de

Für meine Familie,
für meine Nächsten,
für mich!

Vorwort

Wenn Sie nach einem Buch über BOINC suchen, so wird es Ihnen sicherlich schwerfallen, etwas Passendes zu finden. Ich habe es während meiner ersten Schritte mit BOINC getan und war verblüfft: Alles was ich fand waren kleinere Abschnitte in dicken Büchern über völlig andere Technologien. Erstaunlich, denn immerhin existiert BOINC offiziell schon seit Februar 2002. Aber ganz allein ist man dann doch nicht; BOINC besitzt eine wunderbare, hilfsbereite Community und ein stellenweise nicht mehr ganz aktuelles Wiki, bedauerlicherweise ohne tiefer gehende Erklärungen und Beispiele der Programmierschnittstelle. Während der ersten Entwicklungsarbeiten laufen Sie daher möglicherweise gegen die Wand und Sie wissen nicht warum. Mit diesem Buch möchte ich versuchen, Ihnen die ersten Schritte mit BOINC zu vereinfachen, und liefere Ihnen einen nachvollziehbaren Entwicklungsweg, von der Idee bis zum lauffähigen BOINC-Projekt.

Meinen ersten Kontakt mit BOINC hatte ich 2007, damals noch freiwilliger Projektteilnehmer und ich habe ein wenig Rechenleistung für das Lösen von wissenschaftlichen Fragestellungen zur Verfügung gestellt. Nach meinem Erststudium der Informationstechnik an der Fachhochschule Bielefeld habe ich mich direkt in den Masterstudiengang „Optimierung und Simulation“ an derselben Hochschule eingeschrieben. *Seither ist BOINC ein großer Teil meines Lebens und hat mich bis heute fast täglich beschäftigt.* In meiner Masterarbeit habe ich mich gründlich mit den Funktionen und technischen Raffinessen von BOINC auseinandergesetzt, um sogenannte ISV-Anwendungen (Independent-Software-Vendor) mit BOINC zu verheiraten, zu verteilen und Rechner Dritter für die Abarbeitung von Simulationsmodellen einzubeziehen – mit einigem Erfolg.

Seither bin ich wissenschaftlicher Mitarbeiter an der Fachhochschule Bielefeld und promoviere in Kooperation mit der Glyndŵr University in Wales im Bereich des Computing Engineering. Meine Promotionsarbeit beschäftigt sich mit der Erforschung von Ansätzen für einen modellgetriebenen Entwicklungsansatz, um dadurch die Arbeit mit BOINC effizient und so einfach wie möglich zu gestalten. Die Entwicklungszeit soll gering und die Wartungsarbeiten minimal sein.

Wir beschäftigen uns in diesem Buch in vier Abschnitten mit BOINC, so dass Sie in die Lage versetzt werden eigene BOINC-Projekte zu realisieren. Teil I (Grund-

lagen) liefert Ihnen eine Erläuterung der unterschiedlichen aktuellen Entwicklungsansätze und Informationen darüber, was mit BOINC möglich ist. Teil II (Technik) erläutert, wie Sie die BOINC-Quellen für ein eigenes BOINC-Projekt herunterladen und kompilieren. Teil III (BOINC) stellt die Programmierschnittstelle von BOINC vor, zeigt Ihnen Listings zahlreicher Funktionen und in welchem Kontext diese genutzt werden können, weiterhin wird der Unterschied zwischen den Implementierungen von Linux, Macintosh und Windows in Augenschein genommen. Teil IV (Praxis) präsentiert Praxisbeispiele von vier BOINC-Projekten. Von der Webseite zum Buch http://www.visualgrid.de/buch können alle Quellen heruntergeladen und von Ihnen modifiziert und verwendet werden. Die BOINC-Quellen sind nicht zu hundert Prozent frei von Fehlern. Aus diesem Grund gibt es zusätzlich ein Blog http://wp.visualgrid.de mit Einträgen zu diesen Fehlern und möglichen Lösungen. Weitere Informationen bezüglich meiner Arbeiten mit BOINC finden Sie auf meiner Webseite www.christianbenjaminries.de.

Bedanken möchte ich mich bei meinen Mentoren Prof. Dr. rer. nat. Christian Schröder und Prof. Dr. Vic Grout. Dank gebührt Alexandra Marcelina Boggasch und meiner Familie für die seelische Unterstützung in den letzten Monaten und Jahren. Ein Dank an die Arbeitsgruppe „Computational Materials Science and Engineering“ (CMSE): Thomas Hilbig, Thomas Englisch, Mikhail Tolstykh und Lisa Teich. Vielen Dank an das Bundesministerium für Bildung und Forschung (BMBF) und an die Arbeitsgemeinschaft industrieller Forschungsvereinigungen „Otto von Guericke“ e.V. (AiF) für die Möglichkeit der freien Entfaltung innerhalb meiner Arbeiten.

Seit dem ersten Tag bin ich begeistert von BOINC und der Idee, meine Berechnungsprobleme auf der ganzen Erde verteilt zu lösen. Ich hoffe, dass ich Ihnen mit diesem Buch die ersten Schritte mit der Arbeit mit BOINC erleichtere und möglichst gute Tipps geben kann, so dass auch Ihre Projekte von Erfolg gekrönt sind. Zuletzt noch ein kleiner Hinweis auf unser BOINC-Projekt an der Fachhochschule Bielefeld http://spin.fh-bielefeld.de. Wir würden uns sehr freuen, Sie in unserem Team begrüßen zu dürfen!

Bielefeld, März 2012 *Christian Benjamin Ries*

Inhaltsverzeichnis

Teil III BOINC

Teil I
Grundlagen

Kapitel 1
Clustertechnologien

Software is getting slower more rapidly
than hardware becomes faster.

Martin Reiser

Wenn Sie sich Gedanken um die Installation eines Clusters machen, so haben Sie wohl den Wunsch nach mehr Rechenleistung bzw. die Idee, Ihre Rechenlast auf mehrere Rechner möglichst optimal auszulagern (sog. *load-balancing*), oder Sie benötigen einfach mehr Speicherplatz, wie es zum Beispiel durch Storage Area Networks (SANs) erreicht wird. In den nachfolgenden Abschnitten präsentiere ich Ihnen einen kleinen geschichtlichen Überblick von den Anfängen der ersten Cluster bis zu heutigen Systemen und zeige unterschiedliche Möglichkeiten der Umsetzung. Weiterhin stelle ich Ihnen verschiedene Clusterarchitekturen vor, wie die Infrastruktur der Kommunikation zwischen Clusterknoten aussehen kann und mit welchen modernen Softwarebibliotheken in der Regel gearbeitet wird.

Dabei präsentieren sich alte Probleme in einem neuen Gewand. Zur Anfangszeit mussten Wege gefunden werden, die es den Programmierern ermöglichen, komplexe Berechnungen auf einer Maschine auszuführen und diese Aufgabe wiederum mit nur einem Prozessor zu lösen. Die Weiterentwicklung zu Mehrkernprozessoren, wobei zwei (Dual-Core) bzw. vier (Quad-Core) heutiger Stand der Technik sind, machen es nicht einfacher. Was noch gut mit Hilfe von einzelnen Prozessoren möglich war, kann nicht auf triviale Weise direkt auf solchen Mehrkernprozessoren gestartet werden. Eine Erwartungshaltung wie „das Problem wird jetzt schneller gelöst“ kann in den allermeisten Fällen nicht bestätigt werden und ist meist sehr enttäuschend. Die Schritte zur Entwicklung von komplexen Anwendungen werden durch neue Hardwaretechnologien immer aufwendiger. Dies liegt nicht an den Programmiersprachen oder an den Betriebssystemen, denn Mehrkernprozessoren können durch viele beliebige Programmiersprachen gehandhabt werden. Die Problematik steckt vielmehr in der Komplexität solcher Applikationen und den vielen kleinen Rädchen, an denen gedreht werden kann, um ein Optimum an Rechenleistung auf dem Weg zur Lösung zu erhalten.

Stellen Sie sich vor, Sie alleine bauen eine Wand, dabei müssen Sie einfach nur nacheinander die Steine übereinandersetzen. Dies ermöglicht es Ihnen, Reihe für Reihe in die Höhe zu streben, so dass Sie nach jedem Tag eine oder mehr Reihen geschafft haben. Nun stellen Sie sich vor, dass es von Ihnen zehn Kopien gibt oder einfach weitere Angestellte, jede Kopie beschäftigt sich mit dem Erstellen einer

C. B. Ries, *BOINC*, Xpert.press,
DOI 10.1007/978-3-642-23382-8_1,

Reihe. Soll die Mauer nun aus zehn Reihen bestehen, so wird jede Kopie von Ihnen für eine Reihe verantwortlich sein. Diese Aufteilung führt zwangsläufig zu Problemen, denn die Arbeit koordiniert werden, wenn Kopie fünf seine Reihe erstellen will, so müssen die darunterliegenden Reihen nicht zwingend im Ganzen fertig sein, aber die direkt untereinander liegenden sollten zumindest vorhanden sein, so dass auf diesen aufgebaut werden kann. Das heißt, alle Kopien müssen sich untereinander austauschen und müssen jeweils genügend Informationen erhalten um entscheiden zu können, ob und wie die Reihe weiter gebaut werden kann. Vielleicht kann die jeweilige Reihe nicht von einem Anfang links zum Ende rechts nacheinander gebaut werden. Es könnte sein, dass die Mitte schon so weit ist und an der Stelle weitergemacht werden kann. Nichtsdestoweniger muss dieser Umstand den anderen Teilnehmern bekannt gemacht werden, so dass beruhend auf diesen Informationen eine Entscheidungsfindung möglich ist. Die Frage ist auch, was passiert, wenn keine Steine mehr zur Verfügung stehen.

Man erkennt das Dilemma. Wie in jeder guten Projektorganisation bedarf es Techniken, so dass komplexe Vorgänge möglichst vereinfacht werden und eine effiziente Umsetzung ermöglicht wird. Aus diesen Bedürfnissen haben sich Softwarebibliotheken für die unterschiedlichsten Programmiersprachen entwickelt. Zur Vervollständigung werde ich zwei Standardbibliotheken kurz vorstellen: OpenMP in Abschn. 1.3.1.2 und Message Passing Interface (MPI) in Abschn. 1.3.1.3.

In vielen Büchern werden an dieser Stelle das **Amdahl'sche und Gustafson'sches Gesetz** genannt, welche aussagen, wie gut ein Programm parallelisiert werden kann [30, 34, 41]. Es geht in beiden Gesetzen um die Ermittlung des sogenannten Speedup, der als die Division der Zeit der sequenziellen Ausführung durch die Zeit der Ausführung der parallelen Ausführung mit p Prozessoren: $S_p = T_1/T_p$ definiert ist, wobei für $T_P = T_1 \cdot ((1-f)+f/p)$ gilt. Wir werden später sehen, dass wir uns im Fall von BOINC nicht sehr um diese Gesetzmäßigkeit zu kümmern haben. Um es vorwegzunehmen, BOINC ist prinzipiell für die Ausführung auf einzelnen Prozessorkernen ausgelegt. Es werden Mehrkernprozessoren unterstützt und ich zeige Ihnen auch, wie Sie mit diesen arbeiten können, aber eine optimale Umsetzung ist nicht Ziel dieses Buchs.

1.1 Geschichte

Begeben wir uns zurück in die sechziger Jahre des letzten Jahrhunderts. Die ersten Entwicklungsarbeiten, um verteiltes Rechnen zu ermöglichen, existierten und in den nachfolgenden Jahrzehnten stieg die Anhängerschaft der begeisterten Computerspezialisten. Es entstanden die ersten zusammengeschalteten Rechner, um Hochleistungsrechnen (HPC, *High-Performance-Computing*) zu ermöglichen [46]. Der Cray-1 wurde im Los Alamos National Laboratory geboren und in Betrieb genommen. Zugegeben, der damalige Stand der Technik lag weit hinter der heutigen Rechenleistung, denn eine Rechenleistung von immerhin 80 MFLOPS/s beeindruckt heutzutage niemanden.

In diesen Jahren musste noch viel Fleißarbeit und Geld in die Grundlagenforschung gesteckt werden. Prinzipiell kann man drei Bereiche definieren, in denen damals parallel gearbeitet wurde. Einige aus diesen drei Bereichen und nachfolgend aufgelisteten Technologien unterstützen uns heutzutage außerdem bei unserer täglichen Nutzung des Internets, sei es privat oder geschäftlich:

Netzwerktechnik Eine Verbindung zwischen einzelnen Rechnern wurde erst in den frühen Siebzigern durch das Ethernet ermöglicht. Die heutigen Standardkommunikationsprotokolle TCP/IP wurden in den Jahren nach 1978 an einigen wenigen Universitäten übernommen. Erst diese Technologien ermöglichten das Zusammenschalten und die Kommunikation zwischen mehreren Rechnerknoten. Weitere Technologien und Geräte wurden entworfen und entwickelt: *Hubs*, *Switches* und *Router*. Alle diese Komponenten haben Einfluss auf die Performance eines Clusters (vgl. Abschn. 1.2).

Plattformen Zu Beginn hatte ein Rechner nur einen Prozessor, und man nannte das Ganze Single-Core-Prozessor. Stetig wurde versucht die Prozessorgeschwindigkeit zu erhöhen. Dies hat auch jahrelang nach dem Moore'schen Gesetz [12] funktioniert. Der Cray-1 war der erste Clusterrechner im Jahr 1976 [183, Cray-1], der schon extra eine runde Bauweise besaß, um so die Kommunikationswege zu verkürzen. Im Jahr 2004 wurden dann die ersten massenmarkttauglichen Multi-Core-Prozessoren in den Vertrieb gebracht. Die Symbiose ließ nicht lange auf sich warten und heutzutage haben die Clustersysteme keine Single-Core-Prozessoren wie beim Cray-1 mehr, sondern folgen eher dem Trend, über mindestens vier oder mehr Rechenkerne in einem sogenannten Multi-Core-Prozessor zu bündeln. Prozessoren mit 4, 6, 8, 10, 16, 24, 48, 54 und noch mehr Kernen sind alle kommerziell erhältlich. Die Arbeit wird dadurch nicht einfacher und der nachfolgende dritte Bereich „Softwaretechnik" soll helfen, Vereinfachungen beim Umgang mit solchen komplexen Systemen zu entwickeln.

Softwaretechniken Wenn Sie sich mit der Programmierung von Single-Core-Prozessoren, Dual- oder Quad-Core-Prozessoren und zusammengeschalteten Rechnersystemen beschäftigen, dann wird Ihnen irgendwann klar, dass jeder zusätzliche Evolutionsschritt in Richtung mehr Rechenleistung durch parallel geschaltete Prozessoren bzw. Rechenkernen eine höhere Komplexität zur Folge hat und nicht mehr durch konventionelle Programmiersprachen auf einfache Weise unterstützt wird. Ihre Anwendungen arbeiten womöglich auf unterschiedlichen Rechnerplattformen (unterschiedliche Betriebssysteme, eine 32-Bit- oder 64-Bit-Architektur, eine ARM- oder SPARC-Architektur, usw.), die gegebenenfalls weder im selben Raum noch im selben Land zu finden sind. Sie müssen die Kommunikation handhaben, die Synchronisation zwischen Prozessen verwalten, vielleicht eine Berechnung zum gleichen Zeitpunkt starten, die Berechnung auf alle Rechnern pausieren oder direkt stoppen. All diese Problematiken treffen auch zu, wenn Sie mit einem Mehrkernprozessor arbeiten, im Unterschied zu mit einem Netzwerk vernetzten Systemen findet in solchen Prozessoren die Kommunikation zwischen den einzelnen Kernen mit Hilfe unterschiedlicher Systembusse (Datenbus, Adressbus und Steuerbus) oder Kommunikationstechniken (Shared-Memory, FIFOs, PIPES, etc.) statt [35, 43, 44]. Aus dem Wunsch her-

aus, dies alles auf einfache Weise zu ermöglichen, wurden Softwareschnittstellen entwickelt; sehr bekannte sind u. a. die Multi-Thread-Programmierung, Open Multi-Processing (OpenMP), Parallel Virtual Machine (PVM) und Message-Passing-Interface (MPI). Der Abschn. 1.3 wirft einen genaueren Blick auf eine Auswahl dieser Techniken.

Am 17. Mai 1999 wurde am SETI-Institut[1] ein Projekt mit dem fast gleichlautenden Namen SETI@home gegründet [183, SETI@home]. Ziel dieses Projekts war das Aufteilen der Arbeit auf mehrere Rechner, möglichst mit einen geringem Kostenanteil, denn das SETI-Institut wirtschaftet nur aus privaten Mitteln und wird durch keine staatlichen Förderungen am Leben gehalten. Leider konnte SETI@home bis heute keine Erkenntnisse bezüglich außerirdischen Lebens liefern.

Aus den Arbeiten von David P. Anderson (dem Urvater von BOINC), wissenschaftlichen Mitarbeitern von unterschiedlichen Forschungsinstituten, Studierenden verschiedener Universitäten und einer ungezählten Menge weiterer Freiwilliger aus der ganzen Welt hat sich basierend auf der SETI@home-Software das BOINC-Framework (Berkeley Open Infrastructure for Network Computing) hervorgehoben und wird stetig weiterentwickelt. BOINC ist in der Zeitrechnung der Computerindustrie schon ein wenig in die Jahre gekommen. Die Arbeiten zu BOINC wurden im Februar 2002 bekannt gemacht und ein erstes offizielles Release am 10. April 2002 veröffentlicht. Das erste BOINC-basierende Projekt war Predictor@home; es hat seine Arbeit am 10. Juni 2009 eingestellt und existiert seither nicht mehr. Nicht lange nach dem Start von Predictor@home wurden weitere BOINC-Projekte etabliert und lieferten in kürzester Zeit neue wissenschaftliche Erkenntnisse (s. Abschn. 2.4). Zu Beginn standen wissenschaftliche Fragestellungen im Vordergrund, doch nach und nach wurde BOINC auch für das Abarbeiten von industriellen Fragestellungen und das Lösen von nicht-wissenschaftlichen Problemen verwendet, z. B. das Erstellen von Filmen oder die Ermittlung von Schachstrategien [100, 101, 143].

Die Prinzipien von BOINC basieren auf dem Ansatz des Public-Resource Computing (PRC), um dadurch die Rechnerressourcen von freiwilligen Teilnehmern nutzen zu können, um mit deren Hilfe komplexe Berechnungen zumindest kostengünstiger durchzuführen. Hauptziel von PRC ist das Einbeziehen von nicht steuerbaren und kontrollierbaren Rechnerressourcen. Es handelt sich dabei meist um die Rechnerressourcen der heimischen Rechner von Privatpersonen. Daher stammt auch das Suffix „@home“ der meisten BOINC-Projekte, sprich „zu Hause“. Viele Wissenschaftsprojekte erfordern viel Geduld und Ausdauer der Forscher und freiwilligen Helfer. Manche Projekte haben sehr ehrgeizige Ziele, wie z.B. das von SETI@home. Geduld ist umso wichtiger, weil in vielen Projekten eine kaum zu beziffernde Menge an Berechnungen durchgeführt werden soll oder muss. In dem BOINC-Projekt Spinhenge@home kann ein Simulationsmodell, mit tausenden von unterschiedlichen Parametersätzen, Monate benötigen [171].

[1] „Search for Extraterrestrial Intelligence“ zu Deutsch „Suche nach außerirdischer Intelligenz“.

1.2 Infrastruktur

Cluster sind in der Informatik eine Ansammlung von Prozessoren, die durch koordinierte Zusammenarbeit große Probleme schnell lösen können [41]. In der Regel soll eine höhere Performance und mehr Rechenleistung zur Lösung von komplexen Problemen oder mehr Speicherkapazität erreicht werden. Letzteres ist – nicht zwingend – durch ein Storage Area Network (SAN) zu erreichen und wird hier auch nicht weiter behandelt. Hingegen können die ersten beiden Punkte auf dem Weg zur Lösung sehr viele unterschiedliche Ausprägungen annehmen und zudem von der Problemstellung selber abhängen. Ein Cluster „von der Stange“ ist zwar zu gebrauchen, allerdings erhalten Sie als Zusatz vielleicht die Büchse der Pandora mit allen möglichen Lasten und Problemen, welche die Ausgangssituation vielleicht noch verschlimmern:

- Die Steuerung der Kommunikation innerhalb des Clusters nimmt evtl. einen hohen Anteil der Rechenzeit in Anspruch und löst Ihr Berechnungsproblem möglicherweise noch langsamer, weil mutmaßlich zu wenig Speicher für das Zwischenspeichern von Nachrichten vorhanden ist oder die falsche Technologie für die Verkabelung der Rechnerknoten gewählt wurde und dadurch die Kommunikation ausgebremst wird.
- Sie erhalten nicht den gewünschten Peak, wenn Sie ihn wirklich benötigen. Es gibt in der Regel bei der Arbeit auf einem Cluster einen sogenannten *Scheduler*, welcher die Verantwortung des Koordinierens übernimmt, also quasi als Projektmanager agiert. Dieser könnte womöglich standardmäßig einen nicht optimalen Algorithmus verwenden, was dazu führen kann, dass keine ordentliche Lastverteilung innerhalb des Clusters erreicht wird.

Man erhofft sich durch das Zusammenschalten ein schnelleres Lösen von Berechnungsproblemen. Es existieren verschiedene Ansätze für den Aufbau eines Clusters. Abbildung 1.1 zeigt drei mögliche Netzwerktopologien für Clustersysteme. Die Bus-Topologie und die Ring-Topologie sind nicht mehr ganz aktuell und wer-

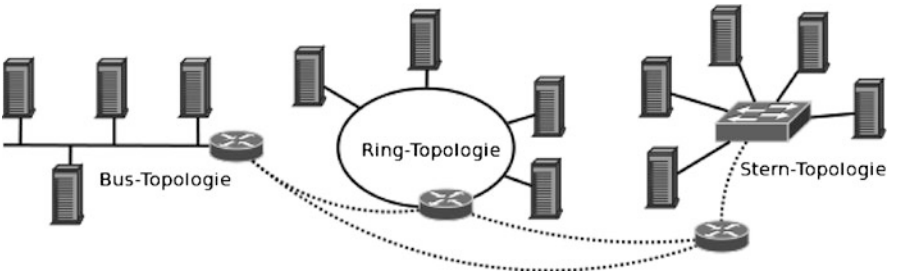


Abb. 1.1 Drei Standardtopologien für den Aufbau von Clustersystemen: (1) Bus-Topologie, (2) Ring-Topologie und (3) Stern-Topologie. Die einzelnen Topologien können autonom aufgebaut sein, eine Kommunikation kann allerdings auch zwischen Computern aus den jeweiligen Topologien stattfinden, verdeutlicht durch die *gestrichelte Linie*

den bei neueren Netzwerkinstallationen schon gar nicht mehr beachtet oder empfohlen. Sie erkennen zudem die gestrichelten Verbindungslinien; es ist natürlich möglich, dass unterschiedliche Netzwerktopologien miteinander verknüpft werden. Unnötig zu sagen, dass solch ein Vorgehen gravierenden Einfluss auf die Performance einer Applikation haben kann. Noch vor ein paar Jahren waren Cluster eine Randerscheinung, doch die Nachfrage nach immer mehr Rechenleistung, um sogenannte „große“ Berechnungsprobleme zu lösen, führte zu der Entwicklung von unterschiedlichen Clustersystemen:

zentral-organisiert Hierbei handelt es sich um ein Cluster, wie Sie es sicherlich schon kennen, wenn Sie schon einmal vom – im Jahr 2002 eingeweihten – Earth-Simulator [119] gehört haben. Für mich waren diese Benchmark-Werte einfach unglaublich, und bedenkt man erst, das der Earth-Simulator eine Taktung von 35,61 Teraflops hat und heute gar nicht mehr in der Liste der Top500 auftaucht, ist das schon erstaunlich [133]. In der Regel handelt sich um große Gebäude mit viel Platz, in denen tausende von Rechnerknoten installiert sind.
BOINC kann u. a. in diese Kategorie eingeordnet werden; auch wenn die Rechnerknoten sich nicht in einem großen Gebäude befinden, so wird ein BOINC-Projekt doch zentral gesteuert.

dezentral-organisiert *BitTorrent* ist ein Peer-to-Peer-Clustersystem, bei dem sich die Teilnehmer als *Seeder* und *Peer* am Netzwerk anmelden. Das heißt, die einzelnen Teilnehmer bauen Verbindungen zu den nächsten Nachbarn auf, Anfragen werden von jedem Knoten weitergeleitet. Wenn also ein Host eine Anfrage, zum Beispiel nach einer Datei, nicht beantworten kann, so wird die Anfrage weitergeleitet und die Antwort an den Fragenden zurückgeliefert.

1.3 Varianten

Es stellt sich doch die Frage: Was kann ich wie lösen und was brauche ich dafür? Nicht alle Probleme lassen sich durch einen einzigen allgemeinen Lösungsansatz lösen. Nachfolgend erhalten Sie einen kurzen Überblick über unterschiedliche softwaretechnische Möglichkeiten, um diese Frage ein wenig für sich beantworten zu können.

1.3.1 High-Performance-Computing

High Performance Computing (HPC) (zu Deutsch Hochleistungsrechnen) ist eine sehr interessante und anspruchsvolle Disziplin. Es existieren eine Handvoll an Softwareschnittstellen, mit denen heutzutage HPC-Anwendungen implementiert und ausgeführt werden. Generell wird zwischen zwei Umsetzungsprinzipien unterschieden – gemeinsam und verteilt genutzter Adressraum –, die sich auch in den Softwareschnittstellen widerspiegeln.

Gemeinsamer Adressraum In Applikationen mit gemeinsamem Adressraum teilen sich die beteiligten Prozesse einen Speicherbereich. In diesem Speicherbereich werden die Daten für die Berechnungen gehalten. Die Applikation ist für die Verwaltung und Synchronisation der Zugriffe auf diese Bereiche verantwortlich. Jeder Prozess kann **alles** „sehen“ und modifizieren. Die Technologien *Multi-Thread-Programmierung* (s. Abschn. 1.3.1.1) und *Open Multi-Processing* (s. Abschn. 1.3.1.2) basieren auf dem Prinzip des gemeinsamen Adressraums.

Verteilter Adressraum In Applikationen mit verteiltem Adressraum besitzen die jeweiligen Prozesse einen eigenen Speicherbereich. Dieser kann nur von den jeweiligen Prozessen modifiziert werden. Der Applikationsentwickler ist dafür zuständig, dass das mathematische Problem von den jeweiligen Prozessen gelöst werden kann. Alle Informationen zur erfolgreichen Lösung müssen den jeweiligen Prozessen zur Verfügung gestellt werden. Message Passing Interface (MPI) (s. Abschn. 1.3.1.3) und BOINC fallen in diese Einordnung.

1.3.1.1 Multi-Thread-Programmierung

Eine der Möglichkeiten zur Programmierung mit gemeinsamem Adressraum ist die Verwendung von Threads, falls diese durch das Betriebssystem unterstützt werden. Viele heutige Betriebssysteme wie UNIX und Windows unterstützen die Programmierung mit Threads als Multi-Thread-Programmierung (*multithreaded programming*) und stellen für den Benutzer Programmierschnittstellen (API, *Application Programming Interface*) zur Verfügung. Threads werden als „leichte Prozesse“ bezeichnet, vgl. [42]. Ein Thread ist eine Art Kopie innerhalb eines Prozesses, der nur einen Teil des Prozesses dupliziert. Dieser beinhaltet u.a. folgende gemeinsam genutzte Daten innerhalb eines Prozesses: Prozessanweisungen und Speicherbereiche. Jeder Thread besitzt seine eigene Kennung, Register-Sets inklusive Programmzähler, Stack-Zeiger und Prioritäten. Alle Threads innerhalb eines Prozesses nutzen den gleichen globalen Hauptspeicher. Damit wird die gemeinsame Nutzung von Informationen erleichtert, wobei in diesem Zusammenhang das Problem der *Synchronisation* auftaucht. Durch die Erstellung von Threads können verteilte Applikationen erstellt und den Anforderungen entsprechend priorisiert werden. Bei einem Prozessor mit mehr als einem Kern kann eine Verteilung der Threads auf die jeweiligen Kerne stattfinden. Dies kann die Ausführung beschleunigen, aber auch zu einem höheren Verwaltungs- und Synchronisationsaufwand führen. Abschnitt 7.4 behandelt Multi-Threads im Rahmen einer BOINC-Anwendung.

1.3.1.2 Open Multi-Processing

Open Multi-Processing (OpenMP) ist eine Spezifikation von Übersetzungsdirektiven, Bibliotheksfunktionen und Umgebungsvariablen, die von einer Gruppe Soft- und Hardwareherstellern mit dem Ziel entworfen wurde, einen einheitlichen Standard für die Programmierung von Parallelrechnern mit gemeinsamem Adressraum

zur Verfügung zu stellen [55]. Das Programmiermodell von OpenMP basiert auf parallel arbeitenden Threads. Die Abarbeitung eines mit Hilfe von OpenMP formulierten Programms beginnt mit der Ausführung eines sogenannten *Master-Threads*, der das Programm so lange sequentiell ausführt, bis das erste parallel-Konstrukt auftritt. Mit den durch OpenMP zur Verfügung gestellten Mechanismen zur Steuerung der Parallelität können Programme formuliert werden, die sowohl sequentiell als auch parallel ausgeführt werden können. Es können Programme geschrieben werden, die nur bei einer parallelen Ausführung das gewünschte Ergebnis errechnen. Der Programmierer ist dafür verantwortlich, dass die Programme korrekt arbeiten. Dies gilt auch für die Vermeidung von Konflikten, *Deadlocks* oder zeitkritischen Abläufen [43].

Listing 1.1 Ausschnitt eines OpenMP-Programm zur parallelen Berechnung einer Matrix-Multiplikation

```
1  #pragma omp parallel private(row, col, i)
   {
     #pragma omp for schedule(static)
     for( row=0; row < size; row++ ) {
       #pragma omp parallel shared(MA, MB, MC, size)
6      {
         #pragma omp for schedule(static)
         for( col=0; col < size; col++ ) {
           MC[row][col] = 0.0;
           for( i=0; i < size; i++ )
11           MC[row][col] += MA[row][i] * MB[i][col];
         }
       }
     }
   }
```

Die verwendeten OpenMP-Anweisungen haben folgende Bedeutungen [55]:

#pragma omp parallel private() Diese Direktive bewirkt, dass der angegebene Anweisungsblock parallel ausgeführt wird. Wird die Arbeit nicht explizit verteilt, so führen alle Threads die gleichen Berechnungen mit evtl. unterschiedlichen privaten Daten aus. Private Daten werden mit dem Parameter `private()` definiert. In diesem Beispiel sind die Variablen `row`, `col`, `i` in jedem Prozess eigenständig und beeinflussen sich nicht gegenseitig.

#pragma omp parallel shared() Diese Direktive ist ähnlich der vorangegangenen mit dem Unterschied, dass hier von den Prozessen gemeinsam genutzte Variablen mit `shared()` definiert werden. Dies betrifft die Matrix-Variablen `Ma`, `MB`, `MC` und die Dimension der Matrizen, die durch `size` definiert wird.

#pragma omp for schedule() Innerhalb eines parallelen Bereichs können die durchzuführenden Berechnungen mit Hilfe von speziellen Direktiven zur Verteilung der Arbeit auf die ausführenden Threads verteilt werden. Diese Direktive beschreibt eine for-Schleife, im Listing 1.1 mit statischer Verteilung der Schrittweite in der for-Schleife.

1.3.1.3 Message Passing Interface

Message Passing Interface (MPI) hat sich quasi als Industriestandard durchgesetzt. MPI ist eine Abstraktion eines Parallelrechners mit verteiltem Speicher. Ein MPI-Programm besteht aus einer Anzahl von Prozessen mit zugeordneten lokalen Daten. Jeder Prozess kann auf seine lokalen Daten zugreifen und mit anderen Prozessen Informationen durch das explizite Verschicken von Nachrichten austauschen. Nachrichten können mit Hilfe von Punkt-zu-Punkt- und globalen Kommunikationsoperationen ausgetauscht werden.

Kosinus-Integration mit MPI

Die Integration einer Kosinus-Funktion kann exakt oder numerisch durchgeführt werden. Die Funktion $y = \cos(x)$, integriert nach dy/dx, liefert folgende Stammfunktion $y' = -\sin(x)|_{UG}^{OG} = \sin(OG) - \sin(UG)$. Für $UG = 0$ und $OG = 2\pi$ ist die exakte Lösung $y' = 0$. In der Regel ist die Stammfunktion nicht so trivial wie die der Kosinus-Funktion. In technischen Anwendungen sind meist gekoppelte Differentialgleichungen in einem Gleichungssystem (GLS) vorhanden, wie z. B. die Navier-Stokes-Gleichungen. Diese sind nicht einfach exakt lösbar und benötigen ein numerisches Lösungsverfahren. Das Integral der Kosinus-Funktion kann numerisch gelöst werden. Zu diesem Zweck werden kleine Rechtecke in den Grenzen UG, OG gebildet, die die Fläche innerhalb der Kosinus-Funktion berechnen, wie in Abb. 1.2 zu sehen. Die Summe der Rechteckflächen ist eine Annäherung an die Lösung des Integrals. Je mehr Unterteilungen in den jeweiligen Prozessen berechnet werden, desto genauer wird das Ergebnis. Das Listing 1.2 liefert eine exemplarische Umsetzung einer numerischen Integration mit Hilfe von MPI. In den Zeilen 12 bis 15

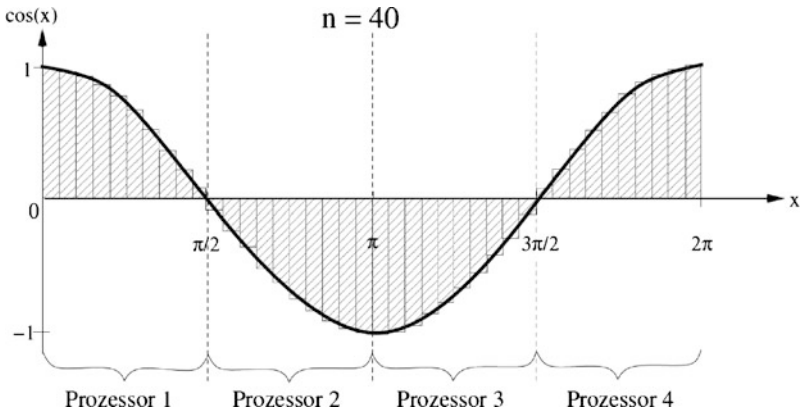


Abb. 1.2 Eine numerische Integration einer Kosinus-Funktion auf vier Prozesse aufgeteilt, mit jeweils 10 Teilberechnungen [6]; ein höherer Teilungsfaktor *n* und mehr teilnehmende Prozessoren sind möglich

wird MPI initialisiert und für die jeweiligen Prozesse die Identifizierungsnummer (ID) ermittelt, da die Funktion `MPI_Init(&argc,&argv)` die Prozesse für die verteilten Rechner erstellt. Mit der ID kann in Zeile 20 die Untergrenze der Prozessunterteilungen berechnet werden. Der linke Teil in Zeile 19 ermittelt die Schrittweite der Iterationen, diese entsprechen der Breite eines Rechtecks. Der rechte Teil in derselben Zeile berechnet den Abstand zur jeweiligen Obergrenze in einer Iteration. Die Zeilen 24 bis 30 werden auf dem Rechner ausgeführt, der die Ausführung des Programms gestartet hat und empfängt in der Zeile 27 die Einzelergebnisse der jeweiligen Prozesse aus Zeile 33 und addiert diese in Zeile 29 auf. Das Ergebnis wird in Zeile 31 ausgegeben.

Listing 1.2 Ausschnitt einer numerischen Kosinus-Integration mit dem Message Passing Interface (MPI) in der Programmiersprache C

```
double integral(double lowerBound, int n, double step) {
  double integ = 0.0, step2 = step/2.;
  for(int j=0; j<n; j++) {
    double lowerBound_ij = lowerBound + j*step;
    integ += cos(lowerBound_ij + step2)*step;
  }
  return (integ);
}

int main(int argc, char**argv) {
  ...
  MPI_Status status;
  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &myID);
  MPI_Comm_size(MPI_COMM_WORLD, &prozessAnzahl);

  int num = n / prozessAnzahl;

  step = (b-a) / n;
  my_range = (b-a) / prozessAnzahl;
  my_a = a + myID * my_range;
  my_result = integral( my_a, num, step);

  if(myID == 0) {
    my_result = 0.0f;
    for (int i=1; i < prozessAnzahl; i++) {
      MPI_Recv(&my_result, 1, MPI_REAL, i, \
               11111, MPI_COMM_WORLD, &status);
      result += my_result;
    }
    printf("Ergebnis der MPI Berechnung = %.18f\n", result);
  } else {
    MPI_Send(&my_result, 1, MPI_REAL, \
             0, 11111, MPI_COMM_WORLD);
  }
  MPI_Finalize();
}
```

1.3.2 Grid-Computing

Grid-Computing ist der kleine Cousin des Cloud-Computing (vgl. Abschn. 1.3.3) in der Familie der Cluster-Systeme und von der Funktionalität her angelehnt an das

Stromnetz (engl. *electrical power grid*). Sie haben in beiden Grids einen Zugriff auf die Ressourcen, und in beiden Fällen soll sich das Netz – im besten Fall – selber regulieren, so dass ein Maximum an Ressourcen zur Verfügung stehen.

Dieses Grid-Paradigma hat sich allerdings nie wirklich am Massenmarkt etabliert. Jenseits des Kommerziellen existiert mit dem wissenschaftlichen Rechnen ein Standbein und die Daseinsberechtigung für das Grid-Computing. Grid-Computing hat seine Wurzeln in unterschiedlichen wissenschaftlichen Betrieben. Wenn kein Geld vorhanden ist, aber eine Zielsetzung möglichst schnell umgesetzt werden muss, so wird man einfallsreich und entwickelt Ideen um seine Wünsche zu erfüllen. Mit BOINC ist dies möglich und ein Grid kann erstellt werden. Die große Herausforderung ist es, die Rechenleistung für spezielle Anwendungsgebiete bzw. Szenarien nutzbar zu machen, die im Internet oder im Intranet verborgen liegen.

Im Jahr 2009 wurden rund 5,4 Mio. und im Jahr 2010 etwa 7,2 Mio. Smartphones verkauft – und das nur in Deutschland. Für das Jahr 2011 wird ein Verkauf von mindestens 10 Mio. Geräten erwartet [194]. Daraus können wir eine kleine Beispielrechnung erstellen. Angenommen, jedes dieser Smartphones hat mindestens 1 GHz, wie das heutige Smartphones auch tatsächlich besitzen [183, Apple A5]. Weiterhin nehmen wir an, uns stehen 50 Prozent der Rechenleistung zur Verfügung, so hätten wir eine theoretische Rechenleistung von $10^6 \cdot (0{,}5 \cdot 10^9)$ GHz zur Verfügung, das entspricht $5.000.000\,\text{Giga-Hz} = 5000\,\text{Tera} - \text{Hz} = 5\,\text{Peta-Hz}$ und ist eine gewaltige Summe, nur für den deutschen Smartphone-Markt.

Dieser Wert beruht auf der Annahme „GFlops = GHz“; dies kann man nicht verallgemeinern, da der Prozessor mehr Arbeit leisten muss, auch wenn Sie als Entwickler aber eigentlich nur eine Anweisung ausführen wollen. Zum Rechnen muss der Prozessor Befehl und Daten aus dem Speicher holen und in seinen Registern unterbringen. Diese vereinfachte Darstellung heißt das *von Neumann-Modell* eines Rechners. Bei einer Ausführung einer Anweisung müssen mindestens folgende Schritte folgen [30]:

- Hole nächsten Befehl aus dem Speicher in das Befehlsregister.
- Interpretiere diesen Befehl.
- Lade evtl. vorhandene Argumente aus dem Speicher in ein Register.
- Führe den Befehl aus und schreibe das Resultat in ein weiteres Register.
- Schreibe ein evtl. vorhandenes Resultat zurück in den Speicher.

Sie sehen, eine Anweisung hat einen Overhead, der nicht zu vernachlässigen ist. Es gibt Benchmarks, mit denen Sie einen ungefähren Wert für die GFlops ermitteln können. Linpack ist eine Software, um solche Benchmark-Messungen durchzuführen. Dies wird durch das Verwenden von Standardalgorithmen ermöglicht, z. B. das Lösen von linearen Gleichungssystemen [142]. In diesem Buch werden wir keine Performance-Messungen durchführen. Erwähnenswert ist dies allerdings, denn auf solchen Messungen beruhen die Statistiken der Top500-Liste.

Zurück zu unseren 5 Peta-Hz; die Rechnung kann man ins Unendliche fortsetzen. Viele Geräte besitzen zusätzlich eine Graphical Processing Unit (GPU) mit 200 MHz und mehr, mit durchaus sehr unterschiedlichen Performance-Werten (Apple iPad 2, Samsung Galaxy S 2, Samsung Infuse 4G, Galaxy Tab 10.1, LG Opti-

mus 3D, Nexus S, Apple iPhone 4) [68]. Bedenken Sie, dass heutzutage eine breite Palette an verschiedenen Geräten mit solchen Chipsätzen ausgestattet sind: Fernseher, Receiver für Satelliten-Empfang, Geräte für die Haussteuerung und viele mehr. Das hört sich alles vielversprechend an, allerdings sind komplexe Berechnungen auf Smartphones eine Herausforderung für sich, denn der Akku eines solchen Gerätes wäre in kürzester Zeit leer und entsprechend wäre das Gerät nicht mehr verfügbar. Der Dauerbetrieb an einer Steckdose ist nötig. Mehr noch, eine Applikation, die auf den einzelnen Geräten arbeiten soll, muss auch die unterschiedlichsten Geräte unterstützen; eine solche heterogene Systemumgebung ist schwierig zu handhaben. In einem solchen Fall dauert die Entwicklung einer Applikation wohl ein wenig länger und steht in keinem positiven und sinnvollen Kosten-Nutzen-Verhältnis.

Bei einem Grid müssen die einzelnen Rechner oder Cluster nicht an einem Ort zur Verfügung gestellt werden. Die Idee ist, dass der Zusammenschluss von regionalen, nationalen und globalen (heterogenen) Rechnern, Daten und Embedded-Ressourcen durch ein Kommunikationsnetz zu einem großen Rechnerverbund zusammengeschaltet werden und an der Lösung eines Problems arbeiten.

Dass solch ein Zusammenschalten von Hunderten, Tausenden oder Millionen – wohl auch heterogenen – Rechnern nicht ganz trivial ist, kann sich sicherlich jeder vorstellen. Seit der Idee in den späten neunziger Jahren wurden eine Reihe von Hilfswerkzeugen in Form von Softwarebibliotheken entwickelt. Mit Hilfe des open-source Globus-Toolkit [7] kann ein Grid geplant und umgesetzt werden. Weitere Werkzeuge und Softwarebibliotheken existieren und erleichtern das Arbeiten mit Grids: g-Eclipse [10], Oracle Grid Engine [157] und gLite [130].

Klassifikation von Grid-Systemen

Ein Grid kann in drei Klassifikationen unterschieden werden [38], welche sich auch in den anderen Clusteransätzen wiederfinden. Alle drei Klassifikationen wurden in der jüngsten Vergangenheit zu sogenannten Cloud-Umgebungen weiterentwickelt. In Abschn. 1.3.3 geben ich Ihnen einen Überblick zu Clouds.

Grid für Berechnungen Ein Grid mit hoher Rechenleistung. Die Rechenleistung wird durch das Zusammenschalten von mehreren Clustern erreicht, welche geografisch nicht am selben Ort sein müssen und weltweit verteilt sein können.

Grid für Datenmengen Ein Grid mit der Möglichkeit, hohe Speicherkapazitäten anzubieten, ähnelt prinzipiell der ersten Klassifikation mit dem Unterschied, dass wir hier von Speicherkapazität anstatt Rechenleistung sprechen.

Grid für Dienste Ein Grid, welches Anwendungen anbietet. Anwender melden sich an einem solchen Grid an und bekommen womöglich eine komplette Systemumgebung zur Verfügung gestellt; auf der lokalen Rechenmaschine befindet sich nur eine Minimalinstallation einer Systemumgebung.

1.3.3 Cloud-Computing

„Cloud computing is partly a business model and partly a usage model“ [13]. Die Aussage verrät etwas über den Hype, welcher hinter dem Schlagwort Cloud-Computing steckt. Jason Stowe gründete 2005 das Unternehmen Cycle Computing [113]. Seine Idee: Kunden hatten nie die Rechenressourcen zur Verfügung, wenn sie gebraucht wurden. Daraufhin entwickelte Stowe eine Cluster-Management-Software, um die Größe eines Clusters ändern zu können, und zwar nach Abhängigkeit der Arbeitsaufträge in einer Warteschlange – der Startschuss für das heutige Cloud-Computing. Stowe kategorisiert die Wünsche der Nutzer in zwei Bereiche: die Sprinter und die Marathonläufer. Der Sprinter wünscht sich, dass seine Arbeitsaufträge die beste Parallelität aufweisen und die bestmögliche Ausführungszeit erhalten; sprich, so schnell wie möglich gelöst werden. In diesem Fall ist jedes Quäntchen Latenzzeit des Speicherzugriffs oder bei der Kommunikation zwischen Rechenknoten zu minimieren, um das Optimum zu erreichen. Der Marathonläufer hingegen hat einfach zu parallelisierende Anwendungen, wobei die Zielsetzung meist darauf beruht, dass eine gewaltige Datenmenge verarbeitet werden kann; nicht zwingend erforderlich ist hierbei eine hohe Performance.

Aber was ist nun Cloud-Computing, zum Beispiel für Sie? Vielleicht haben Sie einen E-Mail-Account bei Google oder arbeiten mit Facebook, schon dann sind Sie in der „Cloud“ unterwegs.

Cloud as a Service

Allgemein kann man drei Systeme unterscheiden, welche heute ausgiebig erforscht werden und in die reichlich Energie fließt:

Infrastructure as a Service (IaaS) Hierbei wird die gesamte Infrastruktur zur Verfügung gestellt. Als Beispiel seien hier Angebote wie Virtuelle Server (VServer) oder Root-Server erwähnt. Sie erhalten auf Wunsch einen kompletten Server, um damit arbeiten zu können. Eine Problematik hier könnte sein, dass Sie zum Beispiel nicht wissen, ob – wenn Sie mehr als einen Server besitzen – diese auch wirklich Nachbarn sind. Planen Sie etwa die Installation einer MPI-Applikation – welche zudem zeitkritisch ist –, so könnte dies zu Problemen führen, wenn Ihnen Ihr Service-Provider womöglich zwei oder mehr Server in unterschiedlichen Rechenzentren, eventuell in unterschiedlichen Städten oder noch schlimmer in unterschiedlichen Ländern zuweist.

Sicherlich können Sie mit Ihrem Service-Provider vorab eine Lösung besprechen, allerdings erwähne ich dies auch nur, damit Sie dafür ein etwas entwickeln und sich nicht wundern, wenn es Probleme gibt und die Rechenzeit vielleicht sogar höher ausfällt als bei Ihrem Rechnersystem im Büro oder zu Hause.

Platform as a Service (PaaS) Es handelt sich hierbei um sogenannte virtuelle Systeme, z. B. VServer. VServer werden heutzutage von zahlreichen Internet-Dienstanbietern angeboten, und es besteht in der Regel die Möglichkeit, zwi-

schen einem auf Windows oder Linux basierenden System zu wählen. Weiterhin besteht oft die Möglichkeit, dass Sie sich weitere Software durch mehrere Mausklicks zusätzlich installieren oder sich direkt auf einer Remote-Desktopumgebung oder einem Remote-Terminal bewegen.

Software as a Service (SaaS) Es handelt sich um Software, die im Internet, eben in der „Cloud", installiert ist. Diese rufen Sie z. B. durch einen Browser auf und können damit arbeiten. Wie schon erwähnt, ist ein Google-Kalender oder Ihre E-Mail-Adresse bei Web.de oder Google eine solche Anwendung.

Die Reihenfolge der Auflistung (IaaS → PaaS → SaaS) spiegelt zudem den Weg des nächsthöheren Abstraktionslevel wider. Je weiter links Sie also einsteigen, desto mehr Aufwand müssen Sie betreiben um die Infrastruktur, die Plattform oder die Software erfolgreich einzurichten und zu warten.

Gefahren

Wie bei jedem Hype gibt es womöglich zu schnelle Entwicklungen und es können schnell gravierende Fehler passieren. In einigen Fällen kam es zu Datenverlust und Problemen der Erreichbarkeit von Online-Firmenpräsenzen, was sicherlich Umsatzeinbußen für mindestens diese Zeit bedeutete – auf den entsprechenden Imageverlust will ich gar nicht erst eingehen. Problematiken in den letzten Monaten und Jahren waren:

- Bei einem US-Partner der Telekom kam es im Jahr 2009 zu Datenverlust bei Datensätzen von tausenden von Kunden [102, 104]. Es kam zum Verlust von E-Mails, Bildern, persönlichen sowie geschäftlichen Kontakten und Terminen. Der Grund war damals ein nicht vorhandenes Backup-System, auf das man aus Kostengründen verzichtet hat.
- Sicherlich ist man nicht immer davor gefeit, dass ein System mal ausfallen kann, allerdings werben die Anbieter von Cloud-Systemen damit, dass diese so gut wie ausfallsicher seien. Leider ist es erst vor Kurzem bei zwei der großen Konkurrenzprodukte von Amazon (AWS Cloud) [66, 67] und Microsoft (Azure) [188] zu großen Problemen gekommen, als ein Blitzschlag die Systeme lahmgelegt hat. Teile der Cloud-Dienste waren stundenlang nicht mehr erreichbar [114].
- Was tun Sie, wenn Ihre Daten bei einem Anbieter sind, der plötzlich Konkurs geht und seinen Dienst einstellt oder womöglich die Konditionen ändert? Im schlimmsten Fall erfahren Sie das womöglich erst, wenn Sie auf die Daten zugreifen wollen und dies nicht mehr möglich ist. Dies ist einigen Kunden vom Musikstreamingdienst Lala passiert [189]: Apple hatte das Unternehmen aufgekauft und den Dienst eingestellt. Folge: die Daten waren weg.

Diese Liste kann beliebig erweitert werden, allerdings sind das alles Punkte, auf die Sie rein theoretisch auch im privaten Bereich achten müssten, was leider viele aus Kostengründen nicht tun wollen oder können.

Kapitel 2
BOINC ist Public-Resource Computing

BOINC (Berkeley Open Infrastructure for Network Computing) is a software system that makes it easy for scientists to create and operate Public-Resource Computing projects. It supports diverse applications, including those with large storage or communication requirements. PC owners can participate in multiple BOINC projects, and can specify how their resources are allocated among these projects.

David P. Anderson [4]

Das Public-Resource Computing (PRC) ist eine vielversprechende Möglichkeit, die im vorherigen Abschn. 1.3.2 erwähnten versteckten Rechenleistungen für wissenschaftliche Anwendungsgebiete nutzbar zu machen. Verschiedene Umsetzungen prägen das Bild von PRC-Projekten, folgende Möglichkeiten stehen als Grundgedanke bei der Verwendung von BOINC zur Verfügung:

Volunteer-Computing Durch das Volunteer-Computing, was soviel bedeutet wie *freiwilliges Rechnen*, melden sich Benutzer an ein oder mehreren Projekten an und spenden sozusagen ihre Rechnerressourcen den jeweiligen wissenschaftlichen Projekten und rechnen für diese.

Desktop-Grids Ein Desktop-Grid kann wie das Volunteer-Computing ausgeprägt sein. Allerdings handelt es sich bei solch einem Grid um ein eigenständiges Netzwerk, z. B. ein Firmennetzwerk oder das Campus-Netzwerk eines Institutes (Universität, Volkshochschule, Bibliothek). In diesem Fall wird auf den Rechnern der Firma oder dem Institut zusätzliche Arbeit verteilt. Sinnvoll erscheint das in dem Zusammenhang, wenn man bedenkt, dass die meisten Rechner sowieso viel Zeit im Wartezustand verbleiben und eher selten durchgehend auf Volllast arbeiten. Die Sekretärin oder der Pförtner müssen im Normalfall keine größeren Applikationen ausführen und daher können ihre Rechner zusätzlich für andere Aufgaben zusätzlich genutzt werden.

2.1 Anforderungen

PRC-Projekte fallen in die Kategorie der Grid-Computing-Installationen, allerdings muss ein solches Projekt zusätzliche Eigenschaften erfüllen [30]:

Granularität Das Internet ist ein sehr heterogenes Netz. Bandbreiten schwanken zwischen einigen GBit/s (nationale Backbones) bis hin zu 56 kBit/s (analoger Telefonanschluss). Die Datenpakete der aktuellen BOINC-Projekte haben Größen zwischen 340 kByte (SETI@home) und einigen MBytes (Rosetta@home).

C. B. Ries, *BOINC*, Xpert.press,
DOI 10.1007/978-3-642-23382-8_2,

Dies stellt natürlich ein Problem dar: Was machen Anwender mit einer eher niedrigen Übertragungsgeschwindigkeit, denn heutige DSL-Geschwindigkeiten sind noch immer nicht überall verfügbar. Es kann dann schon einmal vorkommen, dass das Herunterladen eines Arbeitspaketes länger als die Berechnung selber dauert, und dies sollte möglichst vermieden werden, weil die Anwender dadurch sicherlich frustriert sind. Ein Mechanismus für die Prüfung bzw. Messung der Netzleitung, um daraus zu ermitteln, welche Größe am geeignetsten für den Anwender ist, gibt es leider nicht. Diese Verantwortung tragen bisher noch die Administratoren eines BOINC-Projektes.

Diversität Die teilnehmenden Rechner sind sehr unterschiedlich. Um Windows-PCs, Macintosh-Rechner und diverse Unix-Maschinen nicht auszuschließen, müssen die BOINC-Projektbetreiber für jede dieser Plattformen die passende Client-Software bereitstellen. In der Regel werden mindestens eine Windows- und eine Unix-Variante für das Bearbeiten von Arbeitspaketen zur Verfügung gestellt. In vielen BOINC-Projekten fehlt teilweise einfach die Hardware, um eine Portierung vorzunehmen und zu testen.

Robustheit Die Client-Rechner unterliegen nicht der Kontrolle eines BOINC-Projektes. Man muss damit rechnen, dass übernommene Arbeitspakete nicht bearbeitet werden oder zurückgelieferte Berechnungsergebnisse falsch sind. Selbst Sabotage kann nicht ausgeschlossen werden. Der BOINC-Server erstellt deshalb für ein Arbeitspaket mehrere Aufteilungen und versendet diese an verschiedene Clients. Zur Überprüfung erhalten Clients außerdem regelmäßig Aufgaben, deren Berechnungsergebnisse bekannt sind. Zur Robustheit gehört auch, dass die Client-Software ein vorzeitiges Abschalten des Rechners „überlebt“ und nach dem Einschalten dort weitermacht, wo sie unterbrochen wurde. Dies wird durch das Erstellen von sog. Checkpoint-Dateien ermöglicht.

Attraktivität Die Berechnungssoftware stört den normalen Rechenbetrieb nicht. Auf Unix-Maschinen läuft sie mit niedriger Priorität im Hintergrund, für Windows-Rechner ist eine Bildschirmschoner-Variante erhältlich. Unterstützer müssen allerdings sowohl Zeit (Installation) als auch Geld (Online- und Stromkosten) investieren. Dazu sind die Leute umso eher bereit, je attraktiver das BOINC-Projekt erscheint. Für viele ist auch die wissenschaftliche Fragestellung interessant und beflügelt die Anwender, keine Kosten und Mühen zu scheuen, wenn die BOINC-Teilnehmer Potential an der Fragestellung erkennen. Noch attraktiver wird es, wenn die einzelnen Betreiber der BOINC-Projekte versprechen, dass die Teilnehmer mit den entsprechenden passenden Berechnungsergebnissen in den internationalen Forschungsberichten erwähnt werden.

2.2 Was ist umsetzbar?

Abbildung 2.1 zeigt die prinzipiellen Möglichkeiten, wie Arbeitsaufträge abgearbeitet werden können; dies ist auch davon abhängig, wie die einzelnen Arbeitspakete mit der Außenwelt kommunizieren. Die einzelnen Abarbeitungsmöglichkeiten

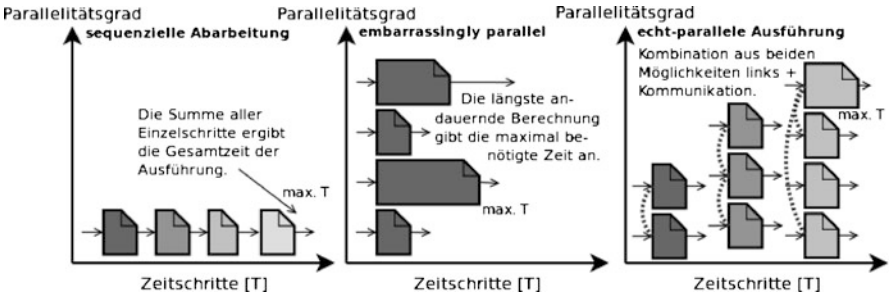


Abb. 2.1 Auf einem Rechencluster lassen sich die unterschiedlichsten Probleme lösen, abhängig vom Problem muss keine oder muss zwangsläufig eine Kommunikation zwischen den Rechenknoten stattfinden. Diese drei Konstellationen sind möglich: (1) sequenzielle Abarbeitung, (2) embarrassingly parallel und (3) echt-parallele Ausführung [32]

werden im Grad der Parallelität über die Zeit dargestellt und beschrieben. Je mehr Arbeitsaufträge übereinander angeordnet sind, desto mehr Arbeit kann in einem Zeitbereich gelöst werden. Von links nach rechts nimmt der Grad der Parallelität zu und auch die Anzahl der Arbeitspakete steigt.

Single-Core-Prozessoren arbeiten sequenziell und können immer nur einen Arbeitsauftrag zu einem bestimmten Zeitpunkt abarbeiten; diese werden nacheinander gestartet. Die *embarrassingly parallel* abzuarbeitenden Arbeitsaufträge sind absolut autonom und können ohne Kommunikation zur Außenwelt gestartet, gelöst und abgeschlossen werden. In diesem Fall können zehn Arbeitsaufträge tatsächlich in einem Zeitfenster gelöst werden, wenn zehn Rechner vorhanden sind, auch Multi-Core-Prozessoren können hier sehr rasch zu einer Lösung kommen. Der letzte Fall ist ein wenig komplizierter und muss in der Realität auch etwas genauer betrachtet werden. Die Abbildung zeigt mehrere parallel ausgeführte Arbeitsaufträge; die gestrichelten Linien sollen die Kommunikation zwischen diesen darstellen. Was ist allerdings zu tun, wenn ein Arbeitspaket aus unbekannten Gründen niemals ein Ergebnis liefert, aber von anderen Aufträgen für die weiteren Berechnungsschritte benötigt wird? In diesem Fall liegt ein klassischer *Deadlock* vor (zu Deutsch Verklemmung).

Für unsere Betrachtungen in diesem Buch ist diese Problematik nicht von Belang. Mit BOINC können zwar alle drei möglichen Lösungswege umgesetzt werden, allerdings ist BOINC nur für die ersten beiden Prinzipien gedacht und konzipiert.

2.3 Einsatzbereiche

Wir haben schon erwähnt, dass BOINC generell für Probleme verwendet werden kann, bei denen keine Kommunikation zwischen einzelnen Rechnerknoten nötig ist und die Aufgabenstellungen eine hohe Granularität besitzen. Dies können Anwendungen sein, bei denen große Datenmengen bearbeitet werden müssen oder lange dauernde Berechnungen, die viele Iterationen durchführen:

Film-Sequenzen/Rendering Eine einfache und relativ schnell umzusetzende Problemstellung ist die Bearbeitung von Film-Sequenzen oder das Rendern von Einzelbildern. Eine Beispielapplikation für die Bearbeitung von Film-Sequenzen finden Sie in Kap. 12 [143]. Wenn ein kompletter Film zum Beispiel mit bestimmten Filtern bearbeitet werden soll, so kann der Film in einzelne Film-Sequenzen umgewandelt werden. Diese Film-Sequenzen können dann in einzelne Archive gepackt werden, um daraus – im BOINC-Bereich – Workunits zu erstellen.

Parameterstudien In Kap. 14 zeigen wir, wie man Parameterstudien mit Hilfe von Legacy-Applikationen – u. a. COMSOL Multiphysics – erstellen kann. Bei Parameterstudien kann man ein Simulationsmodell nehmen und mit unterschiedlichen Parametersätzen füttern, z. B. kann die Geschwindigkeit eines Flugzeugs unter bestimmten Luftdruckverhältnissen variiert werden.

Bionische Methoden Ameisenalgorithmen, oder allg. ausgedrückt Algorithmen der Schwarmintelligenz bzw. Partikelschwarmoptimierung, fallen in den Bereich der Bionischen Methoden der Optimierung und suchen das Optimum einer Problemstellung [5]. Dabei wird der Versuch unternommen, ein Minimum oder Maximum zu finden oder zumindest so lange eine Optimierung durchzuführen, bis ein vorher definiertes Abbruchkriterium erfüllt ist. Letzteres dient dazu, dass die Berechnung nicht endlos lange läuft, wenn sie niemals gegen ein Ergebnis konvergiert. Die Durchführung solcher Algorithmen beinhaltet oft mehrere tausend Iterationen mit den Ergebnissen aus der vorherigen Iteration und können daher perfekt als eigenständige Aufgaben auf Einzelrechnern ausgeführt werden.

Fahrpläne und Stundenplan Die automatische Erstellung von Fahrplänen (Straßenbahn, Busse, etc.) oder von Stundenplänen (Müllabfuhr, Schule, etc.) kann durch bionische Methoden oder durch andere Ansätze, allerdings wieder in mehreren tausend Iterationen, gelöst werden. Die Ergebnismenge wird am Ende auf das Vorhandensein einer optimalen Lösung durchsucht.

Stochastik Verhält sich wie die beiden vorherigen Aufgabenstellungen.

2.4 Forschergruppen und ihre Erfolge

Durch Public-Resource Computing (PRC) wurden schon Erfolge gefeiert, die sicherlich auch auf High-Performance-Systemen (HPC) möglich gewesen wären, allerdings nicht mit solch einem geringen Teil an Kosten. Die University of Westminster [180] in London hat einen virtuellen Supercomputer initialisiert; basierend auf BOINC. Die geschätzten Einsparungen belaufen sich auf rund £500.000 in vier Jahren.

2.4.1 SETI@home: Sind wir allein im All?

SETI@home ist das zweite Projekt, das als PRC-Projekt (Public-Resource Computing) aufgebaut wurde, und hat sich der Suche nach außerirdischen Radiowellensignalen für den Beweis der Existenz von intelligentem Leben außerhalb der Erde

verschrieben. Der Beweis für die Existenz solchen Lebens ist bis heute leider noch nicht geliefert worden. Gegründet wurde das Projekt Mitte 1999, und es ist heute neben den zwei weiteren Projekten MilkyWay@home [150] und Einstein@home (vgl. Abschn. 2.4.2) eines der Projekte, welche sich mit aufgefangenen Signalen aus dem Weltall beschäftigen. Nichtsdestoweniger hat SETI@home den ersten Schritt gemacht und die erforderliche Software geliefert, so dass weitere Wissenschaftler ihre aufwendigen Berechnungen und ihr Datenaufkommen an Freiwillige verteilen können und ein wissenschaftlicher Fortschritt ermöglicht wird. Durch SETI@home wurde das PRC-Rechnen quasi salonfähig.

2.4.2 Einstein@home: Weltraumforschung

Das Projekt Einstein@home [120] beschäftigt sich mit der Suche nach einem Beweis der von Albert Einstein im Jahre 1905 aufgestellten speziellen Relativitätstheorie. Diese Relativitätstheorie besagt, dass wir in einem Universum gefüllt mit Gravitationswellen leben. Explodierende Sterne, zusammenstoßende schwarze Löcher und andere gewaltige Zerstörungen erschaffen solche Gravitationswellen, welche den Raum und die Zeit beeinflussen [120]. Bisher wurde noch kein Beweis für das Vorhandensein von Gravitationswellen durch Einstein@home geliefert.

Dies ist nicht die einzige Aufgabe von Einstein@home; seit März 2009 wird auch nach Radiopulsaren gesucht [72]. Für diese Suche werden Daten vom Arebico-Observatorium genutzt – eventuell kennen Sie das dazugehörige Radioteleskop aus dem James-Bond-Film *GoldenEye* und dem Kinofilm *Contact* [152]. Es handelt sich dabei um ein Teleskop mit einem beachtlichen Durchmesser von 300 m und einer Höhe von 150 m. Radiopulsare sind die extremsten Objekte im Universum, es handelt sich dabei um rasend schnell drehende Neutronensterne, die entweder einen weiteren Neutronenstern oder ein schwarzes Loch umkreisen. Die Verwendung von BOINC und der damit nutzbaren Computerleistung, von den Anfängen im Jahr 2009 mit rund 200.000 Teilnehmern, ermöglichte erst die Suche nach sich noch schneller drehenden Radiopulsaren. Das hängt damit zusammen, dass mehr Daten analysiert werden müssen, um die Suche zu verfeinern; je feiner, desto schnellere Radiopulsare können entdeckt werden. Bis heute[1] sind zehn neue Radiopulsare von Einstein@home entdeckt und bestätigt worden [121]. Als Freiwilliger am Einstein@home-Projekt wird man namentlich in der Danksagung der wissenschaftlichen Publikation erwähnt. Diese ungeschriebene Regel in besonderen Maße wird von den BOINC-Teilnehmern schon fast erwartet, aber auch von den BOINC-Projekten respektiert. Die Motivation für die Teilnahme an den jeweiligen Projekten wird dadurch beflügelt.

[1] Stand am 27. September 2011.

2.4.3 Visu@lGrid: Model-Driven-Engineering

Mit Hilfe der Unified Modeling Language (UML) und der Möglichkeit, sogenannte UML-Profile zu erstellen, soll in näherer Zukunft der Prozess von der ersten Idee eines BOINC-Projekts bis hin zu einer fertigen ausführbaren BOINC-Installation und Lösung von Problemstellungen soweit es geht durch Modelle vereinfacht werden. Erste Schritte zeigen, dass dieser Ansatz vielversprechend ist [17–19]. Aufbauend auf der Grundstruktur der BOINC-Programmierschnittstelle (s. Kap. 7) und der BOINC-Infrastruktur (s. Abschn. 4.1) wird mit Hilfe eines Top-down-Ansatzes die Struktur analysiert und in ein UML-Modell umgewandelt, so dass aus diesem Modell folgende Komponenten automatisch durch einen Code-Generator generiert werden [52, 53]:

Entwicklungsumgebung Eine IDE (Integrated Development Environment) wird mit Hilfe des spezifischen UML-Modells generiert, so dass nur Funktionalitäten und Schnittstellen genutzt werden können, welche benötigt und durch BOINC zur Verfügung gestellt werden. Es handelt sich dabei um eine IDE, die der Domäne eines Entwicklers angepasst ist und nur kontextbezogene Modellierungselemente beinhaltet. Dadurch kann die Entwicklung auf ein Minimum der Entwicklungszeit und Wartung reduziert werden. Kleinere Änderungen bei der Konfiguration eines BOINC-Projektes oder einer Applikation werden automatisch geprüft und aufgelöst, so dass keine Fehler aufkommen können, nur weil womöglich eine weitere zu ändernde Konfiguration vergessen wurde abzuändern.

Applikation Die wissenschaftliche Applikation wird durch Standarddiagramme der UML modelliert, z. B. Sequenzdiagramme oder Statusdiagramme. So kann eine grafische Lösung einer Applikation erfolgen und externe Werkzeuge mit Unterstützung von XMI (XML Metadata Interchange) – um Daten unter verschiedenen Entwicklungsumgebungen austauschen zu können [54] – genutzt werden, u. a. IBMs Rational Rhapsody [136].

2.4.4 Dem Dieb auf der Spur

Mit Hilfe von BOINC konnten zwei Computer wiedergefunden werden. In den Jahren 2007 [76, 127, 167] und 2010 [168] wurde jeweils ein Laptop entwendet. Im ersten Fall hatte der Ehemann der Besitzerin Kimberly Melin eine Installation vom BOINC-Manager durchgeführt und SETI@home als Projekt hinzugefügt. Melin hat daraufhin die Datenbankeinträge von SETI@home überprüft und konnte nach einer Weile sehen, dass neue Berechnungsergebnisse abgespeichert wurden. Bei jeder Kommunikation mit einem BOINC-Projekt wird die IP-Adresse mitgesendet und protokolliert. Gut für Melin, denn sie konnte daraufhin die IP-Adresse an die örtliche Polizei weitergeben, und die konnte in Folge dessen die reale Adresse des verwendeten Internetanschlusses ausfindig machen. Der Rechner wurde wiedergefunden und glücklicherweise waren noch alle Daten von Melin vorhanden, plus zusätz-

liche Musikdateien. Dabei handelte es sich wohl um Dateien des Diebs. Die Suche, das Auffinden und Zurückbringen dauerten rund zweieinhalb Wochen.

Im zweiten Fall betraf es einen BOINC-Benutzer aus Deutschland. Ein iMac wurde in einer Nacht aus den Räumlichkeiten einer Firma entwendet, dies wurde am nächsten Morgen bemerkt und der Kriminalpolizei gemeldet. In den nächsten Tagen hat der Rechner sich fünfmal beim WUProb@home-Projekt [191] mit der Meldung von Berechnungsergebnissen gemeldet. Das Diebstahlopfer hat das verfolgt und die IP-Adresse an die Kriminalpolizei weitergeleitet. Daraufhin hat diese einen Durchsuchungsbefehl beantragt und hat das Notebook sicherstellen können. Zwischen der Feststellung des Diebstahls, der Sicherstellung und Rückgabe des Notebooks an den Besitzer sind rund neun Tage vergangen. Neben der wissenschaftlichen Tätigkeit von BOINC-Projekten dienen diese anscheinend auch wunderbar der Detektivarbeit.

Teil II
Technik

Kapitel 3
Berkeley Open Infrastructure for Network Computing

Zusammenkunft ist ein Anfang
Zusammenhalt ist ein Fortschritt
Zusammenarbeit ist ein Erfolg.

Henry Ford I.

3.1 Einführung in BOINC

BOINC ist ein Framework zur Erstellung von hoch-skalierbaren, komplexen und rechenintensiven Problemen im Sinne des *Public-Resource-Computing* (PRC) Prinzips. Mit diesem Framework wird der Rechenaufwand in kleinere Pakete geteilt. Diese Pakete werden *Workunits* genannt. Jeder kann sich an der Arbeit zur Lösung dieser Workunits beteiligen, ganz freiwillig. Freiwillige (engl. *Volunteers*) registrieren sich bei einem BOINC-Projekt, um teilnehmen zu können. Nach Registrierung müssen sich die Teilnehmer den BOINC-Client installieren, zur einfacheren Verwaltung und Nutzung dieses BOINC-Clients bietet ihnen das BOINC-Entwicklerteam den BOINC-Manager an. Mit der Installation des BOINC-Managers erhalten sie automatisch den BOINC-Client. Der BOINC-Client übernimmt die Kommunikation mit einem BOINC-Projekt und kann durch die Teilnehmer über Remote Procedure Calls (RPCs) gesteuert werden, was dank des BOINC-Managers durch grafische Steuerelemente extrem vereinfacht wird. Wie Sie eine manuelle Steuerung durchführen, zeige ich Ihnen in Abschn. 4.3.1.1.

Innerhalb des BOINC-Managers können Freiwillige nun ihre Authentifizierungsdaten eingeben und die BOINC-Projekte hinzufügen, an denen sie sich beteiligen wollen. Nach erfolgreicher Authentifizierung bei den jeweiligen BOINC-Projekten kann der Freiwillige sich die wissenschaftliche Applikation mit den zugehörigen Arbeitspaketen herunterzuladen, diese ausführen und die Berechnungsergebnisse zurück an das Projekt senden. Dieser Prozess ist absolut transparent und benötigt keinerlei Intervention des BOINC-Teilnehmer; er läuft ganz bequem im Hintergrund. Da sich Freiwillige an solchen Projekte anmelden und teilnehmen können, wird dies auch *Volunteer Computing* genannt. Die Freiwilligen bearbeiten die Arbeitspakete, schicken das Ergebnis an den Projektserver zurück und holen sich ein oder mehrere neue Arbeitspakete. Auf dem Projektserver wird das Ergebnis validiert und bei Gültigkeit zu einem Gesamtergebnis hinzugefügt oder als Einzelergebnis an einem spezifizierten Ort abgelegt. Der Freiwillige kann gleichzeitig an mehreren Projekten teilnehmen, auch das Verwalten der unterschiedlichen Kommunikations-

C. B. Ries, *BOINC*, Xpert.press,
DOI 10.1007/978-3-642-23382-8_3,

kanäle und das möglichst intelligente Scheduling[1] zwischen den einzelnen BOINC-Projekten wird durch den BOINC-Client übernommen. Sie können den Scheduler allerdings durch einige Einstellungen beeinflussen. Informationen über die unterschiedlichen Einstellungsmöglichkeiten finden Sie in Abschn. 4.3.3.

Passionierte Entwickler und Wissenschaftler können sich eine eigene wissenschaftliche Applikation erstellen. Zu diesem Zweck liefert BOINC eine API (Application Programming Interface, zu Deutsch Programmierschnittstelle), welche dem Nutzer ermöglicht, eine wissenschaftliche Applikation zu erstellen. Diese Applikation kann daraufhin zur Lösung von einfachen bis komplexen Berechnungsaufgaben verwendet werden.

Die Funktionalitäten des kompletten BOINC-Frameworks können in mehrere Kategorien unterteilt werden. BOINC ist generell ein Client-/Server-System, und in den Abschn. 3.2 und 3.3 werden die Kategorien in die zwei Gruppen Client und Server eingruppiert. Eine Mischung ist allerdings möglich, zum Beispiel können auf der Serverseite natürlich auch Funktionen für die Ein-/Ausgabe genutzt werden. Die Ein-/Ausgabe wird in diesem Buch allerdings mehr in Bezug auf die Entwicklung einer wissenschaftlichen Applikation beschrieben und ist daher auf der Clientseite einsortiert [2–4].

3.2 Aufgaben des BOINC-Servers

Der BOINC-Server ist der Kern eines BOINC-Projekts und besteht aus mehreren Komponenten, die in Kap. 4 vorgestellt werden. Die Funktionen der Serverkomponenten können in mehrere Bereiche unterteilt werden:

Konfigurationen Der Projektserver kann individuell durch eine zentrale Konfigurationsdatei eingestellt werden. Für das einfachere Handhaben und Austauschen von Konfigurationsdateien und Informationen werden alle BOINC-Konfigurationen im XML-Format vorgenommen. Beispiele für solche Konfigurationen finden Sie im Anhang 15.3.

Projektorganisation Informationen über den Status eines BOINC-Projekts, die Verwaltung der Benutzer und wissenschaftlicher Anwendungen, das Erstellen von Backups der Datenbank(en), das Aktualisieren der Informationen auf der Webseite eines BOINC-Projekts, usw. – alle diese Aufgaben werden von eigenen Applikationen durchgeführt. Im Kap. 4 wird diese Architektur diskutiert vorgestellt und diskutiert.

Arbeitserstellung Wissenschaftliche Applikationen benötigen in der Regel Eingabedaten, mit denen sie arbeiten sollen. BOINC nennt solche Daten Arbeitspakete (engl. Workunit). Jede Workunit muss einem BOINC-Projekt bekannt

[1] Scheduling bedeutet, eine Zeitplanung zu erstellen und zu organisieren. In der Regel ist das Bereitstellen eines Zeitfensters gemeint, in dem ein bestimmter Prozess arbeiten darf. Ist das Ende eines Zeitfensters erreicht, so erhält ein neuer Prozess ein Zeitfenster für die Bearbeitung. Durch das Scheduling wird das Multitasking von modernen Betriebssystemen ermöglicht.

gemacht werden, und für diese Prozedur werden mehrere Verfahren zur Verfügung gestellt. So können Sie BOINCs mitgelieferte Helfer (vgl. Abschn. 9.2.1) wählen oder eine eigene Implementierung entwerfen und umsetzen. Für die Implementierung stehen Ihnen BOINC-API-Funktionen zur Verfügung; wie Sie diese nutzen können, wird in Abschn. 9.2.2 näher behandelt.

Validierer Die Workunits werden an die BOINC-Teilnehmer verteilt, auf den Hosts der Teilnehmer werden diese bearbeitet und ein oder mehrere Ergebnisse zum zugehörigen BOINC-Projekt zurückgesendet. Diese Ergebnisse können natürlich alles mögliche beinhalten; wenn man im Vorfeld abschätzen kann, wie ein solches Ergebnis auszusehen hat oder zumindest den Ergebnisraum eingrenzen kann, so kann man einen Validierer nutzen, um eine Prüfung durchzuführen. Sie können Standardvalidierer von BOINC verwenden oder eine eigene Implementierung entwerfen und implementieren. Der Abschn. 9.3.1 geht auf beide Möglichkeiten ein und zeigt Ihnen die Verwendung an Beispielen.

Assimilierer Vielleicht kennen Sie die Borg aus der Fernsehserie Star Trek [172], dann kennen Sie das sogenannte Assimilieren als Prozess der Eingliederung von biologischen Elementen in das Gefüge der Borg. Der Duden beschreibt[2] dies ebenso [118]. BOINC nutzt diese Benennung für das Einordnen der Berechnungsergebnisse innerhalb der von Ihnen spezifizierten Abspeicherungsorte. Sie können bestimmen, wie mit Berechnungsergebnissen umzugehen ist, wie und wo diese abgespeichert werden sollen. Im Abschn. 9.3.2 wird näher darauf eingegangen und gezeigt, wie eine Implementierung aussehen kann.

3.3 Aufgaben des BOINC-Clients

Der vom Freiwilligen installierte BOINC-Client übernimmt die Steuerung der wissenschaftlichen Applikation und die Kommunikation mit den BOINC-Projekten. Der BOINC-Client übernimmt zahlreiche Funktionalitäten und Verwaltungsaufgaben, die in der nachfolgenden Auflistung kurz erläutert werden:

Initialisierung und Terminierung Das BOINC-Framework wird für die anstehende Arbeit vorbereitet. Dafür stehen Funktionen zur Initialisierung und Beendigung einer Applikation zur Verfügung. Die Initialisierung übernimmt das Einstellen von Standardlaufzeitoptionen, die jederzeit wieder modifiziert werden können. Die Laufzeit der Applikation kann in vier Status unterteilt werden: *Suspend*, *Resume*, *Quit* und *Abort*. Die ersten zwei pausieren beziehungsweise heben eine Pause auf. *Quit* und *Abort* brechen die Ausführung der Applikation ab.

Dateiverwaltung In den wissenschaftlichen Applikationen sollten keine hardcodierten Dateinamen genutzt werden. Es wird bei der Programmierung ein vom Entwickler festgelegter Dateiname gewählt, der eine assoziative Verknüpfung zu Dateien symbolisiert. Bei einem BOINC-Teilnehmer können gleichzeitig mehrere Projektanmeldungen vorhanden sein und die einzelnen Projekte besitzen

[2] Bedeutungsübersicht: (1) (Biologie) aufgenommene Nährstoffe in körpereigene Stoffe umwandeln, (2) (bildungssprachlich) [sich] angleichen, [sich] anpassen.

projektspezifische Ein-/Ausgabedateien. Damit diese Dateien nicht in Konflikt mit andere BOINC-Projekten geraten, werden sogenannte *Slots* für die Ausführung einer wissenschaftlichen Applikation genutzt. Innerhalb dieser Slots wird eine wissenschaftliche Applikation ausgeführt, weiterhin werden in diesen Slots alle Ein-/Ausgabedateien für die Ausführung abgelegt. Dabei müssen die Dateien nicht im Ganzen in den Slot kopiert werden, sondern es können symbolische Links abgelegt werden. Die symbolischen Links sind einfache Textdateien und enthalten einen Dateipfad zu den eigentlich zu nutzenden Dateien. Der BOINC-Projektadministrator kann bei der Erstellung von Arbeitspaketen definieren, ob für Dateien symbolische Links oder eine echte Dateikopie erstellt und daraufhin in einer wissenschaftlichen Applikation verwendet werden soll. Der Abschn. 4.3.2 behandelt dieses Verfahren im Detail.

Eingabe/Ausgabe (Input/Output, I/O) BOINC ist für die Ausführung unter verschiedenen Betriebssystemen entworfen, u. a. Windows, Linux und Mac. Um eine einheitliche Schnittstelle für die Ein-/Ausgabe zu ermöglichen, sollten die im BOINC-Framework implementierten Funktionen verwendet werden, z. B. `boinc_fopen()`. Bei Windows können u. a. Sicherheitsmechanismen den Zugriff verwehren, bei Linux werden bei aufkommenden Fehlern Signale gesendet. Die mitgelieferten Funktionen bieten einen *I/O-Wrapper* an, um unter den drei genannten Betriebssystemen einheitlich Dateien öffnen und schließen zu können. Weiterhin beinhaltet der I/O-Wrapper das Handeln von unerwarteten Fehlern und schließt automatisch eine geöffnete Datei, wenn sich die wissenschaftliche Applikation unerwartet beendet.

Um eine schnelle Abspeicherung in Dateien zu ermöglichen, liefert das BOINC-Framework die Klasse `MFILE` mit. Diese Klasse dient als Puffer. Alle Schreibzugriffe erfolgen in einem temporär angelegten Speicherplatz, der beim Aufruf der Methode `flush()` den Inhalt auf das Dateisystem schreibt.

Checkpointing Eine Applikation kann während der Ausführung jederzeit unterbrochen bzw. pausiert werden. Dies ist z. B. der Fall, wenn die Prioritäten für die Ausführung so gesetzt sind, dass die Applikation nur ausgeführt werden soll, wenn der Anwender nicht am Rechnersystem arbeitet (Idle-Betrieb). Der Checkpoint-Mechanismus speichert den aktuellen Status des zuletzt möglichen abzuspeichernden Berechnungsstands in einer sogenannten *Checkpoint*-Datei, und wenn die Applikation weitergeführt werden soll, so werden die Informationen aus dieser Datei wiederhergestellt. Durch diesen Mechanismus muss die Berechnung nicht immer wieder am Anfang gestartet werden und kann schrittweise, möglicherweise mit einigen Stunden Ausführungsversatz fertiggestellt werden.

Kritische Bereiche In der Ausführung einer Applikation kann es sein, dass Bereiche nicht unterbrochen werden dürfen, z. B. wenn die Datenkonsistenz bei der Synchronisation von verschiedenen Berechnungsabschnitten gegeben sein muss. Mit Hilfe von atomaren Bereichen ist dieser Wunsch zu erfüllen, da diese Bereiche nicht unterbrochen werden können.

Berechnungsstatus Der Fortschritt einer Berechnung kann jederzeit abgefragt werden. Die Beendigung einer Berechnung impliziert einen Punktewert (Credit), der dem Benutzerkonto gutgeschrieben wird. Diese Credits beschreiben das

Verhältnis zwischen zur Verfügung gestellter CPU-Zeit[3] und einem projektspezifischen Maximalpunktwert. Die Berechnung der Credits ist nicht einfach und hat zudem mit einigen Problemen zu kämpfen. Es gibt zwei Möglichkeiten, Credits für ein erfolgreiches Arbeitspaket zu vergeben: (1) durch die manuelle Definition eines Credit-Wertes oder (2) durch die Berechnung des sogenannten Cobblestone (in Deutsch Kopfsteinpflaster). Im nachfolgenden Abschn. 3.4 diskutieren wir die Berechnung des Cobblestone.

Status-/Prozessmeldung Die Funktion `boinc_fraction_done()` kann den aktuellen prozentualen Wert der Abarbeitung setzen, wird dem BOINC-Client gemeldet und kann daraufhin vom BOINC-Manager angezeigt werden.

Verschiedene weitere Daten Diese Daten beschreiben zahlreiche Informationen über die verwendete wissenschaftliche Applikation, das Betriebssystem, die genutzten und noch verfügbaren Rechnerressourcen und vieles mehr. Zur Abfrage muss eine Instanz der Struktur `struct APP_INIT_DATA` mit der Funktion `boinc_get_init_data()` gefüllt werden. Diese Informationen werden vom BOINC-Client zusammengetragen und können jederzeit abgefragt werden.

Prozess-/Zeitberechnungen Mit `boinc_wu_cpu_time()` wird der Gesamtwert der bisher genutzten CPU-Zeit ermittelt, seitdem ein Arbeitspaket gestartet wurde. Nach Berechnungspausen liefert die Funktion `boinc_elapsed_time()` das Delta der verstrichenen Zeit zwischen der Pause und dem aktuellen Zeitpunkt. Mit Hilfe dieser Funktionen können Messungen über Rechnerressourcen im Verhältnis zum zeitlichen Verlauf erstellt werden. In der Interna von BOINC werden diese Messungen genutzt um u. a. zu ermitteln, ob die Werte für die maximale Nutzungsdauer eines Prozessors überschritten wurde und eine Berechnung daraufhin abgebrochen werden muss. Zu bedenken ist, dass die CPU-Zeit nur die wirklich genutzte CPU-Zeit wiederspiegelt – viele Ein-/Ausgaben haben in der Regel keinen großen Einfluss auf diesen Wert. Eine for-Schleife mit vielen Berechnungen hingegen benötigt mehr CPU-Zeit, in der die CPU auch wirklich etwas tut und nicht nur wartet, bis eine Ein-/Ausgabe durchgeführt werden kann.

Standalone-Modus Dieser Modus dient dem Testen und Überprüfen der Funktionen einer wissenschaftlichen Applikation. Mit Hilfe dieses Modus kann eine wissenschaftliche Applikation ohne den BOINC-Client auf Fehler hin untersucht werden, da alle Kommunikationen zum BOINC-Client deaktiviert sind.

Wiederkehrende Funktionsaufrufe Funktionen können während der Ausführung im BOINC-Framework registriert werden (sog. *Callback-Funktionen*) und werden in einem Intervall von einer Sekunde ausgeführt.

Netzwerkkommunikation In der Ausführung der wissenschaftlichen Applikation kann dem BOINC-Client übermittelt werden, dass eine Netzwerkverbindung zur Verfügung gestellt werden muss, z. B. um Ergebnisse zu versenden. Sobald der BOINC-Client eine Verbindung aufbauen kann, wird dies getan und die Kommunikation durchgeführt.

[3] Bei BOINC die Anzahl der Fließkomma- und Ganzzahloperationen für die Lösung einer Rechenaufgabe.

Laufzeitdiagnose Bei der Entwicklung einer Applikation können plattformspezifische Fehler bei den Projektteilnehmern auftreten. Wenn eine wissenschaftliche Applikation abstürzt oder abgebrochen wurde, können der *Stacktrace*[4] und Informationen über die Ausführung (Wert der Beendigung, Signalnummern, Plattforminformationen und die Datei mit Inhalt der Standardfehlerausgabe) an den Projektserver gesendet werden. Der Entwickler einer wissenschaftlichen Applikation kann festlegen, welche Informationen innerhalb der Diagnose erstellt und an ein BOINC-Projekt zurückgesendet werden sollen.

Langzeitberechnungen Wissenschaftliche Applikationen können mehrere Wochen oder Monate laufen. Damit ein Arbeitspaket nicht als verloren angenommen wird, u. a. weil die Deadline überschritten wurde, es allerdings noch durch einen BOINC-Teilnehmer in Bearbeitung ist, kann zwischendurch der Status des Arbeitspakets aktualisiert werden. Dies kann durch *Trickle-Messages* ermöglicht werden, wobei diese Nachrichten wahlfrei aufgebaut sein können. Es empfiehlt sich, den aktuellen Berechnungsstand zu versenden.[5]

Grafikausgaben Die wissenschaftliche Applikation kann um die Möglichkeit eines Bildschirmschoners (engl. *Screensaver*) erweitert werden. Zu diesem Zweck existiert eine Grafikschnittstelle, die vom Entwickler zu implementierende Funktionen definiert und für die schon erwähnten Betriebssysteme Windows, Linux und Mac mit den gleichen Aufrufen genutzt werden kann. Innerhalb des BOINC-Framework werden die OpenGL-Bibliotheken GLU und GLUT verwendet; dazu mehr in Kap. 8.

Graphics Processing Unit (GPU) Die Ausführung auf einer GPU beschleunigt die Ausführungszeit um mindestens einen Faktor zwischen zwei, zehn und höher [134, 190]. Ab der BOINC-Client-Version 6.10.10 werden neben *NVidia* auch *ATI* Grafikkarten unterstützt. Die Verwendung wird zwar durch eine API vereinfacht, allerdings ist die Nutzung noch nicht trivial; dazu mehr in Abschn. 7.5.

3.4 Performancebewertung, Credit-Berechnung

Es ist wirklich schwierig, eine sinnvolle Beschreibung der Umsetzung zu finden, wie denn die Credits für erfolgreich bearbeitete Arbeitspakete berechnet werden. Es gab in der Vergangenheit zahlreiche Diskussionen über dieses Thema, damit die Berechnung und Zuweisung von Credits möglichst fair ist und alle BOINC-Teilnehmer zufrieden gestellt werden können.

Die Problematik ist, dass ein Arbeitspaket auf unterschiedlichen Rechnerplattformen womöglich gravierende Zeitunterschiede besitzt sowie eventuell benötigte Rechenzyklen benötigt. Dies erscheint logisch, wenn man ein gleiches Arbeitspaket auf einer 1 GHz oder 3 GHz Maschine rechnen lässt. Wenn die Einstellungen

[4] Ausgabe und Interpretation des Inhalts des Stacks.

[5] CPU-Zeit; Zusammenfassung der Ergebnisse; Befehle, die für die Ausführung Bedeutung haben.

für die freigegebenen Rechenzeiten gleich sind, so erwarte man sicherlich auf der 3-GHz-Maschine ein schneller zurückgeliefertes Ergebnis.

Jeff Cobb verewigte sich in seiner Definition des *Cobblestone*. Cobblestones sind die Einheiten, mit denen die Credits von Arbeitspaketen berechnet werden. In die Berechnung fließen folgende Benchmark-Werte mit ein, welche standardmäßig alle fünf Tage vom BOINC-Client gemessen werden:

Whetstone Dieser Benchmark liefert eine Messung für die benötigten Rechenzyklen von *Fließkommaoperationen*.

Dhrystone Dieser Benchmark liefert eine Messung für die benötigten Rechenzyklen von *Ganzzahloperationen*.

Aktuell werden die Credits nach der Formel 3.1 errechnet [29]. 86.400 sind die Anzahl der Sekunden eines Tages und wu_cpu_time ist die benötigte CPU-Zeit für die Berechnung eines Arbeitspaketes.

$$\frac{([\text{whetstone}] + [\text{dhrystone}])}{1000} \cdot \frac{100}{(2 \cdot 86.400)} \cdot wu\_cpu\_time \tag{3.1}$$

Das Credit-System befindet sich aktuell im Umbruch.[6] Es soll ein System eingeführt werden, welches sich automatisch an die Systemumgebung anpasst, so dass die Credit-Berechnung fairer wird.

Das Credit-System kann von Ihnen quasi ausgeschaltet werden, wenn Sie in der Eingabeschablone von Arbeitspaketen einen festen Wert für die Credits angeben (nachzulesen im Anhang 15.3.3).

3.5 Verwendung des BOINC-Managers

Der BOINC-Manager ist die direkte Schnittstelle zum Projektteilnehmer und ist eine grafische Anwendung. Mit dieser Anwendung kann der Projektteilnehmer sich an verschiedenen BOINC-Projekten anmelden und verwalten. Schnelles Vorgehen beim Konfigurieren des BOINC-Clients wird ermöglicht, z. B. wie viele Prozessoren generell verwendet werden dürfen, oder wie viel Rechenzeit in Prozent pro Prozessorkern für die wissenschaftlichen Applikationen zur Verfügung stehen soll. Das Einstellen der Anzahl der Prozessoren kann nicht für bestimmte wissenschaftliche Applikationen vorgenommen werden, d. h. eine Applikation kann nicht x viele Prozessoren für die Ausführung reservieren. Wenn im BOINC-Manager drei Prozessoren zur Verwendung freigegeben sind, dann werden diese durch drei verschiedene Arbeitspaketberechnungen womöglich drei verschiedener wissenschaftlicher Applikationen genutzt. In Abschn. 4.3.3 wird der BOINC-Manager näher beschrieben und in Abschn. 4.3.1.1 wird ein weiteres BOINC-Werkzeug vorgestellt, mit dem man ein BOINC-Projekt auch ohne den BOINC-Manager steuern kann.

[6] 18. November 2011.

Kapitel 4
Architektur des BOINC-Systems

Wer hohe Türme bauen will, muss lange beim Fundament verweilen.

Anton Bruckner, 04.09.1824–11.10.1896

4.1 Architektur und Prinzipien

Die Grundstruktur eines BOINC-Projekts basiert auf einer Client-/Serverarchitektur. Abbildung 4.1 gibt einen Überblick der beteiligten Softwarekomponenten.

Die linke Seite stellt die Clientseite dar und kann durch Sie mit der Entwicklung einer wissenschaftlichen Applikation an Ihre Bedürfnisse angepasst werden. Die unteren beiden Komponenten mit den fast gleich klingenden Namen *Slot (Arbeitspaket N)*[1] stellt eine wissenschaftliche Applikation dar, welche u. a. in Kap. 12 vorgestellt und diskutiert wird. Die anderen zwei Komponenten BOINC-Client und BOINC-Manager werden vom Projektteilnehmer installiert und stellen ihm eine einfache Schnittstelle bereit um seine eigenen Ressourcen zur Verfügung zu stellen, wie dies in Kap. 3 diskutiert wird.

Auf der rechten Seite sehen Sie die sieben Komponenten *MySQL-Datenbank*, *Scheduler*, *Feeder*, *Transitioner*, *Validator*, *Assimilator* und *File Deleter*. Bis auf die MySQL-Datenbank sind die Komponenten allesamt BOINC-Applikationen und werden *Dämonen* genannt. Der File Deleter ist optional und muss nicht zwingend vorhanden und gestartet sein, die anderen Dämonen sind allerdings essentiell für die erfolgreiche Abarbeitung eines BOINC-Projekts. Daher bilden diese sechs Dämonen das Rückgrat eines BOINC-Projekts und sind entscheidend für den Erfolg der Abarbeitung. Die nachfolgenden Abschn. 4.1.1 bis 4.3.3 beschreiben die einzelnen Dämonen detaillierter.

4.1.1 Projekt- und Serverseite von BOINC

Wie im vorherigen Abschnitt erwähnt, besteht die Serverseite eines BOINC-Projekts aus mindestens sechs plus x-beliebigen Softwarekomponenten, welche auf der rech-

[1] Das N stellt eine Identifikationsnummer eines Arbeitspakets dar und sagt aus, dass hier die Berechnungsdaten von dem Arbeitspaket abhängig sind.

C. B. Ries, *BOINC*, Xpert.press,
DOI 10.1007/978-3-642-23382-8_4,

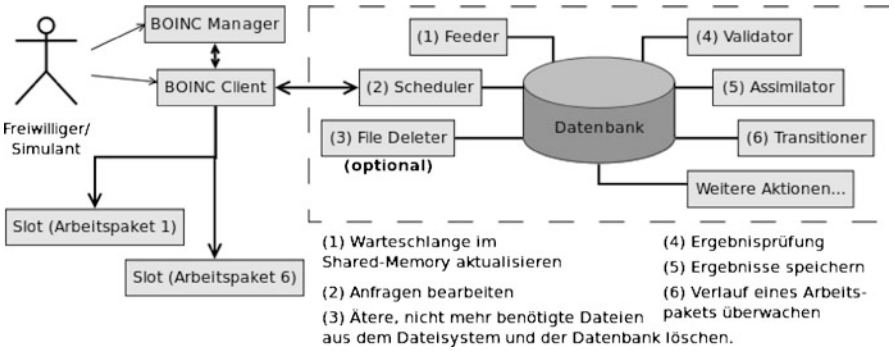


Abb. 4.1 Client-/Serveransicht eines BOINC-Projekts, *links* die Clientseite mit dem BOINC-Client und BOINC-Manager, *rechts* die Serverseite mit den mindestens vorhandenen BOINC-Dämonen: Scheduler, Feeder, Transitioner, Validator und Assimilator. Die *einzelnen Pfeile* zeigen die Kommunikationswege auf, kein Pfeil bedeutet eine mögliche Kommunikation in beide Richtungen. Die einzelnen Dämonen arbeiten autonom zu den jeweils anderen Dämonen und organisieren sich mit Hilfe der Informationen aus der MySQL-Datenbank selbst. Dies trifft auch für die Arbeitspakete 1 und 6 zu, welche ohne Abhängigkeit zu weiteren Applikationen innerhalb des BOINC-Client-Kontexts verarbeitet werden

ten Seite von Abb. 4.1 gezeigt werden. Es können zusätzliche Softwarekomponenten entwickelt, implementiert und zur BOINC-Infrastruktur hinzugefügt werden, z. B. der hier exemplarisch gezeigte File Deleter. Im Kap. 9 wird diese Möglichkeit beschrieben und es wird aufgezeigt, worauf Sie bei der Entwicklung zu achten haben. Die nachfolgende Auflistung beschreibt die bisher genannten Softwarekomponenten:

Datenbank Es handelt sich bei dieser Komponente um eine MySQL-Datenbank. Sie ist das Gedächtnis eines BOINC-Projekts und enthält alle Daten einer wissenschaftlichen Applikation, die Benutzerdaten aller Teilnehmer, alle Host-Informationen der an einem BOINC-Projekt registrierten Rechner, welche Arbeitspakete vorhanden sind und wie weit deren Abarbeitung ist, sowie Informationen über die sog. Credits der BOINC-Teilnehmer. Abschnitt 5.7.1 enthält weitere Informationen über die Struktur der Datenbank und wie die Datenbankstruktur um weitere Datenbanktabellen erweitert werden kann, um zusätzliche Informationen hinzufügen zu können.

Transitioner Überwacht den Lebenszyklus eines Arbeitspakets, indem die Informationen aus den Datenbanktabellen *workunit* und *result* stetig geprüft und im Bedarfsfall aktualisiert werden. Eine Übersicht über die einzelnen Datenbanktabellen finden Sie in Abschn. 9.5. In Abschn. 4.2.3 wird beschrieben, wie der logische Aufbau dieser Prüfung und das Aktualisieren implementiert sind.

Scheduler Der Scheduler bearbeitet Anfragen von Projektteilnehmern, wie z. B. das Registrieren eines einzelnen Benutzers, um sich an einem BOINC-Projekt zu beteiligen. Der Scheduler ist auch für das ordentliche Versenden von Arbeitspaketen und das Empfangen von Berechnungsergebnissen verantwortlich. Damit ist diese Softwarekomponente die Schnittstelle zwischen dem BOINC-Client

und dem BOINC-Projekt. Die Kommunikation ist frei von sicherheitskritischen Informationen. Eine Benutzerauthentifizierung findet über einen öffentlichen Schlüssel statt, der sich aus mehreren Einzeldaten des BOINC-Teilnehmers zusammensetzt.

Der Scheduler selber stellt die oben erwähnten Daten nicht zum Herunterladen bereit, sondern liefert Informationen, wo diese Daten heruntergeladen werden können. Im Normalfall handelt es sich bei diesen Informationen um eine Webadresse, was im Übrigen auch bedeutet, dass die Daten irgendwo im Internet auf einem Rechner zur Verfügung gestellt werden können. Sinnvoll ist das z. B., wenn Sie möglichst kurze Antwortzeiten erreichen möchten. Sie können die Arbeitspakete auf mehreren Servern – verteilt auf dem Globus – bereitstellen und prüfen, in welcher Zeitzone sich der anfragende BOINC-Teilnehmer befindet, und liefern die Webadresse des am nächstliegenden Hosts. Diese Technik wird u. a. vom Einstein@home-Projekt genutzt und ist auch von der dahinter stehenden Entwicklergruppe in den BOINC-Quellen implementiert worden [84]. Es gab in der Vergangenheit auch Bestrebungen, BOINC mit dem Peer-to-Peer-Protokoll BitTorrent zu koppeln, um so eine höhere Performance der Netzwerknutzung zu erreichen [26, 64]. Das Ergebnis ist eine niedrigere Belastung des Netzwerks eines BOINC-Projekts und eine schnellere Verteilung der Arbeitspakete, allerdings wurde die Technik nicht in BOINCs Hauptentwicklungszweig übernommen. Der Abschn. 4.2.7 beschreibt den Scheduler detaillierter.

Feeder Der Feeder fügt Arbeitspakete zum Herunterladen in die Warteschlange (engl. queue) vom Scheduler hinzu. Abschnitt 4.2.2 enthält weitere Informationen über diese Applikation, die interne Struktur und logische Vorgehensweise während der Abarbeitung.

Validator Der Validator (zu Deutsch *Validierer*) prüft die Berechnungsergebnisse, die von den Projektteilnehmern zurückgesendet werden. Die BOINC-Quellen beinhalten zwei Beispiele von einfachen Validierern. Abbildung 4.2 zeigt die standardmäßig mit installierten Validierer, die bei der Erstellung eines BOINC-Projekts hinzugefügt werden. Die Namen der zwei Validierer sind `sample_bitwise_validator` und `sample_trivial_validator`. Abschnitt 4.2.4 enthält weitere Informationen über diese Applikationen und darüber, wie diese arbeiten.

Assimilator Wenn der Validierer ein vom Projektteilnehmer erhaltenes Ergebnis geprüft und für O.K. befunden hat, wird der Assimilierer verwendet, um das Ergebnis permanent abzuspeichern. Die Aktion zum Abspeichern eines Ergebnisses kann vom Administrator eines BOINC-Projekts definiert werden. Wieder liefern die BOINC-Quellen zwei Beispiele für Assimilierer mit. Abbildung 4.2 zeigt die zwei Beispielassimilierer `sample_assimilator` und `sample_dummy_assimilator` auf. Abschnitt 4.2.5 diskutiert die Möglichkeit, wie eigene Assimilierer entworfen und implementiert werden können, um Berechnungsergebnisse möglichst einfach abzuspeichern.

File Deleter Dies ist nur eine zusätzliche Applikation, die optional zu einem BOINC-Projekt hinzugefügt werden kann. Ein Arbeitspaket besteht in der Regel aus der Definition einer wissenschaftlichen Applikation und der Beschreibung,

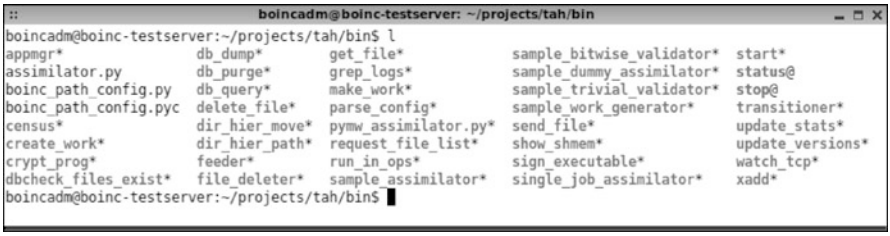


Abb. 4.2 BOINC liefert einige Standardkomponenten für das Validieren und Assimilieren von Arbeitspaketen. Die nachfolgende Liste enthält nur mit einem *Sternchen markierte*, ausführbare Anwendungen: `pymw_assimilator.py` (ein Phyton-Programm), `sample_assimilator`, `sample_bitwise_validator`, `sample_dummy_assimilator`, `sample_trivial_validator` und `single_job_assimilator`

welche Eingabeparameter diese wissenschaftliche Applikation verwenden soll. Dabei ist es grundsätzlich in einem BOINC-Projekt so, dass diese Eingabeparameter in einzelnen Dateien vorliegen, beim Starten einer wissenschaftlichen Applikation ausgelesen und während der Durchführung einer Berechnung verwendet werden. Diese Dateien – je nach Anzahl an vorhandenen Arbeitspaketen – sind gierig und benötigen reichlich Speicherkapazität auf der vorhandenen Festplatte. Es ist daher sinnvoll, diese und alle relevanten Dateien zu löschen, wenn die Bearbeitung eines Arbeitspakets abgeschlossen ist und ein Ergebnis durch den Assimilierer abgespeichert werden konnte. Der File Deleter kann für periodische Ausführung konfiguriert werden. Er überprüft, ob Dateien gelöscht werden können, und tut dies, wenn ein Arbeitspaket fertig bearbeitet ist und die Dateien nicht mehr benötigt werden. Abschnitt 4.2.6 enthält weitere Informationen über diese Applikation.

Die genannten Applikationen übernehmen jeweils eine Aufgabe innerhalb eines BOINC-Projekts. Das spiegelt auch den prinzipiellen Ansatz wider, dass jede Anwendung so *wenig wie möglich*, so *viel wie nötig* an Funktionen besitzen soll, um eine Aufgabe effizient lösen zu können.

4.1.2 Aufgaben eines BOINC-Projektservers

Die BOINC-Architektur ist in verschiedene Aufgabenbereiche unterteilt:

Datenserver (engl. *data server*) Die BOINC-Infrastruktur verwendet für das Speichern aller wichtigen Projektinformationen und für die Prozesse zum Erstellen sowie Verteilen von Arbeitspaketen eine MySQL-Datenbank als Backend-System. In 33 Datenbanktabellen werden alle für die Laufzeit eines BOINC-Projekts wichtigen Daten vorgehalten. Jeder Verarbeitungsschritt – von der Erstellung eines einzelnen Arbeitspakets bis zur fertigen Durchführung einer Berechnung – wird in der Datenbank abgespeichert. Das bedeutet, dass die

Datenbank zu jedem Zeitpunkt Informationen über den Status eines Arbeitspakets liefern kann, zum Beispiel welcher Host für die Berechnung des Arbeitspakets verantwortlich ist oder welche Teilnehmer in den letzten x (Tagen/Stunden/Minuten) gerechnet haben. Eine genauere Untersuchung der einzelnen Datenbanktabellen wird in Abschn. 5.7.1 vorgenommen.

Scheduler Der Scheduler übernimmt die Steuerung für die Verteilung der Arbeitspakete an die BOINC-Projektteilnehmer. Dabei verbinden sich die Teilnehmer mit Hilfe des Hypertext Transfer Protocol (HTTP) zum Scheduler, tauschen Authentifizierungsdaten aus und senden oder empfangen Informationen über die nächsten Arbeitsschritte. Überflüssig zu erwähnen, dass natürlich auch die sichere Variante Hypertext Transfer Protocol Secure (HTTPS) genutzt werden kann. Der Scheduler läuft als Prozess vom Apache Webserver und bietet für Anfragen eine Common-Gateway-Schnittstelle[2] (CGI) an.

Webschnittstelle (engl. *web interface*) Jedes BOINC-Projekt besitzt eine Webseite, auf der sich Benutzer über ein BOINC-Projekt informieren, für ein BOINC-Projekt registrieren und eventuell Antworten auf Fragen finden können. Die Abb. 4.3 und 4.4 zeigen die BOINC-Projektwebseiten von **Spinhenge@home** und **SETI@home**. Es ist klar zu erkennen, dass die Webseiten sehr individuell an die eigenen Wünsche angepasst werden können. Die Webseiten werden dynamisch mit Hilfe von PHP (PHP: Hypertext Preprocessor) erstellt und dem Aufrufer angezeigt. Die Funktionen der Webseiten decken folgende Bereiche ab:

- Es können Informationen bezüglich des Status eines BOINC-Projekts eingeholt werden.
- Aktuelle Informationen können mit Hilfe von RSS (Really Simple Syndication) angezeigt werden[3].
- Benutzer und Anwender können sich bei einem BOINC-Projekt registrieren und so Arbeitspakete vom BOINC-Projekt erhalten.
- In vorhandenen Foren können sich die Benutzer gegenseitig bei Problemen unterstützen oder über die wissenschaftlichen Fragestellungen der einzelnen BOINC-Projekte diskutieren.
- Das persönliche Profil eines jedes BOINC-Teilnehmers kann aufgerufen werden. Die Projektteilnehmer können sich weiterhin in Teams organisieren und zusammen eine Gemeinschaft gründen.
- Die Webseite bietet die Möglichkeit, eine Administration eines BOINC-Projektes durchzuführen. Kapitel 6 beschreibt, wie eine Administration mit Hilfe der Webseite funktioniert.

---

[2] Die CGI-Schnittstelle (Common Gateway Interface – Allgemeine Vermittlungsrechner-Schnittstelle) ist eine Möglichkeit, Programme oder Skripts im Web bereitzustellen, die von HTML-Dateien aus aufgerufen werden und die selbst HTML-Code erzeugen und an einen Web-Browser senden können [165].

[3] RSS ist eine Art von Kurznachrichtendienst. Dabei werden in einer Datei mit Hilfe von XML (Extensible Markup Language) einzelne Einträge abgespeichert, die durch verschiedenste Applikationen angezeigt werden können, zum Beispiel können aktuelle Nachrichten in Microsoft Outlook oder Mozilla Thunderbird stetig eingesehen werden.

Abb. 4.3 Projektwebseite von Spinhenge@home

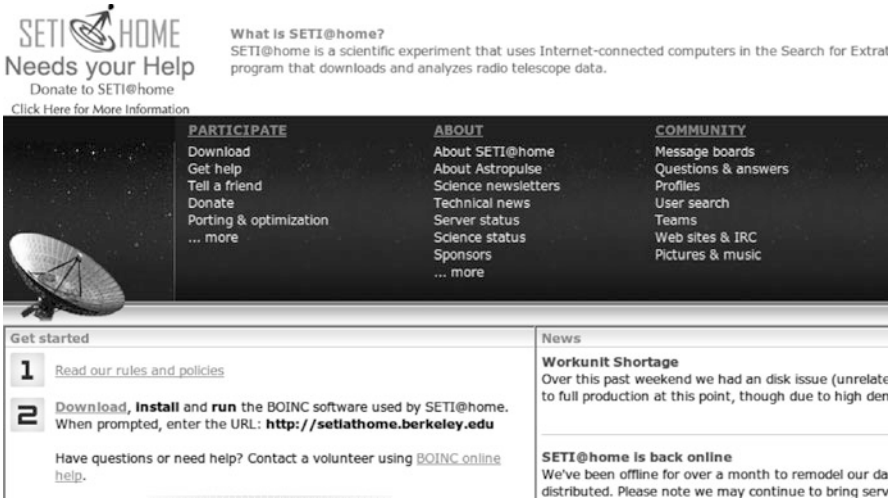


Abb. 4.4 Projektwebseite von SETI@home

Aufgabenserver (engl. *task server*) Ein solcher Server findet u. a. bei SETI@home Einsatz, wobei die von den BOINC-Teilnehmern zu verarbeitenden Daten kontinuierlich vom Aarecibo Observatory (s. Abschn. 2.4.1 und 2.4.2) abgesendet werden. Dieser kontinuierliche Datenstrom wird von einem solchen Dämonen zu kleinen Arbeitspaketen verarbeitet und zum Projekt hinzufügt. Solch ein Server ist natürlich sehr stark von Ihren Eingabedaten abhängig und muss nicht zwingend erforderlich sein. Wenn es Ihnen möglich ist, können Sie schon im Vorfeld alle Arbeitspakete erstellen.

4.2 Komponenten der Serverseite von BOINC

Die BOINC-Infrastruktur besteht aus einer Vielzahl von verschiedenen Anwendungen. Der Grundgedanke ist der, dass jede Applikation eine Aufgabenstellung abarbeitet und die einzelnen Applikationen perfekt dafür entworfen und implementiert werden. Dadurch ist es möglich, dass sich Entwickler nicht zwingend mit allen BOINC-Komponenten zu hundert Prozent auseinandersetzen müssen, um zum Beispiel Erweiterungen zu erstellen oder Fehler zu beseitigen. Jeder kann ein Spezialist werden und dadurch einen signifikanten Beitrag für BOINC liefern.

4.2.1 Verarbeitungsschritte eines BOINC-Projektes

Das BOINC-Framework besitzt mehrere Applikationen, die zusammen das Gesamtsystem eines BOINC-Projekts darstellen. Jeder Aufgabenbereich aus Abschn. 4.1 wird dabei durch eine Applikation repräsentiert. Abbildung 4.5 zeigt den prinzipiellen Verlauf innerhalb eines BOINC-Projekts für die Verarbeitung von Arbeitspaketen. Zu erkennen sind die unterschiedlichen Aufgabengebiete der im letzten Abschn. 4.1.1 und in den nachfolgenden Abschnitten detailliert beschriebenen Applikationen. Dabei wird der Prozess erst dann gestartet, wenn vom Projektadministrator Arbeitspakete zu einem BOINC-Projekt hinzugefügt worden sind. Der Status der Abarbeitung kann durch diesen Arbeitsablauf zu jedem Zeitpunkt ermittelt werden. Es wird davon abgeraten, den Arbeitszyklus direkt in der Datenbank zu manipulieren. Zu diesem Zweck stellt ein BOINC-Projekt eine Webadministrationsseite zur Verfügung. Diese wird in Kap. 6 vorgestellt. Die Abarbeitung eines Arbeitspaketes unterliegt strikten Regelungen. Wenn es in dem Arbeitszyklus zu Unterbrechungen kommt, dann werden keine nachfolgenden Schritte eingeleitet. Es sind hier wirklich Unterbrechungen gemeint, wie zum Beispiel, dass eine Komponente nicht gestartet ist. Wenn ein Berechnungsergebnis vorliegt, welches im nächsten Schritt validiert werden muss und der Validierer nicht gestartet ist, so kann dieses Ergebnis niemals assimiliert werden. Es sind zwischen zwei Arbeitsschritten keine Sprünge möglich, wie dies auch in Abb. 4.5 durch das Verfolgen der Transitionen zu erkennen ist.

Der Bearbeitungsprozess für ein Arbeitspaket ist klar definiert. Zu Beginn erstellt der Projektadministrator ein oder mehrere Arbeitspakete. Für diesen Schritt kann eine eigene Implementierung (vgl. Abschn. 9.2.2) verwendet oder das von BOINC mitgelieferte Skript (vgl. Abschn. 9.2.1) genutzt werden. Daraufhin kommt der Transitioner zum Einsatz, der prüft, ob neue Arbeitspakete vorhanden sind. Wenn ja, werden diese in die Warteschlange für das Herunterladen von den Projektteilnehmern hinzugefügt. Für dieses Hinzufügen ist der Feeder (zu Deutsch *Fütterer*) zuständig, der Informationen über die hinzufügenden Arbeitspakete in seinen Shared-Memory-Bereich schiebt. Wenn nun Anfragen von Teilnehmern (hier *Workers* genannt) eintreffen, wird diese Anfrage vom Scheduler verarbeitet, der dem Worker ein Arbeitspaket zuweist. Der Transitioner ändert den Status eines

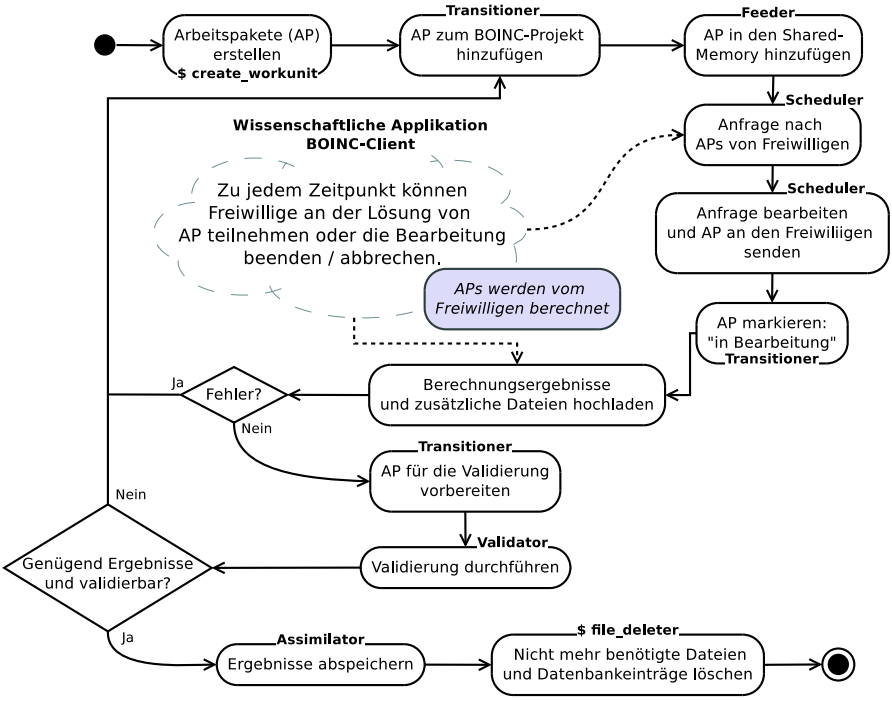


Abb. 4.5 Darstellung eines Ablaufdiagramms der Bearbeitung eines Arbeitspakets innerhalb eines BOINC-Projekts. Der Start ist *oben links*; die Richtung der Abarbeitung wird durch die einzelnen Transitionen zwischen den Kästen beschrieben und endet spätestens *unten in der Mitte* mit dem Löschen von nicht mehr benötigten Dateien [195]

Arbeitspakets auf „wird verarbeitet“ (engl. *in progress*). Nach der Durchführung der Berechnung durch den Worker schickt dieser das Ergebnis zum Projektserver. Die Dateien werden von der CGI-Applikation `cgi_file_uploader` empfangen. Jedes Ergebnis besitzt eine Kennung, die eine Kurzbeschreibung der Ausführung gibt. Dies ist eine einfache Kennung für „erfolgreiche Berechnung“ oder „Fehler bei der Berechnung“.[4] Wenn es einen Fehler bei der Berechnung gab, wird je nach Fehlerart das Arbeitspaket wieder in die Warteschlange von zu bearbeitenden Arbeitspaketen hinzugefügt und wieder an einen Teilnehmer gesendet, so lange bis die Berechnung erfolgreich war oder die Bedingungen für das Abbrechen der Arbeitspaketverteilung erreicht sind. Ist bei der Berechnung allerdings kein Fehler aufgekommen, so wird das Ergebnis validiert und bei erfolgreicher Validierung für die spätere Verwendung durch einen Ingenieur oder Wissenschaftler, mit Hilfe des Assimilierers, abgespeichert.

[4] Informationen bezüglich dieser Fehler finden Sie hardcodiert in den BOINC-Quellen, innerhalb der Datei `db/boinc_db.h`. Ein erfolgreiches Ergebnis wird vom BOINC-Client durch BOINC_OUTCOME_SUCCESS=1 beschrieben, jeder Wert darüber – sprich größer als Eins – beschreibt einen Fehler.

4.2.2 Feeder

Der Feeder erstellt einen Shared-Memory-Bereich, in dem der Feeder Informationen über zu verarbeitende Arbeitspakete vorhält. Es wird versucht, einen ständig vollen Shared-Memory-Bereich zu haben, wenn also ein Arbeitspaket einem Projektteilnehmer für die Berechnung zugewiesen wurde, so wird in Kürze dieser freigewordene Platz durch ein weiteres Arbeitspaket belegt. Dabei wird auch überprüft, ob ein Arbeitspaket nicht ein zweites Mal hinzugefügt wird. Sollte dies der Fall sein, so wird das zusätzliche Arbeitspaket nicht hinzugefügt. Wenn zu einem Zeitpunkt keine neuen Arbeitspakete hinzugefügt werden können – sei es weil die Warteschlange voll ist oder keine neuen Arbeitspakete vorhanden sind –, so wird eine kleine Zeitspanne gewartet und nach Ablauf dieser Zeit ein neuer Versuch gestartet. Dadurch werden nach und nach alle zur Verfügung gestellten Arbeitspakete abgearbeitet.

Um die Performance für die Abarbeitung zu erhöhen, können mehrere Feeder gestartet werden. Beim Starten kann definiert werden, welche Arbeitspakete von den jeweiligen gestarteten Feeder-Instanzen verarbeitet werden sollen. Für die Unterscheidung nutzt der Feeder die Identifikationsnummer von Arbeitspaketen und prüft über eine Modulodivision, ob ein Arbeitspaket von dem jeweiligen Feeder zu verarbeiten ist. Zudem kann definiert werden, für welche wissenschaftliche Applikation die Arbeitspakete bearbeitet werden sollen. Das bedeutet, dass auch hier eine Steigerung der Performance möglich ist, wenn eine Instanz für eine jeweilige wissenschaftliche Applikation gestartet wird.

Weiterhin kann beim Starten die Strategie für die Erstellung einer Reihenfolge für die Auswahl eines Arbeitspakets definiert werden. Dabei kann eine zufällige, eine priorisierte oder vom Erstellungszeitpunkt abhängige Reihenfolge gewählt werden. Diese Einstellungen haben bisher keine signifikante Steigerung oder Verminderung in meinen eigenen BOINC-Projekten ergeben.

4.2.3 Transitioner

Der Transitioner überwacht die Lebenszeit der Arbeitspakete innerhalb eines BOINC-Projekts. Abbildung 4.5 zeigt den ordentlichen Verlauf und die Arbeitsschritte, die auf ein Arbeitspaket einwirken. Teilweise sind mehrere Arbeitspakete notwendig, damit eine Abarbeitung stattfinden kann, wie dies zum Beispiel bei der Validierung von Ergebnissen der Fall sein kann. Dabei werden – je nach Ausführungsmodus – entweder nur einzelne Ergebnisse eventuell auf einen Grenzbereich oder mehrere Ergebnisse gegeneinander geprüft. Der nachfolgende Abschn. 4.2.4 beschreibt diese Anwendung genauer.

Der Transitioner liest den Status eines Arbeitspakets aus der Datenbank aus. Für diesen Zweck enthält die Datenbank mehrere Tabellen, die diese Informationen bereithalten. Abschnitt 5.7 beschreibt die einzelnen Datenbanktabellen eines BOINC-Projekts. Die für den Transitioner wichtigen Tabellen sind `workunit` und `result`.

4.2.3.1 Wichtige Datenbankspalten der Tabelle: workunit

create_time Beschreibt in Sekunden, wann ein Arbeitspaket erstellt wurde.

transition_time Dieser Wert beschreibt einen Zeitstempel von Sekunden seit 1970. Er gibt an, wann ein Arbeitspaket das letzte Mal modifiziert wurde und ob das Arbeitspaket beim nächsten Durchgang eine Prüfung erhalten soll.

need_validate Ist dieser Wert gesetzt, so weiß der Validierer, dass dieses Arbeitspaket genügend Ergebnisse besitzt um geprüft werden zu dürfen, um eventuell im nächsten Schritt durch den Assimilierer abgespeichert zu werden.

target_nresults Ein Arbeitspaket besteht aus einer Anzahl von sogenannten *Results*, die Mindestanzahl an erfolgreichen Berechnungen wird durch diesen Parameter bestimmt. Wenn diese maximale Anzahl überschritten wird, so ist das gesamte Arbeitspaket nicht erfolgreich und wird als solches markiert.

max_error_results Die Anzahl an fehlerhaften Results, bevor das Arbeitspaket als fehlerhaft markiert wird.

max_total_results Der Wert beschreibt die maximale Anzahl an Results für ein Arbeitspaket, welche an BOINC-Teilnehmer gesendet werden können. Wenn der Fall eintritt, dass eventuell ein Arbeitspaket Berechnungen enthält, die nie enden, so beschreibt dieser Wert eine Grenze und bewahrt ein BOINC-Projekt davor, unendlich lange auf erfolgreiche Ergebnisse zu warten. Es kann natürlich passieren, dass ein Wert von drei vielleicht nicht ausreichend ist, wenn mindestens zwei Ergebnisse für die Validierung nötig sind, aber während der Laufzeit des BOINC-Projekts schon zwei Berechnungen verteilt wurden und die BOINC-Teilnehmer nicht die Deadline eingehalten haben. Dies kann so weit führen, dass ein gesamtes BOINC-Projekt nicht erfolgreich ist, wenn sich nur genügend Personen daran beteiligen und immer wieder fehlerhafte Ergebnisse zurücksenden.

max_success_results Die maximale Anzahl an erfolgreichen Results. Wenn dieser Wert erreicht wurde, werden keine weiteren Berechnungsaufgaben an die BOINC-Teilnehmer versendet.

hr_class *hr* steht für *Homogeneous Redundancy* und beschreibt, an was für eine Klasse an Hosts die Berechnungen eines Arbeitspakets verschickt werden. Sie finden in der Datei `sched/hr.cpp` der BOINC-Quellen die definierten Klassenunterteilungen. Es handelt sich dabei um vier Betriebssysteme *(Linux, Windows, Darwin, FreeBSD)* und 80 Prozessortypen *(u. a. Intel Xeon, Celeron, Pentium-Reihe und AMD Athlon-Reihe, Duron und weitere)*.

assimilate_state Dieses Feld kann drei verschiedene Werte besitzen. Null ist der Initialwert, der Wert ist Eins in zwei unterschiedlichen Fällen, abhängig davon, welcher BOINC-Dämon diesen Wert setzt: (1) der Transitioner setzt dieses Feld auf Eins, wenn vorher Null drinstand und wenn das Arbeitspaket fehlerhaft ist, (2) der Validierer setzt Eins, wenn ein Canoncial-Result definiert ist und wenn das Flag für die Assimilierung auf dem Initialwert gesetzt ist. Wenn das Arbeitspaket assimiliert wurde, wird das Feld auf Zwei gesetzt [83].

file_delete_state Zu Beginn hat das Feld den Wert Null, im späteren Verlauf wird der Wert auf Eins gesetzt.

4.2.3.2 Wichtige Datenbankspalten der Tabelle: result

Ein Arbeitspaket wird durch Datenbankeinträge in der `workunit`-Tabelle beschrieben und besitzt ein oder mehr Beschreibungen von Results. Diese Results werden durch Datenbankeinträge in der `result`-Tabelle beschrieben. Erst die Kombination dieser Einträge ermöglicht das Berechnen von Arbeitspaketen auf verschiedenen Hosts und das Validieren von Berechnungsergebnissen der einzelnen Hosts. Sie können bei der Erstellung eines Arbeitspakets spezifizieren, wie viele Results berechnet werden sollen und wie viele für den Validierungsprozess nötig sind. Weitere Informationen dazu finden Sie in den jeweiligen Praxisbeispielen und im Anhang 15.3.3.

outcome Dieses Feld beschreibt das zurückgelieferte Ergebnis, ob es erfolgreich war oder ob ein Fehler vorliegt. Es sind Werte zwischen Null und Sieben in den BOINC-Quellen hardcodiert implementiert:

- RESULT_OUTCOME_INIT=0,
- RESULT_OUTCOME_SUCCESS=1,
- RESULT_OUTCOME_COULDNT_SEND=2,
- RESULT_OUTCOME_CLIENT_ERROR=3,
- RESULT_OUTCOME_NO_REPLY=4,
- RESULT_OUTCOME_DIDNT_NEED=5,
- RESULT_OUTCOME_VALIDATE_ERROR=6 und
- RESULT_OUTCOME_CLIENT_DETACHED=7.

report_deadline Ein Integer-Wert, der das Ende einer Berechnung definiert. Ergebnisse, die nach dieser Grenze an ein BOINC-Projekt zurückgesendet werden, sind nicht mehr gültig und werden verworfen. In der Regel ist die Deadline eine Woche nachdem ein Arbeitspaket an einen BOINC-Teilnehmer versendet wurde.

validate_state Wenn ein Ergebnis erfolgreich validiert wurde, so wird dieses Feld auf Eins gesetzt. Der Initialisierungswert ist Null.

file_delete_state Wenn dieser Wert Null ist, müssen eventuell noch Dateien gelöscht werden. Dieses Feld wird für ein Berechnungsergebnis auf Eins gesetzt, sobald der File Deleter das jeweilige Ergebnis geprüft hat. Dabei ist es nicht relevant, ob Dateien gelöscht wurden; das Feld wird nach der Prüfung auf Eins gesetzt. Der Initialisierungswert ist Null.

4.2.3.3 Arbeitspakete mit einer Mission

Wenn in der `config.xml` eines BOINC-Projekts die Konfiguration `<enable_assignment>1</enable_assignment>` vorgenommen wurde, so werden die Arbeitspakete vom Transitioner daraufhin untersucht, ob der Name eines Arbeitspaketes die Zeichenkette `asgn` (`ASSIGNED_WU_STR`) enthält und wenn ja, so

wird dieses Arbeitspaket gesondert behandelt. In diesem Fall wird das Datenbankfeld `transition_time` auf den maximalen Integer-Wert gesetzt, so dass dieses Arbeitspaket keine Beachtung mehr durch den Transitioner erhält [82].

4.2.4 Validator

Der Validierer überprüft das Ergebnis und setzt das Feld `validate_state` der Datenbanktabelle `result` aus Abschn. 4.2.3.2 auf Eins, wenn die Überprüfung durchgeführt wurde. Der Validierer ist ein Dämon, der, einmal gestartet, bis zum Stoppen in einer Dauerschleife arbeitet und in dieser nach neuen Ergebnissen schaut, um sie zu validieren. Der Entwickler eines BOINC-Projekts kann die Standardvalidierer von BOINC verwenden oder eine eigene Validierungsroutine implementieren. Abschnitt 9.3.1 liefert weitere Informationen, wie Sie eine eigene Implementierung vornehmen oder die Standardvalidierer verwenden können. Die zwei mitgelieferten Validierer bieten folgende Möglichkeiten:

`sample_trivial_validator()`
: Der triviale Validierer ist im Grunde gar kein wirklicher Validierer, sondern dient nur dem Zweck, dass der Status eines Arbeitspakets modifiziert wird, damit der Bearbeitungsprozess nicht unterbrochen wird (s. Abb. 4.5).

`sample_bitwise_validator()`
: Dieser Validierer hat im Gegensatz zum trivialen Validierer eine sehr triviale Logik implementiert, die einen Vergleich zwischen zwei oder mehr Berechnungsergebnissen erstellen kann. Der Validierer prüft die Ergebnisse auf der Bit-Ebene, dadurch müssen die Ergebnisse zu hundert Prozent identisch sein. Allerdings werden hier nicht die Ergebnisse Bit für Bit der Reihe nach geprüft, sondern es werden die schon vorhandenen MD5-Prüfsummen miteinander verglichen.

  Da ein Rechner bei der Berechnung von Fließkommazahlen unterschiedlich runden kann, ist klar, dass wissenschaftliche Applikationen mit Fließkommazahlen als Ergebnis nicht geprüft werden können. Denn die Ergebnisse können sich nach unterschiedlichen Nachkommastellen unterscheiden. Ein Rechner ist begrenzt in der Länge der möglichen abzuspeichernden Nachkommastellen und dadurch kann es in der Folge zu Rundungsfehlern kommen, d. h. ein vorheriges Ergebnis mit anders gerundetem Ergebnis wird für einen nächsten Iterationsschritt verwendet. Dadurch entsteht eine Folge von falsch gerundeten Ergebnissen und das Resultat unterscheidet sich von einem, das durch eine wissenschaftliche Applikation berechnet wurde, die mit den gleichen Eingabeparametern gestartet wurde, aber u. U. einen anderen Prozessor oder eine andere Architektur nutzt. Folgende Prüfungen werden standardmäßig in dieser Reihenfolge durchgeführt:

  - Ist die Anzahl der Ergebnisdateien von zwei Ergebnissen gleich?
  - Sind die einzelnen Ergebnisdateien in derselben Reihenfolge wie bei einer zweiten Liste von Ergebnissen und haben diese die gleichen Inhalte?

Das BOINC-System bietet die Möglichkeit, Arbeitspakete an bestimmte Rechner zu verteilen. Dies ermöglicht das Versenden der Arbeitspakete an Rechner mit gleicher Konfiguration oder Architektur, um den oben genannten Rundungsfehlern entgegenzuwirken. Drei Klassifikationsgruppen werden unterstützt; innerhalb von BOINC „Homogeneous Redundancy“ genannt [1, 28, 85]:

1. Ein deaktiviertes „Homogeneous Redundancy“-System,
2. eine Unterscheidung der Rechner in vier Betriebssysteme, 20 Prozessortypen und
3. eine Unterteilung in nur vier verschiedene Betriebssysteme, gemeint sind Windows, Linux, Mac-PPC und Mac-Intel.

Die Entscheidung, ob und wenn ja, welche Klassifikationsgruppe genutzt werden soll, kann in der `config.xml` eines BOINC-Projekts definiert werden (s. Anhang 15.3.2).
Es ist dem Entwickler eines BOINC-Projekts überlassen, ob er einen dieser zwei Standardvalidierer verwenden will oder ob eine eigene Implementierung mit angepasster Logik in Frage kommt. Die Entwicklung eines eigenen Validierers wird in Abschn. 9.3.1 ausführlich behandelt.

4.2.5 Assimilierer

Der Assimilierer (engl. *assimilator*) dient dem Abspeichern von validierten Berechnungsergebnissen. Wenn Rechenergebnisse einfache Zahlenwerte liefern, dann kann es sinnvoll sein, diese in einer Datenbanktabelle zu speichern. Zu diesen Ergebnissen sollte weiterhin eine Zuordnung der Eingabeparameter möglich sein. Ein Rechenergebnis ist wenig ergiebig, wenn keine Schlüsse mit Hilfe der Berechnungsparameter gezogen werden können. Anders sieht es bei ganzen Berechnungsmodellen aus, wie sie zum Beispiel bei Legacy-Applikationen Verwendung finden. Bei diesen Anwendungen können ganze Modelle zu dem für die Berechnung zuständigen Rechner gesendet und bearbeitet werden. Das Ergebnis wird in der Regel direkt in diesem Modell abgespeichert und deshalb bedarf es einer Abspeicherung dieses ganzen Modells, bestenfalls direkt auf der Festplatte, da eine Abspeicherung in einer Datenbank eventuell nicht in Frage kommt. Die maximale Größe einer MySQL-Tabelle beträgt zwar etwa 64 Terabyte[5], allerdings kann diese Grenze bei einer gewissen Anzahl von Simulationsmodellen schnell erreicht werden. Zum Beispiel können je nach Komplexität der Modelle von der Simulationsapplikation COMSOL Multiphysics[6] schnell Größen von 500 MB erreicht werden. In diesem Fall erscheint es sinnvoller, die Ergebnisse direkt auf einem externen Laufwerk, zum

[5] Tabellengröße für die MySQL-Engines: MyISAM und InnoDb, http://dev.mysql.com/doc/refman/5.1/de/table-size.html.

[6] COMSOL Multiphysics Webseite: www.comsol.de.

Beispiel einem Storage Area Network (SAN)[7] abzuspeichern. Die BOINC-Quellen liefern vier Beispiele für Assimilierer mit:

pymw_assimilator.py
Dieser Assimilierer ist in der Programmiersprache Python implementiert und speichert Berechnungsergebnisse in einen vordefinierten Dateiordner. Der externe Dateiordner kann beim Starten dieses Assimilierers mit angegeben werden. Für diese Angabe muss das Flag `-pymw_dir` mit Pfadangabe angegeben sein. Es ist kein Standardordner angegeben, so dass bei jedem Starten dieses Assimilierers eine Pfadangabe vorhanden sein muss.

sample_assimilator
Mit Hilfe dieses Assimilierers können wie beim ersten die Ergebnisse in einem Dateiordner abgespeichert werden. Falls doppelte Ergebnisse – aufgrund der Konfiguration eines BOINC-Projekts zur Prüfung auf Richtigkeit – vergeben worden sind, so werden die mit einer fortlaufenden Nummerierung versehen. Die Dateiordner für die Abspeicherung sind vorgegeben, im Fall von erfolgreichen Berechnungen werden die Ergebnisse im Ordner `sample_results/` und in einem Fehlerfall in `sample_results/errors` abgespeichert. Beide Ordner befinden sich in einem Unterordner eines BOINC-Projekts.

sample_dummy_assimilator
Dieser Assimilierer ist im Grunde kein wirklicher Assimilierer, er dient wiederum nur der Komplettierung des Ausführungskreislaufs aus den vorherigen Abschnitten. Beim Eingang von ein oder mehr Arbeitspaketen wird das Ergebnis nicht direkt verarbeitet. Es wird nur eine Meldung über die Verarbeitung abgespeichert und der Status für das oder die Arbeitspakete modifiziert, damit der nächste Schritt der Abarbeitung – das Löschen von unnötigen Dateien – für dieses Arbeitspaket oder diese Arbeitspakete gestartet werden kann.

single_job_assimilator
Dieser Assimilierer sollte nur verwendet werden, wenn Ihnen wirklich bewusst ist, was dieser tut. Der Projektadministrator kann kleine Arbeitspakete erstellen, die nur aus einer Aufgabe bestehen und aus diesem Grund *Single-Jobs* heißen. Single-Jobs erhalten eine einzelne Beschreibungsdatei, die einen Dateiordner beschreibt, in dem sich alle Informationen befinden. Nachdem, wie bei den ersten beiden, das Ergebnis in einem Dateiordner abgespeichert wurde, werden alle nicht mehr benötigten Dateien wie die Template-Dateien gelöscht. Das Ergebnis wird im Dateiordner dieses einen Arbeitsauftrags gespeichert.

Wiederum kann eine eigene Implementierung umgesetzt werden; Abs. 9.3.2 erklärt Ihnen, wie.

[7] Ein Storage Area Network (SAN) dient dem Abspeichern einer großen Menge von Daten, die hochperformant und mit Hochgeschwindigkeit erreichbar sein sollen.

4.2.6 File Deleter

Wenn ein Arbeitspaket berechnet wurde und ein Ergebnis vorliegt, werden in vielen Fällen die Eingabedateien nicht mehr benötigt. Die Eingabedateien wurden nur zu dem Zweck erzeugt, dass diese an die einzelnen Teilnehmer zum Ausführen versendet werden können. Nachdem eine erfolgreiche Berechnung und Validierung stattfand, sind diese Dateien im Nachgang überflüssig geworden. Eine Zuordnung der Ergebnisse zu den Eingabeparametern sollte in jedem Fall noch möglich sein, so dass diese Informationen eventuell vom Assimilierer mit abgespeichert werden sollten. Der File Deleter muss nicht von Hand entwickelt werden, die mit BOINC mitgelieferte Implementierung ist in den meisten Fällen ausreichend. Sollten weitere Funktionalitäten gewünscht werden, so kann die Implementierung entsprechend angepasst werden.

4.2.7 Scheduler

Der Scheduler verarbeitet die Anfragen von den an einem BOINC-Projekt Teilnehmenden. Es ist eines der aufwendigsten Applikationen innerhalb des BOINC-Systems.

4.2.8 Webseite

Die Webseite eines BOINC-Projekts lässt sich individuell erstellen und ganz an die eigenen Wünsche anpassen. Nach der automatischen Erstellung eines BOINC-Projekts mit Hilfe des Skripts `make_project` aus Abschn. 5.6 kann man die Einstellungen im Unterordner `html/` vornehmen. Wir werden in diesem Buch keine Beschreibungen oder Beispiele liefern, wie eine Modifikation durchgeführt werden kann oder sollte. Aktuell[8] werden starke Änderungen der Webschnittstelle durch die BOINC-Entwickler umgesetzt.

Dennoch möchte ich Ihnen einen kleinen Überblick über die Möglichkeiten der Einstellungen geben. Allgemeine Webseiteneinstellungen können Sie in der Datei project/project.inc vornehmen. Dies betrifft u. a. folgende Möglichkeiten:

Allgemeine Bezeichnungen Es können Einstellungen von Standardtexten vorgenommen werden, z. B. der Titel eines Projekts oder wer das Copyright an der Webseite besitzt.

Kommunikationswerte An wen e-Mails versendet werden sollen und welche Kontaktadressen der Webadministrator oder Projektadministrator besitzt.

Standardbereiche In der Datei sind mehrere Funktionen implementiert, die jeweils Texte für die Darstellung innerhalb der Webseite zurückgeben. Dies betrifft den oberen Webseitenbereich (Banner), den unteren Webseitenbereich (Footer) und auch, wie die Profil-/Projektinformationen dargestellt werden sollen.

[8] 17. Oktober 2011.

Authentifizierungszugang Ein wichtiger Punkt ist die Möglichkeit, dass Sie in dieser Datei entscheiden können, wie Ihre Webauthentifizierung auszusehen hat: Ob Sie z. B. einen htaccess-Zugang oder eine Datenbankauthentifizierung nutzen wollen. Sie können hier auch eine Implementierung vornehmen, um zu prüfen, von welchem entfernten Host Sie sich verbinden dürfen, z. B. durch die Prüfung der IP-Adresse.

In den zwei Dateien

- project/cache_parameters.inc und
- project/project_specific_prefs.inc

können weitere Einstellungen vorgenommen werden. In beiden Dateien finden Sie Variablen vor, die von der Namensgebung her gut verständlich sind; zusätzliche Kommentare erleichtern den Umgang. Für ein funktionsfähiges BOINC-Projekt ist keine öffentliche Webseite nötig, allerdings ist es von Vorteil, wenn die Webadministrationsseite zur Verfügung gestellt wird. Der Umgang mit dieser Webadministrationsseite wird in Kap. 6 diskutiert.

4.3 BOINC-Komponenten der Clientseite

Wie schon in Abb. 4.1 zu sehen war, kann die Clientseite aus den zwei Komponenten BOINC-Client und BOINC-Manager bestehen. In jedem Fall muss mindestens der BOINC-Client zur Verfügung stehen, da dieser die Kommunikation mit einem BOINC-Projekt ermöglicht und das Starten, Pausieren und Stoppen von wissenschaftlichen Applikationen organisiert.

4.3.1 BOINC-Client

Der BOINC-Client ist für die ordentliche Kommunikation zu allen hinzugefügten BOINC-Projekten zuständig und regelt das Scheduling der wissenschaftlichen Applikationen. Wenn Sie das Softwarepaket [78] für BOINC-Teilnehmer herunterladen und installieren, erhalten Sie den BOINC-Client, den BOINC-Manager und ein Werkzeug, mit dessen Hilfe Sie einen BOINC-Client über eine Netzwerkverbindung kontrollieren können.

Die Kommunikation und die Steuerung vom BOINC-Manager zum BOINC-Client geschieht über eine Netzwerkverbindung; dies kann man in Abb. 4.10 unten rechts erkennen. Dort ist eine Verbindung nach *192.168.1.200* vorhanden; Sie könnten ebenso einen externen BOINC-Client nutzen. Es ist in diesem Fall allerdings nötig, dass der externe BOINC-Clients externe Verbindungs- und RPC-Anfragen zulässt. Doch dazu lässt sich der BOINC-Client ohne Probleme bringen, indem Sie beim manuellen Starten eines BOINC-Client bestimmte Parameter mit anhängen:

```
$ ./boinc --allow_remote_gui_rpc --gui_rpc_port 31416
```

Der erste Parameter erlaubt externe Verbindungen. Da es sich um Netzwerkverbindungen handelt, können Sie zusätzlich definieren, über welchen Verbindungsport dies geschehen soll. Hier wird der Port 31416 angegeben, welcher übrigens auch den Standardwert darstellt. Nach dem erfolgreichen Starten können Sie sich zu dieser BOINC-Client-Instanz verbinden und diese steuern; es ist natürlich nötig, dass der entsprechende Firewall-Port für solche Verbindungen konfiguriert ist.

Eine zusätzliche Sicherheitsblockade ist standardmäßig aktiv; Sie benötigen ein Passwort, um sich verbinden zu können. Es wird nur ein Passwort benötigt und keine weiteren Benutzerdaten. Die Angabe eines solchen Passworts geschieht in der Datei `gui_rpc_auth.cfg` im Dateiordner vom BOINC-Client und enthält einen einfachen Plain-Text; diesen können Sie durch Ihr Wunschpasswort abändern. Wenn die Datei leer ist, können Sie sich ohne Angabe eines Passwortes verbinden; davon rate ich Ihnen ab, weil sich dadurch jeder verbinden kann und alle Einstellungsmöglichkeiten erhält.

4.3.1.1 BOINC-Commander: Fernsteuerung des BOINC-Client

Mit dem BOINC-Commander (BCMD) können in der Version 6.12.34 insgesamt 31 administrative Befehle durchgeführt werden. Der BCMD unterstützt die komplette Steuerung eines BOINC-Clients und kann als Textersatz dienen, wenn keine grafische Oberfläche vorhanden ist und dadurch der Einsatz vom BOINC-Manager entfällt. Wie im vorherigen Abschnitt erwähnt, kann mit dem BCMD auch ein entfernter BOINC-Client gesteuert werden, so dass für die persönlichen Bedürfnisse Administrationsskripte erstellt und genutzt werden können, z. B. kann man über eine Hostauflistung iterieren und so jeden einzelnen Host bei einem BOINC-Projekt anmelden, wie dies in Listing 4.1 exemplarisch gezeigt wird.

Listing 4.1 Iteration über eine Auflistung von Hosts und deren Hinzufügen zum BOINC-Projekt **Spinhenge@home**

```
#!/bin/bash

declare -a hosts=( \
  host1.infake-environment.de \
  host2.infake-environment.de \
  host3.infake-environment.de \
)

BOINC=/home/cr/downloads/BOINC

echo "Count of Hosts: ${#hosts[@]}"
for host in "${hosts[@]}"
do
  ${BOINC}/boinccmd --host ${host} --passwd boincadm \
    --project_attach http://spin.fh-bielefeld.de \
    502826_95f9b93dfa837cca52cc5cea67d835fd
done
```

Sie können sich die Parameter und möglichen Kommandos von BMCD mit

```
$ ./boinccmd --help
```

anzeigen lassen. Die einzelnen Zeilen beinhalten sehr gute Beschreibungen und listen die zum Befehl gehörigen Einstellungsmöglichkeiten auf. Wie im vorherigen

Abschn. 4.3.1 kann ein BOINC-Client durch die Abfrage von Authentifizierungsdaten abgesichert werden. Zeile 14 in Listing 4.1 zeigt die Angabe dieser Daten, erst bei Erfolg werden die nachfolgenden Parameter verarbeitet. Interessant ist hier zudem die Zeile 16; die Beschreibung lautet folgendermaßen:

> –project_attach URL auth attach to project

Zugegeben, diese Beschreibung ist eventuell nicht ganz einleuchtend, `URL` (Uniform Resource Locator) beschreibt die Webadresse von einem spezifischen BOINC-Projekt und ist im Grunde die Master-URL, z. B. http://spin.fh-bielefeld.de. `auth` ist ein Authentifizierungsschlüssel, den es in zwei Varianten gibt (siehe auch Abb. 4.6):

Account key Hierbei handelt es sich um die zufallsgenerierte Zeichenkette aus Listing 4.3 in Zeile 12.

Weak account key Dieser Schlüssel ermöglicht es, dass Sie x-beliebige Rechner zu Ihrem Benutzerkonto hinzufügen können und für sich rechnen lassen. Das heißt, mit Hilfe dieses Schlüssels können Sie sich bei einem BOINC-Projekt authentifizieren, allerdings kann man mit solch einer Authentifizierung keine Einstellungen von dem Benutzerkonto vornehmen, weshalb dies auch als „schwaches Konto“ (*weak account*) bezeichnet wird. Es steht Ihnen frei, meine schwache Identifikation zu nutzen – in diesem Fall handelt es sich um den Schlüssel für Einstein@home.

Es gibt zwei Möglichkeiten, diese Schlüssel mit einem BOINC-Client bekannt zu machen: (1) Sie können eine Datei mit dem Namen „account_PROJEKTADRESSE .xml“ innerhalb Ihrer BOINC-Client-Installation erstellen, ein Beispiel für den möglichen Inhalt finden Sie in Listing 4.2, oder (2) verwenden Sie den schon in Listing 4.1 vorgestellten Befehl, um ein BOINC-Projekt zu einem bestimmten BOINC-Client hinzuzufügen.

Listing 4.2 Datei mit Master-URL und Authentifizierungsschlüssel für das Projekt Einstein@home

```
<account>
  <master_url>http://einstein.phys.uwm.edu/</master_url>
  <authenticator>
    502826_95f9b93dfa837cca52cc5cea67d835fd
  </authenticator>
</account>
```

Für die Applikationen, die Gebrauch von dem schwächeren Schlüssel machen, müssen Sie bei Änderung des Passwortes oder der E-Mail-Adresse diesen Schlüssel bei allen betroffenen Applikationen aktualisieren. Der Grund dafür liegt in der Implementierung zur Erzeugung dieses Schlüssels. Der Schlüssel ist nicht in der Datenbank abgespeichert, sondern wird für jede Verwendung erstellt. Wie dies aussieht, ist in Listing 4.3 gezeigt. Die Funktion `weak_auth` in Zeile 18 erstellt den schwachen Schlüssel, dabei wird der zufallsgenerierte `authenticator` aus Zeile 12 eines Benutzers benötigt sowie die MD5-Hash-Summe von dem Passwort und der E-Mail-Adresse eines Benutzers aus Zeile 3. Sie erkennen womöglich den Unterschied der Programmiersprachen, in diesem Fall sind dies C++ und PHP. Bei den

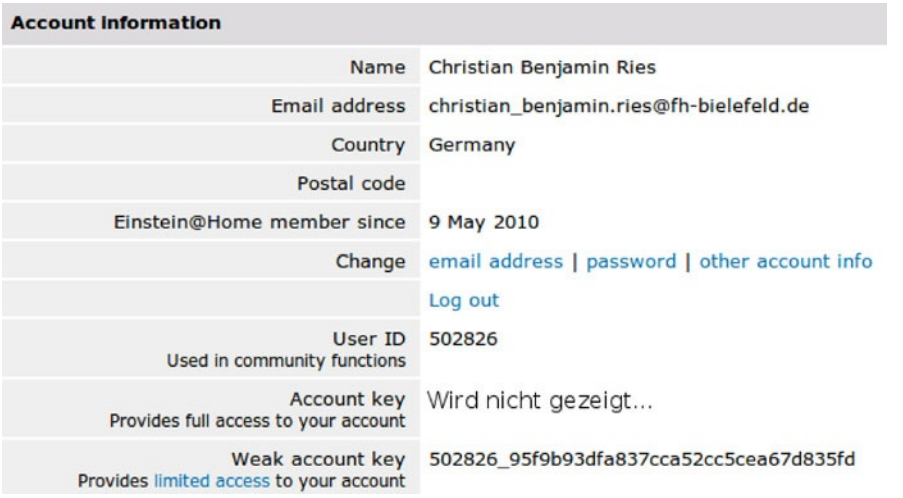


Abb. 4.6 Von BOINC erstellte und nutzbare Account-Keys, einer dient dem Anmelden und der vollen Steuerung von entsprechenden Einstellungen für eine volle Projektanmeldung. Der zweite kann weitergegeben werden, so dass weitere Computer zu einem Benutzerkonto hinzugefügt werden können

Aufrufen aus Listing 4.1 werden RPC-Anfragen erstellt, die alle an entsprechende Webseiten eines BOINC-Projektes weitergeleitet werden, so dass diese dort bearbeitet werden. In diesem Fall wird eine entsprechende Anfrage an die PHP-Datei *create_account.php* weitergeleitet, dort werden die Informationen eines neu zu erstellenden Benutzers eingelesen und daraufhin wird ein Benutzer mit dem erwähnten `authenticator` erstellt. Diese Prozedur wird durch eine XML-Meldung mit dem neuen `authenticator` beendet, so dass die Applikationen diesen nun für die nächsten Schritte verwenden können.

Listing 4.3 Erstellung des *Weak account key*

```
// lib/gui_rpc_client_ops.cpp
static string get_passwd_hash(string passwd, string email_addr) {
  return md5_string(passwd+email_addr);
}

// html/inc/user.inc
function make_user(
    $email_addr, $name, $passwd_hash,
    $country=null, $postal_code=null, $project_prefs=null, $teamid=0
) {
  ...
  $authenticator = random_string();
  $cross_project_id = random_string();
  ...
}

// html/inc/user.inc
function weak_auth($user) {
  $x = md5($user->authenticator.$user->passwd_hash);
  return "{$user->id}_$x";
}
```

4.3.2 Slot-System

Abbildung 4.7 zeigt die Struktur des Slot-Systems auf Clientseite und wie dieses prinzipiell funktioniert. Der BOINC-Manager erstellt für jedes angemeldete BOINC-Projekt innerhalb von `projects/` einen Unterordner, in dem die wissenschaftliche Applikation und die zugehörigen Dateien[9] abgespeichert sind. Der BOINC-Projektverwalter hat womöglich Arbeitspakete erstellt, die alle mindestens eine Datei beinhalten, die sich im Lauf aller vorgesehenen Berechnungen nie verändert und daher immer für die gleiche wissenschaftliche Applikationen verwendet werden kann. In diesem Beispiel existiert die Parameterdatei `parameter.input` mit den zwei beinhalteten, für die Laufzeit wichtigen Parametervariablen:

iterations Diese Variable beschreibt vielleicht die Anzahl an Iterationen, die für eine statistische Berechnung ausgeführt werden sollen.

mode Beschreibt die Berechnungsmethode für die statistische Berechnung, hier nehmen wir eine *Monte Carlo Simulation* als Beispiel an, welche mit der zuvor genannten Anzahl an Iterationen durchgeführt wird.

Diese Parameterdatei kann in dem oben genannten Ordner `projects/` abgespeichert werden. Bei der Ausführung einer wissenschaftlichen Applikation wird eine Datei erstellt, die einen Soft-Link zu dieser Datei enthält. Genau dies tut der BOINC-Client: Wenn Dateien zu einem bestimmten BOINC-Projekt heruntergeladen werden, so werden diese in dem beschriebenen Dateiordner abgelegt, so dass alle Informationen später bei der Berechnung eines Arbeitspakets mit Hilfe von Slots dynamisch hinzugefügt werden können. Die zwei Funktionen aus Listing 7.12, zum Auflösen eines virtuellen Dateinamens zu einem physikalischen Dateinamen arbeiten prinzipiell äquivalent. Im Unterschied zur ersten Funktion kann die zweite mit C++-Standardstrings umgehen und es wird empfohlen, diese Funktion zu ver-

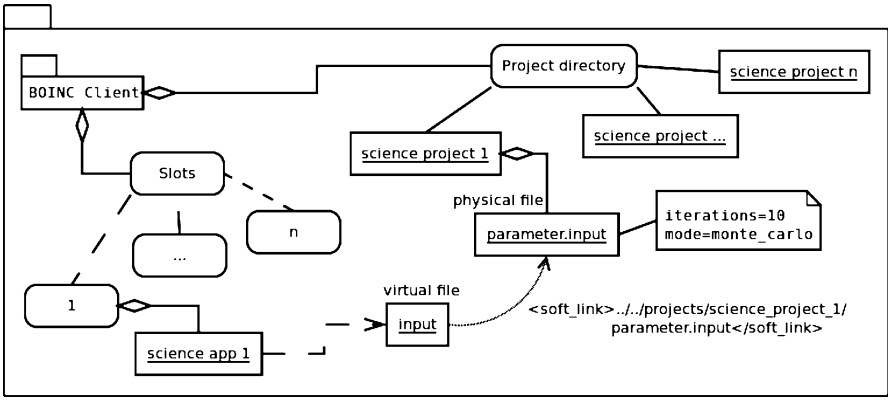


Abb. 4.7 Veranschaulichung der Struktur des Slot-Systems vom BOINC-Client

[9] Bildschirmschoner, Bilddateien, zusätzliche Konfigurationsdateien, usw.

wenden. Dadurch ist der Entwickler von der Verantwortung befreit, sich zu jedem Zeitpunkt bewusst zu sein, wie viel Zeichen der Dateiname haben darf. Die Verwendung beugt Speicherzugriffsfehlern vor.

Listing 4.4 zeigt eine Dateisystemstruktur, in der fünf Projekte zum BOINC-Manager hinzugefügt worden sind. Jedes dieser Projekte enthält mehrere Dateien oder Ordner.

Listing 4.4 Auflistung von Projekten, die einem BOINC-Manager hinzugefügt worden sind

```
projects/
|-- boinc.bakerlab.org_rosetta
|   `--
|-- climateprediction.net
|   `--
|-- einstein.phys.uwm.edu
|   `--
|-- milkyway.cs.rpi.edu_milkyway
|   `--
|-- spin.fh-bielefeld.de
    `--
```

Listing 4.5 zeigt, dass fünf Arbeitspakete vorhanden sind. Dieser erste Überblick gibt keine Auskunft darüber, wie viele Berechnungen im Moment ausgeführt werden. Allerdings wird verdeutlicht, wie essentiell das Verständnis für das Slot-System ist. In den einzelnen Slots werden nicht nur Eingabeparameterdateien über einen Soft-Link beschrieben, sondern sogar die wissenschaftlichen Applikationen selber. In diesem Fall zum Beispiel die zwei aufgelisteten:

- milkyway_0.50_x86_64-pc-linux-gnu__sse2
- einstein_S5GC1HF_1.06_i686-pc-linux-gnu__S5GCESSE2

Durch diesen Vorgang werden Systemressourcen eingespart und das Entwickeln von wissenschaftlichen Applikationen erleichtert. Eingabeparameterdateien müssen nur über einen virtuellen Namen geöffnet werden und verweisen auf unterschiedliche Parameterdateien. Dies erhöht die Flexibilität und ermöglicht es, einen pflegeleichteren Quellcode zu schreiben.

Listing 4.5 Eine Auflistung der Slots für die zum BOINC-Manager hinzugefügten Arbeitspakete

```
slots/
|-- 0
|   `-- astronomy_parameters.txt
/* <soft_link>../../projects/milkyway.cs.rpi.edu_milkyway/
 *  parameters-17-3s-fix.txt</soft_link>
 */
|   `-- milkyway_0.50_x86_64-pc-linux-gnu__sse2
|
/* <soft_link>../../projects/milkyway.cs.rpi.edu_milkyway/
 * milkyway_0.50_x86_64-pc-linux-gnu__sse2</soft_link>
 */
|    ...
|
|-- 1
|   `-- einstein_S5GC1HF_1.06_i686-pc-linux-gnu__S5GCESSE2
/* <soft_link>../../projects/einstein.phys.uwm.edu/
 *  einstein_S5GC1HF_1.06_i686-pc-linux-gnu__S5GCESSE2
 * </soft_link>
 */
```

```
|   `-- h1_1260.60_S5R4
|
/* <soft_link>../../projects/einstein.phys.uwm.edu/
 *  h1_1260.60_S5R4</soft_link>
 */
|   ...
|
|-- 2
|   `-- ...
|-- 3
|   `-- minirosetta_2.17_x86_64-pc-linux-gnu
/* <soft_link>../../projects/boinc.bakerlab.org_rosetta/
 * minirosetta_2.17_x86_64-pc-linux-gnu</soft_link>
 */
|   `-- minirosetta_database.zip
|
/* <soft_link>../../projects/boinc.bakerlab.org_rosetta/
 * minirosetta_database_rev39052.zip</soft_link>
 */
|   `-- minirosetta_database
|       `-- branch_angle
|       `-- chemical
|       ...
|
`-- 4
    `-- ...
```

Listing 4.6 enthält ein einfaches Beispiel zum Öffnen von Dateien, die durch einen Soft-Link beschrieben werden. In dem Beispiel wird ein ZIP-Archiv als Parametereingabedatei erwartet. Der virtuelle Dateiname ist `INPUT_FILE`. `boinc_resolve_filename_s` versucht diese Datei zu öffnen, prüft ob ein Soft-Link vorhanden ist und speichert in `inputArchivePath` entweder den aus dem Soft-Link ausgelesenen physikalischen Dateinamen oder direkt den virtuellen Dateinamen, wenn es sich nicht um einen Soft-Link handelt. Danach kann ganz normal mit dem Dateinamen gearbeitet werden und alle möglichen Dateioperationen auf diese Datei ausgeführt werden.

Listing 4.6 Ein Beispiel für das Arbeiten mit virtuellen und physikalischen Dateinamen

```
...
std::string inputArchivePath;
struct zip *zipFileIn = NULL;
char zipError[512] = {'\0'};

boinc_resolve_filename_s(INPUT_FILE, inputArchivePath);
zipFileIn = zip_open(inputArchivePath.c_str(),
                     ZIP_CHECKCONS, NULL);
if (zipFileIn == NULL) {
  zip_error_to_str(zipError, 512, 0, errno);
  std::cerr << "zip failed: " << zipError << std::endl;
  boinc_finish(-1);
}
...
```

Es sei noch erwähnt, dass diese Funktionen prinzipiell nicht nur innerhalb des Slot-Systems verwendet werden können. Innerhalb der Funktionen wird nicht geprüft, ob man sich im Moment in einem Slot befindet. Dadurch ist es nicht möglich, wahlfreie symbolische Links zu verwenden, denn jede Datei kann einen Soft-Link enthalten, der beim Öffnen automatisch aufgelöst wird.

4.3.2.1 Datenaustauschen zwischen BOINC-Client und wissenschaftlicher Applikation

Der BOINC-Client kann mehrere Instanzen von wissenschaftlichen Applikationen starten. Dabei können die jeweiligen wissenschaftlichen Applikationen verschiedene Projektzugehörigkeiten besitzen. Je nachdem, wie die Einstellungen eines BOINC-Projekts sind, können auch gleiche Arbeitspakete von einer wissenschaftlichen Applikation gestartet werden. Arbeitspakete sind immer spezifisch für eine wissenschaftliche Applikation. Das heißt, ein Arbeitspaket für das BOINC-Projekt „Spinhenge@home“[10] kann nicht von der wissenschaftlichen Applikation „Einstein@home“[11] gestartet werden.

Jede gestartete wissenschaftliche Applikation steht in Kommunikation mit dem BOINC-Client. Auch der BOINC-Manager ist dauerhaft mit dem BOINC-Client verbunden. Der BOINC-Client ist im Grunde das Herzstück beim Anwender. Der BOINC-Client ist verantwortlich für das Handhaben der Arbeitspakete, führt die wissenschaftlichen Applikationen mit allen Eingabe- und Ausgabedateien in einem abgeschlossenen System (einem sog. Slot) aus und sendet die Ergebnisse an ein BOINC-Projekt. Die Kommunikation zwischen BOINC-Manager, BOINC-Client und einer wissenschaftlichen Applikation wird durch die Verwendung von Shared-Memory ermöglicht. Zwischen zusammenhängenden Applikationen kann eine Kommunikation stattfinden. In der Informatik wurden einige Ansätze entwickelt, wie solch eine Kommunikation genutzt werden kann. Allein Windows unterstützt neun verschiedene Ansätze [149]. BOINC verwendet die nachfolgenden Kommunikationsansätze:

File Mapping Diese Art der Interprozesskommunikation (IPC) (engl. *inter-process communication*) findet im BOINC-Framework mit Hilfe des Shared-Memory Verwendung und wird im Abschnitt „Virtueller Speicher mit Memory-Mapped-Files“ näher behandelt.

RPC Remote Procedure Call (RPC) ist ein Client-Server-Entwicklungsparadigma [56, 62]. Es dient dem Nutzen von Funktionalitäten, welche sich innerhalb eines Netzwerks befinden und über eine Netzwerkverbindung aufgerufen werden können. Die Funktionalitäten können sich dabei entweder auf unterschiedlichen Rechnern befinden oder auch auf dem lokalen Rechner, in beiden Fällen muss der Rechner mit der anzubietenden Funktionalität eine Serverinstanz bereitstellen und eingehende Verbindungen akzeptieren. RPC-Implementierungen dienen dabei der Bereitstellung von spezifizierten Schnittstellen mit einem bekannten Verhalten, so dass eine angepasste RPC-Anfrage gestellt und ein bekanntes Ergebnisformat erwartet werden kann.

Sockets Diese Sockets sind die Basis der modernen Kommunikation. Zu Beginn nur auf UNIX oder UNIX-ähnlichen Systemen verfügbar, sind diese heutzutage in jedem modernen Kommunikationssystem vorhanden und je nach Implementierung für unterschiedliche Betriebssysteme dynamisch erweiterbar. BOINC

[10] http://spin.fh-bielefeld.de.

[11] http://einstein.phys.uwm.edu/.

verwendet Sockets zur Erstellung von Kommunikationen zwischen dem BOINC-Client und einem BOINC-Projekt.

Pipes Röhren (engl. *pipes*) sind Kommunikationskanäle. Für jede Kommunikationsrichtung muss eine Röhre geöffnet werden. Die Erstellung von Röhren ist teuer, da im Vorfeld meist der gesamte Prozess kopiert werden muss und die richtigen Einstellungen für die Kommunikationsrichtungen getroffen werden müssen.

Andere Formen Die Liste kann um weitere Techniken erweitert werden, u. a. Clipboard, Object Linking and Embedding (OLE), Component Object Model (COM), Dynamic Data Exchange (DDE), Mailslots und sicher zahlreiche weitere. Die letzten fünf erwähnten sind in der aktuellen BOINC-Version nicht vertreten.

Nicht jeder Ansatz sollte für eine bestimmte Umsetzung verwendet werden. Hierbei ist es wichtig zu entscheiden, welcher Ansatz für ein Problem der geeignetste ist, um ein Problem effizient zu lösen. Zum Beispiel ist es nicht sehr sinnvoll, eine Anwendung mit Mailslot-Unterstützung zu entwickeln, wenn eine *full-duplex*-Kommunikation benötigt wird. Dies wird unter anderem für das Entwickeln eines Bildschirmschoners in Kap. 8 benötigt, wenn Informationen über den aktuellen Berechnungsstatus angezeigt werden sollen. Der Entwickler muss dafür einen Shared-Memory-Speicherplatz erstellen und diesen für den Datenaustausch verwenden. Dabei können alle beteiligten Applikationen Daten auf diesen Speicherplatz speichern und die anderen Applikationen können diesen zu jedem Zeitpunkt auslesen.

Was ist Shared-Memory?

In der IPC können sich mehrere Applikationen über einen reservierten Speicherplatz untereinander Informationen zusenden [149]. Es gibt verschiedene Ansätze, die ein Entwickler bei der Verwendung einer IPC anwenden kann. Die populärsten Ansätze sind das Pipe-Prinzip und der *Shared-Memory*-Ansatz. Beide Verfahren sind unter Windows und Unix/Linux anwendbar, weshalb diese beiden Verfahren hier kurz erläutert werden. Das Pipe-Prinzip wird im BOINC-Framework nicht häufig genutzt, eine Verwendung ist innerhalb von Wrappern in Kap. 13 zu sehen, um Vater- und Kindprozesse miteinander kommunizieren zu lassen. Für die Entwicklung von eigenen Applikationen ist es nicht zwingend erforderlich, diese beiden Techniken zu beherrschen, denn auch ohne kann man recht ordentliche Anwendungen entwickeln.

Pipes, Kommunikationskanäle

Die Kommunikation mit Hilfe einer Pipe (zu Deutsch Röhre) wird exemplarisch in Abb. 4.8 gezeigt, welche eine *half-duplex*- sowie eine *full-duplex*-Variante beinhaltet [47]. Beide Varianten besitzen jeweils einen virtuellen Kommunikationskanal –

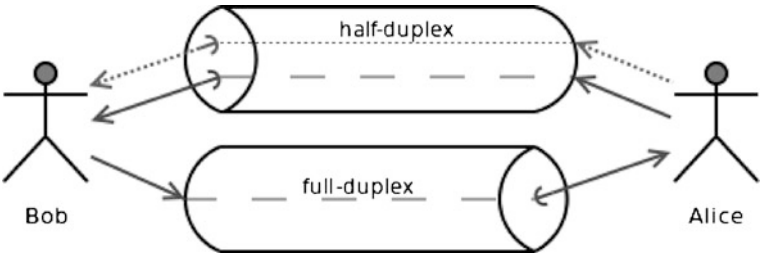


Abb. 4.8 Aufbau der Richtungskanäle bei einer *half-* (*gepunktete Linie*) sowie *full-duplex-* (*gestrichelte Linie*) Kommunikation. Bei einer *half-duplex*-Kommunikation kann Alice zu Bob nur senden und nichts empfangen, eine *full-duplex*-Kommunikation kann eine Information rein theoretisch kreisen lassen, da sich beide Daten zukommen lassen können

die Pipe an sich, über die Daten ausgetauscht werden können; allerdings kann eine Pipe nur eine unidirektionale[12] Kommunikation zulassen.

Bei einer *half-duplex*-Verbindung kann eine Seite immer nur senden oder empfangen, niemals beides gleichzeitig über einen Verbindungskanal. Im Gegensatz dazu steht die *full-duplex*-Variante: Hier können die jeweiligen beteiligten Prozesse sich gegenseitig Daten zusenden und empfangen. In dieser Variante müssen dafür zwei Pipes erstellt werden, so dass die jeweiligen Pipes von einer Seite entweder zum Schreiben oder Lesen verwendet werden und dadurch eine bidirektionale Kommunikation ermöglichen.

Im Allgemeinen ist das Verwenden von Pipes teuer. Für die Erstellung einer Pipe muss der gesamte Prozess, der eine Pipe-Kommunikation erhalten soll, erst kopiert werden. Abhängig von dem entsprechenden Prozess kann dies lange dauern und auch viele weitere Ressourcen für den Prozess beanspruchen. Im nächsten Schritt müssen die Kanäle auf das gewünschte Verhalten (Empfangen, Senden) konfiguriert werden und im Verlauf der Programmausführung immer wieder geprüft werden, ob Daten zum Lesen oder Senden bereitliegen, um diese dann zu verarbeiten.

Virtueller Speicher mit Memory-Mapped-Files

Eine zweite Möglichkeit ist das Verwenden von virtuellem Speicher (Abb. 4.9). Virtueller Speicher wird in Form von Shared-Memory innerhalb der BOINC-Implementierung verwendet. Bei dieser Technik wird ein Speicherbereich im physikalischen Speicher – dem sogenannten Random Access Memory (RAM) – reserviert. Dieser Speicherbereich wird vom Betriebssystem so verwaltet, dass dort ein Abbild der auf der Festplatte abgespeicherten Datei abgebildet wird. Wahlweise kann vom Entwickler definiert werden, ob eine Datei als virtueller Speicher festgelegt wird oder ob der Speicher für sich allein eine virtuelle Abbildung bildet, so dass weitere Applikationen auf diesen Speicher zugreifen können.

[12] Die Kommunikationsrichtung ist in eine Richtung begrenzt.

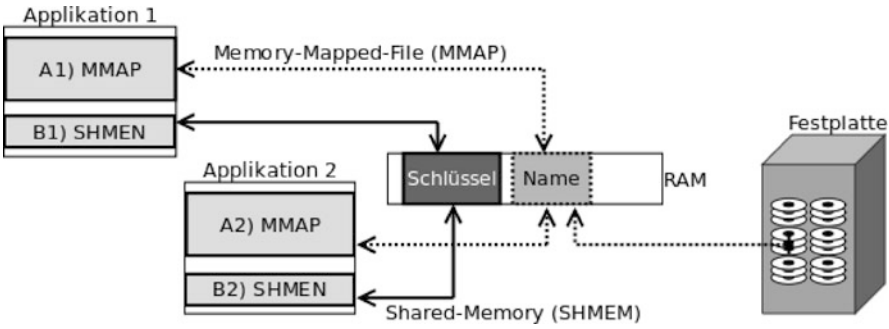


Abb. 4.9 Gegenüberstellung von Shared-Memory und Memory-Mapped-File. `Schlüssel` für Shared-Memory und `Name` für Memory-Mapped-File sind einfache Zeichenketten, wobei der Name den zu nutzenden Dateinamen auf der Festplatte beschreibt und der Schlüssel in der Regel einen systemweiten, eindeutig identifizierbaren Indikator enthält. Die *Pfeile* symbolisieren die Möglichkeit, dass alle teilnehmenden Prozessen lesenden und schreibenden Zugriff haben

4.3.3 BOINC-Manager

Jeder kann sich an einem BOINC-Projekt anmelden und seine Rechnerressourcen zur Verfügung stellen. Der jeweilige Anwender muss sich nur den BOINC-Manager herunterladen und installieren[13]. Der BOINC-Manager ist ein erweitertes Dashboard (vgl. Abb. 4.10), wobei der Anwender auch Schaltfläche u. a. auf der linken Seite zur Auswahl hat und durch diese die Ausführung einer wissenschaftlichen Applikation beeinflussen kann. In dieser Abbildung kann der Anwender ein oder mehrere selektierte Projekte pausieren, weiterarbeiten lassen oder die aktuelle Bearbeitung abbrechen. Weiterhin können Statusmeldungen und Statistiken über die Projekte eingesehen und für jedes Projekt individuelle Einstellungen vorgenommen werden, wie z. B. nur arbeiten, wenn der Rechner n-Minuten nicht an diesem gearbeitet hat.

Der Einstellungsdialog kann im BOINC-Manager durch die Klickreihenfolge: *Tools, Computing preferences* (Version 6.12.34, Englisch) oder *Assistenten, Einstellungen* (deutsche Version) aufgerufen werden. Die Abb. 4.11 bis 4.13 zeigen die drei Fenster für die Einstellungen folgenden Kategorien:

CPU- und GPU Viele der in Abb. 4.11 aufgeführten Einstellungsmöglichkeiten sind sicherlich selbsterklärend, z. B. die Ausführung erlauben, wenn der Computer im Batteriebetrieb ist oder die CPU/GPU gerade durch weitere Prozesse belegt wird. Es kann eine Wartezeit angegeben werden, wann die Ausführung gestartet wird, nachdem der Computeranwender anscheinend nicht mehr an diesem Computer arbeitet, oder ab wie viel freier Prozessorleistung der Computer genutzt werden kann. Weiterhin kann man nur zwischen bestimmten Zeitpunkten, an vordefinierten Tagen die BOINC-Projekte arbeiten lassen, z. B. nur am Wochenende. Im unteren Bereich kann man noch die Anzahl der zur Verfügung

[13] Webseite zum Herunterladen des BOINC-Managers: boinc.berkeley.edu/download.php.

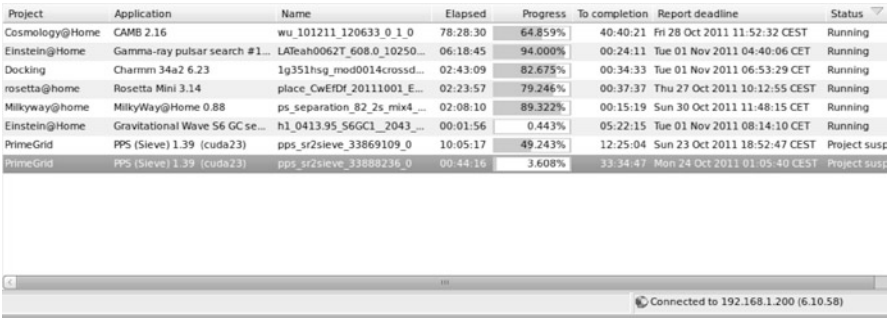


Abb. 4.10 Ansicht des BOINC-Managers, der zu einem externen BOINC-Client verbunden ist und sieben BOINC-Projekt-Registrierungen verwaltet. Sechs dieser sieben Projekte führen im Moment Berechnungen aus, das untere BOINC-Projekt *PrimeGrid* wurde durch den Anwender pausiert und kann zu jedem Zeitpunkt weitergeführt werden

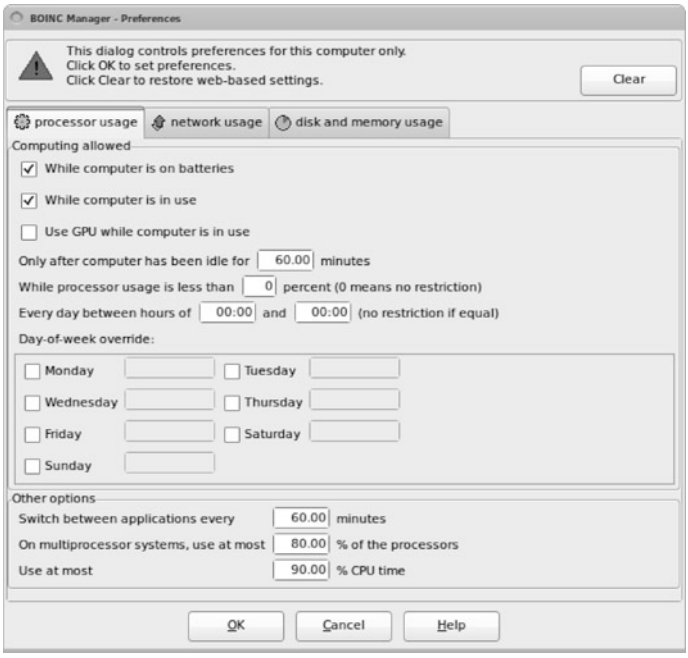


Abb. 4.11 BOINC-Manager-Einstellungsdialog für die CPU- und GPU-Auslastung

gestellten Prozessoren und die maximale CPU-Zeit für alle BOINC-Projekte einstellen.

Netzwerk Möglichkeiten für das Einstellen der maximalen Downloadgeschwindigkeit sowie der maximalen Dateigrößen, in welchen Zeiträumen eine Netzwerkverbindung erlaubt ist und ob eine Nachfrage vor einer neuen Internetver-

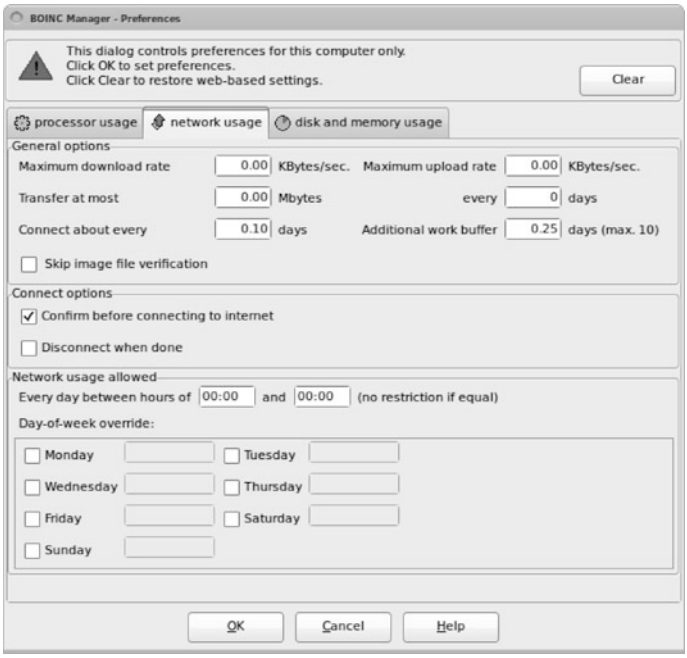


Abb. 4.12 Einstellungsdialog des BOINC-Managers für die Netzwerkkommunikation

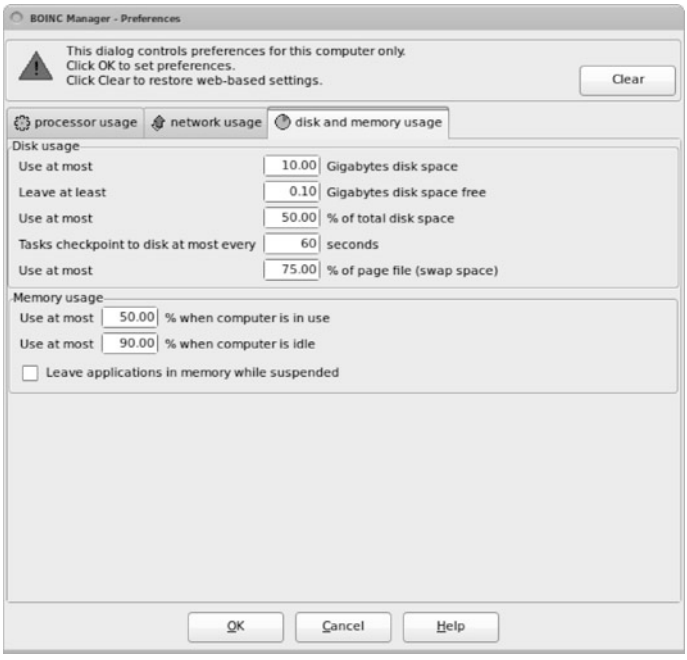


Abb. 4.13 Einstellungsdialog des BOINC-Managers für die Festplatten- und RAM-Nutzung

bindung gestellt werden soll, werden in dem Einstellungsdialog von Abb. 4.12 vorgenommen.

Ressourcen Einfache Einstellungsmöglichkeiten für das Zurverfügungstellen von Speicherressourcen auf einem Computer. Diese Informationen werden bei Anfragen zu BOINC-Projekten mitgesendet, so dass der BOINC-Scheduler entscheidet, ob neue Arbeitspakete an Ihren Computer gesendet werden oder nicht. Sie erhalten in dem Fall, dass nicht genügend Ressourcen freigegeben sind, eine Meldung im Protokollierungsfenster Ihres BOINC-Managers.

Kapitel 5
Serverinstallation

Eine lange Reise beginnt mit dem ersten Schritt.

Chinesisches Sprichwort

Eine der wichtigsten Aufgaben in der Softwareentwicklung ist die Installation von neuen oder abhängigen Softwarebibliotheken und Applikation. Entwickler können Profis in ihrem Gebiet sein, aber wenn eine Softwarekomponente nicht relativ einfach zu installieren ist, dann kann davon ausgegangen werden, dass sie nicht von vielen akzeptiert wird. Eine entsprechende Prüfung auf Useability ist daher Pflicht und sollte in einen Entwicklungsprozess einfließen. Dieses Kapitel beschreibt, wie Sie erfolgreich mit einem Linux-Ubuntu-System einen BOINC-Server installieren und ein erstes BOINC-Projekt erstellen.

5.1 BOINC-Projekte

Fast alle etablierten BOINC-Projekte beschäftigen sich mit der Lösung einer konkreten Problemstellung und wurden speziell für diese ins Leben gerufen. Es gibt zur Zeit[1] etwa 53 Projekte [92]. Die verschiedenen Projekte haben teilweise sehr unterschiedliche Zielrichtungen. Es existieren Projekte für die Medizinforschung (Biochemical Library [75], DrugDiscovery@Home [117], Erforschung von neuen Algorithmen (Enigma@Home [124]), für die Lösung von Problemen der Luft- und Raumfahrt (Constellation [109], MilkyWay@Home [150]), der theoretischen Physik (Spinhenge@home [171], LHC@home [141]) und für Multimedia-Anwendungen (BURP [100]). Jedes dieser Projekte bietet dabei wissenschaftliche Applikationen für die gängigsten Betriebssysteme an und teilweise auch für Berechnungen mit Hilfe von Grafikkartenprozessoren. Letzteres wird leider noch nicht von allen Projekten unterstützt, da es unter Umständen auch von der Problemstellung abhängt, ob eine solche Umsetzung möglich ist. Sie benötigen in Ihrer wissenschaftlichen Applikation Algorithmen, welche sich auch parallelisieren lassen.

[1] November 2011.

C. B. Ries, *BOINC*, Xpert.press,
DOI 10.1007/978-3-642-23382-8_5, © Springer-Verlag Berlin Heidelberg 2012

5.2 Ausgangspunkt für eine BOINC-Server-Installation

Als Ausgangspunkt installieren wir uns in einer virtuellen Maschine ein Ubuntu-Linuxsystem. VirtualBox [181] ermöglicht das Erstellen von virtuellen Maschinen mit einer breiten Supportpalette für die unterschiedlichsten Betriebssysteme, u. a. Windows, Macintosh, Unix oder Linux sowie für 32-Bit- und 64-Bit-Installationen. Als Betriebssystem wählen wir die Ubuntu-Server-Version 10.04 mit Langzeitsupport (engl. *long time support*, LTS) [178].

Sollten Sie in Erwägung ziehen, unter Ihrem aktuellen Benutzerkonto die Schritte zur Installation und Administration durchführen zu wollen, können Sie direkt zu Abschn. 5.3 übergehen. Im gesamten Buch wird allerdings davon ausgegangen, dass ein Benutzer `boincadm` existiert und dass dieser Benutzer sein Heimatverzeichnis in `/home/boincadm` hat.

5.2.1 Testumgebung mit VirtualBox

Für alle meine Testprojekte verwende ich VirtualBox, um einen virtuellen Computer überall zur Verfügung zu haben, da ich diesen ganz praktisch auf meinem Laptop installiert habe. Das bringt enorme Vorteile, u. a. die naheliegenden, dass man quasi überall an seinen BOINC-Projekten und wissenschaftlichen Applikationen weiterarbeiten kann, man kann aber auch eine vorherige Installation wiederherstellen. Für eben einen solchen Fall können mit VirtualBox sog. Snapshots erstellt werden, die den aktuellen Status abspeichern und zu jedem späteren Zeitpunkt wiederhergestellt werden können [181, 182].

Die zuvor erwähnte Ubuntu-Server-LTS-Version wird in der Liste der voll unterstützten Gastinstallationen gelistet; im Allgemeinen werden alle 2.4- und 2.6-Linux-Kernel unterstützt, was bei dieser Distribution kein Problem darstellt.

Da es sich um eine Serverinstallation handelt, können wir die Einstellungen wie folgt vornehmen:

- Keine Unterstützung eines Diskettenlaufwerks,
- CD-Rom und daraufhin die Festplatte bei der Boot-Reihenfolge,
- keine Input/Output APIC (I/O APIC)-Unterstützung, eine Aktivierung verlangsamt in der Regel die Ausführungszeit,
- das Bereitstellen eines Prozessors bzw. Prozessorkerns reicht völlig aus (mehr dürfen es natürlich sein),
- Deaktivierung der 2D/3D-Beschleunigung sowie nur 12 Megabyte für den freigegebene Grafikkartenspeicher,
- eine Festplatte von etwa acht Gigabyte (auch hier können es natürlich mehr sein),
- Deaktivierung der Soundkarte,
- mindestens eine Netzwerkkarte mit „Bridged Adapter“, so dass eine feste IP-Adresse innerhalb des installierten Betriebssystems möglich ist,
- serielle sowie USB-Anschlüsse können deaktiviert werden und
- keine miteinander geteilten Verzeichnisse (Shared-Folder), denn wir sehen die virtuelle Maschine als Netzwerkkomponente und behandeln sie auch so.

Wenn Sie alles soweit eingestellt haben, werden wir im nächsten Abschnitt eine eigene ISO-Datei für die Installation eines Ubuntu-Servers erstellen, die sich automatisch mit allen benötigten Abhängigkeiten aus Abschn. 5.3 installiert.

5.2.2 Kickstart-Installation eines Ubuntu-Servers

Es gibt mehrere Wege, sich einen neuen Rechner zu installieren, und auch die Wahl eines passenden Betriebssystems ist manchmal nicht ganz einfach. Wenn man allerdings Vorgaben bzgl. eines Betriebssystems hat, dann sollte man zumindest immer versuchen sich das Leben zu erleichtern. Wie schon erwähnt verwende ich hier exemplarisch die Ubuntu-Server-LTS 10.04-Version. Der große Vorteil dieser Version ist die Möglichkeit, die Installation durch Kickstart zum großen Teil zu automatisieren; CD/DVD ins Laufwerk, Rechner an, abwarten, fertig.

Die Basis einer Kickstart-Installation ist eine Kickstart-Konfigurationsdatei [161, 177], welche an einem einfachen Texteditor erstellt werden kann. In dieser Datei können Sie zahlreiche Standardeinstellungen vornehmen:

- die Installation von verschiedenen Sprachunterstützungen und die Definition einer Standardsprache für die Tastatureingaben und die Systemumgebung,
- Zeitzonen, Firewall-Regelsätze,
- ob es sich um eine Neuinstallation oder eine Aktualisierung handelt,
- wie die Festplattenpartitionen auszusehen haben,
- welche Netzwerkeinstellungen konfiguriert werden sollen,
- ob der Benutzer *root* deaktiviert ist oder aktiviert sein soll, wofür allerdings eine Passworteingabe nötig ist und
- vieles mehr.

Ein an dieser Stelle sehr wichtiger Punkt ist, dass die für BOINC benötigten Softwarepakete aus Abschn. 5.3 direkt mit installiert werden können und keine späteren Installationen mehr nötig sind,

Das beste an einer Kickstart-Installation ist, dass Sie die Konfiguration aus Listing 5.1 beim Starten der Installation des Ubuntu-Servers nur passend angeben müssen, und Sie erhalten in Kürze eine neue funktionsfähige und für BOINC nutzbare Systemumgebung.

Listing 5.1 Kickstart-Konfiguration für die Installation eines neuen Ubuntu-Servers, der nach der Durchführung sofort nutzbar ist

```
lang de_DE
langsupport en_GB en_US --default=de_DE
keyboard de_nodeadkeys
timezone Europe/Berlin
network --device=eth1 --bootproto=dhcp --hostname=boinc-testserver
rootpw --disabled
user boincadm --password boincadm --fullname "Standard BOINC User"
reboot
text
install
cdrom
bootloader --location=mbr
zerombr yes
```

```
clearpart --all --initlabel

# Disk partitioning information
part /boot --fstype ext2 --size 200
part / --fstype ext4 --size 30000
part swap --size 1024
part /local/data --fstype=ext4 --size=1 --grow

auth --useshadow --enablemd5
firewall --disabled
skipx
firstboot --enable

# Package install information
%packages
gcc
g++
m4
m4-doc
libtool
autoconf
openssl
openssh-server
nfs-common
nfs-kernel-server

%post --interpreter=/bin/bash
/usr/sbin/ntpdate 129.217.160.1
/usr/sbin/hwclock --systohc

## Basic packages (for all hosts) to install:
apt-get install pkg-config --assume-yes
apt-get install automake --assume-yes
apt-get install make --assume-yes
apt-get install subversion --assume-yes
apt-get install dialog --assume-yes
apt-get install bogl-bterm --assume-yes
apt-get install tree --assume-yes
apt-get install python-setuptools --assume-yes
apt-get install python-mysqldb --assume-yes
apt-get install sqlite --assume-yes
apt-get install libsqlite0 --assume-yes
apt-get install libsqlite0-dev --assume-yes
apt-get install nfs-common --assume-yes
apt-get install nfs-kernel-server --assume-yes
apt-get install mysql-client --assume-yes
apt-get install libfcgi-dev --assume-yes
apt-get install libcurl4-openssl-dev --assume-yes
apt-get install libmysqlclient-dev --assume-yes
apt-get install libglu-dev --assume-yes
apt-get install libglut3-dev --assume-yes
apt-get install libxmu-dev --assume-yes
apt-get install libxi-dev --assume-yes
apt-get install libjpeg-dev --assume-yes
apt-get install apache2 --assume-yes
apt-get install apache2-utils --assume-yes
apt-get install libapache2-mod-php5 --assume-yes
apt-get install libapache2-mod-fastcgi --assume-yes
apt-get install libapache2-mod-python --assume-yes
apt-get install php5-cli --assume-yes
apt-get install php5-gd --assume-yes
apt-get install php5-curl --assume-yes
apt-get install php5-mysql --assume-yes
apt-get install php5-mcrypt --assume-yes
apt-get install mysql-server-core --assume-yes
apt-get install mysql-server --assume-yes
```

Eine solche Kickstart-Konfiguration kann an einem beliebigen Ort zur Verfügung gestellt werden: Web-Server, FTP-Server oder auf einem USB-Stick, um nur einige Beispiele zu nennen. Sie müssen VirtualBox nun nur mit der Ubuntu-Server-ISO-Datei starten und die Escape-Taste drücken, wenn das Auswahlmenü für die Installation erscheint. Daraufhin gelangen Sie in eine Prompt-Eingabe, dort geben Sie die nachfolgende Zeile ein und drücken die Enter-Taste.

```
boot: install ks=http://www.visualgrid.de/dl/ks-boinc.cfg
```

Eventuell müssen Sie nach diesem Schritt die MySQL-Datenbank neu konfigurieren. Bei Ubuntu übernimmt das ein einfaches Skript für Sie, und ein Aufruf von

```
$ sudo dpkg-reconfigure mysql-server-5.1
```

reicht aus und Sie können ein Passwort für den Hauptbenutzer konfigurieren. Nach der erfolgreichen Installation können Sie mit dem Abschn. 5.4 weitermachen, die BOINC-Quellen herunterladen und ein erstes BOINC-Projekt erstellen.

Erwähnen möchte ich an dieser Stelle noch zwei weitere Möglichkeiten, eine komplette Installationsroutine zu automatisieren:

M23 Mit diesem Werkzeug ist eine Softwareverteilung und die Softwareverwaltung von mehreren Rechnern möglich, ganz komfortabel über eine Webseite. Es werden zahlreiche Distributionen unterstützt, u. a. Debian, Ubuntu, OpenSUSE, Fedora und Linux Mint. M23 unterstützt das automatische Erstellen von Konfigurationen, um Backups zu erstellen, das Erstellen von Skripts für eine automatische Installation von neuen Rechnern, das Konfigurieren von Firewall-Regeln und vieles mehr [144, 187].

Fully Automatic Installation (FAI) Dieses Softwarepaket besteht aus mehreren Perl- und Shell-Skripts für die Installation von neuen Rechnern.

5.3 Software- und Bibliotheksabhängigkeiten

Es werden einige Softwarekomponenten benötigt, um eine Installation eines BOINC-Projektes durchzuführen und um eigene wissenschaftliche Applikationen entwickeln zu können. Im vorherigen Abschn. 5.2 haben wir gezeigt, wie eine Ubuntu-Server-Installation durchgeführt wird. Diese Schritte zeigen eine Minimalinstallation ohne weitere Softwarekomponenten. Im aktuellen Abschnitt zeigen wir, welche Softwarekomponenten zusätzlich zum installierten System hinzugefügt werden müssen. Diese Beschreibungen basieren auf Empfehlungen aus dem BOINC-Wiki [88] und Erfahrungen während der Entwicklung einiger BOINC-Projekte [16, 19, 20, 171].

Grundsätzlich können die zu installierenden Softwarekomponenten in fünf Schritte unterteilt werden. Da ein BOINC-Projekt im Zusammenschluss von mehreren Rechnern arbeiten kann (vgl. Abschn. 5.8), ist es nicht zwingend erforderlich, auf allen Rechnern alle Softwarekomponenten zu installieren. Generell wird empfohlen, mindestens die Softwarekomponenten aus Listing 5.2 auf allen zu verwendenden Rechnern zu installieren. Bei den nachfolgenden Installationserläuterungen

beschränken wir uns ausschließlich auf die Serverkomponenten. Ein Server benötigt in den meisten Fällen keine Laufzeitbibliotheken für die Erstellung von Desktop-Elementen.

Listing 5.2 enthält die Softwarekomponenten, die minimal installiert werden müssen. Durch diese Komponenten wird es erst ermöglicht, dass die BOINC-Quellen heruntergeladen und kompiliert werden (vgl. Abschn. 5.4). Weiterhin wird ein OpenSSH-Server installiert, so dass eine Remote-Administration der Rechner möglich ist; ebenso werden Compiler (gcc, g++), Maketools (autoconf, automake), Makro-Prozessoren (m4) und Werkzeuge zur Ermittlung von Informationen über Bibliotheksinstallationen (libtool, pkg-config) installiert. Nach diesem Installationsschritt werden alle weiteren Befehle über eine SSH-Verbindung (Secure Shell) ausgeführt.

Listing 5.2 Softwarekomponenten ($\approx 30MB$), die mindestens auf allen Serverrechnern eines BOINC-Projektes vorhanden sein sollten

```
# Minimale Ausstattung:
sudo apt-get install openssl openssh-server \
 m4 m4-doc libtool autoconf gcc g++ \
 pkg-config automake subversion
```

Listing 5.3 installiert benötigte Datenbankapplikationen und Datenbank-Softwarebibliotheken. Die Datenbank ist das Gehirn eines BOINC-Projektes. Alle Daten und Informationen über ein Projekt oder eine wissenschaftliche Applikation werden in dieser Datenbank abgespeichert. Das BOINC-System besitzt Skripts in der Skriptsprache *Python*, und diese Skripts benötigen Zugriff auf die Datenbank. Die zwei aufgelisteten *Python*-Pakete liefern die benötigten Funktionalitäten. Die Client-Bibliothek *mysql-client* liefert Bibliotheken für den Zugriff auf die Datenbank, wenn Software von Drittanbietern diese benötigen und nicht selber in ihrer Installation bereitstellen. Die Installation des MySQL-Datenbankservers erwartet die Eingabe (vgl. Abb. 5.1) und Bestätigung (vgl. Abb. 5.2) eines Passworts für den Datenbankbenutzer `root`. In beiden Fällen geben wir – wie beim Erstellen des Standardbenutzers vom Linux-System – `boincadm` ein.

Listing 5.3 Softwarekomponenten ($\approx 24MB$) für das Verwenden von einer MySQL-Datenbank

```
# Datenbank:
sudo apt-get install \
 mysql-server-core mysql-server \
 python-setuptools python-mysqldb mysql-client
```

Das BOINC-System verwendet für das Verteilen und Einkassieren seiner Arbeitspakete standardmäßig den Webserver Apache. Listing 5.4 enthält die Befehle zum Installieren der benötigten Komponenten für das Starten eines Apache mit der Möglichkeit, dynamische Inhalte mit Hilfe von PHP (PHP: Hypertext Preprocessor) zu erstellen. Weiterhin werden Module für das Verwenden von FastCGI und Python installiert und automatisch durch die Installationsroutinen von `apt-get` aktiviert. Eventuell sollte nach diesem Schritt ein Neustart vom Webserver erfolgen. Der Befehl

```
sudo /etc/init.d/apache restart
```

erledigt das.

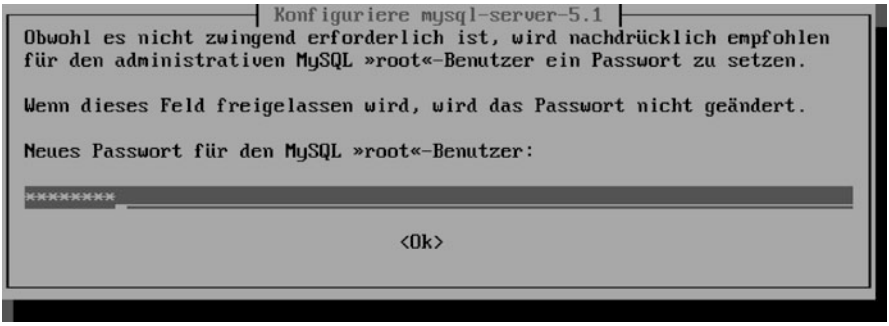


Abb. 5.1 Eingabe eines Passworts für den Datenbankbenutzer `root`

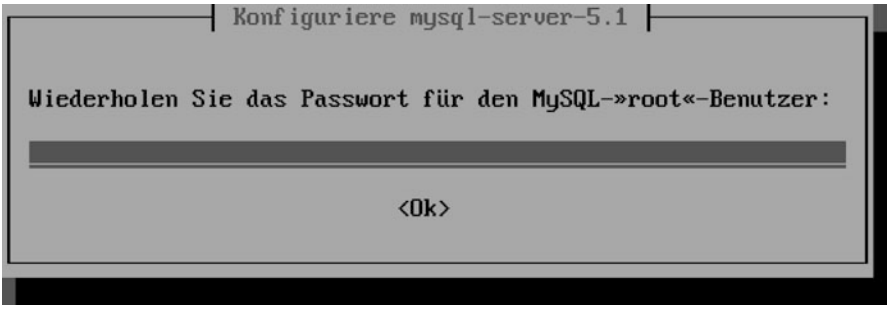


Abb. 5.2 Bestätigung des Passworts für den Datenbankbenutzer `root`

Listing 5.4 Softwarekomponenten ($\approx 16MB$) für das Verwenden des Apache-HTTP-Servers mit PHP5-Unterstützung

```
# Webserver:
sudo apt-get install apache2 \
 apache2-utils libapache2-mod-php5 \
 libapache2-mod-fastcgi libapache2-mod-python

# Dynamische Inhalte erzeugen:
sudo apt-get install php5-cli \
 php5-gd php5-curl php5-mysql \
 php5-mcrypt
```

Für das Kompilieren der BOINC-Quellen werden die Softwarebibliotheken aus Listing 5.5 benötigt. Der Befehl installiert folgende Bibliotheken:

fcgi-dev *Fast Common Gateway Interface* (FastCGI, fcgi) ist eine freie und offene Erweiterung zum Common Gateway Interface (CGI) mit einer hohen Performance (engl. *performance*) in allen Internetanwendungen, ohne die Latenzzeit einer evtl. vorhandenen Schnittstelle eines Webservers. Die Implementierung soll eine schnelle Verarbeitung von Webanfragen ermöglichen. Applikationen mit FastCGI-Nutzung können im Kontext eines Webservers ihren Dienst verrichten oder als eigenständige (engl. *stand-alone*) Applikation gestartet werden.

curl4-openssl-dev Ermöglicht das Entwickeln von Anwendungen mit CURL-Unterstützung. Dabei werden Standardprotokolle wie TCP/IP und UDP/IP unter-

stützt. Weiterhin werden Transportprotokolle wie FTP und HTTP direkt durch die Bibliothek bereitgestellt. Dies ermöglicht die einfache Implementierung in eigene Applikation, ohne sich tiefer mit den entsprechenden Protokollen auseinander setzen zu müssen. Durch OpenSSL wird eine verschlüsselte Kommunikation ermöglicht.

mysqlclient-dev Diese Bibliothek dient der Entwicklung von Anwendungen mit Zugriff auf eine MySQL-Datenbank.

glu-dev, glut3-dev Zwei OpenGL-Bibliotheken zur Erstellung von echtzeitfähigen 3-dimensionalen Anwendungen.

xmu-dev *X11 miscellaneous utility library* wird für das Kompilieren der glu-dev- und glut3-dev-Bibliotheken benötigt.

xi-dev Eine X-Window-System Client-Schnittstelle, die eine Erweiterung für das Eingabesystem vom X-Protokoll zur Verfügung stellt.

jpeg-dev Programmierbibliothek zum Erstellen und Verarbeiten von Bildern im JPEG-Format.

Listing 5.5 Softwarebibliotheken ($\approx 22 MB$), die für das Kompilieren der BOINC-Quellen benötigt werden

```
# Softwarebibliotheken zur Erstellung
# der BOINC-Komponenten:
sudo apt-get install libfcgi-dev \
 libcurl4-openssl-dev libmysqlclient-dev \
 libglu-dev libglut3-dev libxmu-dev \
 libxi-dev libjpeg-dev
```

Listing 5.6 installiert das Network File System (NFS). Dieses wird benötigt, wenn das BOINC-Projekt auf mehreren Rechnern installiert wird. Dadurch können die BOINC-Quellen und BOINC-Anwendungen auf allen Rechnern importiert werden. Dieses Verfahren wird empfohlen und minimiert den Aufwand zur Pflege eines BOINC-Projektes. Abschnitt 5.8 beschreibt die Verwendung des NFS und beschreibt, wie eine Installation und Administration u. a. innerhalb eines Clusters möglich ist.

Listing 5.6 Optionale Softwarekomponenten ($\approx 0{,}6\,MB$) für das Verwenden des Network File System (NFS)

```
# Optional:
sudo apt-get install nfs-common nfs-kernel-server
```

5.4 BOINC-Quellen herunterladen und kompilieren

Die BOINC-Serverquellen werden mit dem Revisionssystem Subversion (SVN) gepflegt. Dieses System ermöglicht das einfache Herunterladen der Serverquellen. Sollten Sie sich dafür entscheiden, ein BOINC zu erstellen, um Forschung zu betreiben, können Sie sich die Quellen direkt mit

Listing 5.7 Subversion-Befehl zum Herunterladen der BOINC-Quellen

```
svn co http://boinc.berkeley.edu/svn/branches/server_stable
```

herunterladen. Es wird ein neuer Unterordner `server_stable`, erstellt in dem sich die BOINC-Quellen befinden. Die Version in diesem Buch basiert auf der Revision 22328 mit der letzten Änderung am 09.09.2010 (vgl. Listing 5.8).

Listing 5.8 Revision und Tag der letzten Änderung der in diesem Buch verwendeten BOINC-Quellen

```
boincadm@boinc-testserver:~/server_stable$ svn info
Pfad: .
URL: http://boinc.berkeley.edu/svn/branches/server_stable
Basis des Projektarchivs: http://boinc.berkeley.edu/svn
UUID des Projektarchivs: 677681c0-c3c0-48b4-bd46-92119ef6d97a
Revision: 22840
Knotentyp: Verzeichnis
Plan: normal
Letzter Autor: davea
Letzte geaenderte Rev: 22328
Letztes Aenderungsdatum: 2010-09-09 01:16:13 +0200 (Do, 09. Sep 2010)
```

Nach einem Wechsel in den neuen Unterordner `server_stable` kann folgende Befehlskette zum Konfigurieren und Kompilieren der BOINC-Quellen eingegeben werden

Listing 5.9 Befehlskette zum Übersetzen der BOINC-Quellen

```
cd server_stable
./_autosetup
./configure --disable-client --disable-manager
make
```

Der `./configure`-Befehl kann dazu verwendet werden zu definieren, welche Softwarekomponenten und Bibliotheken kompiliert werden sollen. In diesem Listing wurde beispielhaft `-disable-manager` mitgegeben, das bestimmt, dass der BOINC-Manager nicht kompiliert werden soll. BOINC besteht aus vier Hauptkomponenten, die als Parameter beim Aufruf von `./configure` einzeln oder gemeinsam deaktiviert werden können:

- Server (–disable-server)
- Bibliotheken (–disable-library)
- Client (–disable-client)
- Manager (–disable-manager)

Die Standardeinstellung erstellt alle Komponenten. Um ein BOINC-Projekt zu entwickeln, werden zunächst nur die ersten zwei Komponenten benötigt, wir deaktivieren deshalb die Erstellung der beiden anderen Komponenten. Diese werden später direkt als binäre Pakete installiert.

5.5 Struktur der BOINC-Quellen

Nach dem Herunterladen und Kompilieren der BOINC-Quellen in Abschn. 5.4 findet sich folgende Struktur im Unterordner `server_stable/` vor (vgl. Abb. 5.3):

api/ Dieser Ordner enthält die allgemein von Ihnen zu verwendende BOINC-API (Application Programming Interface). In der Datei `api/boinc_api.h` sind

die Standardfunktionen der BOINC-API definiert und in dem Archiv `api/libboinc_api.a` als vorkompilierte Objektdateien zusammengefasst, so dass Sie dieses Archiv nur zur Kompilierung eigener Anwendungen mit einbeziehen müssen (s. Kap. 7).

apps/ Enthält Beispiele von wissenschaftlichen Applikationen, die auch als Basis für die eigenen Entwicklungsschritte zur Verfügung stehen. Abschnitt 5.6 beschreibt die Verwendung der `apps/upper_case`-Applikation und wie Sie diese zu einem ersten BOINC-Projekt hinzufügen.

client/ Enthält die Quellen für den BOINC-Client. Es bestehen Abhängigkeiten zu Dateien im `lib/` und `api/`-Ordner.

clientctrl/ In diesem Ordner befindet sich eine Anwendung zum Starten und Stoppen eines BOINC-Clients unter Windows. Diese Anwendung steuert den BOINC-Client, wenn dieser als Windows Service installiert und gestartet ist.

clientgui/ Die Quellen vom BOINC-Manager.

clientscr/ Ein mit Windows und Macintosh OS X kompatibler Bildschirmschoner.

clienttray/ Die Quellen eines Steuerungswerkzeugs, welches zur Systray hinzugefügt werden kann, um Informationen vom BOINC-Client abzufragen und daraufhin kompakt anzuzeigen.

coprocs/ CUDA-Bibliothek und Funktionsprototypen für die CUDA-Nutzung unter Windows.

curl/ Nicht mehr notwendig.

db/ Klassendefinitionen und Strukturen für den Zugriff auf die MySQL-Datenbank. Jede Datenbanktabelle wird durch eine Klasse beschrieben und kann über eine einheitliche Schnittstelle genutzt werden.

doc/ Die Dokumentationen über die BOINC-Architektur, einige Grafiken, die für ein eigenes Projekt verwendet werden dürfen, Seiten durch PHP umgesetzt, die als Vorlage dienen und Manpages.

html/ Die Elemente (HTML-Seiten, PHP-Dateien, Bilder, etc.) für die Webseiten von BOINC-Projekten.

languages/, locale/ Übersetzungen aller visuell dargestellten Elemente, also durch den BOINC-Manager oder auf den BOINC-Webseiten.

lib/ Implementierungen von Hilfsfunktionen und API-Schnittstellen. Es ist unter anderem eine FCGI-API und Kryptografie-API vorhanden.

mac_build/ Skripts zum Kompilieren der BOINC-Quellen unter Macintosh OS X.

mac_installer/ Ein Installationswerkzeug für Macintosh OS X.

packages/ Unbekannt.

py/ Python-Skripts zur Verwaltung eines BOINC-Projektes.

rboinc/ Eine Erweiterung für BOINC, mit der die Erstellung von Arbeitspaketen über eine Netzwerkverbindung möglich ist [9].

samples/ Enthält Beispiele für die Entwicklung von wissenschaftlichen Applikationen. Die Beispiele decken eine breite Palette an Anwendungsfällen ab: Single- und Mehrkernprozessoren, Grafikprozessoren, Wrappernutzung und Bildschirmschoner.

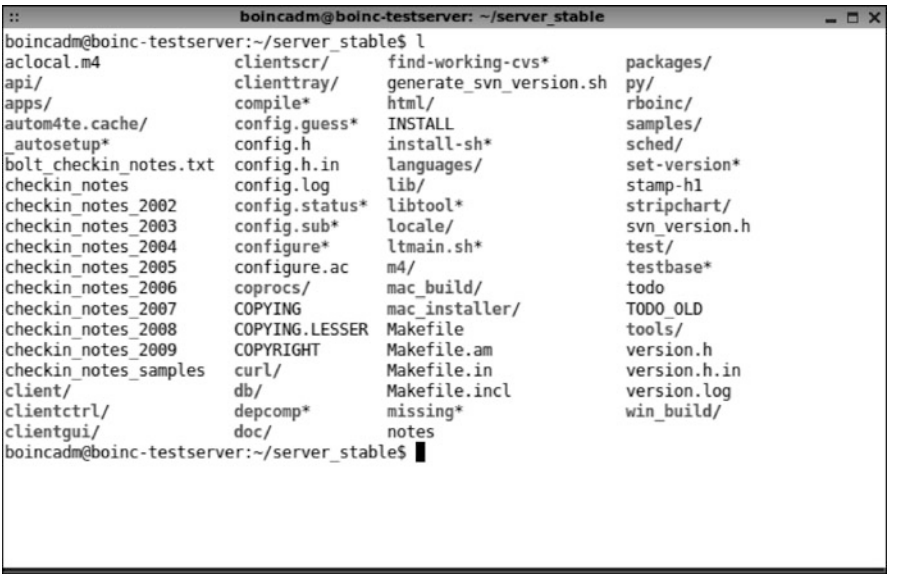


Abb. 5.3 Oberste Ordnerstruktur – BOINCs *root* – innerhalb der BOINC-Quellen

sched/ Die Quellen für den Scheduler Server. Jeder BOINC-Teilnehmer verbindet sich zu diesem Server um Informationen über ein BOINC-Projekt oder Arbeitspakete zu erhalten.

stripchart/ Ein einfaches Skript zur Erstellung eines Statistikgraphen. Was dargestellt werden soll, kann eingestellt werden, und durch eigene Skripts können die zu verwendenden Datensätze frei gewählt werden.

test/ Testskripts und Testanwendungen, um eine Qualitätssicherung zu ermöglichen, allerdings nur für einen kleinen Teil der BOINC-Quellen.

tools/ Werkzeuge für das Abfragen von BOINC-Projektinformationen und für die Verwaltung eines BOINC-Projektes. Das Werkzeug `make_project` dient zum Beispiel dem Erstellen eines neues BOINC-Projektes und wird in Abschn. 5.6 eingeführt.

win_build/ Projektdateien, die mit den gängigsten Entwicklungsumgebungen für Windows verwendet werden können und helfen, mit den BOINC-Quellen zu arbeiten.

5.6 Ein erstes Projekt erstellen

Das BOINC-Projekt liefert mit dem Skript `tools/make_project` ein Werkzeug zur Erstellung eines BOINC-Projektes mit. Es sei angemerkt, dass es nicht ausreicht, dieses Werkzeug zu benutzen, um wirklich ein lauffähiges BOINC-Projekt zu erhalten. Zusätzlich müssen zahlreiche Modifikationen in Konfigurationsdateien vorgenommen werden, eine wissenschaftliche Applikation entwickelt, eventu-

elle weitere Dämonen und Handler implementiert und konfiguriert werden. Dieses Skript ist nur der Beginn, um die ersten Schritte zu einem eigenen BOINC-Projekt zu meistern.

5.6.1 Installationsskript ausführen

Listing 5.10 zeigt die Parameter des Werkzeugs `tools/make_project`, wie diese anzugeben sind und welche Bedeutung sie haben. Dieses Werkzeug bietet offiziell eine volle Unterstützung für Linux-Systemumgebungen.

Listing 5.10 Ausgabe der Hilfestellung von `tools/make_project`

```
boincadm@boinc-testserver:~/server_stable$ ./tools/make_project --help

syntax: ./tools/make_project [options] project ['Project Long Name']

Creates a new project with given name with everything running on a single
server.

Misc options:
  --no_query            accept all directories without querying
  --user_name           default: $USER (boincadm)
  --delete_prev_inst    delete project-root first (from prev installation)
  --drop_db_first       drop database first (from prev installation)
  --test_app            install test application
  --web_only            install web files, no executables (for Bossa, Bolt)
  --srcdir              where to find the source files (default: current
      directory)

Dir-options:
  --project_root        default: HOME/projects/PROJECT
  --key_dir             default: PROJECT_ROOT/keys
  --url_base            default: http://$NODENAME/ (http://boinc-testserver/)

  --html_user_url       default: URL_BASE/PROJECT/
  --html_ops_url        default: URL_BASE/PROJECT_ops/
  --cgi_url             default: URL_BASE/PROJECT_cgi/

Other:
  --db_name             default: PROJECT
  --db_user             default: USER_NAME
  --db_passwd           default: None
  --db_host             default: None

Example command line:
  ./make_project --project_root $HOME/boinc/projects --url_base http://tah.org/
      tah 'Test@home'

Then upload_dir = $HOME/boinc/projects/tah/upload
and  cgi_url    = http://tah.org/tah_cgi/

By default, directory options will be queried if they do not exist yet.
```

Listing 5.11 erstellt ein erstes BOINC-Projekt. Dieses wird standardmäßig im Heimatverzeichnis des Benutzers erstellt, der das Skript aufgerufen hat. Somit befindet sich das neue Projekt unter `/home/boincadm/projects/tah`. Im nächsten Schritt muss der Apache-HTTP-Server für das neue Projekt konfiguriert werden, so dass dieses Projekt mit Hilfe eines Webbrowsers und des BOINC-Managers erreichbar ist. Die Zeilen 5.11ff fordern Sie freundlich auf, die entsprechende README-

Datei – hier `tah.readme` – zu lesen, in der weitere Instruktionen enthalten sind, so dass Sie die erste Installation erfolgreich absolvieren sollten. Nachfolgend eine Zusammenfassung der in der README aufgeführten Hilfestellungen:

- Informationen über das Konfigurieren des Apache-HTTP-Servers (s. Abschn. 5.6.1.1),
- Welche Crontab-Einträge Sie hinzufügen sollen, damit kontinuierlich projektspezifische Aufgaben ausgeführt werden (s. Abschn. 5.6.1.2),
- Das Hinzufügen von Projektinformationen, wie zum Beispiel welche Plattformen unterstützt werden (s. Abschn. 5.6.1.3) und
- das Erstellen einer Zugriffsdatei `.htaccess` um zu steuern, welche Benutzer administrativen Zugriff auf ein BOINC-Projekt erhalten (s. Abschn. 5.6.1.4).

Weiterhin werden die drei Kommandos

1. `bin/start` zum Starten eines BOINC-Projektes,
2. `bin/stop` zum Stoppen eines BOINC-Projektes und
3. `bin/status` zum Ermitteln des Status eines BOINC-Projektes eingeführt.

Listing 5.11 Erstellung eines ersten BOINC-Projektes

```
boincadm@boinc-testserver:~/server_stable$ ./tools/make_project \
--user_name boincadm --delete_prev_inst --drop_db_first \
--db_user root --db_passwd boincadm \
--url_base http://192.168.1.100/ tah 'Test@home'

Creating project 'Test@home' (short name 'tah'):
   PROJECT_ROOT = /home/boincadm/projects/tah/
       URL_BASE = http://192.168.1.100/
  HTML_USER_URL = http://192.168.1.100/tah/
   HTML_OPS_URL = http://192.168.1.100/tah_ops/
        CGI_URL = http://192.168.1.100/tah_cgi/
        KEY_DIR = /home/boincadm/projects/tah/keys/
        DB_NAME = tah
        DB_HOST =

Continue? [Y/n] Y
Keys don't exist in /home/boincadm/projects/tah/keys/; generate them? [Y/n] Y
Setting up server files: generating keys
Setting up server files: copying files
Setting up database: generate database
Setting up server files: writing config files
Setting up server files: linking cgi programs
update_translations finished
Done installing default daemons.

Done creating project.  Please view
/home/boincadm/projects/tah/tah.readme
for important additional instructions.
```

5.6.1.1 Konfiguration des Apache-HTTP-Servers

Bei dem im vorherigen Abschnitt erstellten Projekt wurde eine Konfigurationsdatei für den Apache-HTTP-Server generiert, `tah.httpd.conf`. Diese Datei (vgl. Listing 5.12) enthält Informationen über Aliase zur Webadresse für den Zugriff

auf die Projektwebseite und Administrationswebseite. Die generierte Konfiguration fügen wir unserer Server-Konfiguration `/etc/apache2/httpd.conf` zu

```
sudo bash -c "cat tah,httpd.conf >> /etc/apache2/httpd.conf"
```

und starten den Webserver neu,

```
sudo /etc/init.d/apache2 restart
```

damit die Konfigurationen übernommen werden.

Listing 5.12 Konfiguration für den Apache-HTTP-Server für das Testprojekt in diesem Buch

```
Alias /tah /home/boincadm/projects/tah/html/user
Alias /tah_ops /home/boincadm/projects/tah/html/ops
ScriptAlias /tah_cgi /home/boincadm/projects/tah/cgi-bin

# Note: projects/*/keys/ should NOT be readable!

<Directory "/home/boincadm/projects/tah/html">
  Options Indexes FollowSymlinks MultiViews
  AllowOverride AuthConfig
  Order allow,deny
  Allow from all
</Directory>

<Directory "/home/boincadm/projects/tah/cgi-bin">
  Options ExecCGI
  AllowOverride AuthConfig
  Order allow,deny
  Allow from all
</Directory>
```

Der Apache-HTTP-Server ist in unserem Fall über die Adresse der virtuellen Maschine erreichbar. Diese Adresse ist von Ihren persönlichen Netzwerkeinstellungen abhängig. Hier ist es die IP-Adresse `192.168.1.100`, die automatisch über einen DHCP-Server gesetzt wurde. Zeile 1 in Listing 5.12 setzt einen Alias `/tah`, der auf die Dateien zur Darstellung der Projektwebseite `html/user` zeigt. Zeile 2 beschreibt einen Alias von `/tah_ops` nach `html/ops` und zeigt somit auf die Administrationsseite vom neuen BOINC-Projekt. Die jeweiligen Zielordner `html/user` und `html/user` werden durch die Zeilen 7 bis 12 weiter beschrieben und die Konfigurationsdirektiven werden auf den Webseiten des Apache-Projektes sehr gut beschrieben [69]. Mit `ScriptAlias` wird zuvor ein Alias erstellt, der allerdings automatisch als CGI-Skript ausgeführt wird.

Zu diesem Zeitpunkt ist der Apache-HTTP-Server installiert und ein BOINC-Projekt erstellt. Der HTTP-Server läuft standardmäßig als Benutzer und Gruppe `www-data`. Das BOINC-Projekt wird unter dem Benutzer und der Gruppe `boincadm` erstellt. Dies führt zu Problemen beim Arbeiten mit einigen BOINC-Applikationen und dem Aufrufen von Webseiten. Damit der Benutzer `www-data` auf die Projektwebseiten zugreifen kann, muss `www-data` die Gruppenzugehörigkeit der Gruppe `boincadm` erhalten:

```
boincadm@boinc-testserver:~/projects/tah$ sudo adduser www-data boincadm
Fuege Benutzer >>www-data<< der Gruppe >>boincadm<< hinzu ...
Adding user www-data to group boincadm
Fertig.

boincadm@boinc-testserver:~/projects/tah$ sudo /etc/init.d/apache2 restart
 * Restarting web server apache2                              [ OK ]
```

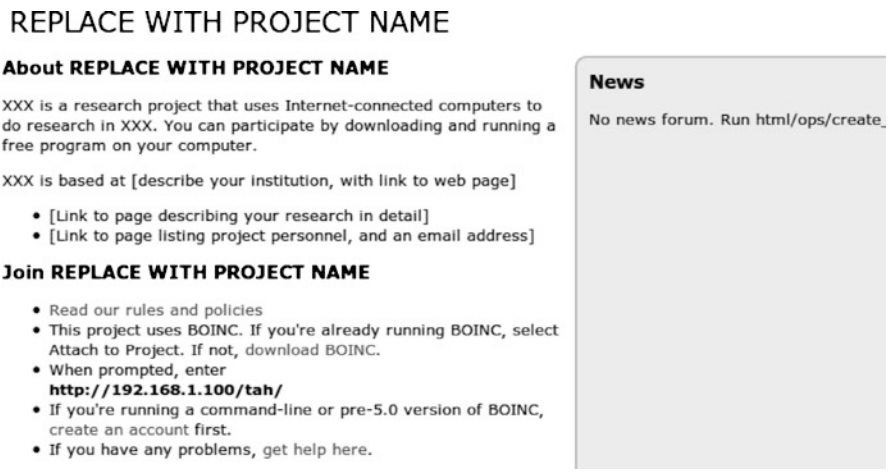


Abb. 5.4 Erster Aufruf der Webseite eines neuen BOINC-Projektes. Man erkennt eindeutig, dass noch einige Einstellungen gemacht werden müssen, viele davon können in der Datei `html/project/project.inc` vorgenommen werden

Das BOINC-Wiki [94] enthält zudem einen Hinweis auf nachzubessernde Zugriffsrechte, die mit den nachfolgenden Kommandos gesetzt werden können:

```
boincadm@boinc-testserver:~/projects/tah$ chmod 02770 upload
boincadm@boinc-testserver:~/projects/tah$ chmod 02770 html/cache
boincadm@boinc-testserver:~/projects/tah$ chmod 02770 html/inc
boincadm@boinc-testserver:~/projects/tah$ chmod 02770 html/languages
boincadm@boinc-testserver:~/projects/tah$ chmod 02770 html/languages/compiled
boincadm@boinc-testserver:~/projects/tah$ chmod 02770 html/user_profile
```

Daraufhin sollte ein erster Zugriff auf die Projektwebseite des neuen BOINC-Projekts möglich sein. Starten Sie einen Webbrowser ihrer Wahl und geben Sie

```
http://192.168.1.100/tah/
```

ein. Abbildung 5.4 zeigt die Webseite, die erscheinen muss. Womöglich wird der Zugriff auf die Projektwebseite mit einem Fehler quittiert. Dies kann daran liegen, dass der Benutzer `www-data` keinen Zugriff auf das Heimatverzeichnis von `boincadm` hat. Je nach Ubuntu-Server-Version ist der Zugriff von vornherein deaktiviert, wie dies nachfolgend gezeigt wird. Das Zugriffsrecht für Gruppenangehörige sowie Fremde ist untersagt [33].

```
root@boinc-testserver:/home# ll
insgesamt 16
drwxr-xr-x  4 root      root      4096 2010-12-13 13:15 ./
drwxr-xr-x 22 root      root      4096 2010-12-14 09:31 ../
drwx------  6 boincadm boincadm 4096 2010-12-14 12:24 boincadm/
drwxr-xr-x  3 root      root      4096 2010-12-13 13:15 .ecryptfs/
```

Dies kann mit dem Befehl

```
boincadm@boinc-testserver:/home$ chmod 755 boincadm/
```

behoben werden.

5.6.1.2 Konfiguration des Cron-Dämon

Das neue BOINC-Projekt besitzt eine Konfigurationsdatei `config.xml`, in der Aufgaben definiert werden können, die nach einem bestimmten Intervall ausgeführt werden sollen. Diese Aufgaben (engl. tasks) werden zwischen den XML-Tags `<task>` und `</task>` und in Abschn. 15.3.2 beschrieben. Der Cron-Dämon wird mit folgender Zeile bereichert:

```
0,5,10,15,20,25,30,35,40,45,50,55 * * * * /home/boincadm/projects/tah/bin/start
    --cron
```

Diese Zeile beschreibt, dass alle fünf Minuten ein Aufruf zum Starten eines BOINC-Projektes erfolgen soll. Der Parameter `--cron` gibt an, dass nur Tasks gestartet werden sollen. Sollte ein Task länger als fünf Minuten benötigen, wird dieser automatisch bei einem weiteren Aufruf übersprungen und es wird mit dem nächsten Task weitergemacht, so lange ein Task noch nicht ausgeführt ist. Das Hinzufügen der oberen Zeile kann mit dem Befehl

```
crontab -e
```

erledigt werden. Beim ersten Starten wird nach einem Editor gefragt; hier können Sie sich einen beliebigen auswählen. Bei der in den vorherigen Abschnitten durchgeführten Ubuntu-Server-Installation stehen folgende zur Verfügung:

```
boincadm@boinc-testserver:~/projects/tah$ crontab -e
no crontab for boincadm - using an empty one

Select an editor.  To change later, run 'select-editor'.
  1. /bin/ed
  2. /bin/nano        <---- easiest
  3. /usr/bin/vim.basic
  4. /usr/bin/vim.tiny

Choose 1-4 [2]: 3
```

Nach dem Hinzufügen von dem neuen Crontab-Eintrag aus Abb. 5.5 wird dies durch eine Meldung bestätigt: *crontab: installing new crontab projects/tah/*. Der Cron-Dämon ist standardmäßig beim Starten der virtuellen Maschine aktiviert, das durch die Eingabe von folgender Befehlszeile geprüft werden kann:

```
boincadm@boinc-testserver:~/projects/tah$ sudo status cron
cron start/running, process 651
```

5.6.1.3 Hinzufügen von Plattforminformationen

Wissenschaftliche Applikationen können speziell für ausgewählte Rechnerplattformen (kurz Plattformen) entwickelt werden. Eine Plattform ist eine Kurzbeschreibung eines Hosts, der zur Ausführung der wissenschaftlichen Applikation verwendet wird. Dabei enthält die Kurzbeschreibung Informationen über das Betriebssystem (Linux, Apple Mac OS X Darwin, Unix Solaris, Windows) und die Rechnerarchitektur (z. B. 32-Bit, 64-Bit, SPARC, Motorola PowerPC). Mit dem Befehl

```
boincadm@boinc-testserver:~/projects/tah$ ./bin/xadd
```

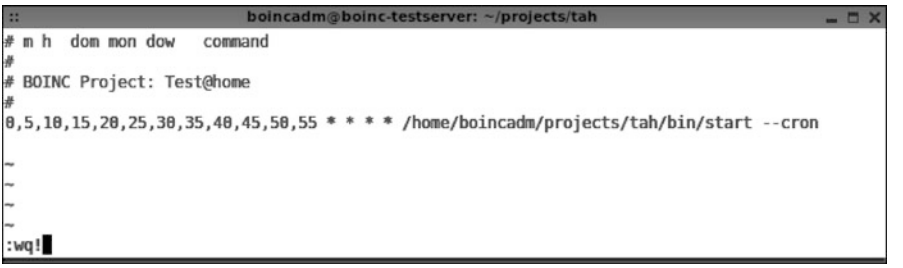


Abb. 5.5 Eingabeeditor für das Hinzufügen eines neuen Crontab-Eintrags

können Standardplattformen hinzugefügt werden. Die Standardplattformen sind:

- **i686-apple-darwin** Mac OS 10.4 or later running on Intel
- **i686-pc-linux-gnu** Linux running on an Intel x86-compatible CPU
- **powerpc-apple-darwin** Mac OS X 10.3 or later running on Motorola PowerPC
- **powerpc64-ps3-linux-gnu** Sony Playstation 3 running Linux
- **sparc-sun-solaris** Solaris 2.8 or later running on a SPARC-compatible CPU
- **sparc-sun-solaris2.7** Solaris 2.7 running on a SPARC-compatible CPU
- **sparc64-sun-solaris** Solaris 2.8 or later running on a SPARC 64-bit CPU
- **windows_intelx86** Microsoft Windows (98 or later) running on an Intel x86-compatible CPU
- **windows_x86_64** Microsoft Windows running on an AMD x86_64 or Intel EM64T CPU
- **x86_64-apple-darwin** Intel 64-bit Mac OS 10.5 or later
- **x86_64-pc-linux-gnu** Linux running on an AMD x86_64 or Intel EM64T CPU

Für die späteren Entwicklungsschritte benötigen wir nicht alle diese Plattformen. Sie können die Plattformdefinitionen gerne im Vorfeld modifizieren. Abschnitt 5.6.2.1 beschreibt, wie die Plattformen modifiziert werden können. Der vorherige Befehl fügt zudem eine Definition für eine wissenschaftliche Applikation `upperCASE` hinzu. Wenn das nicht erwünscht ist, können Sie in Abschn. 5.6.2 nachlesen, wie Sie das im Vorfeld abändern können. Mit den beiden Befehlen

```
boincadm@boinc-testserver:~/projects/tah$ ./bin/appmgr list
boincadm@boinc-testserver:~/projects/tah$ ./bin/appmgr list_platform
```

können sie sich die aktuell hinzugefügten Plattformen und Applikationen anzeigen lassen.

5.6.1.4 `.htaccess` für den administrativen Zugriff

Durch die Eingabe der Webadresse http://192.168.1.100/tah_ops in ihrem Webbrowser haben Sie Zugriff auf die Administrationsseite vom erstellten BOINC-Projekt. Beim Aufrufen werden Sie nach Authentifizierungsdaten gefragt, es wurden allerdings noch keine definiert. Der Webserver erstellt die Abfrage und sendet diese an den aufrufenden Webbrowser. Weiterhin muss der Webserver auch wissen,

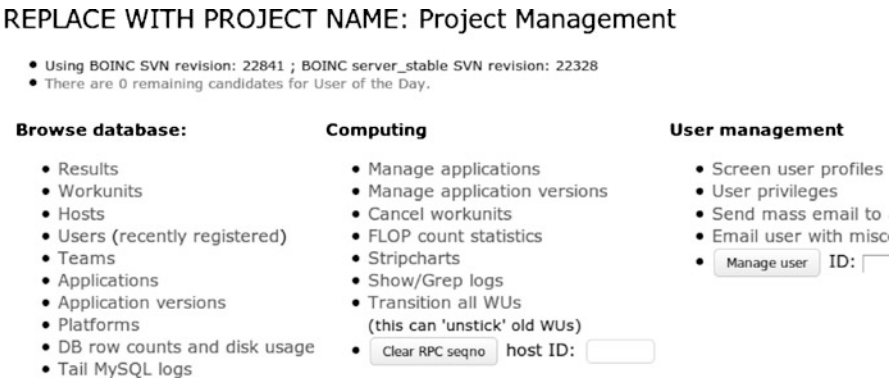


Abb. 5.6 Erste Ansicht der Administrationswebseite der Testinstallation eines BOINC-Projektes

wen er mit welchen Daten authentifizieren kann oder nicht. Dafür muss der Projektadministrator die Datei `.htaccess` erstellen. `.htaccess` enthält eine Beschreibung des Authentifizierungsdialogs. Weiterhin beschreibt die Datei, wo sich die Datei mit der Liste von Benutzernamen mit zugehörigen Passwörtern befindet, die Zugriff auf die Administrationsseite erhalten sollen. Durch die Eingabe von

```
boincadm@boinc-testserver:~/projects/tah$ htpasswd -c \
  html/ops/.htpasswd boincadm
New password:
Re-type new password:
Adding password for user boincadm
```

wird der Benutzer `boincadm` zu dieser Benutzerliste hinzugefügt. Es wird nach einem zu vergebenden Passwort gefragt, unser Testsystem erhält wieder das Passwort `boincadm`. Nach nochmaliger Eingabe der Webadresse http://192.168.1.100/tah_ops und der Eingabe der soeben vergebenen Authentifizierungsdaten erhalten wir Zugriff zur Administrationsseite unseres BOINC-Projektes (s. Abb. 5.6).

5.6.2 Eine erste wissenschaftliche Applikation hinzufügen

Dieser Abschnitt beschreibt, wie Sie die Grundkonfiguration für ein BOINC-Projekt vornehmen können, so dass Sie danach „nur noch“ eine wissenschaftliche Applikation entwickeln und hinzufügen müssen.

5.6.2.1 Verwaltung der Plattformen

Die Standardplattformen sind in der Konfigurationsdatei `project.xml` definiert. Das Plattformdefinitionen sind einzelne XML-Beschreibungen und entsprechen dem Format:

```
PLATFORM:
```

```
  <platform>
    <name>STRING</name>
    <user_friendly_name>STRING</user_friendly_name>
  </platform>

<boinc>
  PLATFORM*
</boinc>
```

Die Platzhalter `STRING` können jede Zeichenkette aufnehmen. Für `name` gibt es vom BOINC-Projekt vordefinierte Plattformen, die bereits in Abschn. 5.6.1.3 aufgelistet sind. Wenn noch keine Plattformen zum BOINC-Projekt hinzugefügt sind oder neue hinzugefügt werden sollen, übernimmt das Kommando `./bin/xadd` die gemachten Änderungen. Sollten Plattformen gelöscht werden, dann kann dies mit

```
boincadm@boinc-testserver:~/projects/tah$ ./bin/appmgr \
  delete_platform PLATTFORMNAME
```

durchgeführt werden. Vorhandene Plattformdefinitionen können nicht einfach überschrieben werden, diese müssen im ersten Schritt gelöscht und dann neu hinzugefügt werden. Wenn in einem Schritt alle Plattformen gelöscht werden sollen, reicht die nachfolgende – etwas komplizierte – Befehlskette:

```
boincadm@boinc-testserver:~/projects/tah$ for a in `./bin/appmgr list_platform
    | awk '{s=$1; print substr(s,1,length(s)-1);}'`
> do
> ./bin/appmgr delete_platform ${a}
> done
Deleted <Platform #36 anonymous>
Deleted <Platform #30 i686-apple-darwin>
Deleted <Platform #27 i686-pc-linux-gnu>
Deleted <Platform #29 powerpc-apple-darwin>
Deleted <Platform #35 powerpc64-ps3-linux-gnu>
Deleted <Platform #33 sparc-sun-solaris>
Deleted <Platform #32 sparc-sun-solaris2.7>
Deleted <Platform #34 sparc64-sun-solaris>
Deleted <Platform #25 windows_intelx86>
Deleted <Platform #26 windows_x86_64>
Deleted <Platform #31 x86_64-apple-darwin>
Deleted <Platform #28 x86_64-pc-linux-gnu>
boincadm@boinc-testserver:~/projects/tah$
```

Eine wissenschaftliche Applikation kann mit

```
boincadm@boinc-testserver:~/projects/tah$ bin/appmgr list
uppercase: upperCASE
-------------------
No homogeneous redundancy. Weight 1.
Versions:

boincadm@boinc-testserver:~/projects/tah$ bin/appmgr delete uppercase
Removed <App#3 uppercase>
```

aus der Datenbank gelöscht werden. Wir definieren uns vier Plattformen für Windows und Linux jeweils mit 32-/64-Bit Unterstützung (s. Abb. 5.7).

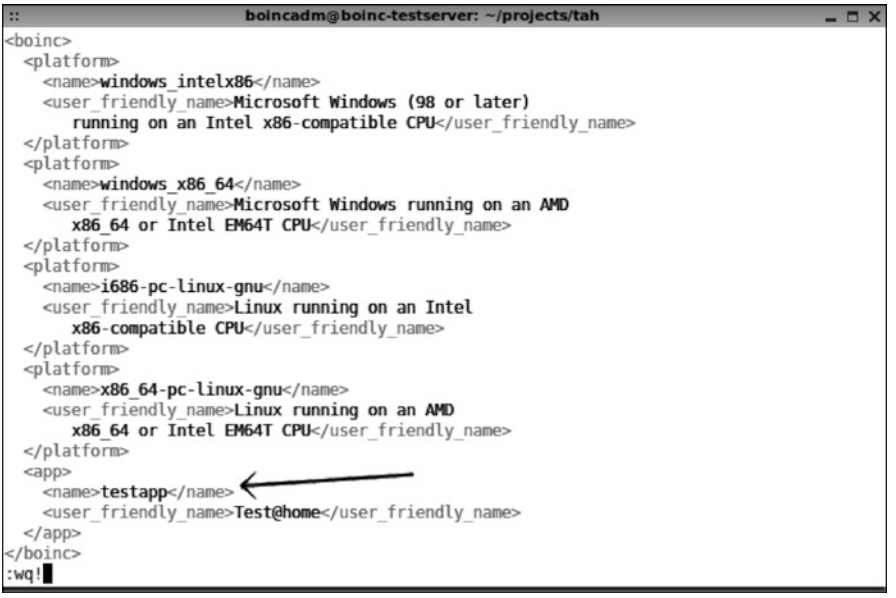


Abb. 5.7 Plattformdefinitionen für den Testbetrieb unseres BOINC-Projektes

5.6.2.2 Wissenschaftliche Applikation eintragen

Die Konfiguration für eine wissenschaftliche Applikation wird auch in `config.xml` vorgenommen. Das XML-Format eines solches Eintrags ist

```
<boinc>
  <app>
    <name>STRING</name>
    <user_friendly_name>STRING</user_friendly_name>
    [ <min_version>INTEGER</min_version> ]
    [ <homogeneous_redundancy>0|1</homogeneous_redundancy> ]
    [ <weight>DOUBLE</weight> ]
    [ <beta>1</beta> ]
  </app>
  PLATFORM*
</boinc>
```

Wir definieren uns eine Testapplikation `testapp`. Abbildung 5.7 zeigt die XML-Definitionen, die zur Projektkonfiguration hinzugefügt wurden. An dieser Stelle haben Sie zwei Möglichkeiten für das weitere Lesen dieses Buchs. Sie können das Lesen in den nachfolgenden Abschnitten weiterführen oder gehen direkt zum Kap. 7, um sich in die Programmierschnittstelle von BOINC einzulesen und eventuell schon eine kleine eigene wissenschaftliche Testapplikation zu entwickeln und dem BOINC-Projekt hinzufügen.

5.6.2.3 Wissenschaftliche Applikation promoten

Wir haben nun alles vorbereitet, um eine wissenschaftliche Applikation zum Projekt hinzuzufügen. In den BOINC-Quellen ist eine Beispielanwendung enthalten. An dieser Stelle werden wir nicht die Implementierung betrachten, sondern nur, welche Aufgabe diese Testapplikation hat. Die Testapplikation ist eine eher fiktive Problemstellung, denn sie erwartet eine Datei mit Plain-Text, welcher im Ergebnis als Ganzes Großbuchstaben enthält. Aus

```
TestTextTestTextTestTextTestTextTestTextTestTextTestTextTestTextTest
TextTestTextTestTextTestTextTestTextTestTextTestTextTestTextTestText
TestTextTestTextTestTextTestTextTestTextTestTextTestTextTestTextTest
TextTestTextTestTextTestTextTestTextTestTextTestTextTestTextTestText
TestTextTestTextTestTextTestTextTestTextTestTextTestTextTestTextTest
TextTestText
```

folgt

```
TESTTEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXTTEST
TEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXT
TESTTEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXTTEST
TEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXT
TESTTEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXTTEST
TEXTTESTTEXT
```

Ich denke nicht, dass Sie jemals solch eine Applikation benötigen, aber es ist ein sehr gutes Beispiel innerhalb eines BOINC-Projektes, um die Funktionalitäten zu testen. Abbildung 5.8 zeigt die mitgelieferte Beispielapplikation, die beim Kompilieren der BOINC-Quellen automatisch mit erstellt wird. `uc2.cpp`, `uc` steht in diesem Fall für `upperCASE`, und `uc2` ist hier die ausführbare Datei. Die ausführbare Datei `uc2` muss nun in eine plattformspezifische Datei umbenannt oder kopiert werden. Es handelt sich in unserem Fall um eine 32-Bit-Applikation für Linux, und unsere Testapplikation hat den Namen `testapp`, daher kopieren wir diese Datei nach `testapp_0.1_i686-pc-linux-gnu`, so wie dies dem Abschn. 5.6.1.3 zu entnehmen ist. Weiterhin handelt es sich um die erste Version unserer Testapplikation, wir haben daher `0.1` als Versionskennzeichnung gewählt. Im nächsten Schritt erstellen wir im Dateiordner `apps/` einen Unterordner `testapp`. Abbildung 5.7 gibt diesen Namen vor; hätten wir in der Konfiguration einen anderen Namen gewählt, so müsste dieser für den Namen des Unterordners verwendet werden. In diesen Ordner kopieren wir nun unsere Testapplikation. Ihre Dateihierarchie und Dateistruktur sollte nun so sein wie in Abb. 5.9. Für unseren Test haben wir folgende Plattformdefinitionen zum Projekt hinzugefügt:

```
i686-pc-linux-gnu: Linux running on an Intel
       x86-compatible CPU
windows_intelx86: Microsoft Windows (98 or later)
       running on an Intel x86-compatible CPU
windows_x86_64: Microsoft Windows running on an AMD
       x86_64 or Intel EM64T CPU
x86_64-pc-linux-gnu: Linux running on an AMD
       x86_64 or Intel EM64T CPU
```

Allerdings haben wir nur eine Applikation für die erste Plattform hinzugefügt, sollten Sie mehr hinzufügen wollen, dann müssen die einzelnen Applikationen passend für die jeweiligen Plattformen kompiliert und entsprechend benannt werden.

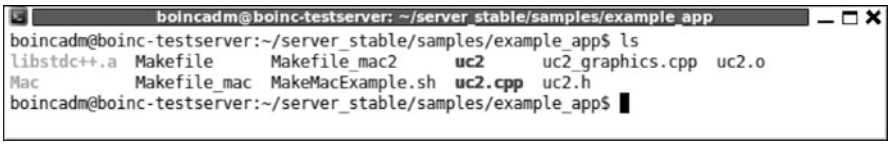


Abb. 5.8 Eine mitgelieferte BOINC-Beispielapplikation `up2`

Mit dem Kommando `./bin/update_versions` fügen Sie die Testapplikation schlussendlich zum BOINC-Projekt zu. Abbildung 5.10 zeigt das Hinzufügen der Testapplikation. Sie können die Abfragen alle mit `Y` für `Yes` beantworten; das Hinzufügen sollte ohne Probleme funktionieren. Die Abfragen haben folgenden Sinn:

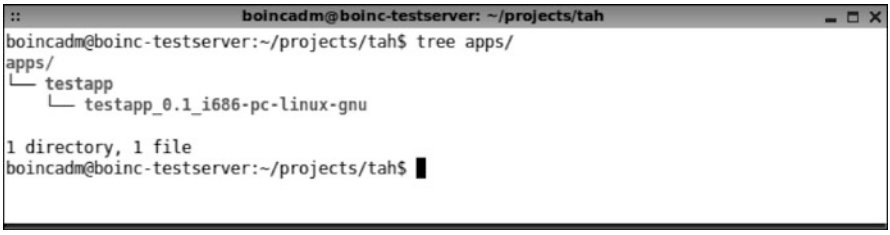


Abb. 5.9 Struktur der Testapplikation innerhalb eines BOINC-Projektes

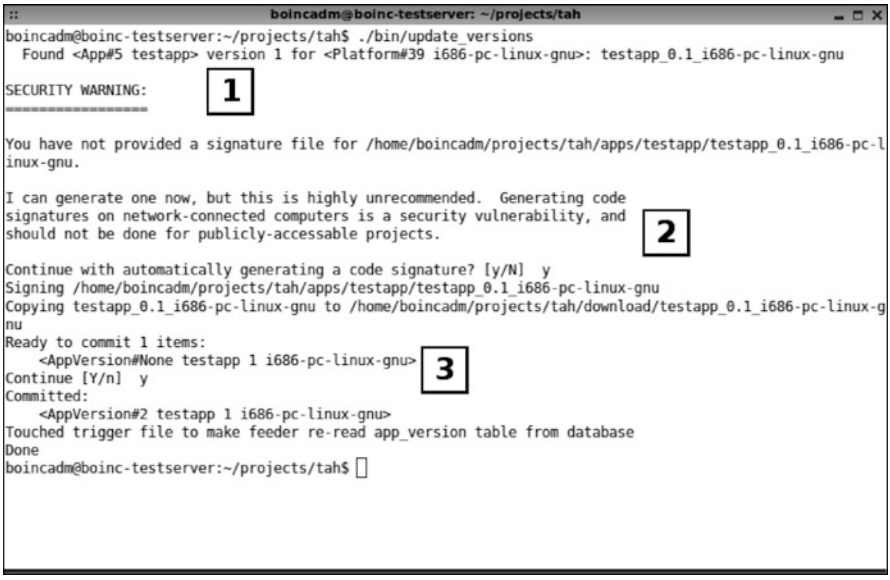


Abb. 5.10 Testapplikation zum BOINC-Projekt hinzufügen

1. Es wurde im Unterordner `apps/testapp` eine Applikation gefunden, welche zum BOINC-Projekt hinzugefügt werden soll.
2. Diese Abfrage erscheint nur, wenn im Vorfeld keine kryptografischen Schlüssel erstellt worden sind. Sollte der Rechner, auf dem Sie das Testprojekt erstellen, mit dem Netzwerk verbunden sein und sollte es sich um ein produktives System handeln, so sollte dieser Schritt nicht mit `Y` bestätigt werden. In diesem Fall sollten die Schlüssel auf einer externen Maschine ohne Netzwerkanschluss erstellt werden und der öffentliche kryptografischen Schlüssel manuell hinzugefügt werden.
3. Im letzten Schritt wird nochmals gefragt, ob Sie die Einstellungen wirklich übernehmen wollen. Eine Bestätigung fügt die gefundene Testapplikation zum BOINC-Projekt hinzu.

Es wird davon abgeraten, dass kryptografischen Schlüssel auf einem Rechner erstellt werden, die mit einem Netzwerk verbunden sind. Es sollte ein separater Rechner verwendet werden, so dass dieser nur für die Generierung von kryptografischen Schlüssel genutzt wird. Für den Transport der öffentlichen Schlüssel sollte ein USB-Stick oder eine CD-Rom verwendet werden und weiterhin sollten nur ausgewählte Personen Zugriff auf diesen Rechner haben. Hacker könnten sonst schädliche Software an die Teilnehmer eines BOINC-Projektes senden, und es kann nicht geprüft werden, ob die Software die Originalsoftware ist.

5.6.2.4 Arbeitspakete zum Projekt hinzufügen

Nach dem Starten unseres Projekts können sich Teilnehmer zwar anmelden und die Testapplikation herunterladen, allerdings werden sie keine Arbeit für uns erledigen. Es fehlen Arbeitspakete und die werden wir als nächstes erstellen. Für die Erstellung von Arbeitspaketen benötigen wir im Fall unserer Testapplikation drei Dateien. Diese müssen manuell erstellt und zum BOINC-Projekt hinzugefügt werden. Dabei handelt es sich um folgende Dateien (die Namensgebung ist unwesentlich):

imput_template.xml Beschreibt den virtuellen Namen einer Eingabedatei und wie oft ein Arbeitspaket berechnet werden muss, bis es vom Validierer geprüft wird.

result_template.xml Beschreibt den virtuellen Namen der Ergebnisdatei und wohin das Ergebnis nach der Berechnung gesendet werden soll.

textfile%04d.txt Enthält unseren Text, welcher in Großbuchstaben umgewandelt werden soll. `%04d` ist dabei ein Platzhalter im Namen, der durch einen inkrementierenden Zahlenwert je erstelltem Arbeitspaket hochgezählt wird. Das erste Arbeitspaket hat also den Namen `textfile0001.txt` und das zweite Arbeitspaket `textfile0002.txt`. Als Text für unsere Tests verwenden wir die Test-Zeichenkette von oben.

Wir erstellen uns ein triviales Programm zur Erstellung von zehn Eingabedateien für Arbeitspakete. Das Programm aus Listing 5.13 reicht für diese Zwecke völlig aus.

Listing 5.13 Programm zur Erstellung von zehn Eingabedateien

```
#include <stdio.h>

#define WU_NAME "textfile%04d.txt"
#define WU_MAX 10

int main( int argc, char **argv) {
  for(unsigned int i=0; i<WU_MAX; i++) {
    char filename[16] = {'\0'};
    sprintf(filename, WU_NAME, i);

    FILE* f=fopen(filename, "w");
    if(f != NULL) {
      fprintf(f,
"TestTextTestTextTestTextTestTextTestTextTestTextTestTextTestTextTestText\n"
"TextTestTextTestTextTestTextTestTextTestTextTestTextTestTextTestTest\n"
"TestTextTestTextTestTextTestTextTestTextTestTextTestTextTestTextTest\n"
"TextTestTextTestTextTestTextTestTextTestTextTestTextTestTextTestTest\n"
"TestTextTestTextTestTextTestTextTestTextTestTextTestTextTestTextTest\n"
"TextTestText"
      );
      fclose(f);
    }
  }
  return 0;
}
```

Nachdem das Programm kompiliert und ausgeführt worden ist, haben wir zehn neue Dateien im aktuellen Arbeitsordner, wie Abb. 5.11 zeigt. Im nächsten Schritt müssen die Dateischablonen `input_template.xml` (s. Listing 5.14) und `result_template.xml` (s. Listing 5.15) erstellt werden. An dieser Stelle werden diese Dateien nicht im Detail beleuchtet, Näheres zu diesem Thema finden Sie in Abschn. 9.2ff. Es sei allerdings erwähnt, dass die Eingabedatei – sprich unsere Dateien mit den `Test`-Zeichenketten – mit dem Dateinamen `in` geöffnet und das Ergebnis der Umwandlung in Großbuchstaben in der Datei mit dem Dateinamen `out` gespeichert wird.

Listing 5.14 `input_template.xml` der Testapplikation

```
<file_info>
  <number>0</number>
</file_info>
<workunit>
  <file_ref>
    <file_number>0</file_number>
    <open_name>in</open_name>
  </file_ref>
  <min_quorum>1</min_quorum>
  <target_nresults>1</target_nresults>
</workunit>
```

<min_quorum> Dieser Wert beschreibt, wie viele Ergebnisse von einem Arbeitspaket vorliegen müssen, bevor der Validierer die Ergebnisse prüft. Dabei können die Berechnungen wahlweise auf einem Rechner oder verschiedenen Rechnern durchgeführt werden.

<target_nresults> Dieser Zahlenwert muss mindestens der Anzahl an `<min_quorum>` [85] entsprechen und sagt aus, wie viele Arbeitspakete von einer Sorte erstellt werden. Zum Beispiel werden fünf erstellt, allerdings wird nur ein Ergebnis benötigt, dann werden die weiteren vier nicht mehr beachtet und auch

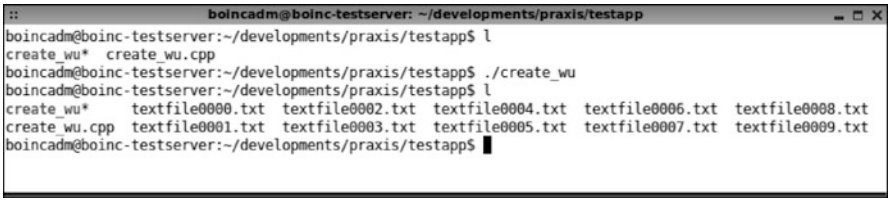


Abb. 5.11 Dateiordner mit zehn erstellten Eingabedateien für Arbeitspakete unserer Testapplikation

nicht mehr an die Teilnehmer versendet. Dadurch können je nach Teilnehmeranzahl schnellere Berechnungsergebnisse geliefert werden.

Listing 5.15 beschreibt, wie mit Ergebnissen umzugehen ist. Der erste Teil beschreibt, welche Namen zum Zugriff des BOINC-Clients auf die Ergebnisse verwendet werden sollen, und dass die Datei eine neue, lokale Datei ist. Für ein korrektes Hochladen darf die Datei in diesem Fall nicht größer als 10.000 Bytes sein.

Listing 5.15 `result_template.xml` der Testapplikation

```
<file_info>
  <name><OUTFILE_0/></name>
  <generated_locally/>
  <upload_when_present/>
  <max_nbytes>10000</max_nbytes>
  <url><UPLOAD_URL/></url>
</file_info>
<result>
  <file_ref>
    <file_name><OUTFILE_0/></file_name>
    <open_name>out</open_name>
  </file_ref>
</result>
```

Eine Kurzbeschreibung einiger Konfigurationen ist:

<OUTFILE_0/>, <generated_locally/> Diese XML-Tags beschreiben einen generierten Namen, der vom BOINC-Framework definiert wird und eindeutig ist, so dass das Ergebnis ohne Konflikte an den BOINC-Server übertragen werden kann. `<generated_locally/>` beschreibt eine neu erstellte Datei auf der Seite des BOINC-Clients.

<upload_when_present/> Dieser XML-Tag beschreibt, dass die Datei zum Projekt hochgeladen werden soll, wenn das entsprechende Arbeitspaket abgearbeitet worden ist.

<max_nbytes/> Durch diesen Parameter wird die Ergebnisgröße auf 100.000 Bytes gesetzt. Wenn das Ergebnis mehr Speicherplatz benötigt, wird ein Fehler gemeldet.

<UPLOAD_URL/> Dieser XML-Tag ist eine Substitution zu der Adresse, an die die Berechnungsergebnisse übertragen werden. Diese Adresse wird automatisch aus den Projektinformationen ermittelt.

Die zwei Dateischablonen kopieren wir in den Unterordner `templates/` unseres BOINC-Testprojekts. Wir sollten nun die Struktur aus Abb. 5.12 haben.

```
::                boincadm@boinc-testserver: ~/projects/tah
boincadm@boinc-testserver:~/projects/tah$ tree apps/ templates/
apps/
└── testapp
    └── testapp_0.1_i686-pc-linux-gnu
templates/
├── input_template.xml
└── result_template.xml

1 directory, 3 files
boincadm@boinc-testserver:~/projects/tah$
```

Abb. 5.12 Struktur der Testapplikation und Dateischablonen

Nun wird es Zeit, ein wenig Arbeit für die Teilnehmer zu erstellen. Bis jetzt haben wir eine Testapplikation zum Textprojekt hinzugefügt, zehn Eingabedateien erzeugt, die Dateischablonen erstellt und in die relevanten Ordner kopiert – nun müssen alle diese Komponenten zu Arbeitspaketen zusammengefügt und dem Projekt mitgeteilt werden!

Abbildung 5.13 gibt einen Überblick über die erstellten Dateien und zeigt in welchem Zusammenhang diese zueinander stehen. Ein Arbeitspaket besteht aus den zuvor erstellten drei Dateien. Die Eingabedatei wird innerhalb der Testapplikation mit dem Dateinamen `in` geöffnet und das Ergebnis wird in der Datei mit dem Namen `out` gespeichert. Es handelt sich bei diesem Mechanismus um das sogenannte Slot-Verfahren, das in Abschn. 4.3.2 näher beschrieben wird. Für die Erstellung von Arbeitspaketen liefert das BOINC-System ein Programm als Hilfestellung mit. Dieses befindet sich im Unterordner `bin/` unseres Testprojekts und hat den Namen `create_work`. Das Programm kann nicht ohne Weiteres einfach aufgerufen werden, es erwartet eine ganze Palette an Aufrufparametern und zudem im Vorfeld ein – durch ein weiteres Programm erfolgtes – Kopieren der Eingabe-

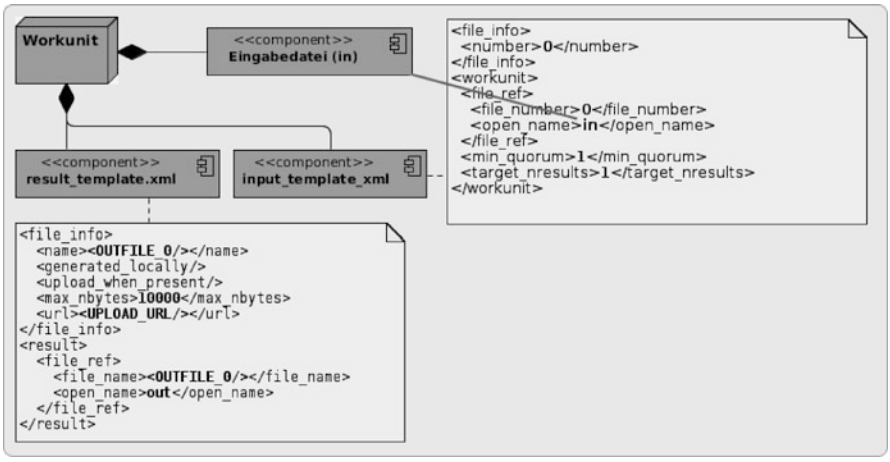


Abb. 5.13 Struktur der Dateischablonen und der Eingabeparameterdatei für die Testapplikation

parameterdateien in die Hierarchie der herunterzuladenden Dateien innerhalb eines BOINC-Projektes. Ein BOINC-Projekt besitzt die zwei Dateiordner `download/` (zu Deutsch Herunterladen) und `upload/` (Hochladen). Der Ordner zum Herunterladen besitzt in der Regel eine Ordnerhierarchie, in die wir die Eingabeparameterdateien hineinkopieren müssen. Das Kopieren übernimmt ein Programm für uns, welches auch automatisch die Eintragung in der Datenbank vornimmt, so dass der Scheduler weiß, wo sich welche Dateien befinden, um ein Herunterladen für die Teilnehmer zu ermöglichen. Das Skript in Listing 5.16 erstellt die Arbeitspakete für unser Testprojekt.

Listing 5.16 Skript zum Erstellen von Arbeitspaketen für das Testprojekt

```
#!/bin/bash

ROOTPROJECT=/home/boincadm/projects/tah

let count=0

for file in textfile*.txt
do
  ((count++))
  WUNAME_INPUT=${file}
  WUPATH_INPUT=`pwd`/${WUNAME_INPUT}
  echo "WU_INPUT: " ${WUPATH_INPUT}

  TPLINPUT=`pwd`/input_template.xml
  TPLRESULT=`pwd`/result_template.xml

  cd ${ROOTPROJECT}
    pwd
    BASEWUNAME_INPUT=`basename ${WUNAME_INPUT}`
    cp ${WUPATH_INPUT} `bin/dir_hier_path ${BASEWUNAME_INPUT}`

    cmd="./bin/create_work -appname testapp \
      -wu_name testapp_wu_${count} \
      -wu_template templates/input_template.xml \
      -result_template templates/result_template.xml \
      ${BASEWUNAME_INPUT}"

    echo $cmd
    $cmd
  cd -
done
```

Listing 5.16 zeigt, dass das Skript direkt ohne Parameter aufgerufen werden kann. Es zeigt allerdings auch, wie viele Schritte für die Erstellung nötig sind. Zeile 20 kopiert die Eingabeparameterdateien in die Ordnerhierarchie unseres Testprojekts. Die Anzahl der hier möglichen Unterordner kann in der Konfiguration `config.xml` durch den Parameter `<uldl_dir_fanout> %d </uldl_dir_fanout>` eingestellt werden, der Standard ist 1024, was bedeutet, dass die Ordnernamen von `0x00` bis `0x3ff.` gehen [86]. Die Zeilen 22 bis 26 erstellen die Arbeitspakete. Das Programm `./bin/create_work` wird mit fünf Parametern aufgerufen; für jedes Arbeitspaket muss ein Aufruf erfolgen. Die Parameter haben folgende Bedeutung:

-appname %s Der Name der Testapplikation, in diesem Fall ist dies `testapp`.

-wu_name %s Jedes Arbeitspaket muss einen eigenen Namen erhalten. In diesem Beispiel verwenden wir dafür einen Zähler, der nach jedem Erstellen eines Arbeitspaket um Eins inkrementiert wird, so haben wir nach der Durchführung die Arbeitspakete mit den Namen von `testapp_wu_1` bis `testapp_wu_10`.

-wu_template %s Dieser Parameter erwartet den Dateinamen unserer Eingabeschablone `input_template.xml`, die in `templates/`-Dateiordner vom Testprojekt liegt.

-result_template %s Die Ergebnisschablone wird durch diesen Parameter übernommen.

INPUT%4_FILES Hier können die Namen der Eingabeparameterdateien angegeben werden. Die Anzahl wird nur durch die maximale Anzahl an zu übergebenen Parametern in der Konsole beschränkt. Es dürfen nur die Namen der Dateien – mit Dateiendung, ohne Pfadangabe – mitgegeben werden. Solch eine Eingrenzung wird auch Basisname einer Datei genannt.

Listing 5.17 zeigt die Ausgabe bei der Erstellung der Arbeitspakete. Jede Zeile, die mit `WU_INPUT` beginnt, beschreibt die Erstellung eines Arbeitspakets. Die Zeile danach ist das Verzeichnis unseres Testprojekts, und danach folgt der Aufruf zum Hinzufügen eines Arbeitspakets mit dem BOINC-Werkzeug `./bin/create_work`.

Listing 5.17 Ausgabe beim Aufruf des Skripts zur Erstellung von Arbeitspaketen für das Testprojekt

```
boincadm@boinc-testserver:~/developments/praxis/testapp$ ./make_work
WU_INPUT: /home/boincadm/developments/praxis/testapp/textfile0000.txt
/home/boincadm/projects/tah
./bin/create_work -appname testapp -wu_name testapp_wu_1 -wu_template templates
    /input_template.xml -result_template templates/result_template.xml
    textfile0000.txt
/home/boincadm/developments/praxis/testapp
...
...
...
WU_INPUT: /home/boincadm/developments/praxis/testapp/textfile0008.txt
/home/boincadm/projects/tah
./bin/create_work -appname testapp -wu_name testapp_wu_9 -wu_template templates
    /input_template.xml -result_template templates/result_template.xml
    textfile0008.txt
/home/boincadm/developments/praxis/testapp
WU_INPUT: /home/boincadm/developments/praxis/testapp/textfile0009.txt
/home/boincadm/projects/tah
./bin/create_work -appname testapp -wu_name testapp_wu_10 -wu_template
    templates/input_template.xml -result_template templates/result_template.
    xml textfile0009.txt
/home/boincadm/developments/praxis/testapp
```

Das Ergebnis unserer vorherigen Arbeitsschritte ist in Abb. 5.14 zu sehen. In der Hierarchie zum Herunterladen von Dateien befinden sich nun 11 Dateien, die Testapplikation und die 10 Eingabeparameterdateien.

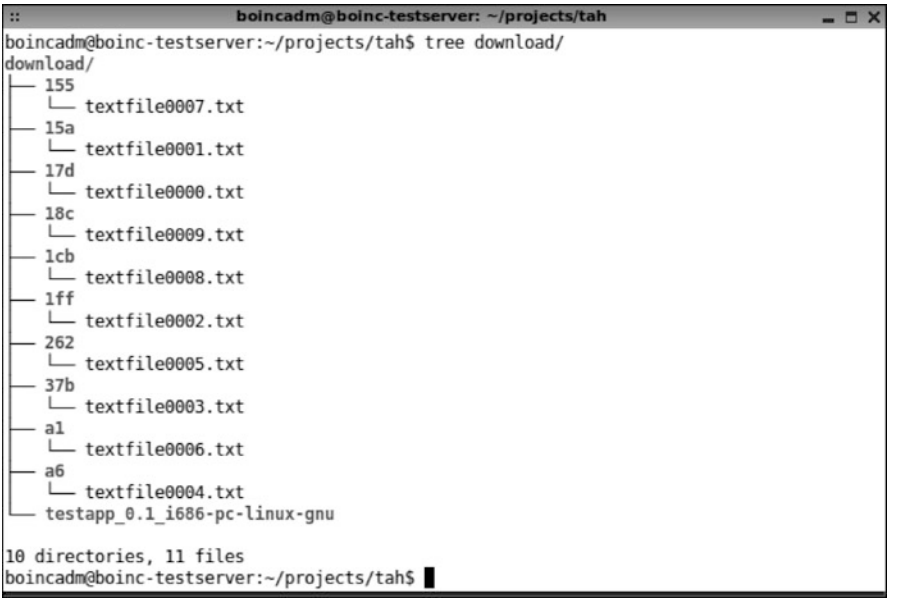


Abb. 5.14 Die Hierarchie der hinzugefügten Parameterdateien innerhalb unseres Testprojekts

5.6.2.5 Dämonen konfigurieren

Aus Abschn. 4.2 wissen wir, dass ein Kreislauf an Applikationen konfiguriert sein muss, so dass ein Arbeitspaket als fertig deklariert werden kann, wenn es einmal zum Projektserver gesendet wurde. In der Standardinstallation fehlen für diesen Zweck zwei Applikationen in diesem Kreislauf: Das ist erstens ein Validierer und zweitens ein Assimilierer. Beide müssen manuell zur Konfiguration `config.xml` eines BOINC-Projektes hinzugefügt werden. Listing 5.18 zeigt die Zeilen, die zwischen die XML-Tags `<daemons>` und `</daemons>` hinzugefügt werden müssen.

Listing 5.18 Konfigurationen für die Dämonen des Testprojekts

```
<daemon>
  <cmd> sample_bitwise_validator -d 2 -app testapp </cmd>
  <output> sample_bitwise_validator.log </output>
  <pid_file> sample_bitwise_validator.pid </pid_file>
</daemon>
<daemon>
  <cmd> sample_assimilator -d 2 -app testapp </cmd>
  <output> sample_assimilator.log </output>
  <pid_file> sample_assimilator.pid </pid_file>
</daemon>
```

Der hier konfigurierte Assimilierer benötigt für das Assimilieren noch das Verzeichnis `sample_results/` im Projektordner. Dieses stellen wir noch kurz bereit.

```
boincadm@boinc-testserver:~/projects/tah$ mkdir sample_results
boincadm@boinc-testserver:~/projects/tah$ cd sample_results/
boincadm@boinc-testserver:~/projects/tah/sample_results$ pwd
```

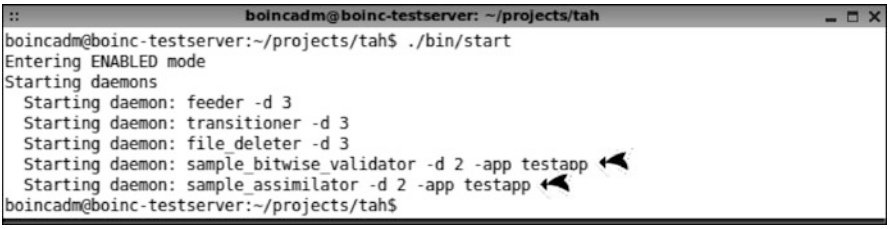


Abb. 5.15 Das BOINC-Testprojekt starten

Abb. 5.16 Den Status des BOINC-Testprojekts ermitteln

```
/home/boincadm/projects/tah/sample_results
```

Sollten Sie das Verzeichnis mit einem anderen Benutzer als `boincadm` erstellen, so sollten Sie noch die Eigentümerrechte ändern, so dass es beim Schreiben in diesen Ordner keine Zugriffsprobleme gibt.

5.6.2.6 Endlich rechnen

Nun sind wir endlich soweit und können unser Projekt starten, die Aufgaben verteilen und Ergebnisse erhalten. Durch den Befehl `./bin/start` im Projektordner können wir das Testprojekt starten; dies kann je nach Einstellung ein paar Sekunden dauern. Abbildung 5.15 zeigt, wie die einzelnen Dämonen gestartet werden und die letzten beiden Zeilen der Ausgabe zeigen, dass der Validierer und der Assimilierer für die Testapplikation `testapp` gestartet werden soll. Der Status unseres BOINC-Projekts kann mit `./bin/status` ermittelt werden. Abbildung 5.16 beinhaltet alle relevanten Informationen über die Dämonen und sonstigen Aufgaben (engl. *tasks*). In der ersten Zeile sehen wir, dass das BOINC-Projekt aktiviert ist. In der Mitte ist der Status der einzelnen Dämonen zu erkennen; alle Dämonen sind gestartet. Der untere Bereich liefert Informationen über die Tasks unseres Testprojekts. Es sind keine Tasks aktiviert, jeder Tasks zeigt `yes` für `disabled` (zu deutsch deaktiviert). Sie können nun den BOINC-Manager starten und sich zum Testprojekt verbinden und Arbeitspakete herunterladen. Unser Projekt ist natürlich nicht in

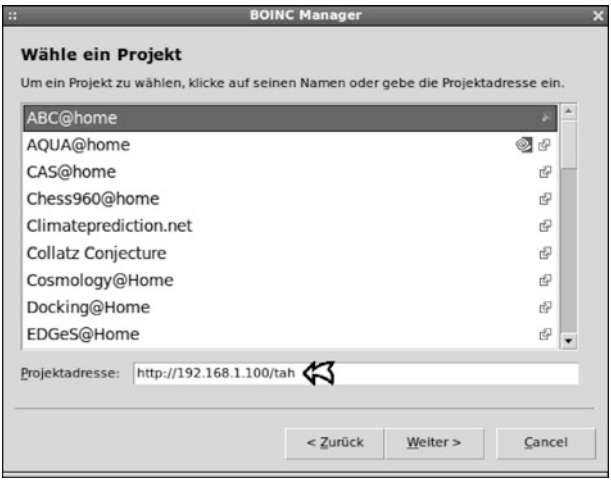


Abb. 5.17 Adresse für das Testprojekt im BOINC-Manager angeben

Abb. 5.18 Eingabe der Authentifizierungsdaten für das Testprojekt

der Liste der öffentlichen Projekte aus Abb. 5.17, daher muss in der unteren Eingabe die Adresse des installierten Testprojekts eingegeben werden. Danach kann auf *Weiter* geklickt werden und es wird versucht, eine Kommunikation mit dem Testprojekt durchzuführen. Wenn eine Kommunikation möglich ist, so werden die Authentifizierungsdaten in Abb. 5.18 abgefragt. Dabei kann wahlweise direkt ein neuer Benutzer angelegt oder ein bisher vorhandener verwendet werden. Danach kann auf *Weiter* geklickt werden und die Daten werden im nächsten Kommunikationsschritt mit dem Testprojekt geprüft. Bei Erfolg kann der Dialog beendet werden, andernfalls sollten die Authentifizierungsdaten überprüft und korrigiert werden. Im

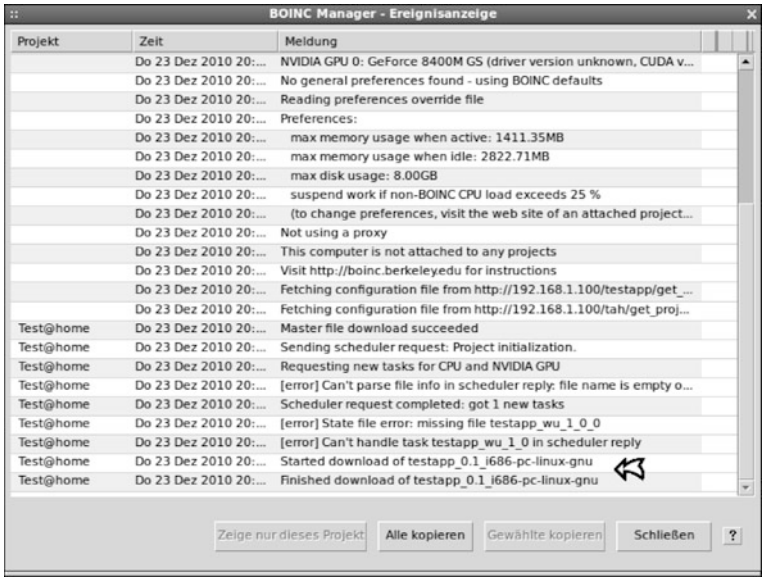


Abb. 5.19 Die erforderlichen Daten für die Berechnung werden vom Testprojekt heruntergeladen

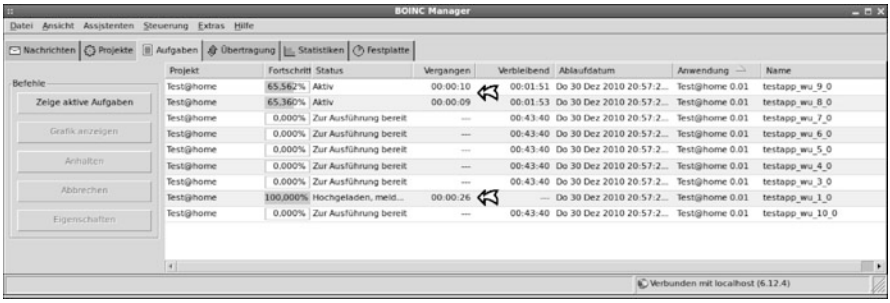


Abb. 5.20 Die Testapplikation wird ausgeführt; zwei Berechnungen sind aktuell in Bearbeitung und eine Berechnung ist fertiggestellt und bereit für das Hochladen zum Testprojekt

Nachrichtenfenster aus Abb. 5.19 werden Informationen über vergangene und den aktuellen Status der Berechnungen angezeigt. In der aktuellen Ausgabe ist zu sehen, wie die Testapplikation vom Testprojekt heruntergeladen wird. Weiterhin werden ein oder mehrere Arbeitspakete heruntergeladen. In Abb. 5.20 ist zu sehen wie diese Arbeitspakete für die Berechnung bereitliegen; zwei sind aktuell am Rechnen und eine Berechnung ist abgeschlossen und bereit für das Hochladen zum Testprojekt. Abbildung 5.21 zeigt das Resultat der gesamten Durchführung aller zehn Arbeitspakete. Die Pakete wurden erfolgreich validiert, assimiliert und zu guter Letzt werden nicht mehr benötigte Eingabeparameterdateien gelöscht. Abbildung 5.22 zeigt die zehn Ergebnisse innerhalb der Ordnerstruktur im Testprojekt. Die Ergebnisse befinden sich im Unterordner `sample_results/` und enthalten jeweils

REPLACE WITH PROJECT NAME: Result summary

10 results

Server state	# results
Inactive	0
Unsent	0
Unsent (in work seq)	0
In Progress	0
Over	10

'Over' results

Outcome	# results
Init	0
Success	10
Couldn't send	0
Client error	0
No reply	0
Didn't need	0
Validate error	0
Client detached	0

'Success' results

Validate state	# results
Initial	0
Valid	10
Invalid	0
Skipped	0
Inconclusive	0
Too late	0

File Delete state	# results
Initial	0
Ready to delete	0
Deleted	10
Delete Error	0
Total files	10

'Client error' results

Client state	# results
Downloading	0
Downloaded	0
Compute error	0
Uploading	0
Uploaded	0
Aborted	0

Abb. 5.21 Die Berechnungen aller zehn Arbeitspakete ist abgeschlossen, validiert, assimiliert und die Dateien nach der Assimilation gelöscht

```
:: boincadm@boinc-testserver: ~/projects/tah/sample_results
boincadm@boinc-testserver:~/projects/tah/sample_results$ l
testapp_wu_1   testapp_wu_2  testapp_wu_4  testapp_wu_6  testapp_wu_8
testapp_wu_10  testapp_wu_3  testapp_wu_5  testapp_wu_7  testapp_wu_9
boincadm@boinc-testserver:~/projects/tah/sample_results$ cat testapp_wu_1
TESTTEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXTTEST
TEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXT
TESTTEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXTTEST
TEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXT
TESTTEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXTTESTTEXTTEST
TEXTTESTTEXTboincadm@boinc-testserver:~/projects/tah/sample_results$
```

Abb. 5.22 Die Ergebnisdateien, nachdem diese validiert und assimiliert wurden

unsere erwarteten Ergebnisse, die jeweils in den Dateien `testapp_wu_1` bis `testapp_wu_10` abgespeichert sind. Ein Ausschnitt `...TESTTEXTTESTT...` aus der Datei `testapp_wu_1` zeigt das.

5.7 Datenbank – das Gedächtnis eines BOINC-Projektes

Abbildung 5.23 zeigt den Blick auf die Datenbankstruktur unseres BOINC-Projektes. Es sind 33 Tabellen erstellt worden, nicht alle werden wirklich für alle BOINC-Projekte benötigt. Allerdings ist das BOINC-System historisch gewachsen. Jeder, der sich einmal mit Qualitätssicherung beschäftigt hat, bekommt eine Gänsehaut, wenn er diese Aussage hört. Aber wie so oft in der Softwareentwicklung war der Erfolg von BOINC nicht abzusehen, und von daher gab es nicht von Anfang an eine

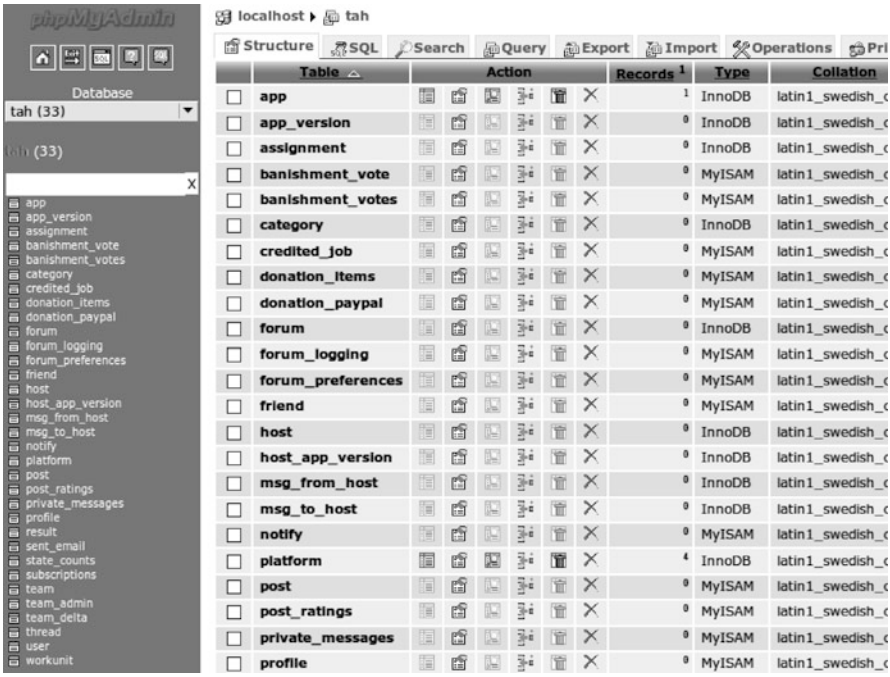


Abb. 5.23 Ansicht der Datenbankstruktur eines BOINC-Projektes mit dem Werkzeug phpMyAdmin [159]

klare Zielrichtung. Weiterhin wird das BOINC-System von einer Vielzahl an Freiwilligen weiterentwickelt und unterstützt und vielleicht kommt irgendwann die Zeit, dass die Datenbank aufgeräumt wird. Wie zuvor erwähnt werden nicht alle Datenbankelemente für alle heutigen BOINC-Projekte benötigt. Einige Strukturen und Tabellen stammen aus der Zeit, als BOINC noch kein offizielles Projekt war. Am Anfang stand ein System für das SETI@home-Projekt [166], woraus sich BOINC entwickelte.

5.7.1 Struktur der Datenbank

Die Datenbank kann in verschiedene Kategorien unterteilt werden:

Applikation Mit den Tabellen `app`, `app_version` und `platform`. Diese Tabellen beschreiben eine zu einem BOINC-Projekt hinzugefügte wissenschaftliche Applikation. Eine wissenschaftliche Applikation muss nur einmalig zur Datenbank hinzugefügt werden, diese Informationen befinden sich dann in der Tabelle `app`. Zu diesem Eintrag gesellen sich in Verbindung mit der Tabelle `app_version`, Spalte `app_version.appid` die einzelnen Applikationen

für die jeweiligen Plattformen, beschrieben durch die Spalte `app_version.platformid`. Tabelle `platform` enthält Informationen, die in Abschn. 5.6.2.1 zum BOINC-Projekt hinzugefügt worden sind.

Forum und Web 2.0 Mit den Tabellen `banishment_vote`, `banishment_votes`, `category`, `forum`, `forum_logging`, `forum_preferences`, `friend`, `notify`, `post`, `post_ratings`, `private_messages`, `profile`, `subscriptions`, `team`, `team_admin`, `team_delta` und `thread`. Wie unschwer zu erkennen ist, gehen die meisten Tabellen in diese Kategorie ein. Durch diese Tabellen wird die Struktur für ein Benutzerforum und den Zusammenschluss von Mitgliedern zu einem Team beschrieben. Das BOINC-Projekt bietet die Möglichkeit, die Team-Zugehörigkeit zu externen Anbietern für die Erstellung von Statistiken der Credits von BOINC-Teilnehmern zu exportieren.[2]

Spendensammlung Mit den Tabellen `donation_items|` und `donation_votes`. Durch diese zwei Tabellen kann ein Spendensystem mit dem Bezahldienstleister PayPal verwendet werden. Um das Spendensystem zu aktivieren, müssen die zwei Zeilen

```
define("PAYPAL_ADDRESS", "paypal@example.com");
define("DONATION_CURRENCY", "EUR")
```

mit ihren Daten in

```
boincadm@boinc-testserver:~/projects/tah/html/project$ vim project.inc
```

eingetragen werden. `DONATION_CURRENCY` kann folgende Währungen enthalten: EUR, CAD, GBP, USD, JPY, AUD, NZD, CHF, HKD, SGD, SEK, DKK, PLN, NOK, HOF und CZK. Weitere Schritte für die Konfiguration sind im BOINC-Wiki nachzulesen und werden in diesem Buch nicht behandelt [77]. Es wird nicht weiter auf das Spendensystem eingegangen.

Hostinformationen Mit den Tabellen `host` und `host_app_version`. Diese Tabellen enthalten zahlreiche Informationen über Hosts, die für die Ausführung einer wissenschaftlichen Applikation verwendet werden, zum Beispiel welche CPU oder wie viel RAM zur Verfügung steht und welche Grafikkarte installiert ist.

Asynchrone Kommunikation Mit den Tabellen `msg_from_host` und `msg_to_host`. Zwischen einzelnen Hosts und dem BOINC-Projekt steht eine asynchrone Kommunikation. Weiterhin können sogenannte Trickle-Messages versendet werden. Bei Trickle-Messages handelt es sich um Anfragen oder Befehle, die eine bestimmte Aktion auf der Seite des BOINC-Teilnehmers oder des BOINC-Projekts zur Folge haben. Zum Beispiel kann es bei länger dauernden Berechnungen vorkommen, dass das BOINC-Projekt zwischendurch nach einem Zwischenstand fragt, um so eventuell vorhandene Credit-Punkte zu vergeben oder Bescheid zu geben, dass ein bestimmtes Arbeitspaket abgebrochen werden soll. Abschnitt 9.4 behandelt Trickle-Messages und wie wissenschaftliche Applikationen mit Trickle-Messages umgehen können.

[2] http://boincstats.com, http://www.allprojectstats.com, http://www.boincsynergy.com.

Berechnungen Mit den Tabellen `assignment`, `result`, `state_counts` und `workunit`. Drei der wichtigsten Tabellen eines BOINC-Projektes sind `result`, `workunit` und `state_count`. In `workunit` werden die Arbeitspakete gespeichert, die die BOINC-Teilnehmer berechnen sollen. In `result` werden die Ergebnisse gespeichert, die von den BOINC-Teilnehmern berechnet wurden. `state_count` enthält den Status der Bearbeitung eines Arbeitspakets, allerdings wird diese Tabelle nicht von vornherein genutzt und erwartet viel Arbeit Ihrerseits. Diese Tabelle entstand aus dem Wunsch heraus, eine höhere Performance zu erhalten, allerdings wird sie von *keiner* BOINC-Applikation nativ unterstützt. Ein Arbeitspaket passiert mehrere Stufen bis zur Fertigstellung, woran mehrere BOINC-Applikationen beteiligt sind (vgl. Abb. 4.1).

Benutzerinformationen `credited_job` gibt an, welche Arbeitspakete von welchem BOINC-Teilnehmer verarbeitet wurden und dass diese zur Credit-Berechnung mit einbezogen werden sollen. `user` enthält die Benutzerinformationen zum Authentifizieren, in welchem Team ein Benutzer ist, welche E-Mail-Adresse er hat und noch weitere Daten. `sent_mail` beschreibt, welcher BOINC-Teilnehmer welche Nachricht erhalten soll, z. B. wenn der Teilnehmer eine falsche Rechner-Konfiguration besitzt, damit er dies korrigieren kann.

Anzumerken ist, dass keine besonderen SQL-Anweisungen benötigt werden, um Datenbankabfragen zu erstellen. Für die Tabellen existieren untereinander keine Fremdschlüssel-Beziehungen oder sonstige tief ins Detail verstrickte Konstellationen. Es wurde versucht, eine simple und einfach zu wartende Datenbankstruktur zu erschaffen, welche von jedem ohne viel Einarbeitungszeit modifiziert werden kann.

5.8 BOINC in einem Cluster installieren

Die Serverkomponenten müssen nicht zwingend auf einem Rechner alleine installiert sein, sondern Sie können mehrere Rechner einbeziehen, um so u. a. eine Lastverteilung zu ermöglichen. Abbildung 5.24 zeigt drei Rechner mit den Hostnamen *vg-challenge-01* bis *vg-challenge-03*. Die Installation eines BOINC-Projekts ist nur auf einem Rechner nötig, in diesem Fall ist dies der Rechner *vg-challenge-01* (auch durch das BOINC-Logo markiert). Nur die config.xml bestimmt darüber, wo die einzelnen BOINC-Applikationen ausgeführt werden sollen. Sie erkennen in den gezeigten Textfeldern, dass die jeweiligen Bereiche für die BOINC-Applikationen – `feeder`, `validator`, `file_deleter` – jeweils einen weiteren XML-Tag `<host>` und die entsprechenden Hostnamen besitzen, auf denen die Applikationen gestartet werden. Der Befehl zum Starten eines BOINC-Projekts `bin/start` liest die `config.xml` aus und verbindet sich über das Systemwerkzeug `ssh` zu den jeweiligen Rechnern. Für diesen Zweck ist es unerlässlich, dass auf jedem Host der SSH-Key vom Haupthost `vg-challenge-01` auf den anderen Rechnern kopiert ist. Dies kann mit dem Befehl `ssh-copy-id` ermöglicht werden.

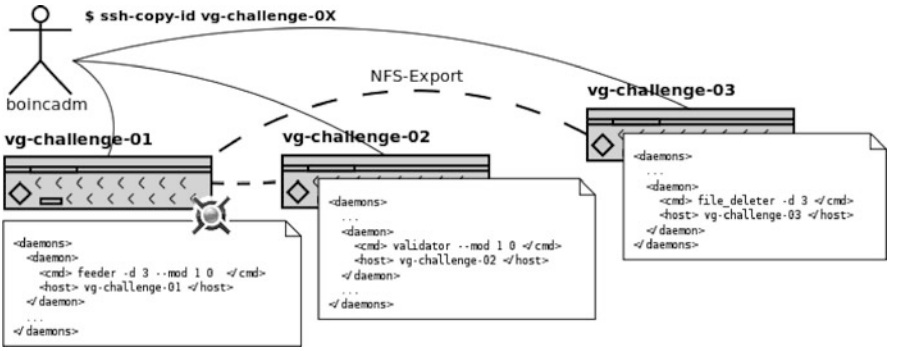


Abb. 5.24 Drei Hosts für die Installation von BOINC auf einem Clustersystem

Bevor Sie das BOINC-Projekt starten, müssen Sie die Installation vom Haupthost auf die weiteren Hosts exportieren, dies geht am einfachsten mit dem NFS-Protokoll. Auf dem Host `vg-challenge-01` fügen Sie in die Datei `/etc/exports` die nachfolgenden Zeilen hinzu:

```
/home/boincadm/projects vg-challenge-02(rw,sync,no_subtree_check) vg-challenge
    -03(rw,sync,no_subtree_check)
```

Daraufhin starten Sie den NFS-Dienst, bei einem Ubuntu-System geht das durch ein Init-Skript:

```
boincadm@vg-challenge-01:~$ sudo /etc/init.d/nfs-kernel-server start
[sudo] password for boincadm:
 * Exporting directories for NFS kernel daemon... [ OK ]
 * Starting NFS kernel daemon                     [ OK ]
```

Auf den anderen beiden Hosts fügen Sie in die Datei `/etc/fstab` folgende Zeile hinzu:

```
vg-challenge-01:/home/boincadm/projects /home/boincadm/projects nfs rsize=8192,
    timeo=14,intr
```

Daraufhin können Sie das Verzeichnis mit dem Befehl `mount` Ihrer Systemumgebung hinzufügen:

```
boincadm@vg-challenge-03:~$ sudo mount /home/boincadm/projects/
```

Ein BOINC-Projekt kann nun gestartet werden, die Statusmeldung in Listing 5.19 sollte bei Ihnen so ähnlich aussehen und zeigen, dass die einzelnen BOINC-Applikationen auf dem jeweiligen Host gestartet sind.

Listing 5.19 Statusanzeige eines BOINC-Projektes, installiert auf drei Rechnern, kombiniert mit einer NFS-Installation

```
boincadm@vg-challenge-01:~/projects/tah$ ./bin/status
BOINC is ENABLED

DAEMON  pid   status   lockfile disabled  commandline
   1    1511  running  locked   no        feeder -d 3 --mod 1 0

TASK        last run ... lock file disabled  commandline
running  ssh vg-challenge-03 cd ~/projects/tah && ./bin/status -v
BOINC is ENABLED
```

```
DAEMON  pid  status   lockfile disabled  commandline
   1   3482  running  locked   no        file_deleter -d 3

TASK       last run ... lock file disabled  commandline
   1 2011/10/22     ... unlocked  no        run_in_ops ./update_uotd.php
   2           ?     ... unlocked  yes       update_stats -update_users
   3 2011/10/22     ... unlocked  no        run_in_ops ./notify.php
running  ssh vg-challenge-02 cd ~/projects/tah  &&  ./bin/status -v
BOINC is ENABLED

DAEMON  pid  status   lockfile ... commandline
   1   3517  running  locked   ... sample_trivial_validator --app tah
   2   3519  running  locked   ... sample_assimilator --app tah
   3   3521  running  locked   ... transitioner -d 3

boincadm@vg-challenge-01:~/projects/tah$
```

Teil III
BOINC

Kapitel 6
Serveradministration

Nichts auf der Welt ist so gerecht verteilt wie der Verstand.
Denn jedermann ist überzeugt, dass er genug davon habe.

René Descartes

Viele administrative Tätigkeiten müssen, sobald ein BOINC-Projekt gestartet ist, nicht mehr durchgeführt werden. Es kommt vor, dass Sie eventuell einen zusätzlichen Rechner zu Ihrem BOINC-Projekt hinzufügen, um eine Lastverteilung zu ermöglichen. Wenn Fehler aufkommen, so bin ich ein persönlicher Freund der Nähe zur Software. BOINC liefert zwar eine Schnittstelle, um Fehler zu durchsuchen, aber nichts ist effizienter, als die Dateien direkt auf einer UNIX/Linux-Maschine zu verarbeiten. Dieses Kapitel bevorzugt Terminalanwendungen, aber gibt auch einen Einblick in die Webadministration.

6.1 Status der Serverkomponenten ermitteln und deuten

Das Erfragen über den Status der auf der Serverseite installierten und ausgeführten bzw. nicht ausgeführten BOINC-Dämonen und Tasks ist relativ trivial. BOINC selbst liefert ein Werkzeug, um sich diese Informationen in einem einfach zu lesenden Format abzufragen. Wie bereits in Abschn. 5.8 gezeigt, kann der Status der BOINC-Infrastruktur mit `bin/status` angezeigt werden. Das Vorhandensein von PID-Dateien im Unterordner `pid_HOSTNAME` ist kein Indiz dafür, dass die Dämonen ordentlich gestartet sind. Beim Aufruf von `bin/start` wird zudem eine Lock-Datei erstellt; diese Datei dient dazu, dass der Dämon nicht nochmal gestartet wird, wenn `bin/start` nochmalig ausgeführt wird. Für Tasks wird nach jedem Aufruf von `bin/start` die Datei `run_state_HOSTNAME.xml` aktualisiert und ein Zeitwert vom letzten Startzeitpunkt hinzugefügt.

6.2 Informationen über die wissenschaftlichen Applikationen

Mit dem Werkzeug `bin/appmgr` können Informationen über die Plattformen und wissenschaftlichen Applikationen angezeigt werden. In Abschnitt 5.6.2.1 habe ich dieses Werkzeug schon vorgestellt. Das Werkzeug bietet folgende Möglichkeiten:

C. B. Ries, *BOINC*, Xpert.press,
DOI 10.1007/978-3-642-23382-8_6,

- Das Hinzufügen einer neuen wissenschaftlichen Applikation, sei es eine neue Version oder eine im Ganzen bisher unbekannte und initialisierte Bekanntgabe. Weiterhin können Plattformdefinitionen hinzugefügt werden. Eine Liste der Standardplattformen ist in `bin/appmgr` hinterlegt und kann initialisiert werden, wenn die `project.xml` nicht alle von BOINC unterstützen Plattformen beinhaltet. Im Anh. 15.3.1 finden Sie die Standardplattformen.
- Die schon vorhandenen Informationen können einzeln gelöscht oder aktualisiert werden.
- Vorhandene Plattforminformationen und die unterschiedlichen Versionen einer wissenschaftlichen Applikation können angezeigt werden.

6.3 Administrationswebseite

Die Administrationswebseite kann durch unterschiedliche Mechanismen vor dem Zugriff unautorisierter Benutzer geschützt werden. In Abschn. 5.6.1.4 haben Sie schon die erste Möglichkeit kennengelernt, durch die Datei `.htaccess` den Zugriff zu regeln. Weitere Zugriffsregelungen sind möglich und werden in der Datei `html/project/project.inc` innerhalb der Funktion `auth_ops_example()` vorgenommen.

Zwei Prüfungen sind schon implementiert; es handelt sich dabei um die Möglichkeit, einzelnen Teilnehmern den Zugriff zu erlauben, welche über deren Identifikator kontrolliert werden, oder um die Privilegien der einzelnen Teilnehmer. Die Privilegien als Entwickler oder Administrator erlauben den Zugriff.

Weitere Zugriffsbeschränkungen sind denkbar, z. B. eine Anbindung an die Benutzerverwaltung einer Institution mit Hilfe von Lightweight Directory Access Protocol (LDAP) [58] oder der generelle Zugriff von einer bestimmten IP-Adresse oder einem Bereich von unterschiedlichen IP-Adressbereichen.

Wir werden nicht alle Bereiche vorstellen, weil keine großen Tricks nötig und Raffinessen möglich sind. Ein einfaches eigenständiges Durchklicken durch alle Bereiche ist der beste Weg, um die Administrationsseite kennenzulernen.

6.3.1 Bereiche der Administrationswebseite

Wenn Sie die Startseite für die Administration, wie in Abb. 6.1 zu sehen, eines BOINC-Projekts aufrufen, so haben Sie die Wahl zwischen vier Kategorien, und jede Administrationsmöglichkeit kann als einzelnes Modul gesehen werden:

Datenbank durchstöbern (engl. *browse database*) In dieser Kategorie können Sie Informationen über Arbeitspakete, deren Berechnungsergebnisse, die registrierten Hosts und Benutzer, wissenschaftliche Applikationen und deren Versionen, die unterstützten Plattformen und den Speicherverbrauch einsehen.

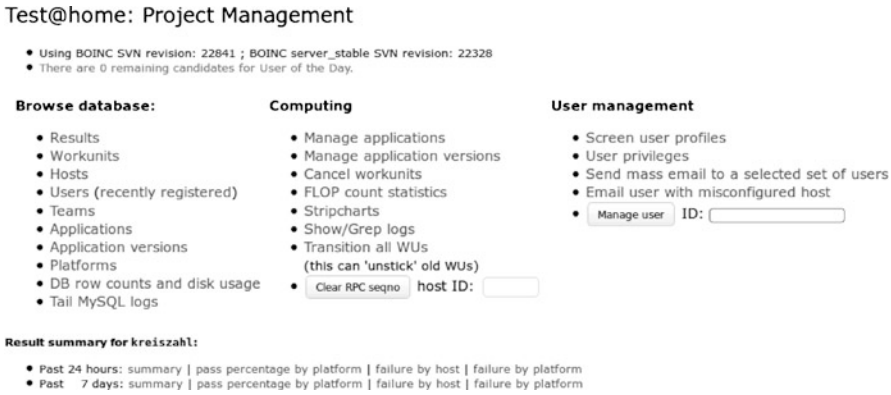


Abb. 6.1 Hauptansicht der Administrationswebseite eines BOINC-Projektes

Berechnungen (engl. *computing*) Wenn Sie wünschen, Arbeitspakete abzubrechen, auch wenn diese schon verteilt sind, so geschieht das in dieser Kategorie. Mehr noch, Sie können wissenschaftliche Applikationen deaktivieren und wieder aktivieren, Sie können erfragen, wie viel Fließkommaoperationen durch die Teilnehmer zur Verfügung gestellt werden – auf Wunsch mit der Einschränkung einer bestimmten Betriebssystemumgebung – und mit Hilfe von regulären Ausdrücken können Sie ausgewählte Informationen abfragen (dazu mehr in Abschn. 6.3.2).

Benutzerverwaltung (engl. *user management*) Hier kann man sich die Benutzerprofile alle Teilnehmer anschauen. In einem weiteren Menüpunkt kann man die Zugriffsrechte einzelner Teilnehmer modifizieren, folgende Zugriffsbeschränkungen bzw. Privilegien können eingestellt werden:

- Ehrenamtlicher Entwickler (engl. *volunteer developer*)
- Ehrenamtlicher Moderator (engl. *volunteer moderator*)
- Ehrenamtlicher Tester (engl. *volunteer tester*)
- Projekttester (engl. *project tester*)
- Projektentwickler (engl. *project developer*)
- Projektwissenschaftler (engl *project scientist*)
- Projektadministrator (engl. *project administrator*).

Standardmäßig sind keine Zugriffsrechte gesetzt und der jeweilige Teilnehmer kann „nur“ Arbeitspakete beziehen und bearbeiten, aber eben das ist die Hauptaufgabe der Teilnehmer.

Ergebniszusammenfassung (engl. *result summary*) Im unteren Bereich der Administrationsseite werden Statistiken – wahlweise – der letzten Stunde, letzten zwölf oder 24 Stunden, einer Woche oder eines Monats angezeigt und können jeweils für eine Gesamtzusammenfassung, einen Prozentwert der erfolgreichen Bearbeitungen auf den jeweiligen Plattformen, Fehler der Hosts und Plattformen ausgewählt werden. Wir können noch einzelne Teilnehmer per e-Mail anschreiben und auf Fehler hinweisen oder eine Massennachricht versenden.

6.3.2 Informationen mit regulären Ausdrücken abfragen

Die mittlere Kategorie *„Computing“* zeigt den Unterpunkt *„Show/Grep logs“*, den wir hier erläutern wollen. Hinter diesem Menüpunkt befindet sich eine einfache HTML-Seite für die Eingabe eines regulären Ausdrucks zur Abfrage von Daten aus allen Dateien, welche sich in `log_HOSTNAME` eines BOINC-Projekts befinden. Weiterhin können Sie näher spezifizieren, aus welchen Dateien Sie die Informationen beziehen wollen und wie viele Ergebniszeilen angezeigt werden sollen.

Im Hintergrund der Abfrage wird das `bin/grep_logs` Perl-Werkzeug ausgeführt. Man kann das Werkzeug auch manuell ausführen und dabei bestimmen, ob das Ausgabeformat in HTML sein soll oder nicht. Das Format eines solchen Aufrufs kann wie folgt aussehen:

```
boincadm@boinc-testserver:~/projects/tah$ ./bin/grep_logs -l 10 \
 'work' log_boinc-testserver/feeder.log
#
# Oder im HTML-Format:
#
boincadm@boinc-testserver:~/projects/tah$ ./bin/grep_logs -html \
  -l 10 \
  'work' log_boinc-testserver/feeder.log
```

Für die Webseite wird der Aufruf mit HTML-Ausgabe gewählt, um daraufhin in Ihrem Browser angezeigt werden zu können, der passende Parameter ist `-html`. `-l` (kleines L) gibt die Anzahl der auszugebenden Ergebnislinien an, abhängig davon, ob ein negativer *(Rückwärtssuche vom Dateiende)* oder positiver *(Vorwärtssuche vom Dateianfang)* Wert übergeben wird, verhält sich die Suche. Der Suchbegriff kann als regulärer Suchbegriff, wie von Perl unterstützt, angegeben werden und muss bei manueller Angabe durch Hochkommata eingeschlossen werden:

```
boincadm@boinc-testserver:~/projects/tah/log_boinc-testserver$ \
   ../bin/grep_logs -l 10 '^2.' sample_assimilator.log

sample_assimilator.log:
  00000: 2010-12-23 19:42:30.7027  Starting
  00001: 2010-12-23 20:58:05.3233  Assimilated 1 workunits.
  00002: 2010-12-23 20:58:55.4152  Assimilated 4 workunits.
  00003: 2010-12-23 20:59:05.4328  Assimilated 1 workunits.
  00004: 2010-12-23 20:59:55.5095  Assimilated 2 workunits.
  00005: 2010-12-23 21:00:15.5781  Assimilated 2 workunits.
  00007: 2010-12-23 23:05:01.6831  Starting
```

`bin/grep_logs` parst eine komplette Zeile in einer Datei und versucht Übereinstimmungen zu ermitteln. Nicht alle Perl-Ausdrücke für die Beschreibung von regulären Ausdrücken werden unterstützt. Die Prüfung auf Übereinstimmungen wird bestens unterstützt, aber nicht das Modifizieren von einzelnen Zeilen. Allerdings ist das auch der Wunsch, damit die Ergebnisse nicht verfälscht werden. Im nachfolgenden einige Beispiele und deren Ergebnisse:

```
#
## 1. Beispiel
#
boincadm@boinc-testserver:~/projects/tah/log_boinc-testserver$ \
      ../bin/grep_logs -l 10 '^\[20.' sample_assimilator.log
```

```
sample_assimilator.log:
  00000: [2010/12/23 19:42:30] Executing command: sample_assimilator -d 2 -app
       testapp
  00007: [2010/12/23 23:05:01] Executing command: sample_assimilator -d 2 -app
       testapp

#
## 2. Beispiel
#
boincadm@boinc-testserver:~/projects/tah/log_boinc-testserver$ \
      ../bin/grep_logs -l 10 \
      'Executing(.*?)testapp' sample_assimilator.log

sample_assimilator.log:
  00000: [2010/12/23 19:42:30] Executing command: sample_assimilator -d 2 -app
       testapp
  00007: [2010/12/23 23:05:01] Executing command: sample_assimilator -d 2 -app
       testapp
```

Das Ergebnis ist in beiden Fällen gleich, allerdings durch unterschiedliche Suchausdrücke ermittelt. Ein guter Einstieg in die Arbeit mit regulären Ausdrücken ist das Buch von J. Lee [37], welches auch online gelesen werden kann.

Kapitel 7
Programmiergrundlagen

„C macht es einfach, sich selbst ins Bein zu schießen; C++ erschwert es, aber wenn es dir gelingt, bläst es dir das ganze Bein weg."

Bjarne Stroustrup, Erfinder von C++

Dieses Kapitel macht Sie zu einem kompetenten Programmierer von BOINC, und Sie können schon in Kürze eine eigene wissenschaftliche Applikation erstellen und weltweit verteilen. Anfangs werden Ihnen die allgemeinen Rahmenbedingungen nähergebracht, an die sich BOINC-Entwickler halten sollten. In den nachfolgenden Abschnitten geht es ans Eingemachte. Sie erfahren die elementarsten technischen Hintergründe für das Verwenden von BOINC und die zur Verfügung gestellten Schnittstellenfunktionen sowie grundlegende vom BOINC-Entwicklerteam getroffene Implementierungsentscheidungen innerhalb des BOINC-Frameworks. Besonderes Augenmerk wird darauf gelegt, dass Sie alle benötigten Kenntnisse vermittelt bekommen, so dass Sie ein solides Grundwissen für die eigene Entwicklung einer wissenschaftlichen Applikation erhalten und schon während der Entwicklungsphase wissen, auf welche Kleinigkeiten Sie achten müssen! Quelltextausschnitte runden dieses Kapitel ab, in denen Sie die Verwendung der einzelnen Funktionen an einem Beispiel nachvollziehen können.

7.1 Voraussetzungen

Das Kapitel enthält einige Beispiele und C/C++-Quelltextfragmente, welche Sie direkt in eigene Entwicklungen übernehmen können. Es wird empfohlen, die BOINC-Quellen für das Lesen und Studieren zur Verfügung zu haben. Dadurch erhalten Sie die Möglichkeit, die interne BOINC-Struktur nachvollziehen zu können. Es werden die Strukturen der Interna diskutiert, doch ist es manchmal von Vorteil, sich ein eigenes Bild von einem Sachverhalt zu machen. Die in diesem Kapitel vorgestellten Quelltextfragmente können nur mit den Originalquellen von BOINC kompiliert werden. Das Kap. 5 behandelt die Installation eines Ubuntu-Servers mit der Installation eines eigenen BOINC-Projekts. Darauf aufbauend können Sie das in diesem Kapitel vermittelte Wissen direkt anwenden und können sich so ein voll funktionsfähiges BOINC-Projekt erstellen.

C. B. Ries, *BOINC*, Xpert.press,
DOI 10.1007/978-3-642-23382-8_7,

7.2 Allgemeine Terminologien und Qualitätssicherung

Bei der Verwendung des BOINC-Frameworks muss auf Definitionen und spezielle Terminologien geachtet werden. Sie als Entwickler sind nicht dazu gezwungen, sich an diese Definitionen und Terminologien zu halten, allerdings ist es von Vorteil, sich diese „Umgangsformen“ anzueignen. Es steigert die Geschwindigkeit beim Lesen und Verstehen der internen BOINC-Struktur. Weiterhin ermöglicht Ihnen dieses Wissen, eventuell vorhandene persönliche Wünsche an das BOINC-Framework selber zu implementieren, die in Folge dessen auch schnell von der BOINC-Entwicklergemeinde gelesen und verstanden werden können. So können Sie Teil der BOINC-Entwicklergemeinde werden und helfen, das BOINC-Framework um Features zu erweitern oder Fehler auszubessern.

7.2.1 Rückgabewerte im BOINC-Framework

Wenn Funktionen im BOINC-Framework einen Rückgabewert liefern, dann beschreibt dieser, ob die Ausführung der Funktion erfolgreich war, oder was in etwa schieflief. Der Rückgabewert liefert Null, wenn die Ausführung erfolgreich war. Jeder Wert, der ungleich Null ist, bedeutet einen Fehler, und weitere Prüfungen sollten sich idealerweise an die Ausführung anschließen. Eine erfolgreiche Ausführung wird mit

Listing 7.1 Rückgabewert für eine erfolgreiche Abarbeitung einer Funktion im BOINC-Framework

```
#include <lib/error_numbers.h>

#define BOINC_SUCCESS          0
```

beschrieben. Weiterhin definiert das BOINC-Framework weitere 135 Fehlernummern für BOINC-Funktionen. Die Fehlernummern haben einen Wert von -100 bis hinunter auf -235 und sind in der Datei `lib/error_numbers.h` definiert. Der Anhang 15.1 beinhaltet eine Auflistung aller Fehlernummern mit den entsprechenden Fehlermeldungen. Durch die Definitionen der Fehlernummern ist eine breite Palette von möglichen unterschiedlichen Fehlern überprüfbar. Die Fehlernummern decken das Abfragen von Fehlern bei Dateisystem-/Systemproblemen, bei der Speicherallozierung, bei RSA-Verschlüsselungen, Kommunikationsabläufen, Zugriffsrechten, Datenbankabfragen und noch einige weitere Fälle ab. Sollten die vorhandenen Fehlernummern nicht ausreichen, so können Sie sich eigene Fehlernummern hinzufügen. Diese können einfach nach unten erweitert werden, d. h. die Fehlernummern nehmen mit jeder weiteren neuen Definition ab. Zudem ist es sinnvoll und von den BOINC-Hauptentwicklern gewünscht, dass eine Beschreibung für jede neue Fehlernummer hinzugefügt wird. Diese Änderung wird in der Datei `lib/str_util.cpp` vorgenommen. Die Funktion

Listing 7.2 Funktion für die Ermittlung einer Fehlerbeschreibung

```
#include <lib/str_util.h>

const char* boincerror(int which_error);
```

liefert eine Beschreibung für jede der definierten Fehlernummern. Die Fehlerbeschreibung sollte kurz und aussagekräftig sein. Zum Beispiel werden Fehler beim Öffnen oder Lesen einer Datei nur durch die Fehlerbeschreibung

```
ERR_READ: read() failed
ERR_OPEN: open() failed
```

beschrieben. Diese Beschreibungen reichen aus, da es sich bei diesen Funktionen um Systemfunktionen handelt, so dass weitere Informationen direkt durch die C-Bibliotheksfunktionen

Listing 7.3 C-Bibliotheksfunktionen für die Fehlerbeschreibung

```
#include <stdio.h>
#include <errno.h>

void perror(const char *s);
const char *sys_errlist[];
int sys_nerr;
int errno;
```

ausgelesen werden können.

7.2.2 Namensgebung von Funktionen und Konstanten

Die hauptsächlichen BOINC-API-Funktionen besitzen das Präfix `boinc_`, wodurch sie direkt zu erkennen sind. Variablen der Klasse `std::string` werden immer als Referenz an einen Funktionsaufruf übergeben. Definitionen von Strukturen besitzen fast ausschließlich einen komplett in GROß geschriebenen Strukturnamen, beispielsweise:

Listing 7.4 Beispiele der im BOINC-Framework vorhandenen Strukturnamen

```
#include <api/boinc_api.h>

typedef struct BOINC_OPTIONS { /* ... */ }
typedef struct BOINC_STATUS { /* ... */ }


#include <lib/filesys.h>

struct FILE_LOCK { /* ... */ }


#include <lib/coproc.h>

struct COPROC_REQ { /* ... */ }
struct COPROC { /* ... */ }
struct CUDA_DEVICE_PROP { /* ... */ }
struct COPROC_CUDA : public COPROC { /* ... */ }
struct COPROC_ATI : public COPROC { /* ... */ }
struct COPROCS { /* ... */ }
```

7.3 Programmierung einer wissenschaftlichen Applikation

Dieser Abschnitt beschreibt große Teile des BOINC-API-Frameworks und welche Funktionalitäten hinter den einzelnen Funktionen stecken. Sie erfahren einiges über die interne Implementierung und wie diese API von Ihnen verwendet werden kann. Nach diesem Abschnitt sind Sie bereit für die Entwicklung einer eigenen wissenschaftlichen Applikation und können selber ein *Public-Resource-Computing*- beziehungsweise *Volunteer-Computing*-Projekt erstellen und hunderte von Freiwilligen für sich arbeiten lassen.

7.3.1 Programmierrahmen

Listing 7.5 enthält einen allgemein gehaltenen Rahmen einer wissenschaftlichen Applikation, welchen Sie nur durch Ihre eigenen Implementierungsdetails erweitern müssen. Dieser Rahmen deckt folgende Funktionalitäten ab, kann als Start für Ihre wissenschaftliche Applikation dienen und muss in dieser Version nur um den Arbeitskern erweitert werden:

Zeile 1 bis 17 Standardmäßig befinden sich im oberen Teil eines C/C++-Programms alle Header-Dateien, hier müssen Sie diese nur um gewünschte und von später genutzten Funktionen benötigte erweitern.

Zeilen 17–29 Eine wissenschaftliche Applikation kann aus mehr als einer Eingabedatei bestehen und auch mehr als eine Ergebnisdatei erstellen. In diesen Zeilen geben Sie die virtuellen Dateinamen (vgl. Abschn. 7.3.5) in einem Array an. Die Anzahl der Dateinamen können Sie mit der Funktion `sizeOf` ermitteln, dafür müssen die Arrays mit `NULL` aufhören. Das Einlesen der Eingabedateien muss in der Zeile 62 implementiert werden und hängt von der Form Ihrer Eingabedateien ab.

Zeilen 31–51 Um einen Checkpoint (vgl. Abschn. 7.3.11) zu erstellen, müssen Sie die Struktur `Checkpoint` um die zu Ihrer wissenschaftlichen Applikation passenden Attribute erweitern. Das Erstellen eines Checkpoints soll mit doCheckpoint möglich sein, das Auslesen kann in den Zeilen 69ff realisiert werden.

Zeilen 54–83 Hier werden Kernberechnungen und Aktionen einer wissenschaftlichen Applikation hinzugefügt. An dieser Stelle sind Ihren Wünschen keine Grenzen gesetzt; Sie können eine einfache Berechnung durchführen, wahlweise auf der CPU oder GPU (vgl. Abschn. 7.4, 7.5) oder Netzwerkbenchmarks umsetzen, um die Verbindungen zu verschiedenen Internetadressen durch Pings zu prüfen – vieles mehr ist denkbar. Es sollte der Fortschritt der Berechnungen ermittelt werden (vgl. Abschn. 7.3.9), ein Checkpoint durchgeführt werden und nach Möglichkeit Zwischenergebnisse sollten abgespeichert werden, so dass diese eventuell während der Abarbeitung schon zum BOINC-Projekt übermittelt werden können. Aufgerufen wird diese Funktion innerhalb der `main`-Funktion in Zeile 97.

Zeilen 86–102 Diese Zeilen beinhalten die `main`-Funktion, mit der Initialisierung und ordentlichen Beendigung einer wissenschaftlichen Applikation.

Listing 7.5 Rahmen einer wissenschaftlichen Applikation

```
// C++
#include <iostream>
#include <string>

// C
#include <stdio.h>
#include <errno.h>

// Third-party
// ...

// BOINC
#include <filesys.h>
#include <boinc_api.h>
#include <error_numbers.h>

const char *inputFilename[24] = {
  "in01", "in02", NULL
};

const char *outputFilename[24] = {
  "out01", NULL
};

inline unsigned int sizeOf(const char **field) {
  unsigned int c = 0;
  while(field[++c] != NULL) ;
  return c;
}

#define CHECKPOINT_FILE "checkpoint.xml"

struct Checkpoint {
  // INSERT: data fields
};

/// Create checkpoint
void doCheckpoint(struct Checkpoint *chk) {
  if(boinc_time_to_checkpoint()) {
    std::string filepath;
    boinc_resolve_filename_s(CHECKPOINT_FILE, filepath);
    FILE *fChk = boinc_fopen(filepath.c_str(), "w+");
    if(fChk) {
      fprintf(fChk, "<checkpoint time=\"%d\">\n", time(NULL));
      // INSERT: more checkpointing
      fprintf(fChk, "</checkpoint>\n");
      fclose(fChk);
    }
    boinc_checkpoint_completed();
  }
}

/// Worker
int doWork() {
  bool working = true;

  unsigned int inCount = sizeOf(inputFilename);
  FILE **in = new FILE*[inCount];
  unsigned int outCount = sizeOf(outputFilename);
  FILE **out = new FILE*[outCount];

  // INSERT: read-out input files
```

```
  // for all files in inputFilename:
  //   std::string filepath;
  //   boinc_resolve_filename_s(inputFilename[0], filepath);
66 //   in[0] = boinc_fopen(filepath.c_str(), "r");
  //   ...

  struct Checkpoint chk;
  // INSERT: read-out checkpoint
71
  do {
    // INSERT: work
    // INSERT: dump output
    // INSERT: fraction-done calculation
76
    working = false;

    doCheckpoint(&chk);
  } while(working);
81
  return BOINC_SUCCESS;
}

/// Main-routine
86 int main(int argc, char **argv) {
  int returnValue = boinc_init();
  if (returnValue) {
    char buf[1024] = {'\0'};
    std::cerr << boinc_msg_prefix(buf)
91            << " boinc_init returned "
              << returnValue << std::endl;
    exit(returnValue);
  }

96 /// Start of your work!
  doWork();
  /// End of your work!

  boinc_fraction_done(1);
101 boinc_finish(0);
}
```

7.3.2 Wissenschaftliche Applikationen initialisieren

Bei vielen modernen Programmier-Frameworks bedarf es einer Initialisierung [30, 131]. Auch jede mit dem BOINC-Framework zu programmierende wissenschaftliche Applikation muss anfangs initialisiert werden. Die Initialisierung übernimmt folgende Aufgaben:

- Setzen der Einstellungen für die Diagnose der Ausführung einer Applikation. Diagnose-Optionen können bei der Fehlersuche oder Prüfung der Berechnungsdurchführung hilfreich sein. Welche Diagnoseoptionen es gibt, wie diese gesetzt werden können und wie mit diesen gearbeitet werden kann, wird in Abschn. 10.1 behandelt.
- Setzt entweder Standardoptionen oder spezifische Optionen für das Laufzeitverhalten einer wissenschaftlichen Applikation.

- Prüft, ob die wissenschaftliche Applikation *standalone* – z. B. zum Testen einer Applikation – oder in Kombination mit dem BOINC-Client gestartet wurde und setzt dementsprechend Flags. Diese Flags sind für spätere Funktionsaufrufe wichtig, die nur dann ausgeführt werden können, wenn eine Kommunikation mit dem BOINC-Client möglich ist. Es ist eher unnötig zu versuchen, den aktuellen Berechnungsstatus zu versenden, oder zu prüfen, ob der BOINC-Client noch erreichbar ist, wenn dieser von vornherein nicht vorhanden ist.
- Es wird ein Prozess (engl. *thread*) erstellt, der in einem Intervall von einer Sekunde stetig Nachrichten verarbeitet, die zwischen einer wissenschaftlichen Applikation und dem BOINC-Client ausgetauscht werden. In diesem Prozess werden Aktionen ausgeführt, die über das Setzen der Optionen in der Struktur `BOINC_OPTIONS` angepasst werden können (s. Abschn. 7.3.2).
- Linux: Die Initialisierung mit `boinc_init_parallel()` kopiert den kompletten Prozess mit der Funktion `fork()`[1]. Abbildung 7.1 zeigt die Richtung der Kommunikation vom Vater- zum Kindprozess. Die in Abschn. 7.3.2 beschriebenen Steuerungsbefehle werden als Signale zum Kindprozess gesendet und können dort durch eigene implementierte Signalbehandlungsfunktionen verarbeitet werden. Abbildung 7.2 verdeutlicht die Verarbeitung der Steuerungsbefehle, die vom BOINC-Client an eine wissenschaftliche Applikation gesendet werden und dort in Signale für die arbeitende Applikation umgewandelt werden. Abschnitt 7.3.2 erläutert die eingehenden Steuerungsbefehle.
- Windows: Die Initialisierung mit `boinc_init_parallel()` verhält sich wie bei den beiden anderen Initialisierungen.

Das BOINC-Framework liefert die drei Funktionen

Listing 7.6 Funktionen für die BOINC-Framework-Initialisierung

```
#include <api/boinc_api.h>

int boinc_init(void);
  /* On successful completion it returns 0.
   * Otherwise, -1 is returned.
   */

int boinc_init_options(BOINC_OPTIONS*);
  /* On successful completion it returns 0.
   * Otherwise, -1 is returned.
   */

int boinc_init_parallel();
  /* On successful completion it returns 0.
   * Otherwise, -1 is returned.
   */
```

durch die die Initialisierung ausgeführt werden kann. In den meisten Fällen reicht die erste Initialisierungsfunktion aus.

[1] Erstellt eine tiefe Kopie des aktuell ausgeführten Prozesses und ist daher sehr teuer. Daher sollte diese Funktion nicht zu häufig verwendet werden. Weitere Informationen sind der Unix/Linux-Manpage zu entnehmen.

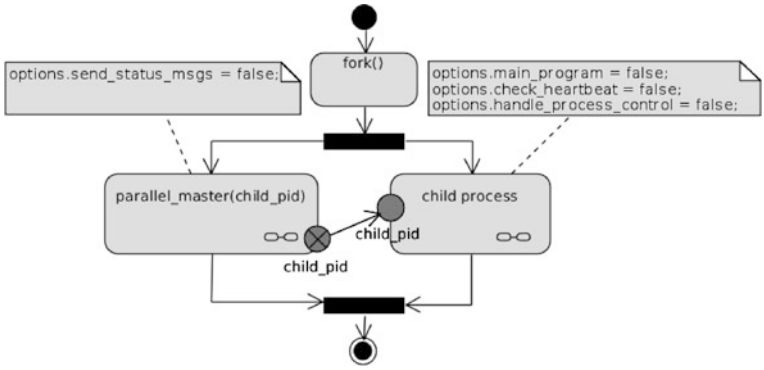


Abb. 7.1 Ablaufdiagramm einer Steuerungsapplikation auf der *linken Seite* und einer Arbeitsapplikation auf der *rechten Seite*

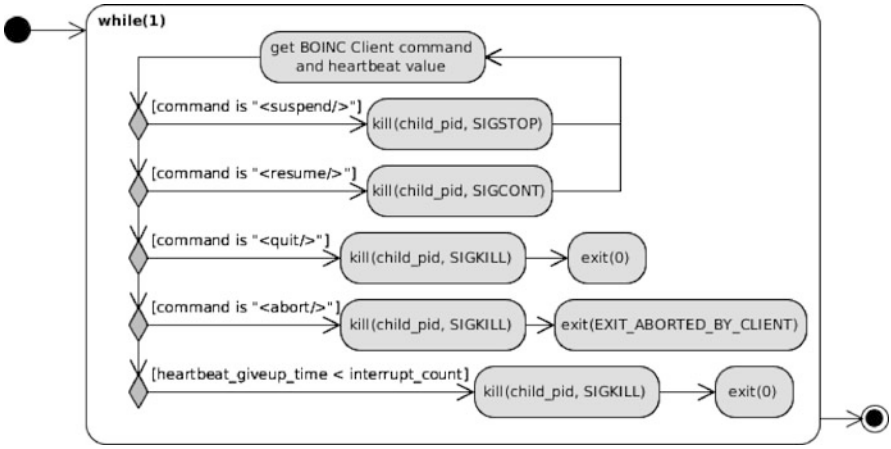


Abb. 7.2 Abfertigung der Befehle, die vom BOINC-Client an eine wissenschaftliche Applikation gesendet und in Signale für die arbeitenden Applikationen umgewandelt werden

Laufzeitverhalten nach eigenen Wünschen spezifizieren

Im vorherigen Kapitel wurde beschrieben, dass die Art der Ausführung einer BOINC-Applikation spezifiziert werden kann. Die Struktur

Listing 7.7 Struktur für die Optionen zum Spezifizieren des Laufzeitverhaltens einer wissenschaftlichen Applikation

```
#include <api/boinc_api.h>

typedef struct BOINC_OPTIONS {
    int backwards_compatible_graphics;
    int normal_thread_priority;
    int main_program;
    int check_heartbeat;
    int handle_trickle_ups;
```

```
    int handle_trickle_downs;
    int handle_process_control;
    int send_status_msgs;
    int direct_process_action;
} BOINC_OPTIONS;
```

beschreibt eine Liste von Optionen, die das Laufzeitverhalten einer wissenschaftlichen Applikation beeinflussen. Standardmäßig sind bis auf `direct_process_action` alle Optionen aktiviert. Sollten die Optionen aktiviert (`true` oder `1`) beziehungsweise deaktiviert (`false` oder `0`) sein, hat dies folgende Konsequenzen:

`backwards_compatible_graphics` Dieser Wert muss explizit deaktiviert werden, wenn Grafikfunktionen verwendet werden sollen, die nur bis zur BOINC-Revision `13697` in den BOINC-Quellen vorhanden waren. Die darauf folgende Revision erhielt eine neuere Schnittstelle für die Programmierung von grafischen Elementen. Es wird empfohlen, nur Funktionen der neueren Grafikschnittstelle zu verwenden. Kapitel 8 beschreibt die Funktionen und die Verwendung der neueren Grafikschnittstelle. Aus Gründen der Abwärtskompatibilität sind die älteren Funktionen weiterhin vorhanden; sie werden in diesem Buch nicht behandelt.

`normal_thread_priority` Änderungen an dieser Option beeinflussen das Laufzeitverhalten nur auf Windows-Systemen. Das BOINC-Framework nutzt die Windows-Funktion `Set ThreadPriority`, um die Priorität einzelner Threads zu setzen [148]. Während der Laufzeit erstellte Threads werden auf die Priorität `THREAD_PRIORITY_IDLE` gesetzt. Die Priorität aktiviert die Optionen

- `IDLE_PRIORITY_CLASS`,
- `BELOW_NORMAL_PRIORITY_CLASS`,
- `NORMAL_PRIORITY_CLASS`,
- `ABOVE_NORMAL_PRIORITY_CLASS`,
- `HIGH_PRIORITY_CLASS` und
- den Wert `16` für `REALTIME_PRIORITY_CLASS`.

für einen Prozess.

`main_program` Eine wissenschaftliche Applikation kann aus mehreren kleineren Applikationen bestehen. Dies kann sinnvoll sein, wenn jede Applikation für sich eine eigene Aufgabe darstellt und nur alle zusammen ein Ergebnis erarbeiten können. So eine Struktur wird Werkzeugkette (engl. *tool chain*) genannt. Die Elemente einer Werkzeugkette können parallel oder sequenziell ausgeführt werden. Beide Ausführungsarten erfordern einen Prozess, der den Überblick über die auszuführenden Applikationen hat und eine kontrollierte Steuerung ermöglicht. Die Steuerungsapplikation stellt hierbei die Hauptanwendung (engl. *main program*) dar und lässt die arbeitenden Applikationen (engl. *worker programs*) für sich tätig werden. So ein Aufbau von Haupt- und Arbeitsprogramm wird auch *compound application* genannt.

`check_heartbeat` Eine wissenschaftliche Applikation wird nur ausgeführt, wenn der BOINC-Client gestartet ist. Um zu prüfen, ob der BOINC-Client nicht

vorzeitig beendet wurde, zum Beispiel durch einen Programmabsturz, wird zwischen einer wissenschaftlichen Applikation und dem BOINC-Client das Ping-Pong-Verfahren angewendet. Dabei tauschen zwei Parteien Lebenszeichen untereinander aus, im Fall von BOINC sind dies Herzschläge (engl. *heartbeats*). Wenn diese Option aktiviert ist, werden diese Lebenszeichen geprüft. Beide Parteien besitzen ein eigenes Verfahren, falls 30 Sekunden lang kein Lebenszeichen eingetroffen ist. Falls die wissenschaftliche Applikation nach der Zeitspanne kein Lebenszeichen vom BOINC-Client empfängt, wird nach Prüfung von `direct_process_action` entweder die wissenschaftliche Applikation beendet oder das Flag `no_heart beat` in der Struktur `BOINC_STATUS` gesetzt. Sollte der BOINC-Client kein Lebenszeichen von einer wissenschaftlichen Applikation erhalten, versucht er den Slot aufzuräumen, in dem die wissenschaftliche Applikation ausgeführt wurde, sendet eine Fehlermeldung an das BOINC-Projekt und startet eine weitere Berechnung eines Arbeitspakets.

`handle_trickle_ups` Abschnitt 3.2 erläutert Trickle-Nachrichten. Der BOINC-Client bearbeitet von der wissenschaftlichen Applikation zu sendende Trickle-Nachrichten, wenn diese Option aktiviert ist.

`handle_trickle_downs` Bei aktivierter Option werden vom BOINC-Projekt zu sendende Trickle-Nachrichten verarbeitet und an die spezifizierten Empfänger gesendet. Es reicht nicht aus, diese Option zu aktivieren, um das Versenden zu aktivieren, weiterhin sind Konfigurationsschritte innerhalb eines BOINC-Projekts nötig. Abschnitt 9.4 gibt einen Überblick über die Schritte zur Aktivierung der Trickle-Messages.

`handle_process_control` Wenn dieser Wert aktiviert ist, wird periodisch geprüft, ob vom BOINC-Client gesendete Steuerungsbefehle vorliegen. Diese Steuerungsbefehle beeinflussen das Laufzeitverhalten einer wissenschaftlichen Applikation. Die Steuerungsbefehle basieren auf einer XML-Definition und können folgende XML-Tags enthalten:

- `<suspend/>`,
- `<resume/>`,
- `<quit/>`,
- `<abort/>`,
- `<reread_app_info/>` und
- `<network_available/>`

Das Verhalten für die ersten vier ist von der Einstellung `direct_process_action` abhängig. Sollte diese Option aktiviert sein, übernimmt das BOINC-Framework das Grundverhalten für die jeweiligen Steuerungsbefehle und versucht die Applikation entweder zu pausieren (engl. *suspend*), weiterzuführen (engl. *resume*), zu beenden (engl. *quit*) oder abzubrechen (engl. *abort*). Weiterhin wird beim Empfang der Steuerungsbefehle die dazu gehörende Statusvariable in der Struktur `BOINC_OPTIONS` gesetzt, die vom Entwickler einer wissenschaftlichen Applikation abgefragt werden kann. Selbst wenn das BOINC-Framework im ersten Schritt die Prozesskontrolle einer wissenschaftlichen Applikation übernimmt, kann der Entwickler doch noch eine weitere Steuerungsinstanz implementieren. Die letzten beiden Steuerungsbefehle weisen die wissenschaftliche

Applikation erstens an, die Projektkonfiguration neu einzulesen, und zweitens wird darüber informiert, dass eine Netzwerkverbindung vorhanden ist.

`send_status_msgs` Bei aktivierter Einstellung wird der aktuelle Status der Berechnung an den BOINC-Client gesendet. Der Status kann im Folgenden vom BOINC-Manager angezeigt werden. Eine deaktivierte Einstellung würde Sinn ergeben, wenn keine Informationen über aktuell durchgeführte Berechnungen benötigt werden und einfach nur gerechnet werden soll.

`direct_process_action` Bei einem positiven Wert werden vom BOINC-Client empfangene Steuerungsbefehle direkt verarbeitet. Sollte diese Option deaktiviert sein, dann ist der Entwickler für die Prozesssteuerung verantwortlich. Der aktuelle Status einer wissenschaftlichen Applikation, das heißt welche Steuerungsbefehle empfangen wurden, kann jederzeit mit der Funktion `boinc_get_status()` (s. Listing 7.9) erfragt werden. Wenn sich der Status ändert, kann darauf eine vom Entwickler spezifizierte Aktion ausgeführt werden, und dies ermöglicht die Entwicklung einer projektspezifischen Prozessverwaltung. Kapitel 13 und 14 behandeln Implementierungen von Wrapper-Umsetzungen für wissenschaftliche Applikationen, um Anwendungen von Drittanbietern (sog. Third-party-Applikationen) oder vorhandene, installierte Applikationen (sog. Legacy-Applikationen) durch diese Technik zu starten und steuerbar zu machen.

Der aktuelle Status einer wissenschaftlichen Applikation kann mit der Funktion

Listing 7.8 Funktionen für die Ermittlung des aktuellen Status einer wissenschaftlichen Applikation

```
#include <api/boinc_api.h>

int boinc_get_status(BOINC_STATUS *s);
  /* Returns 0 */
```

ausgelesen werden. Listing 7.9 zeigt ein Beispiel zum Auslesen. Es wird eine Variable der Struktur erstellt, die später den Status enthalten soll. Die Funktion `boinc_get_status()` gibt immer Null zurück. Eine Fehlerbehandlung ist in diesem Fall nicht nötig.

Listing 7.9 Beispiel zum Auslesen des aktuellen Status einer wissenschaftlichen Applikation

```
BOINC_STATUS currentStatus;
int res = boinc_get_status(&currentStatus);
```

7.3.3 Beenden einer wissenschaftlichen Applikation

Wenn ein Arbeitspaket bearbeitet ist, sollte das Ende der Berechnung mit der Funktion

Listing 7.10 Funktion zum Beenden einer wissenschaftlichen Applikation

```
#include <api/boinc_api.h>

int boinc_finish(int status);
```

Abb. 7.3 Die Datei `boinc_finish_called` beschreibt das Ende der Ausführung

beschrieben werden. Der in Abschn. 7.3.2 erläuterte Prozess, für das stetige Abfragen auf Steuerungsbefehle, wird beendet. Im Slot der Ausführung wird die Datei `boinc_finish_called` erstellt, daraufhin die gesperrte Datei `boinc_lockfile` entsperrt (s. Abb. 7.3, Abschn. 7.3.14 und Listing 7.46) und die Ausführung beendet.

7.3.4 Kommunikation zum BOINC-Client möglich?

Wenn eine neue wissenschaftliche Applikation entwickelt wird, kann man nicht für jedes neue Release oder jede kleine Änderung einen kompletten Test durchführen. Es reicht manchmal auch, nur eine leichtgewichtige Ausführung zu starten. Dies wird „standalone"-Betrieb genannt und trifft dann zu, wenn die wissenschaftliche Applikation direkt vom Entwickler gestartet wird und nicht durch den BOINC-Client. Die Funktionsaufrufe innerhalb vom BOINC-Framework werden durch interne Routinen auf standalone-Betrieb geprüft. Sollten vom Entwickler einer neuen wissenschaftlichen Applikation eigene Fallunterscheidungen gewünscht sein, so können diese mit der Funktion aus Listing 7.11 durchgeführt werden.

Listing 7.11 Funktion zum prüfen, ob sich die wissenschaftliche Applikation im standalone-Betrieb befindet

```
#include <api/boinc_api.h>

int boinc_is_standalone();
  /* Returns 1 if the process is in standalone mode.
   * Otherwise, 0 is returned.
   */
```

7.3.5 Slot-Mechanismus: Virtuelle Dateinamen

Jede wissenschaftliche Applikation wird in einem Slot ausgeführt (s. Abschn. 4.3.2). Nun kann ein BOINC-Projektverwalter Arbeitspakete mit einer, zwei oder mehr Eingabedateien erstellen. Jede einzelne der angegebenen Dateien kann dabei wenig oder viel Speicherplatz benötigen. Es ist von der Anzahl der zu erstellenden Arbeitspakete abhängig, ob dies auf dem BOINC-Projektserver zu Problemen führen kann, zum Beispiel, dass nicht genügend Speicherplatz vorhanden ist und dadurch in nächster Konsequenz keine ordentliche Abarbeitung mehr möglich ist. Bei den Teilnehmern eines BOINC-Projekts kann der Anwender entscheiden, wie viel Speicherplatz für ein angemeldetes BOINC-Projekt verwendet werden darf. Es kann daher vorkommen, dass entweder direkt oder irgendwann keine Berechnungen mehr durchgeführt werden, da die Einstellungen auf Seiten des BOINC-Projektteilnehmers dies nicht zulassen. Wenn zum Beispiel zu wenig Speicherplatz für ein BOINC-Projekt freigegeben wurde und die nachfolgenden Arbeitspakete diese Grenzen überschreiten würden, dann erlaubt das BOINC-Projekt keine Dateien herunterzuladen. Die nachfolgenden Funktionen beschreiben die einheitliche Schnittstelle für die Verwendung von Soft-Links. Soft-Links werden nativ nicht von allen gängigen Betriebssystemen unterstützt, wie zum Beispiel beim Betriebssystem Windows. Da diese Funktionalität allerdings viele Vorteile bietet, zum Beispiel Einsparen von Speicherplatz und Steigerung der Wartbarkeit, wurde durch die BOINC-Entwickler eine generelle Schnittstelle konzipiert. Mit den Funktionen

Listing 7.12 Funktionen, um den physikalischen Dateinamen einer Datei in einem Slot zu ermitteln

```
int boinc_resolve_filename(
    const char *virtual_name, char *physical_name, int len
)
  /* On successful completion it returns 0.
   * Otherwise, ERR_NULL (-116) is returned.
   */

int boinc_resolve_filename_s(
    const char *virtual_name, string& physical_name
)
  /* On successful completion it returns 0.
   * Otherwise, ERR_NULL (-116) is returned.
   */
```

können Pfade zu Dateien aufgelöst werden. Dabei wird geprüft, ob die zu öffnende Datei den XML-Tag `<soft_link>` enthält. Dabei ist zu beachten, dass dieser XML-Tag in der ersten Zeile stehen muss. Entwickler erstellen in der Regel keine Dateien mit enthaltenen Soft-Links. Das Rosetta@home-Projekt [162] verschickt zu jedem Arbeitspaket eine Datenbank mit, bei dieser könnte es eventuell sinnvoll sein Soft-Links zu verwenden, wenn auf Dateien in den Tiefen der Datenbankhierarchie zugegriffen werden soll. Abschnitt 4.3.2 beschreibt das Slot-Verfahren.

Die zwei Funktionen zum Auflösen eines virtuellen zu einem physikalischen Dateinamen arbeiten prinzipiell gleich. Im Unterschied zur ersten Funktion kann die zweite mit C++-Standardzeichenketten (also: `std::string`) umgehen, und es wird empfohlen, diese Funktion zu verwenden. Dadurch ist der Entwickler von

der Verantwortung befreit, sich zu jedem Zeitpunkt bewusst zu sein, wie viel Zeichen der Dateiname haben darf. Die Verwendung von `std::string` beugt Speicherzugriffsfehlern vor.

Listing 7.13 enthält ein einfaches Beispiel zum Öffnen von Dateien, die durch einen Soft-Link beschrieben werden. In dem Beispiel wird ein ZIP-Archiv als Parametereingabedatei erwartet. Der virtuelle Dateiname ist `INPUT_FILE`. `boinc_resolve_filename_s()` versucht diese Datei zu öffnen, prüft, ob ein Soft-Link vorhanden ist, und speichert in `inputArchivePath` entweder den aus dem Soft-Link ausgelesenen physikalischen Dateinamen oder kopiert den virtuellen Dateinamen, wenn es sich nicht um einen Soft-Link handelt. Vor jedem Öffnen einer Datei sollte eine Prüfung mit einer dieser zwei Funktionen durchgeführt werden. Danach kann ganz normal mit der Variablen `inputArchivePath` – in diesem Fall – gearbeitet werden und nach dem Öffnen der Datei können alle möglichen Dateioperationen ausgeführt werden.

Listing 7.13 Ein Beispiel für das Arbeiten mit virtuellen und physikalischen Dateinamen

```
...
std::string inputArchivePath;
struct zip *zipFileIn = NULL;
char zipError[512] = {'\0'};

boinc_resolve_filename_s(INPUT_FILE, inputArchivePath);
zipFileIn = zip_open(inputArchivePath.c_str(),
                     ZIP_CHECKCONS, NULL);
if (zipFileIn == NULL) {
  zip_error_to_str(zipError, 512, 0, errno);
  std::cerr << "zip failed: " << zipError << std::endl;
  boinc_finish(-1);
}
...
```

Es sei noch erwähnt, dass diese Funktionen prinzipiell nicht nur innerhalb des Slot-Systems verwendet werden können. Es wird innerhalb der Funktionen keine Prüfung durchgeführt, ob wir uns in einem Slot befinden. Jede Datei kann daher einen Soft-Link enthalten, der automatisch aufgelöst wird.

7.3.6 Dateioperationen

Das BOINC-Framework bietet dem Entwickler eine komfortable Schnittstelle um Dateizugriffe zu realisieren. Die Dateioperationen im BOINC-Framework stellen dabei eine Abstraktionsschicht zu den Systemfunktionen für Dateioperationen dar. Mit der zusätzlichen Abstraktionsschicht wird eine Cross-Plattform-Unterstützung für die von BOINC unterstützten Betriebssysteme[2] mitgeliefert, so dass in jedem Fall nur ein Funktionsaufruf notwendig ist. Dies erhöht die Wartbarkeit und Portierbarkeit von wissenschaftlichen Applikationen.

[2] Windows, Linux und Mac OS X.

Dateien erstellen, öffnen und löschen

Listing 7.14 zeigt Prototypen von drei Funktionen, mit denen Dateien erstellt, geöffnet und gelöscht werden können. Diese Funktionen sind an die Standardsystemfunktionen angelehnt, welche sich generell nach einer Standarddefinition richten [173].

Listing 7.14 Funktionen, um Dateien zu erstellen, zu öffnen und zu löschen

```
#include <lib/filesys.h>

int boinc_touch_file(const char *path);
  /* Return: Upon successful completion it returns 0.
   * Otherwise, -1 is returned.
   */

FILE* boinc_fopen(const char* path, const char* mode);
  /* Return: Upon successful completion it returns a
   * FILE pointer. Otherwise, ERR_UNLINK (-110)
   * is returned.
   */

int boinc_delete_file(const char*);
  /* Return: Upon successful completion it returns 0.
   * Otherwise, -1 is returned.
   */
```

`boinc_fopen()` entspricht der Funktionalität der Systemfunktion `fopen` und bietet die Möglichkeit, Dateien zu öffnen. Der Parameter `path` beschreibt hierbei den Dateinamen, optional mit Dateipfad, und der zweite Parameter ist der Modus zum Öffnen der Datei. Nachfolgende Modis werden unterstützt:

- **`r`** Öffnet eine Datei zum Lesen (engl. *read*) und setzt den Dateizeiger[3] an den Anfang der Datei.
- **`r+`** Öffnet eine Datei zum Lesen, Schreiben und setzt den Dateizeiger an den Anfang der Datei.
- **`w`** Öffnet eine Datei zum Schreiben (engl. *write*). Beim Öffnen wird der Inhalt der Datei gelöscht und der Dateizeiger an den Anfang der Datei gesetzt.
- **`w+`** Öffnet eine Datei zum Schreiben und Lesen. Wenn die Datei nicht existiert, wird diese erstellt. Ist die Datei vorhanden, wird der Inhalt gelöscht und der Dateizeiger wird immer an den Anfang der Datei gesetzt.
- **`a`** Öffnet die Datei zum Schreiben und setzt den Dateizeiger an das Ende der Datei. Es werden also Daten an eine Datei angehängt (engl. *append*).
- **`a+`** Öffnet die Datei zum Lesen, Schreiben und der Dateizeiger wird an das Ende der Datei gesetzt.

`boinc_fopen()` ist so konzipiert, dass das recht eigenartige Verhalten beim Öffnen von Dateien unter Windows vermieden wird. Eventuell sind auf dem auszu-

[3] Beim Arbeiten mit einer Datei wird eine spezielle Struktur intern mit allen Informationen über eine Datei gesetzt. Eine geöffnete Datei kann zu jedem Zeitpunkt weiter ausgelesen oder beschrieben werden. Der Zugriff kann dabei wahlfrei sein. Zu diesem Zweck speichert die Struktur einen Offset, der Wert dieses Offsets ist relativ zum Beginn der Datei. Dieser Offset ist der sogenannte Dateizeiger, der beschreibt, wo die Bearbeitung innerhalb der Datei stattfindet.

führenden Rechner Programme wie zum Beispiel *FastFind*[4] oder *Diskeeper*[5] installiert. Diese Programme können dazu führen, dass eine Datei zu einem bestimmten Zeitpunkt nicht geöffnet werden kann. Sollte dies der Fall sein, so würden die Systemfunktionen direkt eine Fehlermeldung liefern. Im BOINC-Framework wird in einem Zeitintervall von fünf Sekunden versucht, diese Datei zu öffnen. Erst nach diesen fünf Sekunden wird eine Fehlermeldung zurückgeliefert.

`boinc_touch_file()` erstellt eine neue Datei, wenn diese noch nicht existiert, und `boinc_delete_file()` löscht eine Datei aus dem Dateisystem; Ordner können damit nicht entfernt werden.

Ordner erstellen, löschen und Ordnerinhalte löschen

Listing 7.15 listet vier Funktionen auf, mit denen Ordner erstellt und Ordner sowie Ordnerinhalte gelöscht werden können.

Listing 7.15 Funktionen um Ordner zu erstellen, zu löschen oder nur die Inhalte zu löschen

```
#include <lib/filesys.h>

int boinc_mkdir(const char*);
  /* Upon successful completion it returns 0.
   * Otherwise, ERR_MKDIR (-192) is returned.
   */

int boinc_make_dirs(const char* dirpath, const char* filepath);
  /* Upon successful completion it returns 0.
   * Otherwise, ERR_MKDIR (-192) or ERR_BUFFER_OVERFLOW (-118)
   * is returned.
   */

int boinc_rmdir(const char*);
  /* Upon successful completion it returns 0.
   * Otherwise, ERR_RMDIR (-227) is returned.
   */

int clean_out_dir(const char* dirpath);
  /* Upon successful completion it returns 0.
   * Otherwise, ERR_READDIR (-143), ERR_RMDIR (-227)
   * or -1 is returned.
   */
```

Die Funktionen `boinc_mkdir()` und `boinc_make_dirs()` dienen dem Erstellen von Ordnern. `boinc_mkdir()` erwartet als einzigen Parameter den Namen des zu erstellenden Ordners. `boinc_make_dirs()` benötigt zwei Parameter:

`dirpath` ist der Ordnerpfad, in dem die neuen Ordner erstellt werden sollen, und

[4] Ein Werkzeug, welches mit Microsoft Office mit installiert wird. Dieses Werkzeug wird beim Starten des Rechners ausgeführt und läuft als Hintergrundprozess. Es durchsucht kontinuierlich das Dateisystem nach Microsoft-Office-Dateien und erstellt einen Index für jede Datei. Dieser Index soll das Öffnen von Microsoft-Office-Dateien beschleunigen.

[5] Dieses Werkzeug läuft als Hintergrundprozess und führt kontinuierlich eine Defragmentierung durch. Jeder Schreibzugriff wird daraufhin geprüft, dass die zu schreibenden Bytes möglichst zusammengehörend auf die Festplatte geschrieben werden. Dieses Verfahren soll spätere Zugriffe beschleunigen.

filepath beschreibt die zu erstellenden Ordnernamen. Bei der Verwendung muss beachtet werden, dass die Pfadangaben mit Schrägstrichen / beschrieben werden müssen. Dies gilt auch für das Betriebssystem Windows, bei dem Backslashs verwendet werden. Listing 7.16 beschreibt die Verwendung der zwei Parameter.

Listing 7.16 Veranschaulichung zur Verwendung von `boinc_make_dirs()`

```
/* Aktuelle Ordnerstruktur
 * in /home/boincadm/projects/spinhenge/slots/0
 * --
 * 0/
 * 0 directories, 0 files
 * --
 */
char path[] = "/home/boincadm/projects/spinhenge/slots/0";
char newDirectories1[] = "calculation/first/";
char newDirectories2[] = "calculation/second/tada";
char buf[512] = {'\0'};

if(boinc_make_dirs(path, newDirectories1)) {
  printf(stderr, "%s: could not create directories\n", boinc_msg_prefix(buf));
}

if(boinc_make_dirs(path, newDirectories2)) {
  printf(stderr, "%s: could not create directories\n", boinc_msg_prefix(buf));
}
/* Ordnerstruktur nach der Ausfuehrung
 * in /home/boincadm/projects/spinhenge/slots/0
 * --
 * 0/
 * `-- calculation
 *     |-- first
 *     `-- second
 *         `-- tada
 * 4 directories, 0 files
 * --
 */
```

Mit den Funktionen `boinc_rmdir()` und `clean_out_dir()` können Ordner oder nur der Ordnerinhalt mit den Unterordnern gelöscht werden. Der eigentliche Ordner würde in diesem Fall erhalten bleiben.

Ordner auslesen

Das BOINC-Framework beinhaltet eine Klasse zum Auslesen von Ordnern. Das Auslesen wird frei von Rekursionen durchgeführt, das bedeutet, dass keine Unterordner ausgelesen werden. Für das Auslesen von Unterordnern ist der Entwickler verantwortlich. Listing 7.17 veranschaulicht die Klassendefinition `DirScanner`.

Listing 7.17 Klassendefinition zum Auslesen von Ordnern

```
#include <lib/filesys.h>

class DirScanner {
public:
  DirScanner(std::string const& path);
  ~DirScanner();
  bool scan(std::string& name);
```

```
    /* Returns true when one file has retrieved.
     * Otherwise, false is returned.
     */
};
```

Der Konstruktor wird zum Setzen des auszulesenden Verzeichnisses verwendet. Nachdem das Verzeichnis gesetzt ist, kann mit der Methode `scan` das Auslesen durchgeführt werden. Listing 7.18 zeigt die Anwendung dieser Methoden. Der Rückgabewert von `scan` liefert *true*, wenn ein Dateiname gelesen werden konnte. Bei Fehlern wird *false* zurückgegeben. Dateinamen mit einem Punkt am Anfang werden übersprungen. Der Dekonstruktor schließt das aktuelle, geöffnete Verzeichnis.

Listing 7.18 Ein Beispiel zur Auflistung aller Dateien und Ordner in einem Dateiordner

```
// C++
#include <string>
#include <iostream>
// BOINC
#include <lib/filesys.h>

int main(int argc, char **argv) {
  ...
  DirScanner scanner("/");
  std::string name;
  while(scanner(name)) {
    std::cout << "Dateiname: " << name << std::endl;
  }
  ...
}
```

Dateien kopieren und umbenennen

Listing 7.19 beinhaltet zwei Funktionen zum Umbenennen und Kopieren von Dateien oder nur zum Umbenennen von Ordnern.

Listing 7.19 Funktionen zum Kopieren und Umbenennen von Dateien und Ordnern

```
#include <lib/filesys.h>

int boinc_copy(const char* orig, const char* newf);
  /* Upon successful completion it returns 0.
   * Otherwise, ERR_FOPEN (-108), ERR_FWRITE (-103)
   * or -1 is returned.
   */

int boinc_rename(const char* old, const char* newf);
  /* Upon successful completion it returns 0.
   * Otherwise, ERR_RENAME (-109) or -1 is returned.
   */
```

`boinc_copy()` ist so implementiert, dass unter Windows ein direktes Kopieren der Datei erfolgt. Unter OS/2 ist durch die Eberhard-Mattes-eXtender[6] (EMX)-Erweiterung ein direktes Kopieren möglich. Für UNIX/Linux-Derivate wird eine

[6] Eberhard-Mattes-eXtender (EMX) ist eine Programmierumgebung für DOS und OS/2. Die Programmierumgebung liefert eine POSIX-API und für OS/2 werden Zugriffe auf die OS/2 API ermöglicht http://www.os2site.com/sw/dev/emx/.

eigene Kopierroutine genutzt, um so auch die Zugriffsrechte und Einstellungen der Eigentümer mit zu übernehmen. Sollte die Zieldatei existieren, so wird diese in allen Fällen durch die Quelldatei überschrieben. Ordner können durch diese Funktion nicht kopiert werden. In diesem Fall müssen alle Zielorder einzeln erstellt und die Dateien im Quellordner einzeln zum jeweiligen Zielordner kopiert werden.

`boinc_rename()` versucht eine Datei oder einen Ordner umzubenennen. Sollte dies nicht auf Anhieb funktionieren, wird es fünf Sekunden lang probiert und spätestens dann eine Fehlernummer zurückgegeben. In der Regel passiert es unter Windows, dass ein Umbenennen nicht auf Anhieb klappt, da zum Beispiel ein Virenscanner diese Datei geöffnet hat. Dabei verhält sich diese Funktion wie `boinc_ fopen()`. Geöffnete Dateien können unter Windows nicht umbenannt werden. Unix/ Linux ist in dieser Angelegenheit ein wenig offener, dort kann die Datei umbenannt werden, auch wenn sie öfter geöffnet sein sollte.

Zugriffsrechte einer Datei modifizieren

Die in Listing 7.20 aufgelistete Funktion ist nur unter UNIX/Linux-Derivaten verfügbar und verhält sich wie die Systemfunktion `chown`. Im Gegensatz zu der Systemfunktion werden nur zwei Parameter verlangt: (1) der Dateiname und (2) die neue Gruppenidentifikation der zu ändernden Datei. Standardmäßig wird die Benutzeridentifikation der vorhandenen Datei übernommen und nicht modifiziert.

Listing 7.20 Funktion, um die Zugriffsrechte einer Datei zu modifizieren

```
#include <lib/filesys.h>

int boinc_chown(const char*, gid_t);
  /* Return: Upon successful completion it returns 0.
   * Otherwise, ERR_CHOWN (-223) is returned.
   */
```

Den Typ einer Datei ermitteln

Listing 7.21 enthält drei Funktionen, mit denen der Typ einer Datei ermittelt werden kann. Es wird unterschieden zwischen einer einfachen Datei, einem Ordner und einem symbolischen Link. Symbolische Links werden unter Windows nicht unterstützt, weshalb diese Funktion dort nicht zur Verfügung steht.

Listing 7.21 Funktionen zum Ermitteln des Typs einer Datei

```
#include <lib/filesys.h>

int is_file(const char* path);
int is_dir(const char* path);
// Windows
int is_symlink(const char* path);
 /* Upon successful check they return 1.
  * Otherwise, 0 is returned.
  */
```

Die Funktionen `is_file()` und `is_dir()` verwenden intern die POSIX-Funktion `stat()`, um den Typ der Datei zu ermitteln. Die dritte Funktion `is_symlink` verwendet in der Implementierung die POSIX-Funktion `lstat()`. `lstat()` verhält sich wie `stat()`, mit dem Unterschied, dass diese Funktion symbolische Links verarbeiten kann. Listing 7.22 zeigt die Verwendung dieser drei Funktionen. Zu Beginn wird die Ordnerstruktur gezeigt, die zum Testen verwendet wird. Der erste Buchstabe in der Beschreibung `[drwxr-xr-x]` beschreibt den Typ der Datei.

- `d` steht für Ordner (engl. *directory*),
- `l` steht für symbolischer Link (engl. *symbolic link*) und
- `-` beschreibt eine einfache Datei.

Listing 7.22 Veranschaulichung der Funktionen zum Ermitteln des Typs einer Datei

```
#include <lib/filesys.h>

int main(int argc, char **argv) {
  boinc_init();
  /*
   * File system structure
   * in /home/boincadm/projects/spinhenge/slots/0:
   * --
   * `-- [drwxr-xr-x]  0
   *     |-- [lrwxrwxrwx]  c -> calculation/
   *     `-- [drwxr-xr-x]  calculation
   *         |-- [drwxr-xr-x]  first
   *         |   `-- [-rw-r--r--]  input
   *         `-- [drwxr-xr-x]  second
   *             `-- [drwxr-xr-x]  tada
   * --
   */
  if(!is_file("c/")) {
    fprintf(stderr, "This is not a directory!\n");
  }
  if(!is_file("calculation/")) {
    // error message
  }
  if(is_link("c/") && is_file("c/first/input")) {
    fprintf(stdout, "Everything is fine!\n");
  }
  boinc_finish(BOINC_SUCCESS);
}
```

Dateisystem- und Ordnerinformationen ermitteln

Listing 7.23 listet Funktionen auf, um Informationen über das Dateisystem oder Ordner zu erhalten.

Listing 7.23 Funktionen zum Ermitteln von Dateisystem- und Ordnerinformationen

```
#include <lib/filesys.h>

void boinc_getcwd(char* path);
void relative_to_absolute(const char* relname, char* path);

int boinc_file_exists(const char* path);
  /* Upon successful check it returns true (1).
```

```
    * Otherwise, false (0) is returned.
    */

int boinc_file_or_symlink_exists(const char* path);
  /* Upon successful check it returns true (1).
    * Otherwise, false (0) is returned.
    */

int file_size(const char*, double& size);
  /* Upon successful check it returns 0.
    * Otherwise, ERR_NOT_FOUND (-161) is returned.
    */

int dir_size(const char* dirpath, double& size,
    bool recursive=true);
  /* Upon successful check it returns 0.
    * Otherwise, ERR_OPENDIR (-111) is returned.
    */

int get_filesystem_info(double &total_space,
    double &free_space, char* path=const_cast<char *>("."));
  /* Returns 0 */
```

`boinc_getcwd()` speichert im Parameter `path` den aktuellen Arbeitspfad. Es werden Dateipfade mit einer maximalen Länge von 256 Zeichen unterstützt. Der Entwickler ist für das Bereitstellen von genügend Speicherplatz verantwortlich. Listing 7.24 zeigt ein einfaches Anwendungsbeispiel.

Listing 7.24 Beispiel zur Anwendung von `boinc_getcwd()`

```
1  // C
   #include <stdio.h>
   // BOINC
   #include <lib/filesys.h>

6  int main(int argc, char **argv) {
     boinc_init();
     char path[256] = {'\0'};
     boinc_getcwd(path);
     fprintf(stdout, "Current working directory is: %s\n", path);
11   boinc_finish(BOINC_SUCCESS);
   }
```

`relative_to_absolute()` erwartet zwei Parameter. Der Parameter `relname` ist der zum Parameter `path` relative Dateiname. Diese beiden Parameter werden zusammengefügt. Im ersten Schritt wird mit `boinc_getcwd()` das aktuelle Verzeichnis ermittelt und in `path` gespeichert und im zweiten Schritt der relative Pfad in `relname` an diese Verzeichnisinformation angehängt. Listing 7.25 zeigt ein einfaches Anwendungsbeispiel. Der Parameter `path` sollte genügend Speicherplatz erhalten, damit keine Speicherzugriffsfehler aufkommen können; es sollten mindestens 256 Bytes – möglichst mehr – reserviert werden und eine Prüfung der Zeichenkettenlängen stattfinden. Es ist dabei egal, ob die Datei existiert, es handelt sich bei dieser Funktion nur um eine Zeichenkettenarithmetik.

Listing 7.25 Beispiel zur Anwendung von `relative_to_absolute()`

```
   // Current directory: /home/boincadm/projects/spinhenge/slots/0
   // C
3  #include <string.h>
   // BOINC
```

```
#include <lib/filesys.h>

int main(int argc, char **argv) {
  boinc_init();
  char path[512] = {'\0'};
  if(sizeof(path) <= 256) {
    // error message: reserve more bytes, min. 256 bytes!
  }
  realtive_to_absolute("calculation/input", path);
  // result in path:
  // /home/boincadm/projects/spinhenge/slots/0/calculation/input
  boinc_finish(BOINC_SUCCESS);
}
```

`boinc_file_exists()` und `boinc_file_or_symlink_exists()` erwarten einen Pfad zu einer Datei oder einem Ordner. Es wird geprüft, ob diese Datei oder der Ordner existiert. Unter Windows sind diese beiden Funktionen identisch, da keine symbolischen Links unterstützt werden. Unter Unix/Linux prüft die zweite Funktion zudem, ob es sich um einen symbolischen Link handelt und ob die Datei existiert, zu der dieser symbolische Link zeigt. Die Anwendung ähnelt den Funktionen `is_file()`, `is_dir()` und `is_symlink()` und kann durch eine gleiche Vorgehensweise beim Aufruf verwendet werden (vgl. Listing 7.22).

Mit `file_size` kann die Dateigröße in Bytes ermittelt werden. Symbolische Links werden nicht unterstützt. Listing 7.26 verdeutlicht die Anwendung.

Listing 7.26 Beispiel zur Anwendung von `file_size`

```
...
double fsize = 0.0;
if(file_size("calculation/input", fsize) == ERR_NOT_FOUND) {
  // Arg, something went wrong.
} else {
  std::cout << "File size: " << fsize << std::endl;
}
...
```

`dir_size` ermittelt den Speicherplatz der Dateien in Bytes. Optional wird auch der Speicherplatz aller Unterordner mit den dort vorhandenen Dateien ermittelt. Standardmäßig werden alle Dateien inklusive Unterordner für die Ermittlung einbezogen. Versteckte Dateien werden auch verarbeitet. Diese Funktion sollte mit Vorsicht verwendet werden, je nach Anzahl an Dateien und Unterordnern kann die Ausführung zu Einbußen der Systemperformance führen. Zur Ermittlung der Dateigrößen wird in der Implementierung die Funktion `file_size` verwendet. Listing 7.27 liefert ein Anwendungsbeispiel.

Listing 7.27 Beispiel zur Anwendung von `dir_size`

```
...
double bytes = 0.0;
if(dir_size("calculation/input", bytes, false) == ERR_OPENDIR) {
  // Arg, something went wrong.
} else {
  std::cout << "Directory size: " << bytes << std::endl;
}
...
```

`get_filesystem_info` ermittelt den Speicherplatz, den ein Dateisystem besitzt und den noch freien Speicherplatz. Es kann jedes eingebundene Dateisystem

geprüft werden. Der dritte Parameter `path` dient der Angabe eines eingebundenen Dateisystems. Die jeweiligen Werte in Bytes werden in den Parametern `total` und `free` abgespeichert. Listing 7.28 liefert ein Anwendungsbeispiel.

Listing 7.28 Beispiel zur Anwendung von `get_filesystem_info`

```
/* Mounted devices:
 * ---
 * /dev/sdb1 307663800 259417136  32618232  89% /media/files
 * //home.ad.fh-bielefeld.de/Share_FB7pe/Info
 *           41943040  31637120  10305920  76% /media/share_master
 * //home.ad.fh-bielefeld.de/share_fb3/Forschung/science_schroeder
 *           31457280  17716944  13740336  57% /media/share_visualgrid
 * ---
 */
...
char devices[][128] = {
  "/media/files",
  "/media/share_master",
  "/media/share_visualgrid"
};
for(int i=0; i<3; i++) {
  double totalSpace = 0.0;
  double freeSpace = 0.0;
  get_filesystem_info(totalSpace, freeSpace, devices[i]);
  fprintf(stdout, "dev(%s) -> total: %lf, free: %lf\n",
                  devices[i], totalSpace, freeSpace)
}
...
/* Output:
 * ---
 * dev(/media/files) -> total: 292035368, free: 32618232
 * dev() -> total: 41943040, free: 10305920
 * dev() -> total: 31457280, free: 13740336
 * ---
 */
```

7.3.7 Dateien sperren – Sperrung erstellen und aufheben

Mit der Struktur `FILE_LOCK` kann eine Datei für weitere Zugriffe gesperrt und zu jedem Zeitpunkt wieder entsperrt werden.

Listing 7.29 Struktur zum Sperren von Dateien

```
#include <lib/filesys.h>

struct FILE_LOCK {
#if defined(_WIN32) && !defined(__CYGWIN32__)
  HANDLE handle;
#else
  int fd;
#endif
  bool locked;
  FILE_LOCK();
  ~FILE_LOCK();
  int lock(const char* filename);
    /* Upon successful completion it returns 0.
     * Otherwise, ERR_FCNTL (-154), ERR_OPEN (-121)
     * or -1 (Windows) is returned.
     */
```

```
  int unlock(const char* filename);
    /* Upon successful completion it returns 0.
     * Otherwise, -1 is returned.
     */
};
```

Zum Sperren von Dateien muss die Funktion `lock` verwendet werden. So eine gesperrte Datei kann dann wiederum mit der Funktion `unlock` entsperrt werden. Die Variable `locked` speichert entsprechend dem Zustand `true` für eine gesperrte Datei oder `false` für eine nicht gesperrte Datei. Die BOINC-Implementierung dieser Funktionen unterstützen mit dem *Network File System* (NFS) eingebundene Dateisysteme.

7.3.8 Dateien mit XML-Baum erstellen und verarbeiten

Das BOINC-Framework beinhaltet eine eXtended-Markup-Language (XML)-Implementierung, um Dateien im XML-Format (nachfolgend „XML-Dateien“ genannt) zu lesen und verarbeiten zu können.

7.3.8.1 C++-Klasse XML_Parser

Mit der Klasse `XML_PARSER` ist das Arbeiten mit XML-Dateien ohne großen Aufwand möglich. Die Klasse enthält 10 Methoden. Methoden mit dem Präfix `parse_` dienen dem Auslesen von XML-Tags. Die XML-Tags werden direkt mit einem gewünschten Tag-Namen verglichen, und bei Gleichheit wird der Wert aus dem XML-Tag geparst und abgespeichert. Methoden dieser Art sind bis auf `parse_start` identisch aufgebaut. Sie erwarten drei bis vier Parameter:

`parsed_tag`
: ist der aktuelle XML-Tag einer eingelesenen Zeile (z. B. „mode“ in Listing 7.31),

`start_tag`
: ist die vom Entwickler gesuchte Zeichenkette, mit der der erste Parameter verglichen wird, bei Übereinstimmung geben die `parse_`-Funktionen einen Wert ungleich Null zurück,

`Parameter 3`
: ist eine zur `parse_`-Funktion Datentyp passende Variable, zum Beispiel erwartet `parse_str` eine Variable vom Typ `std::string`, und

`Parameter 4`
: wird nur bei der Funktion `parse_str` verwendet, wenn der dritte Parameter eine Speicheradresse für einzelne Zeichen `char` erwartet. Der vierte Parameter beschreibt die Größe des Speicherbereichs in möglichen Bytes.

Die Methode `get` wird verwendet, um einzelne Tag-Zeilen zu lesen. Die Tag-Zeilen können dabei über mehrere Zeilen gehen. Zeichenketten innerhalb eines XML-Tags werden als zusammenhängende Zeichenketten ausgelesen. Der Anfang und

das Ende einer Zeichenkette wird gesäubert, alle nicht druckbaren Zeichen werden entfernt, z. B. `\n \r \t`. Die letzten beiden Parameter sind optional. Der Parameter `attr_buffer` dient dem Abspeichern der Zeichenkette der gefilterten Attribute und `attr_length` ist die Anzahl an Zeichen, die in `attr_buffer` gespeichert werden können. Die Methode gibt so lange `false` zurück, bis das Ende der XML-Datei erreicht ist. `is_tag` ist `true` wenn die gelesene Information eine Tag-Information enthält.

`get_aux` arbeitet prinzipiell wie `get`, außer dass der Rückgabewert variiert. `get_aux` gibt einen Integer zurück. Dieser Wert beschreibt den Typ der gelesenen Information. Es werden vier Typen unterschieden:

`XML_PARSE_TAG`
: Die gelesene Information ist eine Tag-Information.

`XML_PARSE_COMMENT`
: Die gelesene Information ist ein Kommentar.

`XML_PARSE_EOF`
: Das Ende der XML-Datei ist erreicht.

`XML_PARSE_CDATA`
: Beschreibt gelesene Informationen, die nicht von einem XML-Parser interpretiert werden müssen. Dies betrifft Informationen, die in XML eingebettet Funktionen beschreiben oder sonstige Zeichen enthalten, die beim Parsen Fehler erzeugen würden. `CDATA` steht für *(Unparsed) Character Data* – die Daten sollen nicht verwendet werden.

Listing 7.30 Klassendefinition zum Auslesen und Parsen von XML-Dateien

```
#include <lib/parse.h>

class XML_PARSER {
public:
  MIOFILE* f;
  XML_PARSER(MIOFILE*);

  bool get(char* buf, int len, bool& is_tag,
           char* attr_buffer=0, int attr_length=0);

  int get_aux(char* buf, int len,
              char* attr_buffer, int attr_length);

  bool parse_start(const char* start_tag);

  bool parse_str(char* parsed_tag,
       const char* start_tag, char* buf, int len);

  bool parse_string(char* parsed_tag,
       const char* start_tag, std::string& str);

  bool parse_int(char* parsed_tag,
       const char* start_tag, int& i);

  bool parse_double(char* parsed_tag,
       const char* start_tag, double& x);

  bool parse_bool(char* parsed_tag,
       const char* start_tag, bool& b);

  int element_contents(const char* end_tag,
```

```
        char* buffer, int buffer_length);

  void skip_unexpected(const char* start_tag,
        bool verbose, const char* where);
};
```

Mit `element_contents` kann ein Teilbereich aus der XML-Datei ausgelesen werden. Es wird ab der aktuellen Position bis zum `end_tag` gelesen. Das Teilstück wird in `buffer` bis maximal `buffer_length` viele Zeichen vorgehalten. Der Entwickler ist für eine sinnvolle Angabe der Zeichengröße verantwortlich. Listing 7.33 zeigt, wie diese Funktion verwendet werden kann. In dem Beispiel wird die in Listing 7.31 gezeigte XML-Datei bearbeitet, die Informationen zwischen `<spinhenge>` und `</spinhenge>` werden gelesen und in der Strukturvariablen `buffer` für die spätere Verwendung abgespeichert.

Mit Hilfe von `skip_unexpected` können unerwartete Tags übersprungen werden. Während der Entwicklung eines neuen Software-Release kann es vorkommen, dass weitere XML-Tags zu einer XML-Datei hinzukommen, und zusätzlich wurde eventuell vergessen, die Bedeutung im XML-Parser zu implementieren. Mit dem Ausgeben einer Fehlermeldung kann so ein Fehler schneller gefunden werden. Der Parameter `verbose` kann auf `true` gesetzt eine Ausgabe erzwingen. Mit dem Parameter `where` kann eine weitere Beschreibung für den Ort der fehlerhaften Implementierung beschrieben werden, so dass ein schnelleres Auffinden ermöglicht wird. Listing 7.32 enthält diese Methode und zeigt, wie die Ausgabe aussehen kann.

Der XML-Baum in Listing 7.31 kann mit der Struktur in Listing 7.32 verarbeitet werden. Die Informationen werden in der Struktur vorgehalten und können zu jedem Zeitpunkt neu eingelesen werden.

Listing 7.31 Beispiel einer XML-Datei, wie sie im Spinhenge@home-Projekt verwendet wird (gekürzte Version)

```
<spinhenge>
  <mode attr1="Hello" attr2="World">metropolis </mode>
  <total_mc_cycle >1000</total_mc_cycle >
</spinhenge>
```

Das Listing 7.31 ist eine gekürzte Version der Parameterdatei, wie sie beim Projekt Spinhenge@home [171] verwendet wird. Mit `mode` wird die Berechnungsroutine gewählt und `total_mc_cycle` gibt an, wie viele Iterationen in der Berechnung durchgeführt werden sollen.

Listing 7.32 Beispiel einer Struktur zum Auslesen der XML-Datei für das Spinhenge@home-Projekt

```
typedef struct SpinhengeXML {
  std::string mode;
  int total_mc_cycle;

  inline void init() {
    this->mode = "none";
    this->total_mc_cycle = 10;
  }

  /**
   * @param paramFile Pointer to the xml file.
   * @return Returns ERR_XML_PARSE if something
   *         went wrong. Otherwise, 0 is returned.
```

```
   */
  int parse(FILE *paramFile) {
    init();

    MIOFILE xmlMIOFile;
    xmlMIOFile.init_file(paramFile);
    XML_PARSER p(&xmlMIOFile);

    if(!p.parse_start("spinhenge")) {
      fprintf(stderr, "spinhenge.xml is "
          "not formatted correctly\n");
      return ERR_XML_PARSE;
    }

    bool is_tag = false;
    char tag[128] = {'\0'};
    while (!p.get(tag, sizeof(tag), is_tag)) {
      if (!is_tag) {
        fprintf(stderr,
          "unexpected text in spinhenge.xml: %s\n", tag);
        continue;
      }
      if (!strcmp(tag, "/spinhenge")) return 0;
      if (p.parse_string(tag, "mode", this->mode)) continue;
      if (p.parse_int(tag, "total_mc_cycle",
                      this->total_mc_cycle)) continue;

      /* "<ries>Hello!</ries>" is added to the XML file.
       * When one tag is not expected,
       * following message will printed.
       * --
       * Unrecognized XML in test implementation: ries
       * Skipping: Hello!
       * Skipping: /ries
       * --
       */
      p.skip_unexpected(tag, true, "test implementation");
    }
    return 0;
  }
} SpinhengeXML;
```

Listing 7.33 Beispiel einer Struktur zum Auslesen eines Teilstücks einer XML-Datei für das Spinhenge@home-Projekt

```
/* Output:
 * --
 * buffer: <mode attr1="Hello" attr2="World">metropolis</mode>
 *         <total_mc_cycle>1000</total_mc_cycle>
 * --
 */
// C++
#include <vector>
#include <string>
#include <iostream>

// BOINC
#include <lib/filesys.h>
#include <lib/parse.h>
#include <lib/miofile.h>
#include <lib/error_numbers.h>
#include <api/boinc_api.h>

#define CONFIGURATION_FILE "xmlparse.xml"

typedef struct LMBoincXML {
  char buffer[512];
```

```
  void display() {
    std::cout << "buffer: " << buffer << std::endl;
  }
  int parse(FILE *lmBoincXMLFile) {
    MIOFILE xmlMIOFile;
    xmlMIOFile.init_file(lmBoincXMLFile);
    XML_PARSER p(&xmlMIOFile);

    if(!p.parse_start("lmboinc")) {
      fprintf(stderr, "lmboinc.xml is"
                      " not formatted correctly\n");
      return ERR_XML_PARSE;
    }
    p.element_contents("</lmboinc>", buffer, 512);
    return 0;
  }
} LMBoincXML;

bool initLMBoincXML(LMBoincXML &configuration) {
  std::string xmlFilePath;
  boinc_resolve_filename_s(CONFIGURATION_FILE, xmlFilePath);

  FILE *xmlFile = boinc_fopen(xmlFilePath.c_str(), "r");
  if(xmlFile == NULL) {
    return false;
  }
  return (configuration.parse(xmlFile)?false:true);
}

int main(int argc, char **argv) {
  LMBoincXML lmBoincConfiguration;
  if(!initLMBoincXML(lmBoincConfiguration)) {
    // no message
  }
  lmBoincConfiguration.display();
}
```

7.3.8.2 C-Schnittstelle: Textteile statt Dateien parsen

Nicht immer ist es gewünscht, eine Datei mit XML-Informationen einzulesen und zu parsen. Wenn Sie u. a. Informationen über eine Kommunikationsschnittstelle wie HTTP empfangen, so ist es vorteilhafter, wenn diese Informationen direkt – anstatt vorher abzuspeichern – zur Verfügung stehen. In Listing 7.30 wird die `XML_PARSER`-Klasse vorgestellt, eine C++-Schnittstelle für das Arbeiten mit XML-Dateien. Zusätzlich liefert BOINC eine entsprechende C-Schnittstelle mit, die entweder das Implementieren einer eigenen C++-Klasse für XML ermöglicht, oder das einfache Parsen von XML-Informationen, die in einem Text vorliegen und nicht erst abgespeichert werden sollen, um mit den enthaltenen Informationen arbeiten zu können. Ein einfaches Beispiel finden Sie im Abschn. 9.4.4.3. Die Funktionen für das direkte Parsen von XML-Informationen ähnelt der `XML_PARSER`-Klasse und sind in Kurzform in Listing 7.34 aufgelistet. Diese Schnittstelle ist in den BOINC-Quellen als *deprecated* (zu Deutsch „abgelehnt“) markiert.[7]

[7] In diesem Kontext bedeutet das, dass diese Funktionen nicht weiter gepflegt werden und als veraltet angesehen werden.

Listing 7.34 C-Schnittstelle für das Parsen von XML-Informationen

```
#include <lib/parse.h>

bool match_tag(const char *buf, const char *tag);
bool match_tag(const std::string &s, const char *tag);
```

Überprüfen, ob eine Textzeile ein XML-Tag mit dem Namen `tag` besitzt, bei Existenz wird *true*, ansonsten *false* zurückgeliefert. Der Rückgabewert ist bei allen nachfolgenden Funktionen gleichbedeutend und beschreibt das erfolgreiche Parsen der Informationen oder ob der Vorgang fehlerhaft war.

```
bool parse_bool(const char *buf, const char *tag, bool &result);
```

Versucht die aktuelle Textzeile in `buf` mit dem XML-Tag `tag` in einen Booleschen Wert zu parsen.

```
bool parse_int(const char *buf, const char *tag, int &x);
bool parse_double(const char *buf, const char *tag, double &x);
```

Versucht die aktuelle Textzeile in `buf` mit dem XML-Tag `tag` in einen Integer-Wert bzw. Double-Wert zu parsen.

```
bool parse_str(const char *buf, const char *tag,
               char *dest, int destlen);
bool parse_str(const char *buf, const char *tag, std::string& dest);
```

Versucht die aktuelle Textzeile in `buf` mit dem XML-Tag `tag` in einen Character-Speicher mit der Größe `destlen` zu speichern, wahlweise direkt in eine Objektinstanz der C++-Standardklasse String.

```
void parse_attr(const char *buf, const char *attrname,
                char *out, int len);
```

Eine XML-Zeile kann weitere Attribute beinhalten, z. B.

<person job="Dreamcatcher"city="Berlin"> C.B. Ries </person>

Wenn in `buf` diese Zeile vorliegt, so können die Informationen der einzelnen Attribute mit der Übergabe des entsprechenden Namens für `attrname` geparst werden, der Inhalt steht dann in `out`.

7.3.9 Berechnungsfortschritt

Die Laufzeit einer Berechnung kann von vielen Faktoren abhängen. Berechnungen der Statistik werden öfter ausgeführt, um so ein statistisch stabiles Ergebnis zu erhalten. Solche Algorithmen arbeiten nach dem Prinzip der Monte-Carlo-Algorithmen. Dabei arbeiten diese Algorithmen nach dem „Prinzip der großen Zahlen“, wobei viele Iterationen einer Berechnung durchgeführt werden, mit der Hoffnung, dass das Ergebnis stabil genug wird und zu einem Wert hin konvergiert.

Listing 7.35 Funktionen zum Ermitteln des aktuellen Berechnungsfortschritts

```
#include <api/boinc_api.h>

int boinc_fraction_done(double x);
  /* Returns 0
   */
double boinc_get_fraction_done();
  /* Returns the current fraction done value.
   */

double boinc_elapsed_time();

void boinc_ops_per_cpu_sec(double fp, double i);
void boinc_ops_cumulative(double fp, double i);
void boinc_set_credit_claim(double credit)

int boinc_report_app_status(double, double, double);
```

Die mit diesen Funktionen gesetzten Werte werden in dem im Abschn. 7.3.2 erwähnten Thread zum BOINC-Client gesendet. Routinen in diesem Thread prüfen, ob sich die Anwendung nicht im Standalone-Betrieb befindet und erstellen eine XML-Nachricht. Diese XML-Nachricht enthält die gesetzten Werte und sendet diese an den BOINC-Client. Kapitel 4 beschreibt, wie diese Kommunikation mit Hilfe von Shared-Memory (SHM) durchgeführt wird. Listing 7.36 enthält das XML-Schema, das zwischen dem BOINC-Client und einer wissenschaftlichen Applikation ausgetauscht wird.

Listing 7.36 XML-Daten, die zwischen BOINC-Client und einer wissenschaftlichen Applikation ausgetauscht werden

```
<fraction_done>%e</fraction_done>
<current_cpu_time>%e</current_cpu_time>
<checkpoint_cpu_time>%e</checkpoint_cpu_time>
<fpops_per_cpu_sec>%e</fpops_per_cpu_sec>
<fpops_cumulative>%e</fpops_cumulative>
<intops_per_cpu_sec>%e</intops_per_cpu_sec>
<intops_cumulative>%e</intops_cumulative>
<want_network>1</want_network>
```

`boinc_fraction_done()` dient dem Setzen des aktuellen Berechnungsfortschritts. Der zu übergebende Parameter `x` muss zwischen 0.0 und 1.0 sein. `boinc_get_fraction_done()` gibt den aktuellen Berechnungsfortschritt zurück.

`boinc_elapsed_time()` gibt die Anzahl der Sekunden zurück, die seit dem Start eines Arbeitspakets beziehungsweise seit der letzten Checkpoint-Datei vergangen sind.

`boinc_ops_per_cpu_sec()` setzt die Anzahl an Fließkommaoperationen, die durchschnittlich während einer CPU-Sekunde ausgeführt werden sollen. `boinc_ops_cumulative` setzt die Werte entweder für die Anzahl der Fließkomma- oder Ganzzahloperationen; bei mehrmaligem Aufruf innerhalb kürzester Zeit, z. B. zehn Aufrufe in einer Sekunde, wird nur der letzte Aufruf beachtet.

7.3.10 Atomare Bereiche und Funktionen

Manchmal müssen Aufgaben erledigt werden, die in ihrer Ausführung nicht unterbrochen werden dürfen. Für diesen Fall besitzt das BOINC-Framework Funktionen, um kritische Bereiche (sog. atomare Bereiche – nicht unterbrechbare Bereiche) zu erstellen. Mit den drei Funktionen

Listing 7.37 Funktionen für das Erstellen von atomaren Bereichen

```
#include <api/boinc_api.h>

void boinc_begin_critical_section();
void boinc_end_critical_section();

extern int boinc_try_critical_section();
```

kann ein atomarer Bereich erstellt werden. `boinc_begin_critical_section()` öffnet den atomaren Bereich und `boinc_end_critical_section()` schließt diesen wieder. Bei jedem Öffnen wird intern ein Zähler mit dem Namen `in_critical_section` um Eins inkrementiert. Dieser Zähler gibt zu jedem Zeitpunkt an, ob noch ein atomarer Bereich geöffnet ist und bearbeitet wird. `boinc_end_critical_section()` hingegen dekrementiert diesen Zähler. Ein Wert von Null bedeutet, dass kein atomarer Bereich geöffnet oder bearbeitet wird. `boinc_try_critical_section()` ist nicht im BOINC-Framework implementiert, sondern nur als Prototyp definiert, so dass der Entwickler eine individuelle Implementierung zum Überprüfen auf einen atomaren Bereich erstellen kann. Dadurch ist es möglich, Routinen nur unter bestimmten Bedingungen in einem atomaren Bereich bearbeiten zu lassen. Listing 7.38 zeigt ein einfaches Beispiel, um zu prüfen, ob ein atomarer Bereich begonnen werden kann. In dem Beispiel wird geprüft, ob weniger als drei oder genau drei atomare Bereiche offen sind; wenn ja, gibt die Funktion Null, ansonsten Eins.

Listing 7.38 Beispiel einer Anwendung mit implementierter `boinc_try_critical_section()` Routine

```
/**
 * Returns 1 when less then three atomare areas are open.
 * Otherwise, it returns 0.
 */
int boinc_try_critical_section() {
  if (in_critical_section > 3) {
    return 0;
  }
  return 1;
}
```

Wenn ein atomarer Bereich ausgeführt wird, werden vom BOINC-Client empfangene und für die aktuell ausgeführte wissenschaftliche Applikation bestimmte Nachrichten ignoriert und erst dann bearbeitet, wenn der atomare Bereich beendet ist. Es kann vorkommen, dass der BOINC-Client „unerwartet“ beendet wurde. Dies wird durch sogenannte Herzschlagnachrichten (engl. *heartbeat messages*) geprüft (s. Abschn. 7.3.2). Die wissenschaftliche Applikation prüft ständig den Herzschlag des BOINC-Clients. Sollte der BOINC-Client nicht mehr zur Verfügung stehen, dann muss auch die wissenschaftliche Applikation beendet werden. Sollte die wis-

senschaftliche Applikation ein oder mehr offene atomare Bereiche haben, wird die Beendigung der wissenschaftlichen Applikation so lange ausgesetzt, bis die offenen atomaren Bereiche geschlossen wurden.

7.3.11 Flexible Arbeitszeiten

Der Anwender kann die Ausführung der wissenschaftlichen Applikation jederzeit unterbrechen und zu einem späteren Zeitpunkt fortführen. Natürlich kann es auch vorkommen, dass der BOINC-Manager beendet wurde, auch dann müssen die wissenschaftlichen Applikationen unterbrochen werden. Um nicht bei jedem Neustart das komplette Arbeitspaket neu berechnen zu müssen, bietet es sich an, den Zwischenstand in einer sogenannten Checkpoint-Datei abzuspeichern. Dabei ist es wiederum dem Entwickler überlassen, wie das Dateiformat für die Zwischenspeicherung aussehen soll. Das BOINC-Framework beinhaltet keine speziellen Funktionen, um Checkpoint-Dateien zu erstellen, wohl aber Funktionen, um zu prüfen, ob es sinnvoll ist, eine Checkpoint-Datei zu erstellen oder nicht (s. Listing 7.40).

Listing 7.39 Beispiel zur Erstellung einer Checkpoint-Datei

```
boinc_resolve_filename_s(CHECKPOINT_FILE, checkpointPath);

checkpointFile = boinc_fopen(checkpointPath.c_str(), "w+");
if (checkpointFile != NULL) {
  fprintf(checkpointFile, "%d\n", countFilesDone);
}
fclose(checkpointFile);
```

Das Abspeichern einer Checkpoint-Datei ist teuer, da ein Schreiben auf die Festplatte erfolgt und dieses Gerät im Vergleich zum RAM sehr langsam beim Abspeicherungsprozess ist. Weiterhin ist es nicht sinnvoll, kontinuierlich auf die Festplatte zu schreiben. Mobile Geräte wie Laptops können im Batteriebetrieb laufen, jeder Festplattenzugriff geht auf Kosten der Batterielaufzeit. Das BOINC-Framework besitzt die Konstante `DEFAULT_CHECKPOINT_PERIOD`, die aussagt, dass standardmäßig erst nach 5 Minuten eine neue Checkpoint-Datei erstellt werden darf. Dieser Wert kann vom Projektadministrator oder vom Projektteilnehmer modifiziert werden. Der Wert, der vom Projektadministrator oder Projektteilnehmer über den BOINC-Manager gesetzt werden kann, ist in der Struktur `APP_INIT_DATA` definiert und sollte nicht manuell durch eine Programmierung modifiziert werden:

```
struct APP_INIT_DATA {
  ...
  double checkpoint_period;
  ...
};
```

Die Abb. 7.4 und 7.5 zeigen, wie man den globalen Wert für das Intervall für die Erstellung einer Checkpoint-Datei einstellen kann. Nach der Installation des BOINC-Managers ist dieser Wert standardmäßig auf 180 s eingestellt. Ein gesetzter Wert kann durch die Funktionen

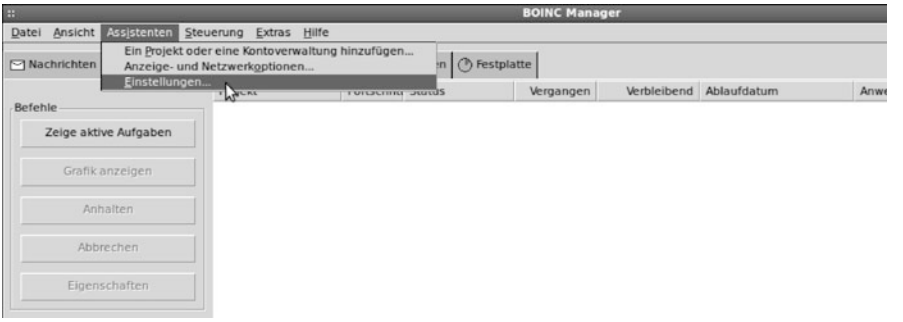


Abb. 7.4 Schaltfläche zum Öffnen des Einstellungsdialogs des BOINC-Managers (Version 6.12.4)

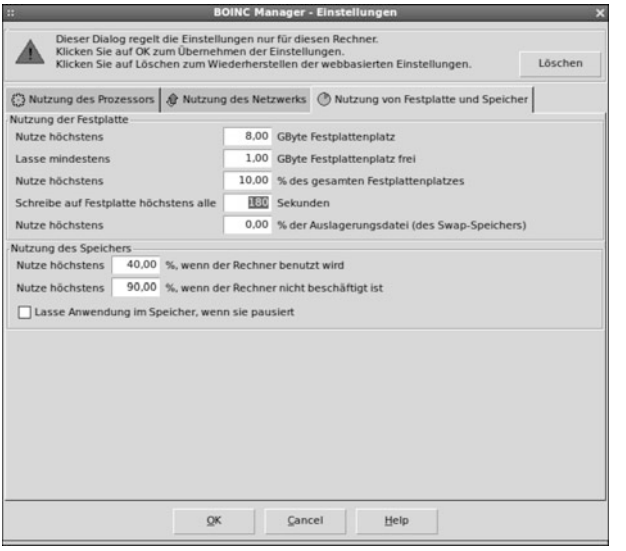


Abb. 7.5 Der Einstellungsdialog des BOINC-Managers (Version 6.12.4) zum Einstellen des Intervalls für die Erstellung einer Checkpoint-Datei

Listing 7.40 Funktionen zum Überprüfen, ob die Zeit abgelaufen ist, nach der eine Checkpoint-Datei erstellt werden darf

```
#include <api/boinc_api.h>

int boinc_time_to_checkpoint();
  /* Returns 1 if a checkpoint should create.
   * Otherwise, 0 is returned.
   */

int boinc_checkpoint_completed(void);
  /* Returns 0 */
```

überprüft werden. Wenn das Intervall überschritten wird, sollte eine Checkpoint-Datei erstellt werden. Das Ende der Erstellung einer Checkpoint-Datei muss mit

boinc_checkpoint_completed() signalisiert werden. Diese Funktion setzt zudem die obere Zeitgrenze neu, nach der wieder eine Checkpoint-Datei erstellt werden sollte. Mit dem Aufruf von boinc_time_to_checkpoint() wird zudem ein atomarer Bereich (vgl. Abschn. 7.3.10) geöffnet, welcher mit dem Aufruf von boinc_checkpoint_completed() wieder geschlossen wird.

Listing 7.41 zeigt ein einfaches Anwendungsbeispiel der zuvor erwähnten Funktionen. In einer einzigen if-Abfrage wird geprüft, ob das Intervall für die Erstellung einer Checkpoint-Datei überschritten wurde. Wenn ja, wird in den inneren Bereich der if-Abfrage gesprungen. Dort kann der Entwickler seine eigene Implementierung zur Erstellung einer Checkpoint-Datei implementieren. Als letzter Schritt sollte mit boinc_checkpoint_completed() signalisiert werden, dass die Erstellung einer Checkpoint-Datei abgeschlossen ist.

Listing 7.41 Anwendungsbeispiel für die Verwendung der BOINC-Funktionen zum Prüfen, ob eine Checkpoint-Datei erstellt werden soll

```
...
if (boinc_time_to_checkpoint()) {
   // Your implementation:
   // Do something to create a checkpoint file.
   boinc_checkpoint_completed();
}
...
```

7.3.12 Prüfung der Netzwerkverbindung

Auch heutzutage hat nicht jeder normale Konsument unbedingt eine dauerhafte Internetverbindung. In Ländern wie Grönland oder Norwegen gibt es weit abgelegene Orte. Nicht immer sind dort schnelle Internetverbindungen verfügbar. Ebenso kann es sein, dass mancherorts keine Kabel verlegt sind und nur teure Funkverbindung die Informationsbeschaffung durch das Internet ermöglicht. Daher ist eventuell nicht immer eine Netzwerkverbindung vorhanden.

Wie in Abschn. 4.3.2.1 beschrieben, besteht ein ständiger Datenaustausch zwischen dem BOINC-Client und einer wissenschaftlichen Applikation, bei der Daten als XML-Nachrichten ausgetauscht werden. Diese Nachrichten können Anfragen nach einer Netzwerkverbindung beinhalten, so dass dem Teilnehmer an einem BOINC-Projekt mitgeteilt wird, dass eine Netzwerkverbindung zur Verfügung gestellt werden soll. Der BOINC-Client prüft kontinuierlich, ob eine Netzwerkverbindung bereitsteht, und tauscht den aktuellen Status mit den wissenschaftlichen Applikationen aus. Die nachfolgenden Funktionen ermöglichen das Prüfen einer Netzwerkverbindung.

Listing 7.42 Funktionen zum Erfragen einer Netzwerkverbindung

```
#include <api/boinc_api.h>

void boinc_need_network();

int boinc_network_poll();
  /* Return 1 if the network is available.
```

```
    * Otherwise, 0 is returned.
    */

void boinc_network_done();
```

Wenn die Funktion `boinc_need_network()` aufgerufen wird, wird `<want_network> 1 </want_network>` zum XML-Baum hinzugefügt. Der BOINC-Client prüft daraufhin, ob eine Internetverbindung besteht beziehungsweise ob eine Verbindung zum BOINC-Projekt hergestellt werden kann. Der BOINC-Client sendet der wissenschaftlichen Applikation die Nachricht `<network_available/>`, wenn eine Verbindung zur Verfügung steht.

7.3.13 Statusmeldungen und Informationsaustausch

Während der Ausführung einer wissenschaftlichen Applikation können Zwischenergebnisse zum BOINC-Projekt übertragen werden. Für diesen Zweck sind die zwei Funktionen

Listing 7.43 Funktionen für das Hochladen von Dateien und Statusberichten an ein BOINC-Projekt

```
int boinc_upload_file(std::string& name);
  /* Upon successful completion 0 is returned.
    * Otherwise, ERR_NULL (-116) or ERR_FOPEN (-108)
    * is returned.
    */

int boinc_upload_status(std::string& name);
  /* Returns the status of the file upload.
    * Otherwise, ERR_NOT_FOUND (-161) is returned.
    */
```

im BOINC-Framework implementiert. Um diese Funktionen nutzen zu können, müssen beim BOINC-Projekt Einstellungen für eine asynchrone Kommunikation durchgeführt werden. Das BOINC-Projekt muss um die Möglichkeit von Trickle-Messages erweitert werden, was zur Folge hat, dass auf der BOINC-Projektseite eine weitere Applikation entweder installiert oder von eigener Hand implementiert werden muss. Abschnitt 9.4 beschreibt, wie ein eigener Trickle-Handler programmiert und installiert werden kann. `boinc_upload_file()` setzt das Flag `have_new_upload_file`, das dafür sorgt, dass im parallel verlaufenden Thread (vgl. Abschn. 7.3.2) eine Nachricht vom BOINC-Client an das BOINC-Projekt verschickt wird. Durch diese Nachricht wird der BOINC-Client angewiesen, im Slot der wissenschaftlichen Applikation zu schauen, ob dort eine Datei zum Hochladen bereitliegt. Listing 7.44 zeigt den relevanten Teil der Funktion `boinc_upload_file()`, die eine Datei mit dem Namen `boinc_ufr_ALTER-DATEINAME` erstellt und für den BOINC-Client bereitstellt. Der BOINC-Client sucht nach Dateien mit dem Präfix `boinc_ufr_` und selektiert diese für das Hochladen. Es können beliebig viele Dateien für das Hochladen bereitstehen.

Listing 7.44 Teilausschnitt der Funktion `bonc_upload_file()`, um eine Datei zum BOINC-Projektserver hochzuladen

```
int boinc_upload_file(std::string& name) {
  ...
  // UPLOAD_FILE_REQ_PREFIX = "boinc_ufr_"
  sprintf(buf, "%s%s", UPLOAD_FILE_REQ_PREFIX,
      name.c_str());
  ...
}
```

7.3.14 Kommunikation zwischen Anwendungen

Abschnitt 4.3.2.1 beschreibt Ansätze, um eine Kommunikation zwischen mehreren Applikationen stattfinden zu lassen. Das BOINC-Framework bietet mehrere Möglichkeiten für diese Aufgabe, baut prinzipiell und nutzt hauptsächlich das Verfahren für Shared-Memory und Memory-Mapped-Dateien; die Ansätze ähneln sich, Shared-Memory teilt mit den Applikationen einen gemeinsamen Speicherplatz und Memory-Mapped-Dateien bilden eine physikalische Datei im Speicher ab, die dann von allen Applikationen genutzt werden kann. Memory-Mapped-Dateien haben den Vorteil, dass bisherige Änderungen – bei einem Absturz einer oder aller Applikationen – nicht gelöscht werden, da diese direkt in der physikalischen Datei vorgenommen wurden.

Prototypen für das Verwenden von Shared-Memory und Memory-Mapping

Für das Verwenden von Shared-Memory mit und ohne Memory-Mapped, werden die Funktionen aus Listing 7.45 vom BOINC-Framework zur Verfügung gestellt.

Listing 7.45 Funktionen für das Erstellen von Shared-Memory-Speicher

```
// Windows
HANDLE create_shmem(LPCTSTR seg_name, int size, void** pp, bool try_global=true
    );
  /* Upon successful creation it returns a pointer.
   * Otherwise, NULL is returned.
   */

HANDLE attach_shmem(LPCTSTR seg_name, void** pp);
  /* Upon successful attach it returns a pointer.
   * Otherwise, NULL is returned.
   */

int detach_shmem(HANDLE hSharedMem, void* p);
  /* Returns 0 */

// min. UNIX V6 Kompatibilitaet: Unix/Linux/Mac
#define MMAPPED_FILE_NAME     "boinc_mmap_file"
int create_shmem_mmap(const char *path, size_t size, void** pp);
  /* Upon successful creation it returns 0.
   * Otherwise, ERR_SHMGET (-144) is returned.
   */
```

```
int attach_shmem_mmap(const char *path, void** pp);
  /* Upon successful attach it returns 0.
   * Otherwise, ERR_SHMAT (-146) is returned.
   */

int detach_shmem_mmap(void* p, size_t size);
  /* Upon successful detach it returns 0.
   * Otherwise, -1 is returned.
   */

// min. UNIX V5 Kompatibilitaet: Unix/Linux/Mac
int create_shmem(key_t key, int size, gid_t gid, void** pp);
  /* Upon successful creation it returns 0.
   * Otherwise, ERR_SHMGET (-144) is returned.
   */

int attach_shmem(key_t, void**);
  /* Upon successful attach it returns 0.
   * Otherwise, ERR_SHMAT (-146) or ERR_SHMGET (-144)
   * is returned.
   */

int detach_shmem(void*);
  /* Upon successful detach it returns 0.
   * Otherwise, -1 is returned.
   */

extern int shmem_info(key_t key);
  /* Not implemented in BOINC, just a prototype
   * for your own implementation.
   */

int destroy_shmem(key_t);
  /* Upon successful creation it returns 0.
   * Otherwise, ERR_SHMCTL (-145) is returned.
   */
```

Ein Shared-Memory wird von den BOINC unterstützenden Betriebssystemen unterschiedlich umgesetzt. Aus diesem Grund beinhaltet Listing 7.45 unterschiedliche Definitionen von Prototypen, d. h. für Windows, UNIX-Derivate und die entsprechenden Versionen, z. B. UNIX mindestens in der Version 5 oder 6.

Die Namensgebung der Funktionsprototypen sind fast identisch, Unterschiede werden bei den Parametersätzen und deren Datentypen gemacht. Im Grunde ist das Einrichten und Verwenden eines Shared-Memory-Bereichs durch das BOINC-Framework sehr gut vereinfacht. Es werden nur eine handvoll Funktionen aus Listing 7.45 benötigt, um eine Kommunikation zwischen Applikationen einzurichten und ablaufen zu lassen. Weiterhin werden die erwähnten moderneren Mechanismen wie Memory-Mapping unterstützt, so dass Dateien direkt durch einen virtuellen Speicher bearbeitet werden können und sogleich mehrere Applikationen Zugriff auf diesen einen Speicherplatz haben.

Listing 7.46 enthält ein Beispiel für das Verwenden einer Memory-Mapped-Datei unter dem Betriebssystem Linux. Das Programm kann direkt zweimal in einem Terminal ausgeführt werden. Beim Starten muss als Parameter eine `1` oder `0` angegeben werden. Dadurch wird zwischen Master (d. h. Eins) und Slave (d. h. Null) unterschieden, so dass diese zwei gestarteten Prozesse miteinander Daten austauschen können. Dabei können mehrere Slave-Prozesse gestartet sein, die wiederum alle denselben Speicherplatz verwenden und auslesen.

In dem Programmbeispiel wird eine Struktur `SHMEM_Exchange` definiert und kann ohne weitere Einschränkungen an die eigenen Bedürfnisse angepasst werden. Diese Struktur enthält die auszutauschenden Datenprototypen. Hier sind das der aktuelle Wert von `fraction_done`, alle vom BOINC-Client gesetzten Konfigurationswerte der Strukturvariablen `app_data` – die für die wissenschaftliche Applikation gesetzt sind – und ein Zähler `count`. Der Zähler wird vom Master bei jeder Aktualisierung um Eins inkrementiert und jeweils – beim Master und Slave – mit einer entsprechenden Meldung ausgegeben.

Listing 7.46 Beispiel zur Erstellung und Nutzung einer Memory-Mapped-Datei unter Linux

```
#include <vector>
#include <string>
#include <iostream>

#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>

// BOINC
#include <api/boinc_api.h>
#include <lib/util.h>
#include <lib/shmem.h>
#include <lib/app_ipc.h>
#include <lib/error_numbers.h>

#define MY_MMAPPED_FILE_NAME "shmem_mmap"

struct SHMEM_Exchange {
  double fraction_done;
  APP_INIT_DATA app_data;
  int count;
};

SHMEM_Exchange *shmem;

APP_INIT_DATA app_init_data;

BOINC_OPTIONS bopts;

void update_shmem() {
  if (!shmem) return;
  if(bopts.main_program) {
    ((SHMEM_Exchange*)shmem)->fraction_done =
                        boinc_get_fraction_done();
    ((SHMEM_Exchange*)shmem)->app_data =
                        app_init_data;
    ((SHMEM_Exchange*)shmem)->count++;
    std::cout << "master: "
              << ((SHMEM_Exchange*)shmem)->count
              << std::endl;
  } else {
    std::cout << "slave: "
              << ((SHMEM_Exchange*)shmem)->count
              << std::endl;
  }
}

int main(int argc, char **argv) {
  if(argv[1] == NULL) {
    std::cout << "Usage: " << argv[0] << " 0|1"
              << std::endl;
    return -1;
  }
```

```
  bopts.main_program = atoi(argv[1]);

  boinc_init_options(&bopts);
58  boinc_get_init_data(app_init_data);

  shmem = new SHMEM_Exchange;

  if(create_shmem_mmap(MMAPPED_FILE_NAME,
63        sizeof(SHMEM_Exchange), (void**)&shmem)) {
    delete shmem;
    shmem = NULL;
    boinc_finish(1);
  }
68
  // do something...
  while(1) {
    // some calculations...
    boinc_sleep(1.0);
73  update_shmem();
  }
  boinc_finish(0);
}
```

7.4 Mehrkernprozessoren für Berechnungen nutzen

7.4.1 Initialisierung

Generell kann die Initialisierung für Mehrkernprozessoren beziehungsweise das Verwenden von mehreren Prozessoren so stattfinden, wie dies bei der Nutzung eines Einzelprozessors der Fall ist. In diesem Fall müssen das Überwachen und die Steuerungsmechanismen vom Entwickler selbst implementiert werden. Das BOINC-Framework bietet dem Entwickler in dem Fall von Mehrkernprozessoren eine Unterstützung. Mit der Funktion `boinc_init_parallel()` aus Listing 7.6 werden zwei Prozesse erzeugt. Ein Prozessor ist für die Abarbeitung von Steuerungsbefehlen zuständig und wandelt diese direkt in Systemsignale um. Der zweite Prozess führt die Berechnungen durch und empfängt die Systemsignale vom ersten Prozess. Es können keine Mehrkernprozessoren für die exklusive Nutzung einer wissenschaftlichen Applikation reserviert werden.

In dem Berechnungsprozess können generell alle Berechnungsarten beziehungsweise Techniken für Berechnungen implementiert sein. Dies bedeutet, dass auch Techniken wie Open Multi-Processing (OpenMP) oder Message Passing Interface (MPI) verwendet werden können. In Abschn. 1.3.1 werden Beispiele für die Verwendung von OpenMP und MPI gezeigt. Es sei an dieser Stelle noch erwähnt, dass das BOINC-Framework vom Grundgedanken her eine Nutzung von MPI nicht vorsieht. Die Zielsetzung von BOINC ist die Unterstützung von kleinen und unabhängigen Arbeitspaketen, die für sich berechnet werden können. Doch wer bestimmt, ob sich nicht hinter einem teilnehmenden Rechner ein Cluster mit MPI-Unterstützung befindet? Da allerdings durch BOINC keine Unterstützung für MPI vorgesehen ist,

ist es leider so, dass der BOINC-Client beziehungsweise der BOINC-Manager keine Prüfung auf MPI-Unterstützung durchführt, und dadurch kann ein BOINC-Projekt nicht entscheiden, ob ein Arbeitspaket mit MPI-Nutzung versendet werden kann.

7.4.2 Prozesse verwalten und steuern

In den BOINC-Quellen wird ein Beispiel für das Nutzen von mehreren Prozessen innerhalb einer wissenschaftlichen Applikation mitgeliefert. Das Beispiel enthält zwei Strukturen:

- `THREAD` und
- `THREAD_SET`.

`THREAD_SET` enthält zwei Funktionen zum Überprüfen, ob alle Prozesse, die rechnen, fertig sind sowie eine Funktion zum Ermitteln, wie viele Recheneinheiten schon berechnet sind. Die Applikation erwartet beim Aufrufen einen Parameter, um anzugeben, wie viele Prozesse gestartet werden sollen, standardmäßig werden vier Prozesse gestartet, die die definierten 16 Berechnungseinheiten bearbeiten. Listing 7.47 zeigt die zwei erwähnten Strukturen. Die Struktur `THREAD_SET` enthält dabei ein Objekt der Container-Klasse `std::vector` mit dem Template-Datentyp eines Zeigers auf die Klasse `THREAD`. Das bedeutet, dass das Objekt der Container-Klasse Zeiger vom Typ `Zeiger auf THREAD` aufnehmen kann.

Listing 7.47 Ein Ausschnitt des BOINC mitgelieferten Beispiels für die Verwendung von Mehrkernprozessoren

```
struct THREAD {
  THREAD_ID id;
  int index;
  int units_done;

  THREAD(THREAD_FUNC func, int i) {
    char buf[256];

    index = i;
    units_done = 0;
#ifdef _WIN32
    id = (HANDLE) _beginthreadex(NULL, 16384,
            func, this, 0, NULL);
    if (!id) {
      fprintf(stderr, "%s Can't start thread\n", boinc_msg_prefix(buf));
      exit(1);
    }
#else
    int retval = pthread_create(&id, 0, func, (void*)this);
    if (retval) {
      fprintf(stderr, "%s can't start thread\n", boinc_msg_prefix(buf));
      exit(1);
    }
#endif
  }
};

struct THREAD_SET {
  vector<THREAD*> threads;
  bool all_done() {
```

```
    for (unsigned int i=0; i<threads.size(); i++) {
      if (threads[i]->id != THREAD_ID_NULL) return false;
    }
34  return true;
  }
  int units_done() {
    int count = 0;
    for (unsigned int i=0; i<threads.size(); i++) {
39    count += threads[i]->units_done;
    }
    return count;
  }
};
```

Das Zuweisen der Zeiger auf eine neue Instanz der Struktur wird in der `main`-Funktion durchgeführt. In Listing 7.48 wird gezeigt, wie in einer `for`-Schleife die Erstellung der zuvor definierten Anzahl von Prozessen in Zeile 18 erfolgt. Das Erstellen geschieht mit dynamisch erstelltem Speicher. Im Aufruf zum Hinzufügen eines Zeigers zum Thread Container wird vorher ein Objekt erstellt, und dieser von `new` zurückgelieferte Zeiger wird abgespeichert. Beim Erstellen wird sogleich der Konstruktor aufgerufen und dadurch mit der Funktion `pthread_create()` in Listing 7.47, Zeile 19 ein arbeitender Prozess erstellt. Die `while`-Schleife in den Zeilen 22ff aus Listing 7.48 erfragt kontinuierlich, wie viele Arbeitsschritte fertig gestellt wurden und berechnet daraus den `fraction_done` (vgl. Listing 7.35) zum Übermitteln an den BOINC-Client, so dass der BOINC-Manager diesen Wert anzeigen kann. In Zeile 24 wird auf die Fertigstellung aller Arbeiten geprüft und dann aus der Schleife gesprungen und mit `boinc_finish()` das Ende der Ausführung für den BOINC-Client signalisiert.

Listing 7.48 `main`-Funktion der Beispielanwendung für die Nutzung von Mehrkernprozessoren

```
/* Call for Linux implementation.
2  */
int main(int argc, char** argv) {
  ...
  boinc_init_parallel();

7   for (i=1; i<argc; i++) {
      if (!strcmp(argv[i], "--nthreads")) {
        nthreads = atoi(argv[++i]);
      } else {
        ...
12    }
    }

    units_per_thread = TOTAL_UNITS/nthreads;

17  THREAD_SET thread_set;
    for (i=0; i<nthreads; i++) {
      thread_set.threads.push_back(new THREAD(worker, i));
    }

22  while (1) {
      double f = thread_set.units_done()/((double)TOTAL_UNITS);
      boinc_fraction_done(f);
      if (thread_set.all_done()) break;
      boinc_sleep(1.0);
27  }
    ...
    boinc_finish(0);
}
```

7.5 Grafikkartenprozessor für Berechnungen nutzen

Es sei im Vorfeld erwähnt, dass es durchaus zu Herausforderungen bei der Entwicklung von wissenschaftlichen Applikationen im Zusammenhang mit Berechnungskernen einer Grafikkarte kommen kann, z. B. in Bezug auf die Performance unterschiedlicher GPU (Graphics Processing Unit)-Architekturen. Sie entwickeln womöglich eine Applikation, welche sich nur mit einer bestimmten GPU optimal ausführen lässt, weil zum Beispiel nur mit dieser genügend Speicher zur Verfügung steht und dadurch unnötiges Freiräumen von Speicherplatz erspart wird.

Erfreulich, dass sich ein Komitee gebildet hat, welches sich das Ziel einer standardisierten Schnittstelle zur Programmierung von Grafikkarten gebildet hat. Unter dem Akronym OpenCL (Open Computing Language) werden Bemühungen unternommen, eine Programmierschnittstelle zu definieren, mit der sich Grafikkarten von verschiedenen Herstellern auf die gleiche Art und Weise programmieren lassen.

Leider unterstützen nicht alle Firmen immer den aktuellen OpenCL-Standard[8]. Somit muss man zum Beispiel beim Kauf der aktuellsten Grafikkarte von NVIDIA, mit CUDA (Compute Unified Device Architecture) entwickeln, um alle Möglichkeiten ausschöpfen zu können. CUDA ist ein zweiter Standard zur Programmierung von Applikationen für die GPU, spezifiziert und entwickelt von NVIDIA, erstmalig im Jahr 2006 vorgestellt. CUDA wird in den nachfolgenden Abschnitten ausschließlich verwendet. Meine Rechner sowie unsere Cluster haben allesamt NVIDIA-Chipsätze und entsprechend besitze ich nur damit Erfahrung.[9] Weiterhin ist der Vorteil bei NVIDIA, dass Sie immer einen aktuellen Treiber erhalten, der ohne Probleme installiert, konfiguriert und gestartet werden kann. Bei anderen namhaften Herstellern gab es bei mir immer Probleme.

7.5.1 GPU-Entwicklungsumgebung: CUDA

In den BOINC-Quellen sind Beispiele für GPU-Berechnungen der Hersteller NVIDIA und ATI enthalten. Prinzipiell führen die Beispiele gleiche Funktionen aus: die Inversion einer Matrix und das Erstellen von Checkpoints während der Ausführung. Allgemein kann man sagen, dass die Beispielanwendungen vom Aufbau her dem upperCASE-Beispiel aus Abschn. 5.6 entsprechen, mit der Ausnahme, dass eine geänderte Berechnungsroutine durchgeführt wird. Anstelle des Umwandelns von Textketten in Großbuchstaben wird hier die Matrix-Inversion umgesetzt.

Die Umkehrung einer Matrix wird inverse Matrix genannt und die Multiplikation einer Matrix mit einer inversen Matrix ist kommutativ (d. h. „vertauschbar“). Der Wunsch nach einer solchen Rechenart liefert die Fragestellung nach x aus der Gleichung $a \cdot x = 1$. Durch das Umstellen nach $x = \frac{1}{x} = x^{-1}, x \in \mathbb{R}$ sehen wir, dass x ungleich Null sein muss, um lösbar zu sein [45].

[8] Zur Zeit OpenCL 1.2, veröffentlicht am 15. November 2011.

[9] NVIDIA Quadro NVS 450 und NVIDIA Tesla M2050.

Leider sind die Beispielanwendungen mit der Matrix-Inversion doch ein wenig zu komplex für das schnelle Verstehen, wenn man vielleicht das erste Mal mit GPU-Programmierung, und dann gleich in Kombination mit BOINC, in Berührung kommt. Ich versuche Ihnen in den nachfolgenden Abschnitten ein einfacher nachzuvollziehendes Beispiel vorzustellen. Sie werden sehen, dass es doch relativ einfach ist, BOINC um GPU-Berechnungen zu bereichern. Es handelt sich dabei um eine Addition von beliebig vielen Matrizen.

Für die Entwicklung von CUDA-Applikationen für NVIDIA-Prozessoren benötigen wir das NVIDIA-SDK (Software Development Kit), welches Sie von der NVIDIA-Webseite herunterladen können [154, 155]. Für Windows und Linux lassen sich zahlreiche Installationsanleitungen im Netz ausmachen, denn teilweise kann schon die Installation dieses NVIDIA-SDK ein Abenteuer sein; füttern Sie einfach Ihre persönliche Lieblingssuchmaschine mit entsprechenden Suchbegriffen.

7.5.1.1 GPU-Architektur

Es wird zwischen *Host* und *Device* unterschieden. Mit Host ist der Computer mit einer CPU oder mehreren CPUs gemeint und Device meint hier die GPU. Eine GPU hat eine grundsätzliche Speicherarchitektur, wie sie in Abb. 7.6 gezeigt wird [36]. Eine Berechnung wird in einem Thread ausgeführt, welcher in einem Block eingebettet ist und wiederum in einem Grid vorliegt. Abhängig von Ihrer GPU und der CUDA-Version kann ein Block 1024 Threads und ein Grid bis zu 65535 Blocks pro Grid enthalten [51]. In Listing 7.50 wird ein Beispiel für die Definition der Grid- und Blockgröße gezeigt. Abhängig davon, wie Sie Daten eventuell zwischen den einzelnen GPU-Bereichen (Grid, Block, Thread) und dem Host austauschen müssen, müssen Sie Ihre Daten in den jeweiligen Speicherbereich ablegen. Generell ist es so, dass Sie Daten vom Host erst in den globalen Speicherbereich der GPU kopieren müssen, daraufhin kann mit den Daten gearbeitet werden. Die Zugriffszeiten auf die einzelnen Speicherbereiche sind sehr unterschiedlich:

global memory Dies ist der globale Speicherbereich, der vom Host beschrieben (W für *write*) und ausgelesen (R für *read*) werden kann. Dieser Speicherbereich enthält zudem noch einen Bereich für konstante Daten. Ein Host hat R/W-Zugriff auf den gesamten Bereich, die GPU hat auf die konstanten Daten nur R-Zugriff und R/W-Zugriff auf die restlichen Daten.

shared memory Dieser Speicherbereich dient dem Austauschen von Daten zwischen Blöcken, was dementsprechend einen R/W-Zugriff erfordert. Ein Host hat keinen Zugriff auf diesen Speicherbereich.

local memory Threads haben lokale Speicherbereiche und ermöglichen schnellen R/W-Zugriff. Dieser Speicherbereich ist von Hosts nicht zugreifbar.

Wenn Sie nur lokal Daten ablegen müssen und damit rechnen wollen, so sollten Sie *local memory* nutzen, der ist am schnellsten. Wenn Sie zwischen einzelnen Threads Daten austauschen müssen, so müssen diese Daten im *shared memory* liegen. Sie

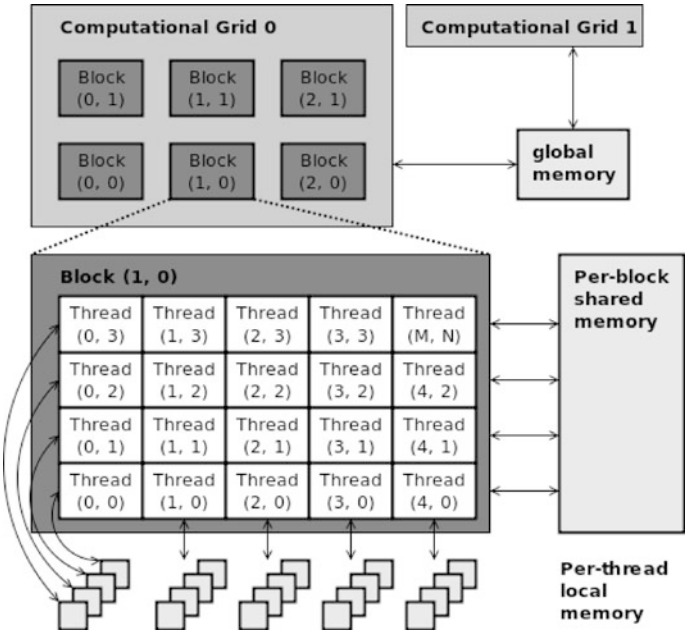


Abb. 7.6 Speicher- und Berechnungskernarchitektur einer GPU

sollten den *global-memory*-Bereich nur nutzen, wenn dieser wirklich benötigt wird, z. B. um Daten zwischen Host und Device auszutauschen.

Der Zugriff auf die jeweiligen Speicherbereiche wird direkt in der Programmierung vorgenommen, indem Sie zum Beispiel das Schlüsselwort `__shared__` verwenden; ein Beispiel finden Sie in Listing 7.49. Es wird zudem gezeigt, wie Sie Variablen entweder auf der GPU oder auf dem Host definieren, so dass der CUDA-Compiler (vgl. Abschn. 7.5.1.2) während der Kompilierung unterscheiden kann, für welches Gerät die einzelnen Direktiven gelten. Selbige Schlüsselwörter gelten auch für Funktionsdefinitionen und können zusätzlich mit `__host__` oder `__global__` definiert werden, so dass eine solche Funktion vom Host aufrufbar ist. Es muss für das Starten einer GPU-Berechnung mindestens eine Funktion als solche definiert sein, damit eine Berechnung stattfinden kann. `__host__` kann dabei nur von der Host-Applikation aufgerufen werden, `__global__` definierte Funktionen vom Host und der GPU, und Funktionen mit `__device__` als Prefix sind nur auf der GPU aufrufbar.

Listing 7.49 Definition von Variablen in unterschiedlichen GPU-Speicherplätzen

```
__device__ __shared__ int sharedVar; // Speicherbereich der GPU: shared memory
__device__ int globalVar; // Speicherbereich der GPU: global memory
int cpuVar; // Speicher auf dem Host

__host__ void doHostWork() { /* ... */ }
__global__ void doWork() { /* ... */ }
__device__ void doLocalWork() { /* ... */ }
```

7.5.1.2 CUDA-Programmierschnittstellen

CUDA bietet zwei Möglichkeiten, GPUs zu nutzen [51]:

CUDA C Diese Möglichkeit beinhaltet einen minimalen Satz von Erweiterungen für die Programmiersprache C. Dadurch können für eine GPU bestimmte Berechnungen in C entwickelt werden und müssen „nur“ durch CUDA-Erweiterungen beschrieben werden. Alle Implementierungen mit solchen CUDA-Erweiterungen müssen mit dem NVIDIA-Compiler `nvcc` kompiliert werden.

CUDA driver API Dies ist eine low-level C-Schnittstelle und liefert Funktionen zum Laden von CUDA-Kernelmodulen, um enthaltene Parameter zu ermitteln und zu prüfen. Allgemein kann man sagen, dass diese API aufwendiger zu verwenden ist, es ist mehr Quellcode nötig, er ist schwieriger zu schreiben und zu debuggen. Allerdings haben Sie mit dieser API mehr Kontrolle über die Ausführung Ihrer Berechnungen.

7.5.2 BOINC mit GPUs initialisieren

Wenn Sie mit GPUs rechnen wollen, so kann die Initialisierung wie bei Applikationen mit CPU-Berechnungen durchgeführt werden. Der einzige Unterschied ist der Ort, an dem Berechnungen durchgeführt werden. Es ist natürlich nicht erforderlich, dass die komplette Abarbeitung der wissenschaftlichen Applikation auf die GPU portiert wird, dies ist in der Regel auch gar nicht nötig und sinnvoll.

Das Schwierigste ist hier, dass Sie Ihren meist sehr speziellen sequentiellen Code in einen möglichst parallel ausführbaren Code umwandeln müssen. Ich kann Ihnen dafür keine goldene Formel vorstellen, da solch ein Schritt sehr unterschiedlich sein kann und teilweise tieferes Wissen über die GPU-Hardware voraussetzt, so dass eine optimale Ausführungsgeschwindigkeit erreicht werden kann. Wir beschränken uns hier auf ein triviales Beispiel, das funktioniert und von Ihnen ohne größere Probleme übernommen werden kann. In unserem Fall müssen wir nur die `doWork`-Funktion aus dem Grundrahmen einer wissenschaftlichen Applikation aus Abschn. 7.3.1 entsprechend anpassen.

Ob eine wissenschaftliche Applikation vom BOINC-Client gestartet wird, hängt von der aktuellen Systemauslastung ab. Der BOINC-Client prüft vorher, ob die erforderlichen Ressourcen zur Verfügung stehen, und startet daraufhin nur die Applikation, welche noch mit den restlichen Ressourcen ausgeführt werden kann. Sie müssen sich nicht darum kümmern, dies erleichtert die Arbeit mit GPUs immens.

7.5.3 Mehr Rechenkraft nutzen

Wie schon erwähnt, muss in einer wissenschaftlichen Applikation grundsätzlich nicht viel umgestaltet werden, so dass eine eventuell vorhandene Applikation um

Tab. 7.1 Beispiele für drei Eingabedateien mit jeweils einer 5×5-Matrix

Eingabedatei 1	Eingabedatei 2	Eingabedatei 3
5	5	5
1 2 −3 4 5	7 8 4 3 2	−1 2 3 4 2
2 1 2 3 4	9 0 7 3 6	2 −1 2 −3 4
3 2 1 2 3	3 3 1 2 −1	3 −2 −1 2 3
4 3 2 1 2	−4 3 8 1 2	4 3 2 −1 2
5 4 3 2 1	−5 4 2 0 0	5 8 9 1 −1

die Fähigkeit der GPU-Berechnung erweitert wird. Die Grundstruktur kann erhalten bleiben, sprich wie eine wissenschaftliche Applikation initialisiert oder beendet wird, wie die Eingabedateien und Ergebnisdateien erstellt werden sollen, und so weiter. Wichtigster Bestandteil einer GPU-Unterstützung ist der Kern der Berechnung, dieser sollte parallel ausführbar sein. Sollte das nicht möglich sein, so wird die Berechnung zwar durchgeführt, allerdings immer noch sequentiell, allerdings ist auch dann ein Speedup möglich. Generell gilt, zuerst die GPU-Berechnung kompilieren und erst im Nachgang eine Optimierung durchführen! Das Listing 7.50 enthält eine GPU-Umsetzung einer einfachen Matrix-Addition von $n \in \mathbb{N}^*, n >= 2$ Matrizen $A_{(N,N)} \forall N \in \mathbb{N}^*$. Die Matrix-Addition ist definiert als $(A + B)_{i,j} = A_{i,j} + B_{i,j}$, wobei gilt $1 \leq i \leq m$ und $1 \leq j \leq n$ und beide müssen entsprechend dieselben Dimensionen besitzen, d. h. Matrix A hat dieselbe Anzahl an Spalten und Zeilen wie Matrix B. In unserer Testapplikation müssen die Matrizen zudem quadratischer Natur sein, sprich die Spalten- und Zeilenanzahl muss identisch sein.

In unserem Beispiel definieren wir drei Eingabedateien in Zeile 14: `in01`, `in02` und `in03`. Jede einzelne Eingabedatei muss nach unserer Spezifikation in der ersten Zeile die Dimension der danach enthaltenen Matrix beschreiben. Tabelle 7.1 enthält die verwendeten Testdaten.

Die Zeilen 25ff enthalten die GPU-Kernfunktion für die Addition zweier Matrizen. Die drei ersten Parameter sind C-Zeiger auf die Speicherstellen innerhalb der GPU-Speicherblöcke und müssen vor der Benutzung in den Zeilen 67ff initialisiert und alloziert werden. `*m1` und `*m2` sind die zu addierenden Matrizen, das Ergebnis wird in `*mres` abgespeichert. Die Dimension aller quadratischen Matrizen ist `N`.

Ab Zeile 41ff finden Sie die `doWork`-Funktion, welche in der `main`-Funktion aufgerufen wird. Innerhalb dieser werden in den Zeilen 50ff die Eingabedateien eingelesen. Zeile 56 enthält das Einlesen der Dimension einer Matrix in die Variable `N`. `N` wird daraufhin in Zeile 59 in einer Schleife genutzt, um alle einzelnen Matrixelement nach und nach einzulesen. Wenn ein Wert falsch formatiert ist oder eventuell fehlt, so wird `ERR_WRONG_SIZE` als Rückgabewert von `doWork` geliefert und die Ausführung der wissenschaftlichen Applikation abgebrochen.

Für die GPU-Berechnung geben wir in den Zeilen 80 und 81 die Block- und Gridgröße an. Wir definieren uns hier somit ein aus Zeile 22 zu erkennendes 5×5 großes Grid und teilen unsere einzelnen Matrixadditionen der Matrixelemente passend auf, so dass jeder Berechnungsblock innerhalb eines Grid die gleiche Anzahl

an Berechnungen erhält. Verwendet werden diese Werte in Zeile 85, beim Aufruf der GPU-Kernfunktion `matrixAdd()`.

Bisher sind die Matrizen nur dem Host bekannt und liegen in der Variablen `**m` vor. Für die Matrixaddition auf der GPU müssen die einzelnen Matrizen in den Speicher der GPU transferiert werden. Dies geschieht durch den Aufruf von `cudaMemcpy()`. Erst dann kann mit diesen Matrizen gerechnet werden. Sollte dies nicht im Vorfeld passieren und Sie versuchen mit den lokalen Speicheradressen beim Aufruf von `matrixAdd()` zu arbeiten, so stürzt die wissenschaftliche Applikation leider nicht ab, Sie erhalten ein undefiniertes Verhalten und entsprechend kein korrektes Berechnungsergebnis. Die Zeilen 83 bis 92 addieren alle Eingabematrizen zusammen und speichern das Ergebnis in `resultDevice`.

Dieses Ergebnis müssen wir der CPU bekannt machen und kopieren daher den Speicherbereich vom Ergebnis in den Zeilen 94 und 95 vom GPU-Speicher zum zugreifbaren Bereich des Hosts. Das Ergebnis liegt nun lokal in der Variablen `localResult` vor und kann in den Zeilen 105ff in der Ergebnisdatei `out` abgespeichert werden und daraufhin vom BOINC-Client an das entsprechende BOINC-Projekt zurückgesendet werden.

Listing 7.50 BOINC-Beispielanwendung mit der Verwendung einer GPU

```
// C++
#include <iostream>
#include <string>

// C
#include <stdio.h>
#include <errno.h>

// BOINC
#include <filesys.h>
#include <boinc_api.h>
#include <error_numbers.h>

const char *inputFilename[12] = {
  "in01", "in02", "in03", NULL
};
const char *outputFilename[12] = {
  "out", NULL
};

/// CUDA kernel runtime parameter
const int blocksize = 5;

/// CUDA kernel
__global__ void matrixAdd(int *m1, int *m2, int *mres, int N) {
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  int j = blockIdx.y * blockDim.y + threadIdx.y;
  int index = i + j*N;
  if(i < N && j < N) {
    mres[index] = m1[index] + m2[index];
  }
}

inline unsigned int sizeOf(const char **field) {
  unsigned int c = 0;
  while(field[++c] != NULL) ;
  return c;
}

/// Worker
```

```
int doWork() {
  unsigned int inCount = sizeOf(inputFilename);
  FILE **in = new FILE*[inCount];
  unsigned int outCount = sizeOf(outputFilename);
  FILE **out = new FILE*[outCount];

  int **m = new int*[inCount];
  int N = 0; // dimension of one matrix is N*N

  for(int i=0; i<inCount; i++) {
    std::string filepath;
    boinc_resolve_filename_s(inputFilename[i], filepath);
    FILE *fin = boinc_fopen(filepath.c_str(), "r");
    if(fin == NULL) return ERR_FOPEN;

    if(fscanf(fin, "%d", &N) != 1)
      return ERR_WRONG_SIZE;

    m[i] = new int[N*N];
    for(int ii=0; ii<N*N; ii++) {
      if(fscanf(fin, "%d", &(m[i][ii])) != 1) {
        return ERR_WRONG_SIZE;
      }
    }
  }

  int *resultDevice, *m1Device, *m2Device;
  const int size = N*N*sizeof(int);
  if(cudaMalloc( (void**)&resultDevice, size)
                  == cudaErrorMemoryAllocation ||
     cudaMalloc( (void**)&m1Device, size)
                  == cudaErrorMemoryAllocation ||
     cudaMalloc( (void**)&m2Device, size)
                  == cudaErrorMemoryAllocation)
  {
    std::cerr << "cudaMalloc failed" << std::endl;
    boinc_temporary_exit(60);
  }

  dim3 dimBlock(blocksize, blocksize);
  dim3 dimGrid(N/dimBlock.x, N/dimBlock.y);

  cudaMemcpy(m1Device, m[0], size, cudaMemcpyHostToDevice);
  cudaMemcpy(m2Device, m[1], size, cudaMemcpyHostToDevice);
  matrixAdd<<<dimGrid, dimBlock>>>
            (m1Device, m2Device, resultDevice, N);

  for(int i=2; i<inCount; i++) {
    cudaMemcpy(m1Device, m[i], size, cudaMemcpyHostToDevice);
    matrixAdd<<<dimGrid, dimBlock>>>
              (resultDevice, m1Device, resultDevice, N);
  }

  int *localResult = new int[N*N];
  int ret = cudaMemcpy(localResult, resultDevice,
                       size, cudaMemcpyDeviceToHost);

  cudaFree(resultDevice);
  cudaFree(m1Device);
  cudaFree(m2Device);

  delete [] m;

  /// dump output
  std::string filepath;
  boinc_resolve_filename_s(outputFilename[0], filepath);
  FILE *fout = boinc_fopen(filepath.c_str(), "w+");
```

```
  if(fout) {
    for(int i=0; i<N*N; i++) {
      if(i > 0 && i%N == 0) fprintf(fout, "\n");
      fprintf(fout, "%d ", localResult[i]);
    }
    fprintf(fout, "\n");
    fclose(fout);
  }

  return BOINC_SUCCESS;
}

/// Main-routine
int main(int argc, char **argv) {
  int returnValue = boinc_init();
  if (returnValue) {
    char buf[1024] = {'\0'};
    std::cerr << boinc_msg_prefix(buf)
              << " boinc_init returned "
              << returnValue << std::endl;
    exit(returnValue);
  }

  /// Start of your work!
  int ret = doWork();
  /// End of your work!

  boinc_fraction_done(1);
  boinc_finish(0);
}
```

7.5.4 CUDA-Applikation kompilieren

NVIDIA liefert einen GCC/G++-Wrapper `nvcc` mit, um CUDA-Applikationen zum Beispiel unter Linux kompilieren zu können. Dieser Wrapper kümmert sich um das Inkludieren der CUDA-Headerdateien, so dass Sie sich zumindest darüber keine Gedanken machen müssen. Ein simples Makefile für das in Listing 7.50 vorgestellte CUDA-Beispiel finden Sie in Listing 7.51. Damit `nvcc` dieses als CUDA-Applikation erkennt, muss die Dateiendung `.cu` der Datei angehängt werden.

Listing 7.51 Makefile für das Erstellen einer CUDA-Applikation mit BOINC

```
INCLUDES=-I./boincsrc/api -I./boincsrc/lib
LIBARIES=-L./boincsrc/api -lboinc_api -L./boincsrc/lib -lboinc ${L}

all: matrixaddBoinc.cu
  nvcc -g -O2 -o matrixaddBoinc matrixaddBoinc.cu \
    ${INCLUDES} ${LIBARIES}
```

Kapitel 8
Die Welt ist bunt – auch BOINC!

Ein Bild sagt mehr als tausend Worte.

8.1 Ein wenig OpenGL

Von Haus wird BOINC mit Unterstützung für die Programmierung einer innerhalb der wissenschaftlichen Applikation mit implementierten oder zusätzlich implementierten OpenGL-Anwendung ausgeliefert. Es handelt sich dabei um eine Abstraktion der – eigentlich von OpenGL – geforderten Funktionen. Um dies zu verstehen, muss im Vorfeld eine kleine Einführung zu OpenGL und von BOINC genutzten Bibliotheken gegeben werden.

In der letzten stabilen BOINC-Revision 22203 unterstützt die BOINC-Grafikbibliothek folgende Entwicklungsumgebungen: Windows, MinGW[1], Cygwin[2], MacOS und Linux. Entsprechend der Entwicklungsumgebung werden die passenden Header-Dateien für die genutzten Bibliotheken verwendet.

8.1.1 OpenGL-Bibliotheken für Linux

Für Linux müssen mindestens folgende Bibliotheken verfügbar sein:

OpenGL Utility Library (GLU) Aktuell liegt die GLU-Spezifikation in der Version 1.3 vor [48]. Diese Bibliothek liefert Routinen, um einfache Primitive zeichnen zu lassen, u. a. Rechtecke, Kreise oder einfache Linien. Alle Primitiven können durch Koordinaten im 3-D-Raum beschrieben werden und können zahlreiche zusätzliche Attribute erhalten, z. B. Farbe oder Linienstärke. Weiterhin liefert GLU Routinen, mit denen die Kameraperspektive einfacher beschrieben werden kann, so dass die Transformation des Blickfeldes mit Hilfe von GLU berechnet und eine korrekte Darstellung – zu jedem Zeitpunkt – ermöglicht wird.

[1] http://www.mingw.org – MinGW | Minimalistic GNU for Windows.

[2] http://www.cygwin.com – Linux-like environment for Windows making it possible to port software running on POSIX systems (such as Linux, BSD, and Unix systems) to Windows.

C. B. Ries, *BOINC*, Xpert.press,
DOI 10.1007/978-3-642-23382-8_8,

OpenGL Utility Toolkit (GLUT) Die aktuelle GLUT-Spezifikation liegt in der Version 3 vor [49]. Mit Hilfe von GLUT werden Arbeiten mit dem Eingabe-/Ausgabesystem eines Betriebssystems erleichtert, z. B. erleichtert GLUT die Erstellung von Fenstern, um die Darstellung von 3-D-Welten zu ermöglichen. Zusätzlich enthält GLUT Methoden, mit Hilfe derer man Maus- und Tastatureingaben verarbeiten kann, u. a. können Mausklicks in passende Koordinaten im 3-D-Raum umgerechnet werden. Weiterhin ermöglicht GLUT:

1. Die Erstellung von Kontextmenüs,
2. das Hinzufügen von Text[3],
3. das Setzen und Lesen von Fensterattributen und
4. noch mehr Vereinfachungen.

8.1.2 OpenGL-Bibliotheken für Windows

Für Windows-Umgebungen muss zusätzlich zu den in Abschn. 8.1.1 erwähnten Bibliotheken noch die folgende Bibliothek zur Verfügung stehen:

OpenGL Auxiliary Library (GLAUX) Diese Bibliothek ähnelt GLUT und stellt auch eine Schnittstelle für die Verwendung von Eingabe-/Ausgabekanälen dar. Die GLAUX legt unter anderem Konstanten für Tasten der Tastatur und Mauseingabe fest. Es kann ein Standarddisplay für die Darstellung erstellt werden; Standardprimitiven können gezeichnet werden.

8.1.3 OpenGL-Bibliotheken für MacOS X

Die in Abschn. 8.1.1 und 8.1.2 erwähnten Bibliotheken werden auch für die Entwicklungsumgebung unter Macintosh OS X benötigt.

8.2 BOINC-Grafikschnittstelle

Wie schon zu Beginn dieses Kapitels erwähnt, liefert BOINC eine Abstraktionsebene, so dass nicht alle Details der in den vorherigen Abschnitten behandelten Bibliotheksroutinen bekannt sein müssen. BOINC kapselt diese in einer einheitlichen API-Schnittstelle, welche innerhalb der unterschiedlichen Entwicklungsumgebungen immer gleiche Namen für Callback-Routinen und Funktionen bereithält.

Die OpenGL-Spezifikationen von GLU und GLUT spezifizieren ein Standardverhalten von implementierten Funktionen. Es gibt zwei verschiedene Ansätze bei den Implementierungen, einige sind direkt durch den Entwickler nutzbar und andere müssen extern zur Verfügung gestellt werden. Das Erstellen eines Fensters für die Darstellung der grafischen 3-D-Welt und Neuladen beziehungsweise Neuzeichnen

[3] Allerdings ist die Umsetzung sehr rudimentär und unterstützt nur einfaches Linienschreiben oder das Verwenden von Bitmap-Daten.

von Änderungen in dieser Welt wird durch Funktionen bereitgestellt. Der zweite Fall betrifft Funktionen wie unter anderem das Auslesen der Tastatur- oder Mauseingaben. Für diesen Zweck müssen eigene Implementierungen vorgenommen werden; damit diese Funktionen durch die OpenGL-Bibliotheken genutzt werden, müssen diese der Bibliothek während der Laufzeit bekannt gemacht werden. Das Bekanntmachen geschieht durch spezifizierte Funktionsaufrufe mit entsprechenden Parametersätzen. Solch eine Implementierung von zu übergebenden Funktionsnamen nennt man Callback-Routinen. Im Fall von GLUT – nur als Beispiel genannt – wären das unter anderem:

void glutDisplacFunc(void (*func)(void)) Der Parameter muss ein Zeiger auf eine Funktion mit keinem Rückgabewert und ohne Aufruf von Parametern sein. Die zu übergebende Funktion ist für das Zeichnen der 3-D-Welt zuständig[4].

glutKeyboardFunc(void (*func)(...)) Jeder Tastendruck ruft die bei dieser Funktion übergebene Funktionsimplementierung auf. Die xy-Koordinaten beschreiben die Position der Maus zum Zeitpunkt des Tastendrucks, relativ zum Anzeigefenster der 3-D-Welt.

8.2.1 Funktionen der BOINC-Grafikschnittstelle

Die BOINC-API für Grafikfunktionen wird in zwei Versionen geliefert, die erste, eine „veraltete“, nicht mehr von den Entwicklern gepflegte aber noch nutzbare Version [97] und die neuere, empfohlene zweite Version [80]. Wir werden uns in diesem Teil nur mit der zweiten Version beschäftigen. Die zweite Version wurde dem BOINC-Entwicklungszweig am 23. Mai 2007, Revision 12735[5], von David Anderson hinzugefügt. Die letzte geänderte Revision ist 24341 und enthält zwölf Funktionen, um die Verwaltung der Grafikdarstellung auf einem Host zu vereinfachen.

Prinzipiell hält sich die Umsetzung der Grafikschnittstelle stark an die GLUT-Prinzipien, intern werden Funktionen der GLUT-Bibliothek fast durchgehend genutzt. Das bedeutet, dass alle Funktionalitäten durch Callback-Routinen repräsentiert und implementiert werden müssen. In den nachfolgenden Abschnitten erhalten Sie eine Übersicht der Callback-Routinen, welche mindestens durch Sie zur Verfügung gestellt werden müssen. Weiterhin ist es Ihnen überlassen, welche Informationen – wenn überhaupt – Sie anzeigen lassen wollen, oder wie die grafische Darstellung aussehen soll. BOINC bietet Ihnen nur einen einfachen Entwicklungsrahmen, so dass Ihre Grafikanwendungen unter unterschiedlichen Systemumgebungen gestartet werden können.

8.2.1.1 Initialisierung einer BOINC-Grafikanwendung

Zu bedenken gilt, dass die zweite Version der BOINC-Grafikschnittstelle eine aktive Dauerschleife ist. Weiterhin sollte die Grafikanwendung als eigenständige Anwen-

[4] Es sei angemerkt, dass die Funktion allen möglichen Inhalt haben kann und nicht rein auf die Darstellung der 3-D-Welt beschränkt ist. Dies gilt unter anderem für alle Callback-Routinen.

[5] http://boinc.berkeley.edu/trac/changeset/12735/trunk/boinc/api/graphics2.h.

dung implementiert werden. Listing 8.1 liefert die Initialisierungsfunktion für die Grafikanwendung und erwartet drei zu übergebende Parameter. Die ersten beiden Parameter `argc` und `argv` sind der main-Funktion von C/C++ nachempfunden, sprich `argc` enthält die Anzahl der Parameter, die beim Starten mit übergeben, wurden und `argv` enthält die Parameter. `title` ist eine Zeichenkette, welche im oberen Bereich des Fensters für die Grafikausgabe angezeigt wird.

Listing 8.1 Funktion für das Initialisieren einer BOINC-Grafikanwendung

```
#include <api/graphics2.h>

void boinc_graphics_loop(int argc, char **argv, const char *title);

void app_graphics_init(void);
  /**
   * Muss von der Grafikanwendung angeboten werden
   * und ermoeglicht das Einstellen von eigenen
   * Laufzeitverhalten. Diese Funktion wird im Verlauf
   * der Abarbeitung von boinc_graphics_loop aufgerufen.
   */
```

Der Initialisierungsaufruf ruft intern die GLUT-Funktion `glutMainLoop()` auf. Sobald die Initialisierungsfunktion aufgerufen wurde, werden keine nachfolgenden Anweisungen abgearbeitet, da `glutMainLoop()` in eine Ereignisschleife übergeht und nur noch alle von BOINC festgelegten Callback-Funktionen aufruft.

Crux der GLUT-Initialisierung

Es hat sich während der Recherchen gezeigt, dass es womöglich zu einer kryptischen Fehlermeldung kommen kann, wenn Sie das BOINC-Framework mit OpenGL- und Transparenz-Support initialisieren wollen:

> Internal error <FBConfig with necessary capabilities not found> in function fgOpenWindow

Wenn Sie diesen Fehler erhalten, haben Sie zwei Möglichkeiten:

1. Installieren Sie Ihren Grafikkartentreiber neu, so dass eventuell aktuelle Versionen zur Verfügung stehen und Ihre Anwendung diese beim Kompilieren auch nutzt.
2. Die zweite Möglichkeit ist ein wenig aufwendiger und Sie müssen die BOINC-Quellen modifizieren[6]. Der Fehler kam bei mir mit der freeglut[7]-Version 2.6.0 auf. In der Datei `api/graphics2_unix.cpp` finden Sie den Aufruf von glutInitDisplayMode[8] und beim Aufruf kommentieren Sie `GLUT_ALPHA` aus, nun sollte das Starten funktionieren. Listing 8.2 zeigt den modifizierten Funktionsaufruf. Nun können Sie Ihre Anwendung starten, allerdings fehlt die Unterstützung für transparente Objekte innerhalb Ihrer Anwendung.

[6] http://visualgrid.de/wp/2011/10/11/internal-error-in-function-fgopenwindow/.

[7] http://freeglut.sourceforge.net/index.php#download.

[8] http://www.opengl.org/documentation/specs/glut/spec3/node12.html.

Listing 8.2 Modifikation der GLUT-Initialisierung ohne Transparenz

```
static void boinc_glut_init(int *argc, char** argv) {
    ...
    glutInit (argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH /* | GLUT_ALPHA */
        );
    glutInitWindowPosition(xpos, ypos);
    glutInitWindowSize(width, height);
}
```

Standardeinstellungen bei der Grafikinitialisierung

Die Initialisierung stellt folgende Standardwerte der OpenGL-Umgebung ein:

Tiefentest Durch das Aktivieren des Tiefentests werden Tiefenvergleiche durchgeführt und der Tiefenpuffer aktualisiert. Dies bedeutet, dass Objekte, welche durch andere überdeckt werden, nicht gezeichnet werden und dadurch eine höhere Performance der 3-D-Umgebung möglich ist. Dies wird intern durch die nachfolgende Anweisung ermöglicht: `glEnable(GL_DEPTH_TEST);`

gepufferte Fensterausgabe Von vornherein wird eine gepufferte Fensterausgabe aktiviert, so dass alle Modifikationen der grafischen Ausgabe in einem Rutsch angezeigt werden. Wenn die Ausgabe nicht gepuffert ist, so kann es ein starkes Flackern geben, was in der Regel nicht gewünscht ist.

Farbe und Transparenz Es werden Farben und Einstellungen für die Transparenz ermöglicht, dadurch können Objekte einen Alpha-Wert erhalten, so dass diese Hintergrundobjekte durchschimmern lassen. Nützlich unter anderem für Objekte, die Glas oder Wasser darstellen sollen.

Dimension der Grafikausgabe Das Grafikfenster wird standardmäßig an die Position $\text{position} = (100, 100)$ mit der Pixelgröße von $\text{dimension} = (600, 400)$ gesetzt. Diese Werte können überschrieben werden, indem man die Datei `gfx_info` zu einem Projekt hinzufügt und das Format aus Listing 8.3 besitzt.

Timer-Funktion Es wird alle $30 \cdot 10^{-3} s$ eine Timer-Funktion aufgerufen. Diese Timer-Funktion prüft kontinuierlich, ob die wissenschaftliche Applikation noch läuft oder nicht[9]. In der hier behandelten zweiten Version der BOINC-API-Grafikschnittstelle wird allerdings geprüft, ob schon eine Instanz eines Bildschirmschoners gestartet ist oder der BOINC-Manager eventuell nicht mehr läuft. In beiden Fällen wird die Ausführung der Grafikanwendung beendet, in der die Prüfung durchgeführt wird. Zusätzlich prüft die Timer-Funktion, ob ein Neuzeichnen der grafischen Szene nötig ist, und tut dies gegebenenfalls. Eine Fensterdarstellung, also nicht im Vollbildmodus der grafischen Ausgabe, wird daraufhin überprüft, ob das Fenster eine Position und Dimension besitzt. Sind diese Werte geändert, so werden sie in der `glx_info` mit dem Format aus Listing 8.3 abgespeichert. Bei einem Neustart der Grafikausgabe wird die vorherige Position und Dimension verwendet.

Standardhintergrund Wenn keine Grafikszene von Ihnen beschrieben wird, so wird eine Standarddatei mit dem logischen Namen `background` aus dem

[9] Diese Prüfung hat nur Einfluss auf die erste Version der BOINC-API-Grafikschnittstelle.

Dateiordner der Grafikanwendung versucht zu laden und angezeigt. Diese Datei kann in folgenden Dateiformaten vorliegen: JPEG, Bitmap, PPM, RGB und TGA. Es sei an dieser Stelle angemerkt, dass es sich bei dem Dateinamen um einen logischen Dateinamen handelt, welcher mit `boinc_resolve_filename()` aufgelöst wird (vgl. Abschn. 7.3.5).

Listing 8.3 Formatdefinition für die Datei `gfx_info`

```
xpos:=INTEGER ypos:=INTEGER width:=INTEGER height:=INTEGER
```

8.2.1.2 Tastatur- und Mauseingaben abfragen

Die BOINC-API bietet vier Funktionen an, mit deren Hilfe die Tastatureingaben und Mauseingaben inklusive Positionsbestimmung abgefragt werden können. Die Tastatureingabe kann dabei auf zwei unterschiedliche Ereignisse unterschieden werden: ein Tastendruck (engl. *press*) oder das Loslassen der Taste (engl. *release*). Für das Abfragen von Mausereignissen stehen auch zwei Funktionen bereit, eine für das Verarbeiten von Mausbewegungen (engl. *move*) und die zweite für das Abfragen, welche Taste der Maus gedrückt wurde.

Listing 8.4 Funktionsdefinitionen für das Verarbeiten von Tastatur- und Mauseingaben

```
#include <api/graphics2.h>

void boinc_app_key_press(int key, int param);
  /**
   * key enthaelt den Tastencode der gedrueckten Taste als Integer.
   *
   * param hat unterschiedliche Werte. Unter Linux ist der Wert
   * standardmaessig auf Null gesetzt, Windows uebergibt
   * den sog. 'LPARAM lParam'-Wert.
   */

void boinc_app_key_release(int key, int param);
  /**
   * key enthaelt den Tastencode der gedrueckten Taste als Integer.
   *
   * param hat unterschiedliche Werte. Unter Linux ist der Wert
   * standardmaessig auf Null gesetzt, Windows uebergibt
   * den sog. 'LPARAM lParam'-Wert.
   */

void boinc_app_mouse_button(int x, int y, int which, int is_down);
  /**
   * x und y sind die aktuelle Position des Mauszeigers,
   * relative zum Fenster des Mausereignisses.
   *
   * which beschreibt welche Taste der Maus gedrueckt wurde,
   * moegliche Werte sind: GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON
   * oder GLUT_RIGHT_BUTTON.
   *
   * is_down ist Eins, wenn die Taste gedrueckt ist,
   * ansonsten ist der Wert Null.
   */

void boinc_app_mouse_move(int x, int y, int left, int middle, int right);
  /**
   * x und y sind die aktuelle Position des Mauszeigers,
   * relative zum Fenster des Mausereignisses.
   *
   * left, middle, right geben Auskunft darueber, welche
```

```
    * Taste der Maus im Moment der Mausbewegung geklickt
    * ist. Die Werte sind exklusiv und es kann nur maximal
    * eine Maustaste gedrueckt sein. Der Wert Eins beschreibt
    * eine gedrueckte und der Wert Null eine nicht gedrueckte
44  * Maustaste.
    */
```

LPARAM lParam

Wenn Sie sich auskennen mit der Programmierung von Applikationen unter Windows, so können Sie diesen Teilabschnitt getrost überspringen. Das Microsoft Development Network (MSDN) liefert indirekt eine Beschreibung des zweiten Parameters `param`[10] aus dem Listing 8.4 für das Abarbeiten der Tastatureingaben. UNIX/Linux-Systeme setzen diesen Parameter standardmäßig auf Null, unter Windows hängt der Wert des Parameters vom Typ[11] des Ereignisses ab. In der Datei `api/graphics2_win.cpp` werden die Ereignisse abgehandelt. Es gilt zu beachten, dass `lParam` für die nachfolgend aufgelisteten Tastaturereignisse ein Bitfeld von sieben Werten beschreibt. Die Tab. 8.1 beschreibt die Struktur des Bitfeldes der gelisteten Tastaturereignisse. Nachfolgende Tastaturereignisse werden durch die Callback-Funktion `WndProc()` von Windows beachtet und entsprechende Werte für `lParam` gesetzt:

WM_KEYDOWN Das 30ste Bit ist Eins, wenn vor dem aktuellen Ereignis die Taste schon gedrückt war, ansonsten ist das Bit auf Null gesetzt.

WM_KEYUP Das 30ste Bit ist immer auf Eins gesetzt.

Tab. 8.1 Bitfeld für den Parameter `lParam` der Tastaturereignisse `WM_KEYDOWN` und `WM_KEYUP`

Bits	Beschreibung
0–15	Wiederholungsrate bei gedrückter Taste.
16–23	Diese 8 Bits enthalten den Wert, welchen die Tastatur liefert, wenn eine Taste gedrückt und losgelassen wird. Entsprechend dieser Definitionen von 8-Bit können maximal 256 Zeichen unterschieden werden. Durch die Kombination mit dem 24sten Bit kann dieser Wert um ein Vielfaches erweitert werden.
24	Gibt Auskunft, ob es sich um eine erweiterte Taste handelt, z. B. rechte ALT-Taste. Das Bit ist Eins, wenn es eine erweiterte Taste ist, ansonsten ist das Bit auf Null gesetzt.
25–28	Reserviert und sollte nicht benutzt werden.
29	Beim Tastaturereignis `WM_KEYDOWN` ist dieses Bit immer Null und bei `WM_KEYUP` immer Eins.
30	Gibt Auskunft über den Zustand der Taste vor dem aktuellen Ereignis.
31	Beim Tastaturereignis `WM_KEYDOWN` ist dieses Bit immer Null und bei `WM_KEYUP` immer Eins.

[10] http://msdn.microsoft.com/en-us/library/ms633573%28VS.85%29.aspx.

[11] http://msdn.microsoft.com/en-us/library/ms644927%28v=VS.85%29.aspx.

8.2.1.3 Zeichnen, Neuzeichnen, Größenänderung

Ein bewegte Grafikszene besteht aus mehr als einem Bild und auch bei einer Größenänderung des Anzeigefensters – sei es im einfachen Fenster oder im Vollbildmodus – bedarf es eines Neuzeichnens der grafischen Szenerie. Das Listing 8.5 beschreibt die zwei BOINC-API-Funktionen, um die Szenerie neu zu zeichnen und auf die Größenänderungen einzugehen. Sie als Entwickler müssen diese Funktionen implementieren, welche durch das BOINC-Framework während der Laufzeit aufgerufen werden. Sollte keine eigene Implementierung vorhanden sein, so werden Dummys verwendet, die keine ausgefeilte Grafikdarstellung nutzen, sondern einen einfachen Hintergrund anzeigen (siehe oben).

Listing 8.5 Funktionsdefinitionen für das Verarbeiten von Tastatur- und Mauseingaben

```
#include <api/graphics2.h>

void app_graphics_render(int xs, int ys, double time_of_day);
  /**
   * xs und ys beschreiben die Weite und Hoehe des aktuellen
   * Grafikfensters. time_of_day beinhaltet die Sekunden
   * seit 1970.
   */

void app_graphics_resize(int width, int height);
  /**
   * Sollte die Neuskalierung der grafischen Szene
   * durchfuehren, {\ua} das ordentliche Seitenverhaeltnis
   * der Perspektive. Wenn dies nicht ordentlich durchgefuehrt
   * wird, so kann es zu Verzerrungen in der Grafikszene kommen.
   * Die neue Groesse wird durch die zwei Parameter width
   * und height beschrieben.
   */
```

8.3 Hilfefunktionen für den OpenGL-Umgang

Die BOINC-API enthält in der Datei `api/gutil.h` einige OpenGL-Hilfsfunktionen. Es handelt sich dabei um Funktionen, um unter anderem in die verschiedenen OpenGL-Modi umzuschalten:

`mode_texture()` Das Erstellen von Texturen mit OpenGL ist ohne die Berechnung von Leuchteffekten empfohlen, weshalb diese Funktion u. a. die Lichtberechnung für alle Lichtquellen deaktiviert.

`mode_ortho()` Aktiviert die orthographische Berechnung von Koordinaten, d. h. es wird die Nutzung von zweidimensionalen Koordinaten aktiviert. Dies ist von Vorteil, wenn man eine Art Dashboard im vorderen Bereich der dreidimensionalen Szene hinzufügen will. Nach erfolgreicher Durchführung der orthografischen Darstellung müssen Sie `ortho_done()` aufrufen.

`mode_shaded(float*)` Aktiviert die Möglichkeit, Farbtiefen zu verwenden um, u. a. Schatteneffekte erstellen zu können.

`mode_unshaded()` Das Gegenteil der vorherigen Funktion, um Objekte und Darstellungen ohne Schatteneffekt zu erstellen.

`mode_lines()` Schaltet Blendungseffekte ein und die Beleuchtung aus, um so Linien ohne Effekte zeichnen zu können. Dies kann sinnvoll sein, wenn Sie vielleicht zweidimensionale Rahmen oder ein Gitter zeichnen wollen.

Zusätzlich existieren Funktionen, um die unterschiedlichen OpenGL-Matrizen zu erhalten:

`get_matrix(double*)` Diese Funktion ermittelt die Modelview-Matrix, ohne sie zu modifizieren.

`get_projection(double*)` Mit Hilfe dieser Funktion kann die Projektionsmatrix ermittelt werden.

`get_viewport(int*)` Diese Funktion speichert die Viewport-Matrix im übergebenen Integer-Feld.

`get_2d_positions()` Mit Hilfe dieser Funktion können die Objektkoordinaten in Fensterkoordinaten umgewandelt werden.

8.3.1 HLS/RGB-Konvertierungen

Die C++-Klasse `COLOR` dient dem Speichern eines RGB[12]-Wertes (Rot, Grün, Blau), und falls diese Informationen als HLS[13]-Werte (Hue, Lightness, Saturation) vorliegen, können diese mit der Funktion aus Listing 8.6 umgewandelt werden.

Listing 8.6 HLS-Werte in RGB-Werte konvertieren

```
#include <api/gutil.h>

struct COLOR {
  // r = Rot, g = Gruen, b = Blau, a = Alpha (Transparenz)
  float r, g, b, a;
  COLOR(float rr=0, float gg=0, float bb=0, float aa=0)
    : r(rr), g(gg), b(bb), a(aa)
  { }
};

void HLStoRGB(double H, double L, double S, COLOR& c);
```

8.3.2 Text in der Grafikszene

Listing 8.7 listet alle BOINC-API-Funktionen, mit deren Hilfe man Text in der Grafikszene hinzufügen kann. Es sei angemerkt, dass das Nutzen von Text in OpenGL keine triviale Angelegenheit ist. In der Regel ist es erforderlich, dass man seine

[12] [147] RGB ist ein additives Farbmodell, d. h., je mehr Rot, Grün und Blau Sie hinzufügen, umso mehr nähert sich die Farbe Weiß an. Der Wert für jeden „Kanal“ (Rot, Grün oder Blau) kann zwischen 0 (keine Farbe) und 255 (vollständig gesättigte Farbe) liegen.

[13] [147] Das HLS-Farbmodell (Hue, Lightness, Saturation) wird häufig als intuitiver empfunden, da die Farben auf Grundlage von Farbton, Helligkeit und Sättigung definiert werden. Der Farbton wird in Grad angegeben (0 bis 360 Grad), und Sättigung und Helligkeit werden mit Prozentwerten zwischen 0 und 100 Prozent angegeben.

gewünschte Schriftart durch einzelne Grafiken für jeden Buchstaben und jede Ziffer erstellt und erst dann innerhalb von OpenGL verwenden kann. Problematisch an dieser Technik ist u. a., dass nicht jede Schriftgröße durch eine Darstellung beschrieben werden kann; es müssen also für jede zu verwendende Schriftgröße einzelne Grafiken erstellt werden. Es wird an dieser Stelle empfohlen, die BOINC-Funktionen zu verwenden, welche eine erhebliche Vereinfachung darstellen. Weiterhin ist es theoretisch auch möglich, diese Funktionen in externen Anwendungen zu verwenden. Eine Beispielanwendung, die das Verwenden der in Listing 8.7 aufgeführten Funktionen nutzt, wird im Abschn. 8.4 vorgestellt. Die in Listing 8.7 aufgeführten Funktionen erwarten meist die gleichen Parameter:

pos Steht für Position und ist ein dreidimensionales Feld und enthält die x-, y- und z-Koordinate in der OpenGL-Szene.

height Ein Wert für die Texthöhe.

width Dieser Wert definiert die Linienstärke der einzelnen zu zeichnenden Characters.

text Eine zur OpenGL-Szene hinzufügende Character-Zeichenkette. Generell wird zumindest die Escape-Sequenz ‚$\backslash n$‘ gefiltert [158] und gesondert behandelt, mit Ausnahme der Funktionen `draw_text_line` und `draw_rotated_text` können das alle.

spacing Wenn eine mehrzeilige Textdarstellung von der Funktion unterstützt wird, so beschreibt dieser Wert den Abstand zwischen zwei Textzeilen.

Die Funktionendefinition zeigen u. a., dass nicht alle Parameter in den BOINC-Quellen verwendet werden, z. B. haben so die Funktionen `draw_text_right` und `draw_text_new_3d` die zwei Parameter `height` und `width`, diese sind allerdings auskommentiert und finden keine Verwendung, also kann ein arbiträrer Zahlenwert übergeben werden. Der Abschn. 8.4 enthält eine einfache Umsetzung eines Bildschirmschoners für ein BOINC-Projekt.

Listing 8.7 HLS-Werte in RGB-Werte konvertieren

```
#include <api/gutil.h>

#define TEXT_LEFT       0
#define TEXT_CENTER     1
#define TEXT_RIGHT      2

/**
 * Alle nachfolgenden Funktionen schreiben einen Text
 * in die OpenGL-Szene.
 */

void draw_text_line(
  float *pos, float height, float width,
  const char *text, int justify=TEXT_LEFT
);

void draw_text(
  float *pos, float height, float width,
  float spacing, const char *text
);

void draw_text_new(
  float *pos, float /*height*/, float /*width*/,
```

```
  float spacing, const char *text
);

void draw_text_right(
  float *pos, float height, float width,
  float spacing, const char *text
);
  /** Der Text wird rechtbuendig in die OpenGL-Szene geschrieben.
   */

void draw_text_new_3d(
  float *pos, float /*height*/, float /*width*/,
  float spacing, const char *text
);

void draw_rotated_text(
  float *pos, float height, float width,
  float /*spacing*/, const char *text,
  float rotation, float* rotation_vector
);
  /** Laesst einen Text um den Wert 'rotation' rotieren,
   * eine Bewebung ist erst in einem kontinuierlich
   * aufsteigenden oder fallenden Wert moeglich.
   * Der Parameter 'rotation_vector' ist ein dreidimensionales
   * Feld und definiert um welche Achse sich der Text drehen soll.
   */

void draw_text_panel(
  float *_pos, float *size, float margin, COLOR color,
  float char_height, float line_width, float line_spacing,
  const char *text
);
  /** Zeichnet ein Panel mit einer bestimmten Farbe
   * und enthaellt einen bestimmten Text.
   */
```

8.3.3 OpenGL-Helfer: Progressbar, Panels, Starfield

Es gibt noch weitere sehr hübsche Helfer, um u. a. den aktuellen Wert des Berechnungsstatus anzeigen zu können; dies kann mit einer einfachen Fortschrittsanzeige (engl. *Progressbar*) erfolgen. Das Listing 8.8 listet die Klassendefinitionen der OpenGL-Helfer auf und mit welchen öffentlichen Methoden diese initialisiert und genutzt werden können.

Listing 8.8 Kleine Helfer für den Umgang mit OpenGL und BOINC

```
#include <api/gutil.h>

class PROGRESS_2D {
public:
  void set_pos(float *);
  void init(float *pos, float len, float width,
            float inner_width, float *color, float *innerColor);
  void draw(float);
};

class RIBBON_GRAPH {
public:
  void set_pos(float*);
  void init(float *pos, float *size, float *color,
            float *tick_color, float tick_yfrac=0.2);
```

```
  void draw(float *data, int len, bool with_ticks=false);
  void add_tick(float x, int index);
};

class MOVING_TEXT_PANEL {
public:
  void init(float *pos, float *size, COLOR &color,
            double dtheta, double ch, double lw,
            double ls, double margin);
  void draw();
  void set_text(int lineno, const char *t);
  void get_pos(int lineno, float *pos);
  static void sort(MOVING_TEXT_PANEL *tp, int n);
  void move(double dt);
};

class STARFIELD {
public:
  void build_stars(int nstars, float movespeed);
  void update_stars(float delta);
};
```

8.4 Entwicklung eines Bildschirmschoners

Listing 8.9 zeigt das Minimum eines BOINC-Bildschirmschoners und Sie müssen die einzelnen Funktionsbereiche nun nur noch mit Inhalt füllen, der eine grafische Szene zaubert und evtl. mit dem Benutzer vor dem Bildschirm interagieren soll. Die Standardausgaben sind natürlich nicht nötig und ohnehin würden diese die Performance beeinflussen, da Ein-/Ausgabe teuer ist. Allerdings erhalten Sie mit Hilfe der Ausgaben die Möglichkeit zu sehen, wann die einzelnen Funktionen durch das BOINC-Framework aufgerufen werden.

Listing 8.9 Implementierung eines einfachen Rahmens für die Entwicklung eines BOINC-Bildschirmschoner

```
// BOINC
#include <api/boinc_api.h>
#include <api/graphics2.h>

// C++
#include <iostream>

void app_graphics_init() {
  std::cout << "app_graphics_init" << std::endl;
}
void app_graphics_render(int xs, int ys, double time_of_day) {
  std::cout << "app_graphics_render" << std::endl;
}
void app_graphics_resize(int width, int height) {
  std::cout << "app_graphics_resize" << std::endl;
}
void boinc_app_key_press(int key, int param) {
  std::cout << "boinc_app_key_press" << std::endl;
}
void boinc_app_key_release(int key, int param) {
  std::cout << "boinc_app_key_release" << std::endl;
}
void boinc_app_mouse_button(int x, int y, int which, int is_down) {
  std::cout << "boinc_app_mouse_button" << std::endl;
}
```

```
void boinc_app_mouse_move(int x, int y, int left, int middle, int right) {
  std::cout << "boinc_app_mouse_move" << std::endl;
}
int main( int argc, char **argv) {
  boinc_parse_init_data_file();
  boinc_graphics_loop(argc, argv, "simpleScreensaver");
}
```

Wenige Zeilen eines Makefiles führen zum Erfolg, wie dies in Listing 8.10 gezeigte Makefile mit Hilfe eines Aufrufs unter einem Linux-System beweist.

Listing 8.10 Wenige Zeilen eines Makefiles lassen einen BOINC-Bildschirmschoner entstehen

```
FLAGS=-O2 -Wall
INCLUDES=-I./boincsrc -I./boincsrc/api -I./boincsrc/lib
LIBARIES=-L./boincsrc/api -lboinc_api -lboinc_graphics2 \
  -L./boincsrc/lib -lboinc -lGL -lGLU -lglut -ljpeg -lpthread

all: ${S}
  g++ ${FLAGS} -o simpleScreensaver main.cpp ${INCLUDES} ${LIBARIES}
```

8.4.1 Bewegungen in der OpenGL-Szene

Ich möchte Ihnen natürlich nicht nur eine Liste aller verfügbaren BOINC-Funktionen geben, denn wie sagt schon ein Sprichwort: *„Ein Bild sagt mehr als tausend Worte“*. In diesem Abschnitt erstellen wir uns eine kleine Bildschirmschonapplikation und werden sehen, wie einfach das ist, dank BOINC. Das Listing 8.11 enthält Funktionen aus dem vorherigen Listing 8.9 mit zusätzlichen Anweisungen für die Darstellung von verschiedenen grafischen Elementen.

Im Listing finden Sie innerhalb der Funktion `app_graphics_render()` fünf Beispiele, um mit den BOINC-Grafikfunktionen arbeiten zu können.

1. Beispiel Es werden in einer for-Schleife 11 Zeilen desselben Textes dargestellt. Das Anzeigen wird durch die Zeile 27 vorgenommen, der y-Wert ist im Feld p durch den Index Eins adressierbar und wird nach jedem Schleifendurchlauf ersetzt, so dass der Text an der richtigen Position dargestellt wird.

2. Beispiel Eine Implementierung eines einfach rotierenden grünen Texts. Die Achse, um die der Text rotieren soll, ist im Feld `r` mit dem Wert $1.0f$ definiert, d. h. es findet eine Rotation um die y-Achse statt. Die Geschwindigkeit der Rotation können Sie selbst bestimmen, Sie müssen nach jedem Durchwahl nur einen passenden Wert als sechsten Parameter übergeben. In diesem Beispiel wird eine kontinuierliche Drehung um zehn Grad verwendet. Der rotierende Text ist rechts-mittig in der Ergebnisabbildung zu sehen (Abb. 8.1), allerdings mit einer kleinen Verzerrung, da der Text im Moment der Aufnahme eine kleine Rotation besitzt.

3. Beispiel Dieses Beispiel demonstriert das Koordinatensystem unserer OpenGL-Szene. Es wird in den jeweiligen Ecken eine Beschreibung ausgegeben, die jeweiligen Koordinaten, an denen der Text erscheinen soll, werden durch ein einfaches Feld in Zeile 45 ermöglicht. In Folge werden die einzelnen Werte der Achsen neu gesetzt und für das Schreiben eines Textes verwendet.

4. Beispiel Wenn Sie das Weltall mögen, dann werden Sie sicher Freund der Klasse `STARFIELD`. Zum Anfang wird in Zeile 1 eine statische Variable erzeugt, welche in Zeile 5 initialisiert und in Zeile 56 kontinuierlich aufgerufen wird, so dass die Sterne neu gezeichnet werden; dadurch ergibt sich der Eindruck, dass wir geradeaus in den Bildschirm hineinfliegen. Nach dem Aktualisieren der Sterne müssen wir `glPointSize()` aufrufen, so dass es im Verlauf nicht zu Darstellungsproblemen kommt. Intern wird von der Zeichenroutine für die Sterne `glPointSize()` aufgerufen, allerdings ein vorheriger Wert nicht wiederhergestellt, so dass es womöglich zu Problemen von nachfolgenden OpenGL-Routinen kommt.

5. Beispiel Das letzte Beispiel zeichnet eine Fortschrittsanzeige im oberen mittleren Bereich der OpenGL-Szene und wird bei jedem Neuzeichnen der Szenerie um einen Wert erhöht. Diese Erhöhung ist hier noch hard-codiert, kann allerdings auch die Informationen des Fortschritts einer wissenschaftlichen Applikation anzeigen. Dazu müssen Sie nur die Funktionen für die Kommunikation mit Shared-Memory aus Abschn. 7.3.14 für Ihre Implementierung zu Rate ziehen.

Listing 8.11 Beispielverwendung der BOINC-Grafikbibliothek

```
static STARFIELD stars;
static PROGRESS_2D progress;

void app_graphics_init() {
  stars.build_stars(500, 1.f);

  float progressPosition[3] = { -0.25f, 1.f, 0.f };
  float color[3] = { 0.6f, 0.6f, 0.6f };
  float innerColor[3] = { 0.f, 0.f, 0.f };
  progress.init(progressPosition, 0.5f, 0.1, 0.1f, color, innerColor);
}

void app_graphics_render(int xs, int ys, double time_of_day) {
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glLoadIdentity();
  glClearColor(1.0f, 1.0f, 1.0f, 0.0f);

/* 1st Example: 11 lines of text
 */
  glColor3f(0.f, 0.f, 1.f);
  float p[3] = { -1.0f, 1.0f, 0.0f };
  int index = 0;
  for(float y=0.5f; y>-0.5f; y-=0.1f) {
    p[1] = y - 0.1f;
    char buf[64] = {'\0'};
    snprintf(buf, 64, "%02d: www.christianbenjaminries.de", ++index);
    draw_text_line(p, 0.05f, 1.0f, buf);
  }

/* 2rd Example: rotating text
 */
  glColor3f(0.f, 1.f, 0.f);
  float p2[3] = { 0.4f, 0.0f, 0.0f };
  float r[3] = { 0.0f, 1.0f, 0.0f };
  static float angle = 0.0f; angle += 10.0f;
  draw_rotated_text(p2, 0.1f, 1.0f, 0.0f, "Hello World!", angle, r);

/* 3th Example: show edge coords
 */
  #define FHEIGHT 0.05f
  #define FWIDTH 2.f
```

```
  #define SPACING 0.05f

  glColor3f(0.0f, 0.0f, 0.0f);
  float p3[3] = { -1.f, 0.95f, 0.f };
  draw_text(p3, FHEIGHT, FWIDTH, SPACING, "oben-links (-1.f, 1.f)");
  p3[0] = 0.43f;
  draw_text(p3, FHEIGHT, FWIDTH, SPACING, "oben-rechts (1.f, 1.f)");
  p3[0] = -1.f; p3[1] = -0.95f;
  draw_text(p3, FHEIGHT, FWIDTH, SPACING, "unten-links (-1.f, -1.f)");
  p3[0] = 0.38f;
  draw_text(p3, FHEIGHT, FWIDTH, SPACING, "unten-rechts (1.f, -1.f)");

/* 4th Example: starfield
 */
  stars.update_stars(50.f);
  glPointSize(1.0f);

/* 5th Example: progressbar
 */
  static float progressValue = 0.0f;
  progressValue += 0.01f;
  if(progressValue > 1.0f) progressValue = 0.0f;
  progress.draw(progressValue);
  glColor3f(1.f, 1.f, 1.f);
  float progressTextPosition[3] = { -0.23f, 0.93f, 0.f };
  char buf[12] = {'\0'};
  snprintf(buf, 12, "%0.0f %%", progressValue*100.f);
  draw_text(progressTextPosition, FHEIGHT, FWIDTH, SPACING, buf);
}

void app_graphics_resize(int width, int height) {
  glViewport(0, 0, width, height);
}
```

Das Ergebnis aus Listing 8.11 finden Sie in Abb. 8.1.

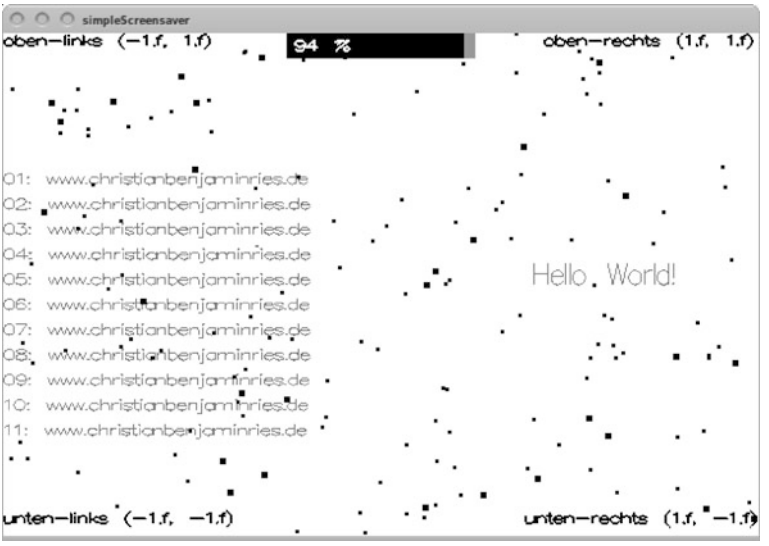


Abb. 8.1 Darstellung eines einfachen Bildschirmschoners ausschließlich auf Basis der Grafikfunktionen von BOINC, zu erkennen ist die eher anspruchslose Lösung des Rendering. Es liegt nicht am Druck, die Darstellung ist wirklich nicht sehr schön

Kapitel 9
Der BOINC-Server als Schaltzentrale

Der Computer arbeitet deshalb so schnell, weil er nicht denkt.

Gabriel Laub, 24.10.1928–03.02.1998
Polnisch-deutscher Schriftsteller

9.1 Sicherheitsaspekte und Authentifizierung

Die Kommunikation zwischen den Benutzern, Simulanten und Anwendern mit dem Projektserver findet auf zwei Wegen statt. Zum einen existiert eine Konfiguration für eine sichere Verbindung, durch das Hinzufügen des Secure Socket Layer (SSL) zum Apache-HTTP-Server (Hypertext Transfer Protocol). SSL wird in diesem Fall als Modul hinzugefügt und kann sofort für die gesamte Webserverkommunikation verwendet werden [70]. Zum anderen findet eine unverschlüsselte Kommunikation statt, die für die Benutzer vorhanden ist.

Die sichere Verbindung dient der Erstellung von Parameterstudien. Das Erstellen von Parameterstudien bedarf besonderer Zugriffsrechte, weshalb Authentifizierungsdaten zwischen dem Simulanten und dem Projektserver transferiert werden. Das HTTP ist ein Klartextprotokoll, d. h. jeder der auf der Netzwerkverbindung den Datentransfer abhört, kann diese Authentifizierungsdaten mitlesen und für sich verwenden und so das System korrumpieren.

Die Benutzer benötigen keine sichere Verbindung. Das BOINC-Framework ist von der Architektur so beschaffen, dass keine sicherheitskritischen Daten versendet werden. Jeder am BOINC-Projekt teilnehmende Benutzer wird durch einen Hash-Wert[1] eindeutig beschrieben. Durch diesen Hash-Wert kann ein Host in Kombination mit der E-Mail-Adresse des Benutzers, respektive der des Simulanten bei vorgegebener Installation, eindeutig als Benutzer innerhalb eines Projekts zugewiesen werden. Sollte der Wert nicht zu dem in der Datenbank passen, wird der Zugriff zum Projektscheduler abgelehnt. Dadurch ist die Sicherheit gegen unerlaubte Zugriffe gewährleistet.

Die Authentifizierung des Benutzers geschieht wiederum durch einen Hash-Wert, basierend auf einem MD5-Wert (Message Digest 5) und wird aus der zusammengesetzten Zeichenkette aus Passwort und E-Mail-Adresse gebildet. Das Listing 9.1 zeigt den C++-Programmausschnitt zur Erstellung dieses Hash-Wertes.

[1] Dieser Hash-Wert bildet sich aus dem Domainnamen, der IP-Adresse (Internet Protocol), dem freien Speicherplatz auf der Festplatte und der aktuellen Zeit.

C. B. Ries, *BOINC*, Xpert.press,
DOI 10.1007/978-3-642-23382-8_9, © Springer-Verlag Berlin Heidelberg 2012

Zeile 12 zeigt außerdem, wie dieser Wert in Verbindung mit dem Benutzernamen zur Authentifizierung des Benutzers durch eine SQL-Abfrage realisiert wird.

Der Benutzer muss nicht für alle Anfragen die Authentifizierungsdaten mitgeben. Bei falschen Authentifizierungsdaten wird eine entsprechende Fehlermeldung ausgegeben. Diese weist allerdings nur darauf hin, dass die Kombination aus Benutzernamen, E-Mail-Adresse und Passwort nicht mit den Daten in der Datenbank übereinstimmt.

Weitere Sicherheitsrestriktionen sind denkbar. Der Netzwerkadressbereich für die Simulanten könnte eingeschränkt werden, so dass nur bestimmte Abteilungen in einer Firma oder Institution auf die Administrationsschnittstellen zugreifen können. Diese Art der Absicherung wurde in diesem Buch nicht weiter beachtet. Sie hat nichts mit dem *BOINC-Framework* zu tun, sondern kann durch weitere Funktionen abgedeckt werden, u. a. Betriebssystemfunktionen wie *$ipchains* oder *$iptables*.

Listing 9.1 Bildung eines Hash-Wertes für ein Passwort zur Authentifizierung an einem BOINC-Projekt

```
1  ...
   remove_quotes(username);
   remove_quotes(email);
   remove_quotes(password);

6  hashwrapper *h = new md5wrapper();
   if(h == NULL) {
     return 0;
   }
   password = h->getHashFromString(password+email);
11
   std::string q = "where name='" + username
                 + "' and passwd_hash='" + password + "'";
   ...
```

9.2 Arbeitspakete

Es gibt zwei Möglichkeiten, Arbeitspakete für eine wissenschaftliche Applikation zu erstellen. Dabei ist es wichtig, dass die wissenschaftliche Applikation erfolgreich zu einem BOINC-Projekt hinzugefügt wurde und von BOINC-Teilnehmern heruntergeladen werden kann.

9.2.1 Skript zur Erstellung von Arbeitspaketen

Das BOINC-Framework liefert wieder ein Werkzeug mit, durch das die Erstellung von neuen oder weiteren Arbeitspaketen erleichtert wird. Mit Hilfe von

```
boincadm@boinc-testserver:~/projects/tah$ ./bin/create_work
```

können Arbeitspakete (engl. *Workunits*, Kurzform *WU*) erstellt werden. Bevor ein Aufruf dieses Werkzeugs getätigt werden kann, müssen einige Vorarbeiten durchge-

führt werden. Je nach Typ des WU kann das in viel Arbeit ausarten. Grundsätzlich ist es ratsam, die Eingabedateien mit den Parametern für die jeweiligen wissenschaftlichen Applikationen im Vorfeld zu erstellen. In dem Fall kann ein generalisiertes Skript erstellt werden, das diese Dateien in einer Schleife abarbeitet und zu der jeweiligen wissenschaftlichen Applikation hinzufügt.

9.2.1.1 Parametersatz für create_work

Der Parametersatz für *create_work* ist nicht trivial zu verinnerlichen. Es ist auch nicht nur ein einziger Aufruf von *create_work* nötig, um WUs zu einem BOINC-Projekt hinzuzufügen. Bei create_work gilt Folgendes zu beachten:

- Es dürfen und können keine WUs erstellt werden, die einen duplizierten Eigennamen besitzen.
- Der Aufruf muss im jeweiligen Projektordner aufgerufen werden.

Sie müssen regelrecht ein eigenes Programm um *create_work* implementieren. Aber alles der Reihe nach. In Listing 9.2 haben Sie die von *create_work* erwarteten Aufrufparameter.

Listing 9.2 Skript zum Erstellen von Arbeitspaketen

```
./bin/create_work -appname ${PROJECTNAME} \
      -wu_name ${WUNAME} \
      -wu_template templates/input.tpl \
      -result_template templates/result.tpl \
      ${INFILE_01} ... ${INFILE_XX}
```

Das Listing 9.2 enthält mindestens drei Variablen, welche Sie zu setzen haben. Für die Parameter *-wu_template* und *-result_template* könnten zudem Variablen hinzugefügt werden.

`PROJECTNAME`
Der Name eines BOINC-Projekts, zu dem Sie Arbeitspakete hinzufügen wollen.

`WUNAME`
Der eindeutige Name einer WU. Dies kann zum Beispiel durch eine fortlaufende Identifikationsnummer oder das Verwenden einer Zufallszahl ermöglicht werden. Sinnvoll ist die Verwendung eines Namens, der das Simulationsmodell und eventuell den Parametersatz beschreibt, zum Beispiel *core_shell_nanoparticles_$ID* bei Spinhenge@home.

`INFILE_01 ... INFILE_XX`
In der Regel besitzen die wissenschaftlichen Applikationen Eingabedateien, mit denen gerechnet werden soll. Diese Dateien werden als letzte Parameter dem Aufruf *create_work* mitgegeben. Die Reihenfolge definiert sich durch die Angaben in der Eingabeschablone, hier durch den Parameter *-wu_template* übergeben. Das in Listing 9.3 gezeigte Anwendungsbeispiel nutzt die Konfigurationseinstellungen aus der Datei *configuration.xml* während der Laufzeit der wissenschaftlichen Applikation.

Mit dem Aufruf von *create_work* können x-beliebige und unendlich[2] viele Arbeitspakete zu einem BOINC-Projekt hinzugefügt werden. Das Listing 9.3 zeigt, wie ein Programm beziehungsweise Skript aussehen kann, um x-beliebige WUs zu erstellen. Ein weiteres Beispiel finden Sie in Listing 11.9 des Praxisbeispiels für die statistische Berechnung der Kreiszahl π.

Listing 9.3 Programm beziehungsweise Skript zur Erstellung von Arbeitspaketen (engl. *Workunits*)

```
#!/bin/bash
ROOTPROJECT=${HOME}/projects/pah
CFGFILE=`pwd`/configuration.xml

for i in {211..220}
do
  cd ${ROOTPROJECT}
    cp ${CFGFILE} `bin/dir_hier_path configuration.xml`
    cmd="./bin/create_work -appname pingathome \
      -wu_name pingathome_wu_${i} \
      -wu_template templates/PingAtHome-input.tpl \
      -result_template templates/PingAtHome-result.tpl \
      configuration.xml"
    echo $cmd
    $cmd
  cd -
  echo ""
done
```

Das Beispiel-Skript aus Listing 9.3 erstellt zehn neue Arbeitspakete und fügt sie dem BOINC-Projekt Ping@home[3] zu. Die Variable ROOTPROJECT enthält den Pfad zum Dateiordner des BOINC-Projektes und CFGFILE enthält den Pfad zur Konfigurationsdatei, welche zur Laufzeit von der wissenschaftlichen Applikation genutzt wird. Innerhalb der for-Schleife wird in den BOINC-Projektordner gewechselt, wo dann *create_work* aufgerufen wird. Nach dem Erstellen der WUs wird in das Verzeichnis des Orts der Abspeicherung des gezeigten Skripts zurückgewechselt. Der Kopieraufruf in Zeile 8 kopiert die Konfigurationsdatei *configuration.xml* in ein Hierarchieverzeichnis des BOINC-Projekts für herunterzuladende Dateien. Das dabei verwendete Tool *bin/dir_hier_path* ist ein von BOINC mitgeliefertes Werkzeug für das automatische Ermitteln eines passenden Unterordners innerhalb der Download-Hierarchie.

9.2.2 Erstellung von Arbeitspaketen

Weiterhin kann der Entwickler ein eigenes Werkzeug für die Erstellung von Arbeitspaketen implementieren. Das BOINC-Framework bietet dafür Funktionsrümpfe an, die in die eigenen Entwicklungszweige mit einbezogen werden können. Ein solches Verfahren ist zum Beispiel dann wünschenswert, wenn nicht von vornherein alle Arbeitspakete vorhanden sind und erst nach und nach erstellt werden. Sicherlich, es

[2] Natürlich sind wir auf die physikalischen Grenzen beschränkt.

[3] Ping@home ist ein nicht-öffentliches BOINC-Projekt für die Zeitmessung von Nachrichtentransfers zwischen BOINC-Teilnehmern.

ist möglich, diese mit dem Skript aus Listing 9.3 zu erstellen. Es kann allerdings auch einfacher und schneller gehen, wenn man das Ganze in C/C++ umsetzt.

Listing 9.4 zeigt den überaus langen – und komplizierten – Aufruf der Funktion `create_work` zum Erstellen von Arbeitspaketen. Für den Anfang lassen wir die letzten beiden Parameter `command_line` und `additional_xml` aus, da diese sowieso NULL gesetzt sind und nur optional sind.

Listing 9.4 Funktionsdefinition `create_work` zum Erstellen von Arbeitspaketen

```
int create_work(
  DB_WORKUNIT &wu,
  const char *wu_template,
  const char *result_template_filename,
  const char *result_template_filepath,
  const char **infiles,
  int ninfiles,
  SCHED_CONFIG &sched,
  const char* command_line = NULL,
  const char* additional_xml = NULL
);
```

Es bleiben uns noch sieben Parameter, einige erkennen wir aus dem `make_work`-Skript wieder, andere sind allerdings unbekannt, aber wichtig für den Aufruf der Funktion.

`wu`
: Dieser Parameter versteht sich womöglich von selbst, denn *create_work* soll eine Arbeitspaket erstellen und `wu` ist der Parameter, welcher die Informationen und Beschreibungen dieses neuen Arbeitspakets enthält. Sie kreieren sich also eine Datenstruktur mit Informationen über das Arbeitspaket, z. B. der Name des neuen Arbeitspakets, übergeben diese Informationen dem Funktionsaufruf und lassen die Informationen in die Datenbank eintragen.

`wu_template`
: Dieser Parameter enthält den Inhalt der von Ihnen definierten Eingabedatei und muss unbedingt die Beschreibung der Eingabedateien enthalten, sprich wie viele Dateien vorhanden sind und mit welchen Dateinamen diese von einer wissenschaftlichen Applikation geöffnet werden können. Das Format der Eingabedatei finden Sie im Kap. 15.3.

`result_template(_filename und _filepath)`
: In einigen Beispielen einer solchen Anwendung zur Erstellung von Arbeitspaketen ist dieser Parameter und der nachfolgende Parameter `result_template_filepath` mit den gleichen Werten besetzt. Das suggeriert, dass es sich hierbei um eine identische Eigenschaft handelt; das ist aber nicht der Fall.
  Im ersten Fall wird der Name in die Datenbank in das Tabellenfeld `result_template_file` mit übernommen und geprüft, ob der Dateiname nicht eventuell zu lang ist, die maximale Länge beträgt *BLOB_SIZE=65536-1* Zeichen; dies sollte für Ihre Zwecke ausreichen. Wie Sie sehen, wird nicht geprüft, ob eine Datei vorhanden ist; beachten Sie, dass sich jegliche Dateien in der Download-Hierarchie Ihres BOINC-Projekts befinden müssen (vgl. Listing 9.5, Zeile 81).
  Der zweite Parameter wird nur auf die Möglichkeit zum Lesen der Datei geprüft; die Datei muss daher existieren. Wie schon erwähnt muss die Datei sich in der

Download-Hierarchie befinden. Es könnten völlig abstruse Pfade verwendet werden und die Funktion würde auch nicht zwingend einen Fehler melden und die Arbeitspakete wären in der Datenbank eingetragen; allerdings würden Fehlermeldungen aufkommen, wenn ein BOINC-Teilnehmer sich Arbeitspakete herunterladen will, diese wären schlicht nicht vorhanden. Daher einfach immer erst die Dateien in die Download-Hierarchie kopieren (vgl. Listing 9.5, Zeile 81) und danach `create_work` mit den passenden Pfaden aufrufen.

`infiles`
Enthält die Informationen der Eingabedateien für ein Arbeitspaket; diese können sich an einem beliebigen Ort befinden und zur Aufrufzeit zugreifbar sein, denn *create_work* prüft die Existenz dieser Dateien und erstellt einen MD5-Hashwert, um während der Laufzeit zu prüfen, ob eine Eingabedatei korrumpiert wurde.

`ninfiles`
Die Anzahl der Eingabedateien, die in `infiles` vorliegen und zu einem Arbeitspaket gehören.

`sched`
Dieser Parameter erwartet die Scheduler-Konfiguration, um u. a. zu finden, wo die Download-Hierarchie vorliegt, wenn eine gleichnamige Datei vorliegt, der MD5-Hashwert der vorhandenen Datei verwendet werden soll[4] und die URLs definiert sind, wo ein Arbeitspaket herunterzuladen/hochzuladen ist.

9.2.2.1 create_work Beispiel und Erläuterung

Listing 9.5 veranschaulicht, wie *create_work* in Ihrer Applikation genutzt werden kann. Das Beispiel erstellt Arbeitspakete mit einer Eingabedatei, ein weiteres Beispiel in Listing 9.7 zeigt, wie Sie Arbeitspakete mit mehr als einer Eingabedatei erzeugen können.

Erläuterung eines einfachen Beispiels

Nachfolgende Betrachtungen beziehen sich auf das Listing 9.5 und geben einen Einblick, wie `create_work` in Ihren eigenen Applikationen genutzt werden kann. Das Beispiel ist voll funktionsfähig; es ist allerdings teilweise um einige Zeilen gekürzt, wenn es zum Beispiel um eine Fehlerverarbeitung oder die Anzeige von mehreren Textzeilen geht. Start der Ausführung wird mit der obligatorischen main-Funktion eingeleitet, die Zeilen 137ff zeigen, wie mit Hilfe der BOINC-API und der darin enthaltenen Funktion `is_arg` übergebene Startparameter verarbeitet werden können. Die Implementierung ist trivial; es werden folgende Parameterformen erkannt:

[4] Sehr sinnvoll, um die Ausführungszeit von *create_work* zu erhöhen, denn Festplattenzugriffszeiten sind teuer und das Erstellen von MD5-Hashwerten, gerade bei größeren Dateien, sehr zeitaufwendig.

param Intern wird vor jeden übergebenen Parameter ‚–‘ gesetzt, und der Zeichenkettenvergleich prüft daraufhin auf die Existenz von ‚*–param*‘.

–param Auch bei dieser Form wird ‚–‘ vor den Parameter gesetzt und es folgt daraus ‚*—param*‘. Der Zeiger auf die Zeichenkette wird allerdings nach rechts verschoben, so dass eine Prüfung auf die Existenz von ‚*–param*‘ erfolgt.

Die Startparameter sollen beschreiben, wie der Inhalt der neuen Arbeitspakete auszusehen hat. Für diesen Zweck haben wir im oberen Bereich vier Variablen, um das Laufzeitverhalten eines Berechnungsmodells zu definieren:

`runs`
Anzahl der Gesamtberechnungsdurchläufe.

`startIteration, endIteration, stepIteration`
Diese drei Variablen beschreiben grundsätzlich eine for-Schleife in Zeile 34, wobei von `startIteration` bis hoch zu `endIteration` hochgezählt wird; in Schritten von `stepIteration`. Innerhalb dieser for-Schleife werden die angesprochenen Eingabedateien mit folgendem Dateinamenformat erzeugt: `wu_%02d_%06d_configuration.xml`. Der erste Platzhalter wird ersetzt durch die Identifikationsnummer der wissenschaftlichen Applikation, für die die Arbeitspakete bestimmt sind. Mit dem zweiten Parameter wird ein durch ein Inkrement beschriebener Zahlenwert genutzt, so dass die Eingabedateien eindeutige Namen besitzen.

Die Parameter sind an das Praxisbeispiel aus Kap. 11 angelehnt und sagen aus, wie viele Iterationen eine Berechnung haben soll und wie viele komplette Berechnungen ausgeführt werden sollen. Die einzelnen Iterationsschritte werden in einer Eingabedatei abgespeichert, welche später von der wissenschaftlichen Applikation eingelesen wird. Das Ergebnis eines solchen Schritts, exemplarisch für eine Datei, finden Sie in Listing 9.6.

Innerhalb der main-Funktion wird `letsGo` aufgerufen, hauptverantwortlich für die Erstellung der Arbeitspakete. Die zuvor erwähnte Identifikationsnummer der wissenschaftlichen Applikation muss ermittelt werden, denn Ihnen ist in der Regel nur der Name der Applikation bekannt. Zeile 60 erstellt ein Objekt, um die Tabelleninformation einer Applikation abfragen zu können, woraufhin in der nächsten Zeile eine Abfrage `where name='kreiszahl'` durchgeführt wird. Nach erfolgreicher Abfrage liegt die Information in `app.id` vor.

Jedes Arbeitspaket muss innerhalb eines BOINC-Projekts eindeutig referenzierbar sein. Dafür muss sich der Name des Arbeitspakets, abgespeichert in `wu.name`, von allen anderen bisher erstellten Arbeitspaketen unterscheiden. In dem Fall, dass wir nicht alle Arbeitspakete von vornherein erstellen, könnten wir die Anzahl der bisher erstellten Arbeitspakete ermitteln, diesen Wert inkrementieren und daraufhin für die nachfolgende Erstellung verwenden. Es sei angemerkt, dass dies natürlich keine perfekte Lösung darstellt, aber wenn Sie schon wissen, dass Sie die Arbeitspakete sequentiell erstellen, so beschreibt die Anzahl der bisher erstellen Arbeitspakete plus Eins immer einen neuen, eindeutigen Wert. Wenn Sie auf Nummer sicher gehen wollen, so können Sie den gezählten Wert in einen neuen Arbeitspaketnamen

umwandeln, prüfen, ob dieser existiert, und dann im Bedarfsfall dies so lange wiederholen, bis Sie einen neuen Namen erhalten. In den Zeilen 69ff werden eine Integervariable für diese Information sowie, eine SQL-Abfrage erstellt, um mit Hilfe der Struktur-Methode `count` die Anzahl der von der SQL-Abfrage betroffenen Datenreihen zu zählen. Intern wird diese Methode zur SQL-Abfrage

```
SELECT COUNT(*) FROM workunit WHERE appid=SIEHEOBEN
```

umgewandelt und ausgeführt. Weitere Informationen zu der BOINC-API, um eine Erweiterung der Datenbankabstraktion umzusetzen, finden Sie in Abschn. 9.5.1.

In der Variablen `infilesNames` haben wir unsere Dateinamen der neuen Eingabeparameter zwischengespeichert. Die Dateien müssen in die Download-Hierarchie kopiert werden. BOINC liefert für diesen Zweck eine C-Funktion `dir_hier_path` mit, so dass das Einordnen in einen passenden Dateiordner automatisiert wird. Zeile 81 übernimmt diese Arbeit für uns, es wird die Struktur-Methode `download_path` aufgerufen; intern wird diese zu `dir_hier_path`; und voilà haben wir einen Pfad zu einem passenden Unterordner in unserer Download-Hierarchie, z. B.

```
/home/boincadm/spin_data/download/234
```

dort können wir nun eine der Eingabedateien hineinkopieren, wie es in Zeile 83 durchgeführt wird.

Die Zeilen 89ff beschreiben ein neues Arbeitspaket, und auch die oben erwähnte Identifikationsnummer der wissenschaftlichen Applikation wird hinzugefügt. Zusätzlich werden einige Zahlenwerte gesetzt; der hier gezeigte Teil sind die Werte, welche Sie mindestens definieren müssen. Die vom Parameter `wu_tpl` beschriebene und physikalisch vorliegende Eingabeschablone wird im Vorfeld eingelesen und die dort enthaltene Konfiguration in die wu-Struktur übernommen. Daraufhin werden die in Ihren Zeilen definierten Zahlenwerte den einzelnen Bereichen zugeordnet und für die Arbeitspaketerstellung verwendet. Das heißt, die hier vorhandenen Einstellungen müssen entweder in der Eingabeschablone oder als C-Anweisung vorliegen!

In Zeile 102 werden die jeweiligen Arbeitspakete erstellt und schlussendlich werden in Zeile 111 die Eingabe- und Ergebnisschablonen in den BOINC-Projektordner mit den Schablonen `templates/` kopiert; dies ist wichtig, weil die Validierungs- und Assimilierungsprozesse zumindest die Ergebnisschablone benötigen, um zu prüfen, ob alle benötigten Ergebnisdateien auch vorliegen.

Listing 9.5 Beispielanwendung für das Erstellen von Arbeitspaketen mit einer Eingabedatei

```
// BOINC
#include <tools/backend_lib.h>
#include <sched/sched_util.h>
#include <lib/error_numbers.h>
#include <lib/filesys.h>
#include <db/boinc_db.h>

// C++
#include <iostream>
#include <string>
#include <vector>
```

```
unsigned int runs = 0;
unsigned int startIteration = 0;
unsigned int endIteration = 0;
unsigned int stepIteration = 0;

#define APPNAME "kreiszahl"
#define WUINPUTNAME "wu_%02d_%06d_configuration.xml"
#define WUNAME "wu_%02d_%06d"

/**
 * Creates input files for one scientific application.
 * @param infiles Name of the created files are stored in it.
 * @param appid Identification of scientific application.
 */
void createInfiles(std::vector<std::string> &infiles, int appid) {
  char xmlfmt[] = "<configuration>\n"
    "<runs>%d</runs>\n"
    "<iterations>%d</iterations>\n"
    "</configuration>\n";
  char xmlres[512] = {'\0'};

  for(unsigned int iter=startIteration, index=1;
      iter <= endIteration;
      iter += stepIteration, index++)
  {
    snprintf(xmlres, 512, xmlfmt, runs, iter);
    char filename[512] = {'\0'};
    snprintf(filename, 512, WUINPUTNAME, appid, index);
    FILE *f = fopen(filename, "wb");
    if(f) {
      fprintf(f, "%s", xmlres);
      fclose(f);
      infiles.push_back(filename);
    }
  }
}

/**
 * @return Exit code
 */
int letsGo() {
  SCHED_CONFIG config;
  config.parse_file();

  boinc_db.open(config.db_name, config.db_host,
                config.db_user, config.db_passwd);

  DB_APP app;
  app.lookup("where name='"APPNAME"'");

  std::vector<std::string> infilesNames;
  createInfiles(infilesNames, app.id);

  DB_WORKUNIT wu;
  wu.clear();

  int wu_count = 0; char q[32] = {'\0'};
  sprintf(q, "where appid=%d", app.id);
  wu.count(wu_count, q);

  for(unsigned int index=0; index < infilesNames.size(); index++) {
    wu.clear();

    char *infiles[1];
    infiles[0] = (char*)infilesNames.at(index).c_str();
    std::cout << "infiles[0]=" << infiles[0] << std::endl;
```

```
79
    char dlpath[1024] = {'\0'};
    config.download_path(infiles[0], dlpath);
    std::cout << "dlpath=" << dlpath << std::endl;
    boinc_copy(infiles[0], dlpath);
84
    char wu_name[32] = {'\0'};
    sprintf(wu_name, WUNAME, app.id, ++wu_count);
    strcpy(wu.name, wu_name);

89  wu.appid = app.id;
    wu.min_quorum = 2;
    wu.target_nresults = 2;
    wu.rsc_fpops_est = 1e10;
    wu.rsc_fpops_bound = 1e11;
94  wu.rsc_disk_bound = 1e8;
    wu.max_error_results = 2;

    char *wu_tpl = NULL;
    if(read_file_malloc("tplInput_Kreiszahl", wu_tpl)
99      != BOINC_SUCCESS)
    { /* Fehlerbehandlung */ }

    int create_work_res = create_work(
      wu, wu_tpl,
104   "templates/tplResult_Kreiszahl",
      "tplResult_Kreiszahl",
      (const char**)infiles, 1, config
    );
    if(create_work_res != BOINC_SUCCESS) {
109   /* Fehlerbehandlung */
    } else {
      boinc_copy("tplResult_Kreiszahl",
                 "../templates/tplResult_Kreiszahl");
      boinc_copy("tplInput_Kreiszahl",
114              "../templates/tplInput_Kreiszahl");
    }
  }

  return 0;
119 }

/**
 * Function will stop execution.
 * @param exitCode
124 */
void usage(int exitCode) {
  /* Anzeige der Aufrufparameter */
  exit(exitCode);
}
129
/**
 * @param argc
 * @param argv
 * @return Exit code
134 */
int main(int argc, char **argv) {
  if(argc < 2) usage(1);
  for(int i=1; i<argc; i++) {
    if(is_arg(argv[i], "start")) {
139   if(!argv[++i]) usage(1);
      startIteration = atoi(argv[i]);
    } else if(is_arg(argv[i], "end")) {
      if(!argv[++i]) usage(1);
      endIteration = atoi(argv[i]);
144 } else if(is_arg(argv[i], "step")) {
      if(!argv[++i]) usage(1);
```

```
      stepIteration = atoi(argv[i]);
    } else if(is_arg(argv[i], "runs")) {
      if(!argv[++i]) usage(1);
      runs = atoi(argv[i]);
    } else {
      fprintf(stderr, "unknown command line argument: %s\n", argv[i]);
      exit(1);
    }
  }

  return letsGo();
}
```

Listing 9.6 Beispiel einer Ergebnisdatei, die mit Hilfe von Listing 9.5 erstellt wird

```
<configuration>
<runs>10</runs>
<iterations>100</iterations>
</configuration>
```

Das Listing 9.7 zeigt die abgeänderten Stellen, um Arbeitspakete mit mehr als einer Eingabedatei zu erstellen. Es enthält nicht die Erstellung dieser Dateien, diese müssen schon vorliegen – es wird behandelt; wie der Aufruf von `create_work` mit den dazugehörigen Rahmen erstellt werden muss. Es gilt zu beachten: Wenn Sie von der vorherigen Version auf eine Erstellung für mehr als eine Datei umstellen, so müssen Sie natürlich auch die Eingabeschablone anpassen. Wenn Sie das versäumen, so erhalten Sie eine Meldung wie in Listing 9.8 und ich muss zugeben, dass mir das auch öfter passiert.

Listing 9.7 Beispielanwendung für das Erstellen von Arbeitspaketen mit mehr als einer Eingabedatei

```
...
#define EXTRAFILES 2
const char *extraFiles[EXTRAFILES] = {
  "database.zip",
  "model.zip"
};
...
int letsGo() {
  ...
  for(unsigned int index=0; index < infilesNames.size(); index++) {
    wu.clear();

    char **infiles = new char*[EXTRAFILES+1];
    infiles[0] = (char*)infilesNames.at(index).c_str();
    for(int i=0; i<EXTRAFILES; i++) {
      infiles[i+1] = (char*)extraFiles[i];
    }
    ...
    for(int i=0; i<EXTRAFILES+1; i++) {
      char dlpath[1024] = {'\0'};
      config.download_path(infiles[i], dlpath);
      boinc_copy(infiles[i], dlpath);
      std::cout << "dlpath=" << dlpath << std::endl;
    }
    ...
    char *wu_tpl = NULL;
    if(read_file_malloc("tplInput_Kreiszahl", wu_tpl)
        != BOINC_SUCCESS)
    { /* Fehlerbehandlung */ }

    int create_work_res = create_work(
```

```
      wu, wu_tpl,
      "templates/tplResult_Kreiszahl",
      "tplResult_Kreiszahl",
      (const char**)infiles, EXTRAFILES+1,
      config
    );
    ...
  }

  return 0;
}

int main(int argc, char **argv) {
  ...
  return letsGo();
}
```

Listing 9.8 Fehlermeldung über eine inkorrekte Syntax der Eingabedatei

```
boincadm@boinc-testserver:~/projects/tah/ries$ ./mainCreateWork \
                                -start 10 -end 100 -step 10 -runs 10
process_wu_template: 3 input files listed, but template has 1
process_wu_template: -1
process_wu_template: 3 input files listed, but template has 1
process_wu_template: -1
```

9.2.3 Arbeitspakete aus der Ferne erstellen

Es gibt mehrere Ansätze, den Prozess der Arbeitspaketerstellung zu vereinfachen [8, 14–16, 27]; das Problem ist, dass jedes BOINC-Projekt individuelle Wünsche hat, welche von den einzelnen Ansätzen meist nicht unterstützt werden. Die Unterschiede gliedern sich in den folgenden Bereichen ein:

Authentifizierung Erlaubt der Ansatz die Unterscheidung zwischen verschiedenen Benutzerrollen, kann nur eine bestimmte Rollenzugehörigkeit die Berechnungsergebnisse abfragen oder jeder?

Zugriffsebene Wie die Schnittstelle zum Hinzufügen von Arbeitspaketen aussieht, unterscheidet sich in: Webschnittstelle, grafische Benutzeroberfläche und Eingabe von Befehlen über ein Terminal.

Packaging Ich fasse unter den Begriff die Möglichkeit, dass einige Schnittstellen erlauben, die wissenschaftliche Applikation mit hinzuzufügen, während andere Schnittstellen „nur“ das Erstellen von Arbeitspaketen ermöglichen.

Ergebnisverarbeitung Einige Schnittstellen erlauben das Herunterladen der Ergebnisse von einer zusätzlichen Webseite und bei anderen können die Ergebnisse durch ein Terminal-Skript geladen werden.

Ich möchte Ihnen in den nachfolgenden beiden Abschnitten zwei Ansätze vorstellen.

9.2.3.1 rBOINC

rBOINC oder auch *Remote BOINC* kann dazu genutzt werden, mit Hilfe von Skripts über eine Netzwerkverbindung Arbeitspakete zu einem BOINC-Projekt hinzuzufügen. Es entstammt dem GPUGRID.net-Projekt [135]. Für diesen Zweck werden auf

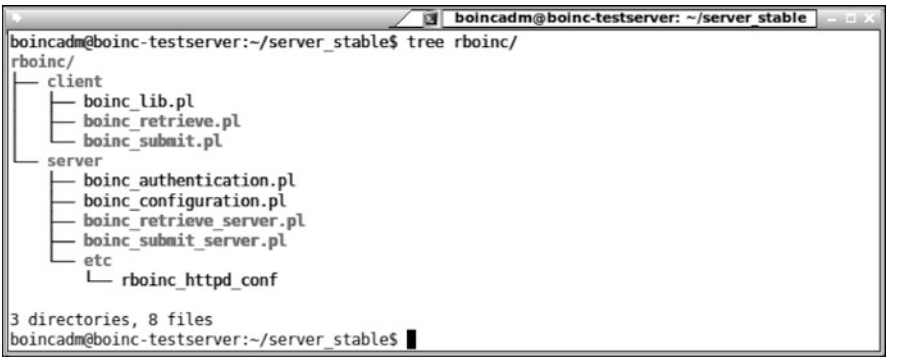


Abb. 9.1 Wichtige Dateien für das Installieren und Nutzen von rBOINC

der Serverseite zwei CGI-Skripts installiert, mit denen sich ein Benutzer verbindet, um entsprechende Anfragen zu senden. Eine Beispiel für die Anwendung von rBOINC finden Sie im Abschn. 13.4.2.

Installation von rBOINC

rBOINC liegt im Unterordner `rboinc/` den BOINC-Quellen bei. Abbildung 9.1 zeigt die acht relevanten Dateien, welche in zwei Bereiche unterteilt sind: *Client* und *Server*. Die zuvor erwähnten zwei CGI-Skripts sind `boinc_retrieve_server.pl` und `boinc_submit_server.pl`. Die Dateiendung `.pl` verrät, dass es sich um Perl-Skripts handelt. Diese beiden Skripts plus die Konfigurationsdateien `boinc_authentication.pl` und `boinc_-configuration.pl` müssen in den CGI-Ordner eines BOINC-Projektes kopiert werden; letztere müssen Sie noch an Ihre Systemumgebung anpassen. Listing 9.9 zeigt eine Beispielanwendung für die Testumgebung.

Listing 9.9 rBOINC-Konfiguration

```
return {
  DAV_DIR  =>  "$cgi/../DAV",
  PROJECT_DIR  =>  "/home/boincadm/projects/tah",
  WORKFLOW_DIR  =>  "/home/boincadm/projects/tah/workflow_results",
  DAV_URL  =>  "http://192.168.1.100/tah_dav/",
  ...
  DEFAULT_APP_NAME => 'wrapperCINOLA',
  ...
};
```

Damit diese Perl-Skripts durch einen Anwender aufrufbar sind, ist der Apache-HTTP-Server zu modifizieren. Dazu fügen Sie die Zeilen aus Listing 9.10 zur `httpd.conf` hinzu, doch bevor Sie den Apache-HTTP-Server neu starten, sollten Sie durch folgenden Befehl die Module für DAV-Support (Distributed Authoring and Versioning) [60] aktivieren:

```
boincadm@boinc-testserver:~$ sudo a2enmod dav_fs dav_lock
Considering dependency dav for dav_fs:
Enabling module dav.
Enabling module dav_fs.
Enabling module dav_lock.
Run '/etc/init.d/apache2 restart' to activate new configuration!
boincadm@boinc-testserver:~$
```

Bevor Sie nun neu starten, erstellen Sie im BOINC-Projektordner die Dateiordner `DAV/` und `workflow_results/` und setzen die entsprechenden Zugriffsrechte:

```
mkdir workflow_results/
mkdir DAV/
sudo chown -R www-data:www-data workflow_results/ DAV/ download/
```

Nun kann der Apache-HTTP-Server neu gestartet werden:

```
boincadm@boinc-testserver:~$ sudo /etc/init.d/apache2 restart
 * Restarting web server apache2                                    [ OK ]
boincadm@boinc-testserver:~$
```

Listing 9.10 Konfiguration des Apache-HTTP-Servers für die Nutzung von rBOINC

```
## rBOINC for Test@home
Alias /tah_dav /home/boincadm/projects/tah/DAV
ScriptAlias /tah_rboinc /home/boincadm/projects/tah/cgi-bin

DAVLockDB /usr/share/apache2/var/DAVLock
<Directory "/home/boincadm/projects/tah/DAV">
  Dav On
  Order Allow,Deny
  Allow from all
  AuthType Basic
  AuthUserFile "/home/boincadm/projects/tah/html/ops/.htpasswd"
  AuthName "Test@home DAV"
  Options +MultiViews +Indexes +FollowSymLinks
</Directory>
```

Eingabeschablonen für rBOINC markieren

rBOINC untersucht beim Hinzufügen von Arbeitspaketen, welche Variablen von Ihnen gesetzt werden müssen. Für diesen Zweck wird die Eingabeschablone nach XML-Tags `<rboinc .../>` durchsucht und entsprechend auf gefundene Zeilen reagiert. Die erste Zeile der Schablone muss bei Verwendung mit rBOINC eine Anfangsbeschreibung erhalten:

```
<rboinc application="kreiszahl"
        description="Add workunits for Kreiszahl@home."/>
```

Zusätzlich muss jede Dateireferenz einen weiteren rBOINC XML-Tag erhalten:

```
<workunit>
  <file_ref>
    <file_number>0</file_number>
    <open_name>configuration.xml</open_name>
    <rboinc parameter_name="cfg"
            parameter_description="Configuration for Kreiszahl@home."/>
  </file_ref>
  <file_ref>
    <file_number>1</file_number>
    <rboinc parameter_name="dummy"
```

```
            parameter_description="Do not set this variable."
            optional="true"/>
  </file_ref>
</workunit>
```

Sie wundern sich vielleicht über die zweite Dateibeschreibung, dies ist ein Workaround für eine fehlerhafte Implementierung in rBOINC[5]. Wenn Sie nur eine Datei beschreiben, so erhalten Sie folgende Fehlermeldung:

```
boincadm@boinc-testserver:~/server_stable/rboinc/client$ ./boinc_submit.pl -url
      http://boinc-testserver/tah_rboinc -app_name kreiszahl -help_parameters
Not an ARRAY reference at /home/boincadm/server_stable/rboinc/client/boinc_lib.
     pl line 107.
```

Wenn die Konfiguration durchgeführt wurde, können Sie nun erfragen, wie Arbeitspakete mit Hilfe von rBOINC hinzugefügt werden können:

```
boincadm@boinc-testserver:~/server_stable/rboinc/client$ ./boinc_submit.pl -url
      http://boinc-testserver/tah_rboinc -app_name kreiszahl -help_parameters

Remote application queue 'kreiszahl'
Description: Add workunits for Kreiszahl@home.
Application on server: 'kreiszahl'

Options defined for this application queue:
   -dummy  (optional) Do not set this variable.
   -cfg    Configuration for Kreiszahl@home.

boincadm@boinc-testserver:~/server_stable/rboinc/client$
```

9.2.3.2 BOINCs angedachte API

Ich rate Ihnen aktuell[6] von der Nutzung dieser API ab [81]. Dieses Buch beschränkt sich auf die BOINC-Quellen der Revision 22328, in der diese API noch nicht vorhanden ist. Als kleinen Ausblick auf die Zukunft und als Hinweis darauf, dass eine einheitliche Schnittstelle in der BOINC-Entwicklergemeinde entsteht, wodurch Sie nicht zwingend eine eigene Implementierung durchführen sollten wird im Folgenden diese angedachte API kurz beschrieben. Wenn Sie die neuen API-Funktionen testen wollen, so müssen Sie sich die SVN-Trunk-Version der BOINC-Quellen herunterladen.

Die Idee der neuen Schnittstelle ist eine API, welche sich in einem Webportal integrieren lässt. Dadurch ist es möglich, unterschiedliche Zugriffsmöglichkeiten zu nutzen und das bequem durch HTTP-Requests. cURL ist eine Bibliothek [111] die HTTP und HTTPS unterstützt und zusätzlich für die gängigsten Plattformen zur Verfügung steht: *Solaris, NetBSD, FreeBSD, OpenBSD, Darwin, HPUX, IRIX, AIX, Tru64, Linux, UnixWare, HURD, Windows, Amiga, OS/2, BeOs, Mac OS X, Ultrix, QNX, OpenVMS, RISC OS, Novell NetWare, DOS und weitere.* Auch mobile Geräte mit Android können genutzt werden [174].

Die eigentliche Schnittstelle ist in PHP umgesetzt und enthält folgende Funktionen:

[5] Bis einschließlich der BOINC-Revision 24359 vom 9. Oktober 2011.

[6] 20. Oktober 2011.

`boinc_(submit, estimate, query)_batch()`
Fügt einen Batch zum BOINC-Projekt hinzu, fragt die ungefähre Zeit für die Abarbeitung eines Batch ab und erfragt aktuelle Informationen eines Batch.

`boinc_query_batch(), boinc_query_job()`
Fragt Informationen eines Batch oder eines einzelnen Arbeitspaketes ab.

`boinc_abort_batch()`
Bricht die Ausführung eines Batch ab.

`boinc_get_output_file(), ..._files()`
Eine Adresse zum Herunterladen einer Ergebnisdatei oder eines ZIP-Archivs mit mehreren Ergebnisdateien.

`boinc_retire_batch()`
Löscht die zu einem Batch nötigen Dateien und Datenbankeinträge.

Die Idee ist, dass Sie diese Funktionen verwenden und eine Schnittstelle um diese Funktionen herum implementieren. Das heißt, Sie haben nun die Möglichkeit, eine Webschnittstelle wie bei Leiden [15] oder eine grafische Oberfläche [8, 16] zu implementieren.

9.3 Individuelle Dämonen implementieren

Dank der veröffentlichten Quellen des BOINC-Systems können alle Funktionen eingesehen und für die eigenen Bedürfnisse angepasst werden. In den Unterabschnitten dieses Abschnitts können sie nachlesen, wie Sie sich Ihre eigenen Validierungsfunktionen und für ein BOINC-Projekt passenden Assimilierer programmieren können. Abhängig vom Format der Berechnungsergebnisse können so ganz individuelle Prüfverfahren und Abspeicherungslösungen definiert werden. Im Unterordner *sched/* der BOINC-Quellen finden Sie Beispielimplementierungen. Ich empfehle Ihnen, jeweils eine Kopie der Quellen zu erstellen und erst dann an Ihre Bedürfnisse anzupassen.

9.3.1 Validierer – Prüfung der Ergebnisse

sample_bitwise_validator und *sample_trivial_validator* sind zwei mögliche Validierer, welche Sie direkt innerhalb Ihrer BOINC-Projekte verwenden können. Wenn Sie einmal in die Quelldateien schauen, dann werden Sie keine main-Funktion finden. Der Grund dafür liegt in der Idee des Validierer-Frameworks (entspricht auch dem Assimilierer-Framework im Abschn. 9.3.2).

Die main-Funktion befindet sich in der Datei *sched/validator.cpp*; sie liefert die Basis für das Starten der Validierung von Berechnungsergebnissen und verarbeitet die übergebenen Startparameter, welche Sie in der BOINC-Projektkonfiguration *config.xml* eines jeden BOINC-Projekts definieren können.

Die BOINC-API liefert eine low-/ und high-level-Abstraktionsschicht. Zuerst stelle ich Ihnen die vereinfachte Version vor, da ich bisher noch keine Anwendung

gesehen habe, bei der die low-level-API unbedingt Verwendung finden musste. Listing 9.11 zeigt die drei Funktionen, die Sie mindestens implementieren müssen, um einen Validierer für Ihr BOINC-Projekt entwickeln zu können.

Das liest sich gut, aber was bedeutet dies nun? Ganz simpel ausgedrückt, alle von den BOINC-Teilnehmern errechneten Ergebnisse werden an ein BOINC-Projekt gesendet. Das Ergebnis befindet sich in der Regel in einer Datei, welche sich innerhalb des BOINC-Projekts im upload-Verzeichnis wiederfindet. Informationen zu diesen Ergebnissen werden – mit zahlreichen weiteren Informationen – in der BOINC-Datenbank abgespeichert. Es gilt nun diese beiden verteilten Informationen wieder zusammenzuführen und in Folge für den nächsten Schritt – die Assimilierung – zu validieren. Die nachfolgenden Abschn. 9.3.1.1 und 9.3.1.2 befassen sich mit zwei Abstraktionsebenen, mit den Sie eigene Validierer implementieren können. Die im ersten Abschn. 9.3.1.1 beschriebenen BOINC-API-Funktionen können als high-level-Abstraktion gesehen werden, da diese schon mehrere Standardaktionen im Vorfeld abdeckt und eigentlich nur noch die direkte Validierung durchführt. Wünschen Sie mehr Kontrolle über den Validierungsprozess, so sollten Sie zusätzlich zu der high-level-Abstraktion auch die low-level-Abstraktion nutzen; dazu in den einzelnen Abschnitten mehr.

9.3.1.1 High-level Validator-API

Die high-level-Abstraktionsschicht kann als Rahmen einer Kurzgeschichte gesehen werden: Sie haben eine Einleitung (1. Schritt), einen Plot (2. Schritt) und den Abgang (3. Schritt), inmitten Ihrer Berechnungen. Einzig dafür ist die high-level-Validierer-API zuständig und angedacht. Wünschen oder benötigen Sie eine stärkere Kontrolle, z. B. um die Berechnung der Credits einer WU zu errechnen, so müssen Sie zusätzlich Funktionen aus der low-level-Abstraktionsschicht nutzen. Das Listing 9.11 enthält die drei Funktionen, mit denen die Kurzgeschichte mit Leben gefüllt werden kann.

`1. Schritt: init_result()`

Der erste Parameter hält die Daten für ein Berechnungsergebnis aus der Datenbank vor, u. a. die Erstellungszeit (DB-Spalte: *create_time*), den Host (DB-Spalte: *hostid*), der für die Berechnung verantwortlich ist, und noch viele mehr. Kleine Helfer ermöglichen uns das einfache Ermitteln der Ergebnisdateien. Abschnitt 9.2 erläutert eine Auswahl von Funktionen, welche das Entwickeln eines eigenen Validierers vereinfachen können. Wie Sie die Informationen für den Vergleich der Berechnungsergebnisse bereithalten, liegt ganz bei Ihnen und ist sicherlich auch von dem Berechnungsergebnis selbst abhängig zu machen. In jedem Fall müssen Sie Ihre Datenstruktur dem Zeiger data mitgeben, so dass diese Struktur im nächsten Validierungsschritt, in der Funktion *compare_result*, auch verwendet wird.

Weiterhin sei angemerkt, dass eventuell nicht jedes BOINC-Projekt einen zutiefst ausgeklügelten Validierer benötigt. Das Praxisbeispiel aus Kap. 12 verwendet etwa den von BOINC mitgelieferten Standardvalidierer *sample_bitwise_vali-*

dator, da nur zwei oder mehr Bilddateien miteinander verglichen werden und es mindestens zwei identische geben muss. In unserem eigenen BOINC-Projekt Spinhenge@home[7] verwenden wir zum Beispiel den BOINC-Standardvalidierer *sample_trivial_validator* und prüfen nur, ob eine bestimmte Rechen- bzw. genutzte CPU-Zeit überschritten wurde[8].

2. Schritt: `compare_results()`
Diese Funktion dient der eigentlichen Überprüfung von Berechnungsergebnissen. Der Parametersatz erwartet zwei Ergebnisse, es handelt sich dabei um ein „neues" Ergebnis und das sogenannte Canonical-Ergebnis (vgl. Abschn. 9.3.1.2). Wenn nur eine Berechnung von den BOINC-Teilnehmern durchgeführt werden soll – d. h. `min_quorum=1` –, so ist das erste übergebene Ergebnis gleich dem Canonical-Ergebnis.

3. Schritt: `cleanup_result()`
Die im 1. Schritt erstellten Datenstrukturen sollten nicht weiter im Speicher belassen werden. In einem größeren Projekt kann das schnell die Ressourcen verbrauchen und daraufhin die Performance des Projekts vermindern. Daher werden die temporär im Computerspeicher vorliegenden Daten entfernt, so dass dieser für andere Ergebnisse genutzt werden kann.

Listing 9.11 High-level-Version der BOINC-Validator-API

```
#include <sched/validate_util2.h>

int init_result(RESULT &result, void *&data);
  /**
   * Returns 0 on success.
   * Returns ERR_OPENDIR if there was a transient error,
   * {\eg} the output file is on a network volume that is not
   * available. The validator will try this result again later.
   * Any other return value indicates a permanent error, {\eg}
   * an output file is missing, or has a syntax error. The
   * result will be marked as invalid.
   */

int compare_results(
  RESULT &r1, void *data1,
  RESULT const &r2, void *data2,
  bool &match
);
  /**
   * Returns 1 if the two results are equivalent,
   * otherwise 0 will be returned.
   */

int cleanup_result(RESULT const &result, void *data);
  /**
   * Returns 0 when runs successfully, something
   * else when an error invoked.
   */
```

[7] http://spin.fh-bielefeld.de.

[8] Das ist darin begründet, dass wir nicht wissen, wie das Ergebnis der Berechnungen auszusehen hat. Spinhenge@home basiert auf Zufallszahlen, die Berechnungen bilden Modelle ab, welche wissenschaftlich untersucht werden und über die im Vorfeld nichts Genaueres ausgesagt werden kann. Es kann nichts geprüft werden, das nicht bekannt ist. Erst im Nachgang, wenn alle Ergebnisse vorliegen, versuchen wir daraus etwas zu lesen und zu lernen.

Wir stellen Ihnen an dieser Stelle das von BOINC mitgelieferte Beispiel eines sehr einfachen Validierers vor. Das Listing 9.12 enthält die relevanten Zeilen dieses trivialen Validierers. Es wäre ohne weiteres möglichen, diesen noch kürzer zu fassen: Anstatt der Prüfung in Zeile 22 können Sie den Wert von *match* direkt auf `true` oder `false` setzen und entsprechend die Validierung von vornherein positiv beziehungsweise negativ ausgehen lassen.

Listing 9.12 Beispiel eines trivialen Validierers zur Prüfung, ob ein Minimum der Rechenzeit in Sekunden überschritten wurde

```
// C/C++
#include <cstdlib>

// BOINC
#include <config.h>
#include <validate_util.h>

static const double MIN_CPU_TIME = 10;

int init_result(
  RESULT& /*result*/,
  void*& /*data*/)
{
    return 0;
}

int compare_results(
    RESULT & r1, void* /*data1*/,
    RESULT const& r2, void* /*data2*/,
    bool& match
) {
    match = (r1.cpu_time >= MIN_CPU_TIME && r2.cpu_time >= MIN_CPU_TIME);
    return 0;
}

int cleanup_result(RESULT const&, void*) {
    return 0;
}
```

Da es sich bei dieser high-level-Abstraktion um eine weitere Abstraktionsschicht handelt, müssen auch die unteren Schichten mit in die Applikation gebunden werden. Für das erfolgreiche Übersetzen in eine ausführbare Applikation benötigen Sie folgende Dateien:

- sched/validator.cpp, sched/validator.h
- sched/validate_util.cpp, sched/validate_util.h
- sched/validate_util2.cpp, sched/validate_util2.h

Validierer kompilieren

Das Kompilieren von Anwendungen mit Nutzen von BOINC-Funktionen zieht immer einen längeren Aufruf hinter sich her. Beim Kompilieren eines Validierers müssen die Dateiordner für die zuvor erwähnten Dateien definiert und die entsprechenden Dateien mit zum Prozess angehängt werden. Listing 9.13 zeigt ein einfaches Makefile, um einen Validierer zu kompilieren. Zeile 1 bindet die MySQL-Bibliotheken und Header-Dateien zum Kompilierungsprozess, denn der Datentyp

RESULT aus Listing 9.12 ist eine Datenstruktur zur Beschreibung der MySQL-Tabelle *result*, welche die Beschreibung der Berechnungsergebnisse enthält. Ich erstelle mir immer einen symbolischen Link zu den BOINC-Quellen; hier in dem Listing 9.13 in Zeile 2 ist das der symbolische Link *./boincsrc*.

Listing 9.13 Makefile für das Kompilieren eines Validierers

```
CFLAGS += -O2 -Wall `mysql_config --cflags --libs`
BOINCSRC = ./boincsrc
CC = g++

INCLUDES += \
  -I${BOINCSRC} \
  -I${BOINCSRC}/api \
  -I${BOINCSRC}/lib \
  -I${BOINCSRC}/sched \
  -I${BOINCSRC}/db

LIBS += \
  -L${BOINCSRC}/sched -lsched \
  -L${BOINCSRC}/lib -lboinc \
  -L${BOINCSRC}/api -lboinc_api \

SRC += \
  ${BOINCSRC}/sched/validator.cpp \
  ${BOINCSRC}/sched/validate_util.cpp \
  ${BOINCSRC}/sched/validate_util2.cpp \
  sample_trivial_validator.cpp

all: trivialValidator

trivialValidator: ${SRC}
  ${CC} ${CFLAGS} ${INCLUDES} -o $@ ${SRC} ${LIBS}

clean:
  rm -rf *.o *~
```

Wenn Sie mehr als eine zusätzliche BOINC-Applikation entwickeln, dann ist es erstrebenswert, einen allgemeinen Standard bezüglich des Kompilierungsprozesses zu haben. Meine bisherigen Erfahrungen waren sehr gut. Ich habe eine Makefile, einen symbolischen Link zu den BOINC-Quellen und gebe manuell an, welche Dateien ich gerne zu meiner neuen BOINC-Applikation hinzufügen will. Die Zeilen 18 bis 20 listen meine gewünschten Dateien auf, die sich alle in den BOINC-Quellen befinden und zu meinem trivialen Validator hinzugelinkt werden müssen.

9.3.1.2 Low-level Validator-API

Kurzbeschreibung der Validator-API. Für den Kompilierungsprozess eines Validierers, welcher nur die low-level-Abstraktionsschicht verwenden soll, kann die *sched/validator.cpp* mit drei zusätzlichen Funktionen gefüttert werden:

`compute_granted_credit()`
: Der Name sagt eigentlich aus, was die Funktion tun soll, allerdings wird diese Funktion nirgends von dem BOINC-Framework aufgerufen und hätte daher keinen Einfluss auf den Programmablauf. Was Sie allerdings machen können: Sie

können diese Funktion in den anderen fünf BOINC-Validate-API-Aufrufen nutzen und entsprechend – bei einem validen Berechnungsergebnis – die passenden Credit-Punkte für eine Berechnung kalkulieren und den BOINC-Teilnehmern gutschreiben.

`check_set()`

Falls eine eigene Implementierung dieser Funktion für Sie in Frage kommt, dann konsultieren Sie bitte das BOINC-Wiki, um nähere Informationen über die Musskriterien Ihrer Umsetzung zu erfahren [87]. Die Funktion muss mit fünf Parametern aufgerufen werden:

1. **wu:** Enthält die Informationen eines Arbeitspakets: Erstellungszeitpunkt (Datenbankspalte: *create_time*), in welchem Status das aktuelle Arbeitspaket ist (Datenbankspalten: *need_validate*, *file_delete_state*, etc.) und vieles mehr. Die Berechnungsergebnisse in **result** sind Ergebnisse dieses Arbeitspakets.
2. **result:** Dieser Parameter ist ein Vektorfeld und enthält nur Berechnungsergebnisse, bei denen eine erfolgreiche Berechnung durchgeführt wurde. Die einzelnen Berechnungsergebnisse sind einzelne Datensätze auf der BOINC-Datenbank der Tabelle *result*. Abhängig davon, ob Fehler bei der Bearbeitung der Berechnungsergebnisse passieren – z. B. es fehlen Ergebnisdateien oder Ergebnisdateien haben kein validen Inhalt –, müssen entsprechende Status-Flags für den einzelnen Datensatz in die Tabelle mit übernommen werden.
3. **canonicalid:** Enthält bei erfolgreicher Validierung eines Arbeitspakets die Identifikation des Berechnungsergebnisses, welches als Canonical-Result – in der Regel das am genauesten berechnete Ergebnis – auserkoren wurde.
4. **credit:** Falls ein Ergebnis erfolgreich validiert wurde, so steht in dieser Variablen die Anzahl der Credits für die BOINC-Teilnehmer, welche erfolgreich Berechnungen für ein Arbeitspaket durchgeführt haben.
5. **retry:** Dieser Parameter sollte `true` gesetzt werden, um ein nochmaliges Abarbeiten eines Arbeitspakets zu ermöglichen. Dies kann unter anderem dann sinnvoll sein, wenn Dateiordner temporär nicht verfügbar sind (z. B. ein mit NFS zum Dateisystem hinzugefügter Dateiordner fehlt).

`check_pair()`

Wenn neuere Berechnungsergebnisse eines Arbeitspakets vorliegen, allerdings schon ein Canonical-Result definiert ist, so wird das neue Ergebnis mit diesem verglichen und geprüft, ob es mit diesem übereinstimmt, und entsprechend werden Credits vergeben.

Innerhalb dieser Funktion werden die Funktionen *init_result*, *compare_results* und *cleanup_result* aus dem Listing 9.14 der high-level-Abstraktionsschicht aufgerufen.

Es wird empfohlen, dass keine der genannten Funktionen Dateien löscht oder auf die Datenbank zugreift [87].

Listing 9.14 Low-level-Version der BOINC-Validator-API

```
// BOINC
#include <sched/validate_util2.h>

// C++
#include <vector>

double compute_granted_credit(
  WORKUNIT &wu,
  std::vector<RESULT> &results
);
  /**
   * Diese Funktionsdefinition wird nirgend in den BOINC-Quellen
   * implementiert oder aufgerufen, daher waere eine Implementierung
   * durch Sie ohne Einfluss auf die Ausfuehrung.
   */

int check_set(
  std::vector<RESULT> &results, WORKUNIT &wu,
  int &canonicalid, double &credit_deprecated, bool &retry
);
  /**
   * Returns always 0.
   * Given a set of results, check for a canoncial result,
   * {\ie} a set of at least min_quorum/2+1 results for which
   * that are equivalent according to check_pair().
   */

void check_pair(RESULT &r1, RESULT &r2, bool &retry);
  /**
   * r1 is the new result and r2
   * is canoncial result.
   */
```

Das Kompilieren eines Validierers mit Verwendung der low-level-Abstraktionsschicht ähnelt dem Prozess aus Abschn. 9.3.1.1, es muss nur ein anderer Satz von BOINC-Framework-Dateien zum eigentlichen Validierer hinzugelinkt werden. Listing 9.15 zeigt die gekürzte Version des Makefile mit den signifikanten Stellen. Sie sehen, dass die einzige Änderung in der Definition der Variablen SRC in Zeile 2 gemacht wurde; die Datei mit der high-level-Abstraktionsschicht wurde entfernt.

Listing 9.15 Makefile für das Kompilieren eines low-level-Validierers

```
...
SRC += \
  ${BOINCSRC}/sched/validator.cpp \
  ${BOINCSRC}/sched/validate_util.cpp \
  lowlevel_validator.cpp

all: lowlevelValidator

lowlevelValidator: ${SRC}
  ${CC} ${CFLAGS} ${INCLUDES} -o $@ ${SRC} ${LIBS}
```

Schlussworte zum Validierer

Sie haben gesehen, dass Sie viele Möglichkeiten haben und an jeder Stelle des Validierungsprozesses Ihre eigenen Wünsche implementieren können. BOINC schreibt Ihnen nicht vor, wie Sie Ihre Berechnungsergebnisse zu validieren haben. Das ist

auch gar nicht möglich, denn wer weiß schon besser Bescheid darüber, wie die Validierung für Ihre Ergebnisse auszusehen hat, als Sie selbst. Allerdings muss nicht für jede Fragestellung oder jedes BOINC-Projekt eine spezielle Validierungslösung entworfen und implementiert werden. Es reicht vielleicht schon, einen der Standardvalidierer zu verwenden; Sie werden sehr dankbar dafür sein, denn es wird Ihnen viel Frust und, was noch viel wichtiger ist, Zeit ersparen.

9.3.2 Assimilierer – Ergebnisse abspeichern

Nach der erfolgreichen Validierung eines oder mehrerer Berechnungsergebnisse heißt es abspeichern. Sie wollen die Ergebnisse nutzen, um damit ein Problem zu lösen oder um neue wissenschaftliche Erkenntnisse zu gewinnen. Ich kann Ihnen jetzt schon versprechen, Sie müssen nicht wirklich viel tun, um eine erfolgreiche Assimilierung zu ermöglichen.

Zuerst müssen Sie sich klar darüber werden, wie Sie Ihre Berechnungsergebnisse abspeichern wollen. Folgende Möglichkeiten ergeben sich in der Regel:

`Keine Abspeicherung`
: Auch dieser Fall kann Sie treffen, wenn Sie vielleicht Trickle-Messages (vgl. Abschn. 9.4) nutzen und durch diese Nachrichten Ergebnisse an Ihr BOINC-Projekt liefern lassen. Allerdings benötigen Sie den Assimilierungsschritt, so dass im nächsten Schritt die Fertigstellung eines Arbeitspakets signalisiert werden kann (vgl. Abschn. 9.3.2).

`Dateiabspeicherung`
: In der Regel haben Sie mindestens eine Ergebnisdatei, und diese können Sie ganz nach Belieben innerhalb Ihres Dateisystems abspeichern. Dabei kann der Abspeicherungsort ein lokales Verzeichnis, ein durch NFS eingebundenes, oder vielleicht sogar ein USB-Gerät sein. Es ist natürlich auch möglich, dass Sie die Ergebnisse direkt verarbeiten und von daher die dazugehörigen Dateien direkt löschen können, z. B. wenn Berechnungen eines Standorts direkt in einer Karte dargestellt werden. Seien Sie bei diesem möglichen Verfahren aber vorsichtig, vielleicht benötigen Sie später eine weitere Interpretation oder Darstellung der Ergebnisse, oder eventuell haben Sie einen Fehler in der direkten Interpretation der Ergebnisse. Sind die Daten einmal weg, hat man das Nachsehen. Von daher würde ich persönlich die Ergebnisse immer permanent abspeichern, wobei Festplatten heutzutage auch perfekt für diese Vorhaben geeignet sind und preislich absolut erschwinglich sind.

  Bevorzugen Sie diese Art der Abspeicherung, so müssen Sie keine weiteren Implementierungsarbeiten durchführen. BOINC liefert Ihnen mit `sample_assimilator` eine Lösung mit, welche zum Abspeichern der Berechnungsergebnisse in einem Dateiordner zuständig ist. Standardwerte in dieser Lösung sind das Abspeichern der Berechnungsergebnisse im Dateiordner `sample_result/`; Ergebnisse, die fehlerhaft waren, werden in `sample_result/errors` abgespeichert. Sie können nun einfach symbolische Links auf diese Pfade setzen und

dadurch erreichen, dass die Dateien nach Ihren Wunsch an einem beliebigen Ort abgespeichert werden.

`Datenbankabspeicherung`
Eine weitere Möglichkeit ist das Abspeichern in einer Datenbank – unter anderem sehr sinnvoll, wenn in Ihren Ergebnisdateien nur Zahlenwerte stehen und keine binären Daten oder vielleicht serialisierte Berechnungsmodelle inklusive Ergebnisse; dies ist zum Beispiel bei COMSOL Multphysics der Fall (vgl. Kap. 14).
Generell können Sie Ihre Daten in jeder vorhandenen Datenbank abspeichern. Eine schöne Bibliothek für C++ ist die Bibliothek SOCI[9], mit der Sie eine Implementierung entwickeln und Zugriff auf die folgenden Datenbanken erhalten: MySQL, OCBC (generic backend), Oracle, PostgreSQL und SQLite3. Ein einfaches Beispiel für diese Persistenzschicht finden Sie in Listing 9.16. Weiterhin ist es natürlich möglich die Persistenzschicht der BOINC-API zu verwenden; Abschn. 11.2.5 eines Praxisbeispiels zeigt, wie Sie diese Persistenzschicht um eigene Strukturen erweitern und für Ihre Aufgaben nutzen können.

Beispiel eines Assimilierers

Listing 9.16 zeigt Ihnen den generellen Aufbau eines eigenen Assimilierers. Die wichtigste Funktion für einen solchen ist `assimilate_handler`, welche Sie in Ihrem eigenen Assimilierer gegen `sched/assimilate.cpp` linken müssen. Für die zuvor erwähnte Persistenzschicht von SOCI müssen Sie die Header-Dateien in Zeile 4ff einbinden.

Zum Beginn der Assimilierung in der Funktion `assimilate_handler` ermitteln wir die Datenbankdaten in Zeile 30 und versuchen uns über eine SOCI-Session mit dieser in Zeile 39 zu verbinden. Die nachfolgenden Zeilen lesen die Ergebnisdateien ein und parsen die Informationen in eine passende Struktur; hier `XML_RESULTS`. Die Informationen in dieser Struktur können nur in Zeile 57 in die passende Datenbanktabelle geschrieben werden. Der Clou bei SOCI ist die einfache Art und Weise, wie dies geschieht; wenn Sie C++-Streams kennen, so werden Sie SOCI lieben!

Die SOCI-Session besitzt einen überladenen Stream-Operator <<, mit dem Sie nun SQL-Abfragen durchführen können. In unserem Beispiel schreiben wir neue Datensätze mit der Abfrage

sql « "insert into results_pi(iterations, value)" « " values(:iter, :val)",
soci::use(xmlResults.iterations[j]), soci::use(xmlResults.values[j]);

in die passende Tabelle. Sie können die Reihenfolge der Werte (engl. *values*) in der passenden Reihenfolge angeben oder explizit durch die Namen übergeben; alles nachzulesen in der guten Dokumentation von SOCI [170].

[9] http://soci.sourceforge.net – SOCI – The C++ Database Access Library.

Listing 9.16 Beispiel eines einfachen Assimilierers, der die Ergebnisse in eine MySQL-Datenbank schreibt

```
// BOINC includes
...
// SOCI
#include <soci/soci.h>
#include <soci/mysql/soci-mysql.h>

struct XML_RESULTS {
  std::vector<int> runs;
  std::vector<int> iterations;
  std::vector<double> values;
  ...
};

int write_error(std::string p, soci::session &sql) {
  // SOCI
  sql << "insert into results_pi(iterations, value,"
      << " description) values(:iter, :val, :dd)",
          soci::use(-1), soci::use(-1.0), soci::use(p);
  return 0;
}

int assimilate_handler(
    WORKUNIT &wu,
    vector<RESULT> &/*results*/,
    RESULT &canonical_result
) {
  unsigned int i;

  // SOCI
  SCHED_CONFIG config;
  config.parse_file();

  std::string cs;
  cs += " host=" + std::string(config.db_host);
  cs += " db=" + std::string(config.db_name);
  cs += " user=" + std::string(config.db_user);
  cs += " password='" + std::string(config.db_passwd) + "'";

  soci::session sql("mysql", cs.c_str());

  if (wu.canonical_resultid) {
    vector<FILE_INFO> output_files;
    get_output_file_infos(canonical_result, output_files);

    unsigned int n = output_files.size();
    for (i=0; i<n; i++) {
      FILE_INFO &fi = output_files[i];

      FILE* f = boinc_fopen(fi.path.c_str(), "r");
      if(f != NULL) {
        XML_RESULTS xmlResults;
        xmlResults.parse(f);
        fclose(f);

        for(unsigned int j=0; j<xmlResults.runs.size(); j++) {
          // SOCI
          sql << "insert into results_pi(iterations, value)"
              << " values(:iter, :val)",
                  soci::use(xmlResults.iterations[j]),
                  soci::use(xmlResults.values[j]);
        }
      }
    }
  } else {
    char buf[1024] = {'\0'};
```

```
    sprintf(buf, "%s: 0x%x\n", wu.name, wu.error_mask);
    return write_error(buf, sql);
  }
  return 0;
}
```

Während des Kompilierungsfortschrittes müssen Sie natürlich die passenden SOCI-Bibliotheken mitgeben. Im Listing 9.17 finden Sie ein einfaches Makefile für diesen Zweck. Es enthält die wichtigen Bibliotheken `soci_core*` und `soci_mysql*`, welche zum Assimilierer hinzugelinkt werden müssen.

Listing 9.17 Makefile für das Kompilieren eines Assimilierers

```
S+=assimilator.cpp pi_assimilator_soci.cpp validate_util.cpp

INCLUDES=-I./ `mysql_config --cflags` \
  -I./boincsrc -I./boincsrc/db -I./boincsrc/sched \
  -I./boincsrc/lib -I/usr/include/soci

LIBRARIES=\
  -L./boincsrc/sched -lsched -L./boincsrc/api -lboinc_api \
  -L./boincsrc/lib -lboinc -pthread `mysql_config --libs` \
  -lsoci_core-gcc-3_0 -lsoci_mysql-gcc-3_0 -ldl

all: ${S}
  g++ -O2 -Wall -o pi_assimilator_soci ${S} ${INCLUDES} ${LIBRARIES}
```

9.4 Trickle-Messages: eine asynchrone Kommunikation

Das Vorreiterprojekt für die Verwendung von asynchronen Nachrichten ist das Klimaberechnungsprojekt Climateprediction [112]. Die Problematik ist, dass die Berechnungen zu lange dauern würden und die Deadline für das Zurücksenden in jedem Fall nicht eingehalten werden kann. Außerdem ist es der Wunsch der BOINC-Freiwilligen, zügig Credits für ihre Berechnungen zu erhalten. Kurzerhand wurde ein relativ einfaches System für den Versand von asynchronen Nachrichten – also das Senden und Empfangen von zeitlich versetzten Daten ohne das Blockieren von Prozessen, beispielsweise durch Warten auf die Antwort des Empfängers [184] – implementiert.

Die asynchrone Kommunikation wird in vielen Fällen dafür verwendet, neue Informationen zu erhalten oder auch einzelne Befehle abzusenden. Daher auch die Namensgebung Trickle-Messages. *Trickle* heißt übersetzt „der Bruchteil“ und *Messages* sind Nachrichten. Der Name weist schon darauf hin, dass diese BOINC-Nachrichten nur dafür gedacht sind, einzelne (Kurz-)Informationen zu übertragen.

Die BOINC-API sieht nur 1024-Zeichen – sprich 1k-Byte – für den Austausch pro Trickle-Nachricht vor. Das reicht bei weitem nicht aus, um komplexe Zwischenergebnisse auszutauschen oder neue Berechnungsmodelle zu versenden. Wie schon erwähnt, wurde es nicht für solche Zwecke entworfen. Climateprediction hat mehrere Projekte mit unterschiedlichen Simulationsmodellen zur selben Zeit aktiv,

u. a. Slab-Model[10], Sulphur-Cycle-Model[11] und ein Transient-Coupled-Model[12]. Die genannten Simulationsmodelle benötigen unterschiedlich lang, um einen Parametersatz zu berechnen. Für Climateprediction-Zwecke reichen die 1k-Byte absolut, denn es werden in diesem Fall nur einzelne sogenannte Trickle-Numbers[13] versendet; diese spezifizieren, wie viel Prozent von einem Modell schon berechnet wurden. Die Trickle-Number 4_3 bedeutet zum Beispiel, dass 59,15 % eines Modells für das Slab-Simulationsmodell fertiggestellt wurden. Entsprechend können dem BOINC-Freiwilligen Credits angerechnet werden.

Eigentlich sind Klimaberechnungen ein Berechnungsproblem für „richtige" Clusterlösungen. Gehören definitiv in die Kategorie der Probleme für High-Performance-Cluster (HPC), weil eigentlich die Berechnungen eines Modells auf alle Knoten eines Clusters aufgeteilt und berechnet werden. In einem solchen Fall entstehen Abhängigkeiten zwischen den Cluster-Knoten, weil die einzelnen Teile eines Modells eventuell ein Berechnungsergebnis eines anderen Cluster-Knotens benötigen, um weitere Berechnungen durchzuführen. Diese Aufteilung ist bei BOINC nicht sinnvoll, da die Berechnungszeit höher wäre, als wenn man das Modell als Ganzes berechnet, denn heutzutage ist der Flaschenhals bei solch einer Nutzung noch immer das Internet, das weitaus langsamer als ein lokales Netzwerk ist.

In den nachfolgenden Abschnitten erhalten Sie einen Überblick über die BOINC-API, um die Möglichkeit von asynchronen Nachrichten in Ihren BOINC-Projekten nutzen zu können.

9.4.1 Aktivieren von Trickle-Messages

Das Aktivieren der Möglichkeit von Trickle-Messages geschieht auf der Serverseite. Abschnitt 15.3.2 beschreibt die Konfigurationsdateien eines BOINC-Projekts; für dieses Feature muss die `config.xml` angepasst werden. Ein einfaches Hinzufügen der in Listing 9.18 aufgeführten XML-Sequenz innerhalb des *<config>... </config>* und nachfolgendes Neustarten des BOINC-Servers reicht aus.

Listing 9.18 XML-Sequenz zum Aktivieren von Trickle-Messages

```
<boinc>
  <config>
    ...
    <msg_to_host/>
    <one_result_per_host_per_wu/>
    ...
  </config>
  <tasks>...</tasks>
  <daemons>...</daemons>
</boinc>
```

[10] http://www.boinc-wiki.info/Slab_Model.

[11] http://www.climateprediction.net/science/s-cycle.php.

[12] http://www.boinc-wiki.info/Transient_Coupled_Model.

[13] http://www.boinc-wiki.info/Trickle.

Sie haben nun Trickle-Messages aktiviert, allerdings haben Sie noch keine Vorkehrungen getroffen, wie diese auf der BOINC-Serverseite gehandhabt werden oder von einer wissenschaftlichen Applikation verarbeitet und genutzt werden sollen. Zusätzlich stellen wir ein, dass jeder Host nur ein Arbeitspaket erhalten soll, so dass im nachfolgenden Beispiel eine ordentliche Verarbeitung stattfinden kann und kein unnötiger Mehraufwand entsteht.

9.4.2 Kommunikationsprotokoll von Trickle-Messages

Grundsätzlich sind die einzelnen Zeilen einer Trickle-Message durch ‚\n' als End-of-Line (EOL) terminiert.

9.4.2.1 Trickle-Message: Host zum BOINC-Projekt

Listing 9.19 enthält das XML-Format der Trickle-Message von einem BOINC-Client zum BOINC-Projekt. Sie erkennen, `YOURDATA*` kann durch Sie definiert und beschrieben werden. Allerdings sind die Gesamtdaten einer Trickle-Message auf 1024 Bit beschränkt – nicht viel Freiraum für größere Datensätze. Wenn Sie mehr Daten übertragen wollen, so können Sie das BOINC-Framework allerdings entsprechend anpassen, sprich Sie müssen die BOINC-Quellen modifizieren.[14]

Listing 9.19 Aufbau einer Trickle-Message vom BOINC-Client zum BOINC-Server

```
<msg_from_host>
  <result_name> STRING </result_name>
  <time> INTEGER </time>
  YOURDATA*
</msg_from_host>

YOURDATA:
  <einFeld> Meine Daten! </einFeld>
```

9.4.2.2 Trickle-Message: BOINC-Projekt zum Host

Das Format der Trickle-down-Nachricht von einem BOINC-Server zu den BOINC-Teilnehmern kann in der Datei `client/cs_trickle.cpp`, innerhalb der Funktion `handle_trickle_down()` nachgelesen werden. Listing 9.20 zeigt das Format. Es ist zwingend erforderlich, dass Sie das XML-Tag `result_name` mit einem passenden Wert besetzen, da eine Nachricht sonst nicht zugestellt werden kann; hier also mit einem bestimmten Arbeitspaket, um eventuell die Bearbeitung dieses Arbeitspaketes zu unterbrechen. Es steht Ihnen frei, die Nachricht mit weiteren Informationen zu befüllen, in diesem Fall mit `YOURDATA` beschrieben.

[14] In der BOINC-Quelldatei `lib/app_ipc.h` müssen Sie die Konstante `MSG_CHANNEL_SIZE` entsprechend anpassen.

Listing 9.20 Aufbau einer Trickle-Message vom BOINC-Server zum BOINC-Client

```
<trickle_down>
  <result_name> NAME </result_name>
  <time> INTEGER </time> <!-- optional -->
  YOURDATA*
</trickle_down>

YOURDATA:
  <einFeld> Meine Daten! </einFeld>
```

Wenn eine Nachricht an einen Host gesendet wird, der das Arbeitspaket mit dem Namen `result_name` nicht bearbeitet, so wird das durch die Fehlernummer `ERR_NULL` signalisiert. Ein wenig verständlicher ist die entsprechende Meldung im BOINC-Manager

[error] handle_trickle_down failed: unexpected null pointer

9.4.3 Sequenz einer Trickle-Message

In den nachfolgenden Abschnitten möchte ich Ihnen ein Chat-System vorstellen, welches auf den BOINC-Komponenten für das Verarbeiten von Trickle-Messages aufbaut. Sicherlich werden Sie im Lauf der Diskussion erkennen, dass dies eine eher weniger praktikable Umsetzung ist, die wohl niemals die breite Masse an Benutzern befriedigen wird. Der Grund ist einfach, es fehlt der flüssige Nachrichtenaustausch, denn Anfragen an ein BOINC-Projekt können nur auf ein Intervall von einer Sekunde eingestellt werden. Das heißt, wir haben mindestens eine Latenzzeit von einer Sekunde und wenn nicht einmal dieser Wert eingestellt ist, so haben wir einen Standardwert zwischen einer Minute und vier Stunden.[15] Eingestellt werden kann der Wert durch Sie innerhalb der `config.xml` mit Hilfe des Parameters `<next_rpc_delay>`.

Die Idee des Chat-Systems ist, dass sich die Teilnehmer eines BOINC-Projekts untereinander Nachrichten zusenden können. Theoretisch kann der Nachrichtenaustausch über eine GUI-Anwendung funktionieren. Die Abb. 9.2 und 9.3 zeigen den prinzipiellen Ablauf von der Eingabe einer Nachricht über das Austauschen zwischen BOINC-Client und BOINC-Projekt sowie das Prüfen, ob ein Empfänger existiert, bis hin zum Zustellen dieser Nachricht. Die GUI-Anwendung prüft in den Slots eines BOINC-Clients, ob dort die Datei `trickleChat` vorliegt, und der Anwender kann entscheiden, ob dieser Slot in Folge für das Chatten genutzt wird. In der GUI-Anwendung kann der Anwender nun einen Empfänger und eine Nachricht eintragen; diese Informationen werden innerhalb des Slots in der Datei `msgChat` abgelegt. Der BOINC-Client bzw. die wissenschaftliche Applikation in dem Slot, hier die Applikation aus Abschn. 9.4.4.2, prüft kontinuierlich, ob eine Datei mit diesem Namen existiert, und wenn ja, so werden die zwei Informationen eingelesen. Diese werden in einen XML-Dialekt, wie die obere Nachricht in Abb. 9.3 zeigt, konvertiert und daraufhin an das entsprechende BOINC-Projekt

[15] Verantwortlich sind für diese Werte die Definitionen `SCHED_RETRY_DELAY_MIN` und `SCHED_RETRY_DELAY_MAX` in der BOINC-Quelldatei `client/scheduler_op.cpp`.

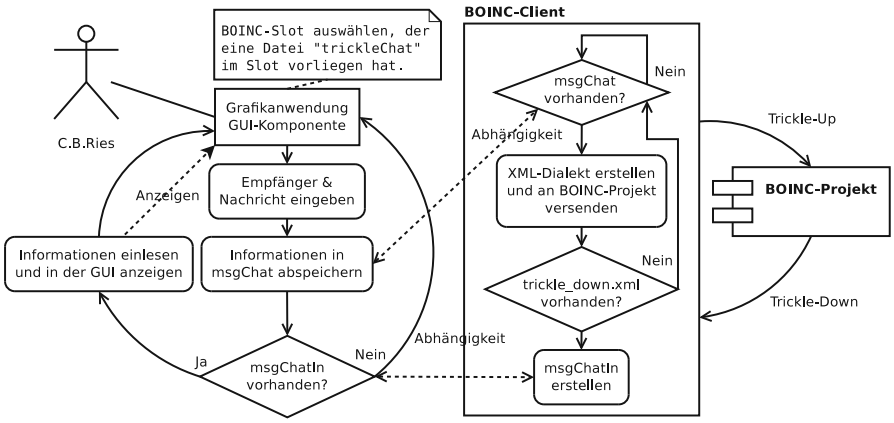


Abb. 9.2 Trickle-Messages mit Hilfe eines Chat-Dienstes für die Teilnehmer eines BOINC-Projektes

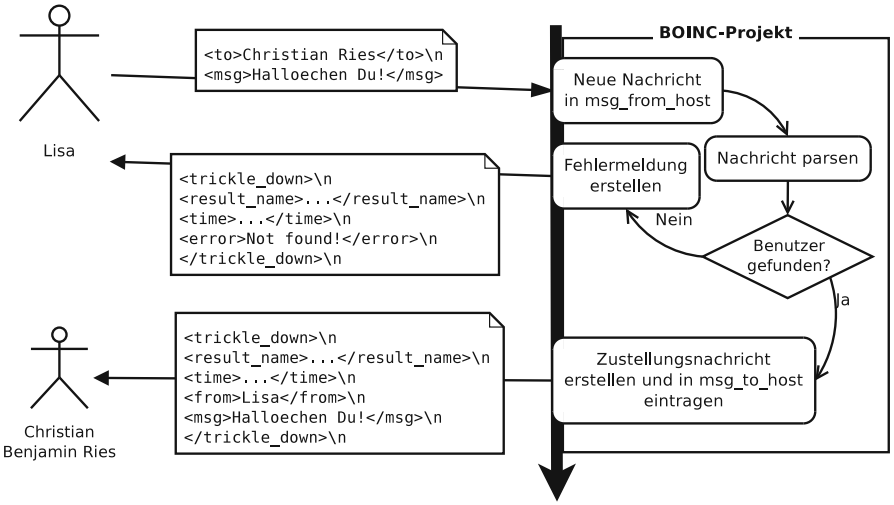


Abb. 9.3 Trickle-Sequence einer Kommunikation zwischen zwei BOINC-Teilnehmern

gesendet. Das BOINC-Projekt in Abb. 9.3 versucht dann die Hosts des Adressaten zu ermitteln. Sollte der Empfänger nicht gefunden werden, wird eine Fehlermeldung – mittlere Nachricht – an den Sender geschickt. Wenn der Empfänger existiert, so wird eine entsprechende Nachricht in der Projektdatenbank eingetragen und im nächsten RPC-Zyklus vom BOINC-Client entsprechend an die Hostrechner versendet. Wiederum in Abb. 9.2 schaut der Chat-Client, ob eine Trickle-Message vom BOINC-Projekt empfangen wurde. Diese ist gekennzeichnet durch eine vorhandene `trickle_down_XXX.xml`-Datei innerhalb des Slots. Wenn ja, so wird der Inhalt ausgelesen, in die Datei `msgChatIn` konvertiert und daraufhin von der GUI-Anwendung angezeigt.

9.4.4 BOINC-Komponenten für Trickle-Messages

Es ist sicherlich einleuchtend, dass mindestens zwei Komponenten vorhanden sein müssen, um eine Kommunikation zu ermöglichen. Im Fall von Trickle-Messages muss ein BOINC-Projekt mindestens einen zusätzlichen Dämon erhalten, der die eingehenden/ausgehenden Trickle-Messages vom/zum BOINC-Client verarbeitet. Die zwei Abschn. 9.4.4.1 und 9.4.4.2 liefern eine Testanwendung für diesen Fall und präsentieren, wie ein Trickle-Dienst eingerichtet werden kann.

9.4.4.1 BOINC-Server: Trickle-Dämon

BOINC liefert ein Beispiel mit, welches Nachrichten von allen BOINC-Teilnehmern verarbeitet und die gesamte Nachricht wieder an die BOINC-Teilnehmer zurücksendet; eine Art Ping-Pong. Dieses Beispiel kann als Basis für eigene Entwicklungen dienen, denn es enthält schon die main-Funktion und Hauptschleife für das Abfragen und Ermitteln von neuen Trickle-Messages innerhalb der Projektdatenbank.

Listing 9.21 enthält einen Trickle-Dämon, mit dem ein Chatserver umgesetzt wird, sprich Sie können auf einem Host eine Nachricht an einen weiteren BOINC-Teilnehmer senden und vice versa.

Listing 9.21 trickleHandler.cpp: Trickle-Dämon für das Verarbeiten von empfangenen Trickle-Messages für das Chatten zwischen mehreren BOINC-Teilnehmern

```
// BOINC
#include <util.h>
#include <boinc_db.h>
#include <str_util.h>
#include <error_numbers.h>

// BOINC Scheduler
#include <sched_config.h>
#include <sched_util.h>

// C++
#include <iostream>
#include <sstream>
#include <string>
#include <vector>

// Trickle-Handler for Chat@home
#include <chatMsg.h>

char variety[] = "msg";

/**
 * @param mfh Information of one Trickle-Message from a host.
 * @return If successfully 0 is returned, otherwise an higher value.
 */
int handle_trickle(MSG_FROM_HOST& mfh) {
  DB_HOST hostFrom;
  DB_USER userFrom;
  hostFrom.lookup_id(mfh.hostid);
  userFrom.lookup_id(hostFrom.userid);
```

Durch dieses Konstrukt kann der Absender jeder einzelnen in der Datenbank vorliegenden Trickle-Message ermittelt werden. Die Informationen eines Benutzers lie-

gen nun in der Objektinstanz `userFrom` vor und könnten im Folgenden auch dazu verwendet werden, die vom Absender aktuell bearbeiteten Arbeitspakete zu modifizieren oder dem Benutzer schon im Vorfeld Credits zuzuweisen.

```
CHATMSG m;
switch(m.parse(mfh.xml)) {
  case 1: /* Fehlerbehandlung */ break;
  case 0: {
    m.show();

    DB_MSG_TO_HOST mth;
    mth.clear();
    mth.handled = false;
    mth.create_time = time(0);
```

Diese vier Zeilen leiten die Erstellung einer Antwortnachricht zum Empfänger ein, wobei `handled` aussagt, dass diese Nachricht nicht durch den Scheduler bearbeitet wurde. `create_time` ist der Erstellungszeitraum; dieser Name steht später auch als Kennung im Trickle-down-Dateinamen.

```
    // iterate all host-ids
    char q[1024] = {'\0'};
    snprintf(q, 1024, "WHERE name='%s'", m.msgTo.c_str());
    DB_USER userTo;
    if(userTo.lookup(q)) {
      // Could not find user.
      /* Fehlerbehandlung, Nachricht zum Sender. */
      fprintf(stderr, "Could not find user: %s\n", m.msgTo.c_str());
      return 0;
    }
```

Es wird geprüft, ob der Benutzer innerhalb des BOINC-Projektes existiert, zu dem die Nachricht versendet werden soll. Wenn dieser nicht existiert, so wird eine Fehlernachricht an den Absender geschickt, wie dies auch durch die mittlere Nachricht in Abb. 9.3 zu sehen ist.

```
    DB_HOST userHost;
    char qHost[1024] = {'\0'};
    snprintf(qHost, 1024, "WHERE userid=%d ORDER BY userid ASC", userTo.id);
    std::cerr << "qHost=" << qHost << std::endl;
```

Wenn der Empfänger existiert, so werden alle von ihm am BOINC-Projekt angemeldeten Hostrechner erfragt, so dass an alle die Nachricht geschickt wird. Dem Absender ist nicht bewusst, wohin er die Nachricht senden soll, er will diese einfach nur an einen bestimmten Teilnehmer versenden. Daher muss der Trickle-Dämon entscheiden, wie mit einer solchen Anfrage umzugehen ist. Vorsichtshalber wird an alle Hostrechner gesendet. Nachfolgend wird über die Liste der registrierten Hostrechner iteriert.

```
    while(userHost.enumerate(qHost) == 0) {
      mth.hostid = userHost.id;

      DB_RESULT userResult;
      char qResult[1024] = {'\0'};
      snprintf(qResult, 1024,
               "WHERE hostid=%d AND userid=%d "
                   "AND received_time=0 "
                   "AND server_state=%d "
                   "AND outcome=%d ORDER BY name ASC",
               userHost.id, userTo.id,
               RESULT_SERVER_STATE_IN_PROGRESS /* 4 */,
```

```
                RESULT_OUTCOME_INIT /* 0 */
        );
        std::cerr << "qResult=" << qResult << std::endl;
```

Wir erfragen, ob ein Hostrechner ein zu bearbeitendes Arbeitspaket besitzt. Diese Abfrage ist daher wichtig, weil eine Trickle-down-Message für ein bestimmtes Arbeitspaket adressiert sein muss. Der BOINC-Client weiß dann, wohin eine solche Nachricht weitergeleitet muss, ergo in welchem Slot die Nachricht zur Verfügung gestellt werden muss. Aus diesen Bedingungen prüfen wir, ob eine Nachricht:

1. ob eine Nachricht noch keinen Zeitstempel besitzt, der eine Aussage über den Empfang macht,
2. ob das Arbeitspaket aktuell noch in Bearbeitung (`RESULT_SERVER_STATE_IN_PROGRESS`) ist und nicht durch einen weiteren Prozess abgebrochen wurde
3. und ob noch ein Ergebnis erwartet wird und daher der Ausgangswert auf dem Initialisierungswert gesetzt ist (`RESULT_OUTCOME_INIT`).

```
70      while(userResult.enumerate(qResult) == 0) {
          // Check if result is in progress and not cancelled.
          DB_WORKUNIT resultWU;
          char qWU[64] = {'\0'};
          snprintf(qWU, 64, "WHERE id=%d", userResult.workunitid);
75        std::cerr << "qWU=" << qWU << std::endl;
          resultWU.lookup(qWU);
          if(resultWU.error_mask != 0) {
            continue;
          }
```

Die Prüfungen sind noch nicht zu Ende. Selbst wenn wir nun alle Teile eines Arbeitspaketes kennen, die bei einem Teilnehmer berechnet werden, so kann das Arbeitspaket selbst schon als fehlerbehaftet deklariert worden sein, dies ist durch die `DB_RESULT`-Tabelle nicht zu erfragen. Aus diesem Grund muss für jeden Eintrag von `DB_RESULT` noch der Eigentümer – sprich das Arbeitspaket – ermittelt werden. Wenn dieses existiert und das Fehlerflag `error_mask` gleich Null ist, so kann dem bestimmten Teilnehmer die Chat-Nachricht zugestellt werden.

```
80        sprintf(mth.variety, "%s", variety);
          char msgOut[1024] = {'\0'};
          snprintf(
            msgOut, 1024,
            "<trickle_down>\n"
85          "<result_name>%s</result_name>\n"
            "<time>%d</time>\n"
            "<from>%s</from>\n"
            "<chat>%s</chat>\n"
            "</trickle_down>\n",
90          userResult.name, (int)time(NULL),
            userFrom.name, m.message.c_str())
          ;
          sprintf(mth.xml, "%s", msgOut);
          std::cerr << "[begin]" << std::endl;
95        std::cerr << "mth.xml=" << msgOut << "[end]" << std::endl;

          int retval = mth.insert();
```

Die Nachricht wird in den passenden XML-Dialekt übertragen, in die Datenbanktabelle `msg_to_host` eingetragen und, bei der nächsten Scheduler-Anfrage des

Empfängers, diesem auch zugestellt. In den oberen Zeilen erkennen Sie die untere Nachricht aus Abb. 9.3 wieder.

```
          if(retval) {
            /* Fehlerbehandlung */
            continue;
          }
        }
      }
      userHost.end_enumerate();
    } break;
  }
  return 0;
}
```

Die nachfolgende Funktion `do_trickle_scan()` prüft, ob eine Trickle-Message von einem BOINC-Teilnehmer vorliegt. Dabei werden nicht alle Trickle-Messages verarbeitet, sobald eine vorliegt. Durch die Angabe einer Varietät (engl. *variety*) in der SQL-Abfrage werden nur Chat-Nachrichten erfragt, bei denen `variety` gleich „msg“ sein muss, zudem darf die Nachricht schon bearbeitet sein. Sind diese Voraussetzungen erfüllt, so wird die Nachricht an den schon oben beschriebenen Prozess weitergeleitet, um daraufhin den Eintrag in der behandelten Tabelle zu aktualisieren, so dass die Nachricht noch einmal bearbeitet wird. Sollte der Prozess Fehler aufwerfen, so wird die Funktion beendet.

```
/**
 * Handles trickles messages when of is received.
 * @return TRUE when one Trickle-Message found,
 *         otherwise FALSE.
 */
bool do_trickle_scan() {
  bool found=false;

  char buf[256] = {'\0'};
  sprintf(buf, "where variety='%s' and handled=0", variety);

  while (1) {
    DB_MSG_FROM_HOST mfh;
    int retval = mfh.enumerate(buf);

    if(retval) {
      if(retval != ERR_DB_NOT_FOUND) {
        fprintf(stderr, "lost DB conn\n");
        exit(1);
      }
      break;
    }
    retval = handle_trickle(mfh);
    if(!retval) {
      mfh.handled = true;
      mfh.update();
    }
    found = true;
  }
  return found;
}

/**
 * @param one_pass Defines if handler will exit after first run.
 * @return 0 if anything worked perfect.
 */
int main_loop(bool one_pass) {
```

```
  bool did_something;

  int retval = boinc_db.open(
    config.db_name, config.db_host, config.db_user, config.db_passwd
  );
  if(retval) {
    log_messages.printf(MSG_CRITICAL, "boinc_db.open failed: %d\n", retval);
    exit(1);
  }
```

Es wird eine Datenbankverbindung initialisiert und daraufhin in einer while-Schleife alle fünf Sekunden geprüft, ob neue Trickle-Messages vorliegen. Die Funktion `check_stop_daemons()`[16] prüft, ob die Datei `stop_daemons` im Hauptordner eines BOINC-Projekts vorliegt; wenn ja, dann wird die Ausführung des Dämon beendet.

```
  while(1) {
    check_stop_daemons();
    did_something = do_trickle_scan();
    if (!did_something) {
      sleep(5);
    }
  }
  return 0;
}

/**
 */
void usage() {
  // ...
}

int main(int argc, char** argv) {
  int i, retval;
  bool one_pass = false;

  check_stop_daemons();

  for (i=1; i<argc; i++) {
    if (!strcmp(argv[i], "-d")) {
      if (!argv[++i]) {
        log_messages.printf(MSG_CRITICAL,
          "%s requires an argument\n\n", argv[--i]);
          usage(); exit(1);
      }
      int dl = atoi(argv[i]);
      log_messages.set_debug_level(dl);
      if (dl == 4) g_print_queries = true;
    } else {
      log_messages.printf(MSG_CRITICAL,
        "unknown command line argument: %s\n\n", argv[i]);
        usage(); exit(1);
    }
  }

  retval = config.parse_file();
  if (retval) {
    log_messages.printf(MSG_CRITICAL,
      "Can't parse config.xml: %s\n", boincerror(retval)
    );
    exit(1);
  }
```

[16] Definiert in der BOINC-Quelldatei `sched/sched_util.cpp`.

```
  install_stop_signal_handler();
  main_loop(one_pass);
}
```

9.4.4.2 BOINC-Client: Verarbeitung von Trickle-Messages

Der in Listing 9.22 enthaltene Trickle-Client ist eine Schnittstelle zwischen dem Anwender und einem BOINC-Projekt, um Nachrichten zwischen den BOINC-Teilnehmern eines BOINC-Projekts austauschen zu können.

Listing 9.22 trickleClient.cpp: Verarbeitet vom Benutzer erstellte Chat-Nachrichten, sendet diese zu einem BOINC-Projekt und stellt die eingehenden Nachrichten dem Benutzer zur Verfügung

```
// C++
#include <iostream>
#include <string>
#include <vector>

// C
#include <stdio.h>
#include <errno.h>

// BOINC
#include <boinc_api.h>
#include <filesys.h>
#include <util.h>

// Trickle-Handler for Chat@home
#include <chatMsg.h>

int main(int argc, char **argv)
{
  BOINC_OPTIONS options;
  options.handle_trickle_ups = true;
  options.handle_trickle_downs = true;
```

Diese zwei Optionen müssen auf *true* gesetzt werden, so dass das BOINC-Framework weiß, dass intern auf das Vorhandensein von Trickle-Messages geprüft werden soll. In diesem Zusammenhang können Sie entscheiden, ob nur Nachrichten empfangen, gesendet (unidirektional) oder beides (bidirektional) möglich sein soll.

```
  int returnValue = boinc_init_options(&options);
  if (returnValue) {
    std::cerr << "boinc_init returned " << returnValue << std::endl;
    exit(returnValue);
  }

  boinc_touch_file("trickleChat");

  while(1) {
    char bName[64] = {'\0'};
    char bMsg[512]  = {'\0'};

    if(boinc_file_exists("msgChat")) {
      FILE *f = boinc_fopen("msgChat", "r");
      if(f) {
        char *cp = NULL;
        cp = fgets(bName, 64, f);
        cp[strlen(cp)-1] = '\0';
```

```
        cp = fgets(bMsg, 512, f);
        cp[strlen(cp)-1] = '\0';
        fclose(f);
      }
      boinc_delete_file("msgChat");
    }
```

Es wird geprüft, ob sich im Slot eine Datei `msgChat` befindet. Diese muss von einem Anwender zur Verfügung gestellt werden und enthält eine zu versendende Nachricht und den Empfänger der Nachricht. Das Format ist ein trivialer Zweizeiler in der Form:

```
Christian Benjamin Ries
Halloechen du!
```

Der Inhalt dieser Datei wird in einen XML-Dialekt konvertiert. Das Ergebnis finden Sie in Abb. 9.3 in der ersten Nachricht von der BenutzerIn an das BOINC-Projekt. Nach dem Einlesen der Datei wird diese gelöscht, so dass sie nicht in der darauf folgenden Iteration der while-Schleife nochmals als Nachricht verschickt wird.

```
    if(strlen(bName) > 0 && strlen(bMsg) > 0) {
      char msgUp[1024] = {'\0'};
      snprintf(
        msgUp, 1024,
        "<to>%s</to>\n<chat>%s</chat>\n",
        bName, bMsg
      );
```

Der XML-Dialekt wird erstellt und mit den Informationen aus der Datei `msgChat` befüllt.

```
      int r = boinc_send_trickle_up((char*)"msg", msgUp);
      if(r) {
        fprintf(stderr, "Could not send trickle up message: %d\n", r);
      }
```

Der Aufruf von `boinc_send_trickle_up()` erstellt im Slot eine Datei mit dem Namen `trickle_up.xml`. Dieser Dateiname ist in der BOINC-Quelldatei `lib/app_ipc.h` hart-codiert. Wenn `options.handle_trickle_ups` auf *true* gesetzt wurde, so wird intern nach jeder Sekunde geprüft, ob die erwähnte XML-Datei vorliegt. Wenn ja, so wird sie an das BOINC-Projekt versendet und daraufhin im Slot gelöscht. Sollte eine Trickle-Message vorliegen und es wird nochmals die entsprechende Funktion zum Erstellen aufgerufen, so wird die vorherige Trickle-Message überschrieben. Wenn die einzelnen Nachrichten eine wichtige Rolle spielen, so wäre die Erstellung einer Queue von Vorteil.

```
    }

    char trickleFilename[32] = {'\0'};
    int retval = boinc_receive_trickle_down(trickleFilename, 32);
```

Dieser Aufruf führt eine Überprüfung auf die Existenz einer Datei im jeweiligen Slot aus, dabei werden alle Dateinamen geprüft und geschaut, ob das Prefix eines Dateinamens `trickle_down` entspricht. Der volle Dateiname wird vom BOINC-Client in der Quelldatei `client/cs_client.cpp` kreiert und hat die Form: `sprintf(path,"%s/trickle_down_send_time);` Der Dateiname

gibt dementsprechend auch Auskunft darüber, wann diese Nachricht versendet wurde, selbige Information befindet sich auch im XML-Dialekt innerhalb dieser Datei. Sie können durch diesen Mechanismus mehrere Trickle-Messages empfangen, die dank dieser Dateinamen geordnet — aufsteigend nach dem Zeitwert – durch diese Funktion verarbeitet werden.

```
    std::string trickleMsg;
    FILE *trickleFile = boinc_fopen(trickleFilename, "r");
    if(trickleFile) {
      char b[512] = {'\0'};
      while(fgets(b, 512, trickleFile)) {
        trickleMsg += b;
      }
      fclose(trickleFile);
      boinc_delete_file(trickleFilename);
    }
```

Die Trickle-Message öffnen und die XML-Zeilen einlesen. Daraufhin die Nachricht löschen, so dass im nächsten Iterationsschritt auch eine nachfolgende Nachricht Beachtung findet.

```
    if(retval) {
      CHATMSG m;
      switch(m.parse(trickleMsg)) {
        case 2:
        case 1: /* Fehlerbehandlung */ break;
        case 0: {
          m.show();
          FILE *f = boinc_fopen("msgChatIn", "w");
          if(f) {
            fprintf(f, "%s\n", m.msgFrom.c_str());
            fprintf(f, "%s\n", m.message.c_str());
            fclose(f);
          } else {
            fprintf(stderr, "Could not open msgChatIn file for writting.\n");
            fprintf(stderr, "Got message(%s) from %s\n",
              m.message.c_str(), m.msgFrom.c_str());
          }
        } break;
      }
    }
```

Die Nachricht in die Datei `msgChatIn` übertragen; das Format entspricht dem oben gezeigten Zweizeiler und unterscheidet sich nur in der ersten Zeile. Hier steht nun nicht der Empfänger, sondern der Absender.

```
    std::cerr.flush(),
    boinc_sleep(1);
  }

  boinc_fraction_done(1);
  boinc_finish(0);
}
```

9.4.4.3 Trickle-Messages: CHATMSG

Das Listing 9.23 enthält die in den vorherigen Listings 9.21 und 9.22 genutzte Struktur, um Trickle-Messages zu parsen und deren Informationen für die spätere Verwendung zu „lagern“.

Listing 9.23 chatMsg.h

```
struct CHATMSG {
  std::string msgFrom;
  std::string msgTo;
  std::string message;

  CHATMSG();

  /** @return Error code: 0 (success), 1 (no data), 2 (wrong msg format)
   */
  int parse(std::string xmlmsg);
  void show();
};
```

Diese drei Variablen dienen dem Abspeichern der Kommunikationsdaten. Dabei ist es abhängig von der Kommunikationsrichtung, welche der drei Variablen tatsächlich Verwendung finden. `message` wird in jedem Fall verwendet; danach ist es abhängig von der Station der Abarbeitung, ob eine Nachricht gelesen oder versendet wird.

Wenn der Trickle-Client eine Nachricht empfängt, so ist die `msgTo`-Variable leer und in `msgFrom` steht der Name des Absenders; es handelt sich dabei um den Benutzernamen des Senders. Beim Versenden einer Nachricht wird `msgFrom` leer gelassen – der Trickle-Handler kann den Absender aus der Datenbank holen – in `msgTo` steht der Empfänger in Form des Benutzernamens.

`message` wird abhängig von der Kommunikationsrichtung mit Daten eines bestimmten Formats gefüllt. Das Format mit einer beispielhaften Kommunikation zwischen zwei BOINC-Teilnehmern finden Sie in Abb. 9.3.

`parse()` liest die Information einer Trickle-Message ein. Wenn keine Daten vorhanden oder das Nachrichtenformat in einer falschen Form vorliegt, so wird das mit einer Fehlernummer signalisiert.

Listing 9.24 chatMsg.cpp

```
// Trickle-Handler for Chat@home
#include <chatMsg.h>

// C++
#include <iostream>
#include <sstream>
#include <string>

// BOINC
#include <parse.h>

StdVectorString &split(
  std::string &s, char delim,
  StdVectorString &elems)
{
  std::stringstream ss(s);
  std::string item;
  while(std::getline(ss, item, delim)) {
    elems.push_back(item);
  }
  return elems;
}

StdVectorString split(std::string &s, char delim) {
  StdVectorString elems;
  return split(s, delim, elems);
}
```

Die beiden Funktionen `split` dienen dazu, eine Zeichenkette zu teilen. In diesem Beispiel werden diese Funktionen zum Teilen der Trickle-Messages genutzt, wobei das *Newline-Character* als Trennungszeichen fungiert, so dass die einzelnen Zeilen einer Nachricht gefiltert werden können.

```
CHATMSG::CHATMSG() {
  message = "No message.";
}

int CHATMSG::parse(std::string xmlmsg) {
  StdVectorString lines = split(xmlmsg, '\n');
  if(lines.size() <= 0) return 1;
  StdVectorString::iterator it = lines.begin();
  for( ; it != lines.end(); it++) {
    const char *p = it->c_str();
    if(parse_str(p, "to", msgTo)) continue;
    if(parse_str(p, "from", msgFrom)) continue;
    if(parse_str(p, "chat", message)) continue;
  }
  return 0;
}
```

`parse` dient dem Filtern der Nachrichtenteile einer Trickle-Message. Sie kennen die einzelnen Funktionsbereiche schon aus Abschn. 7.3.8, im Gegensatz zu der Erläuterung in den Grundlagen wird hier ein einfacher Text und keine Datei mit XML-Informationen genutzt.

```
void CHATMSG::show() { /* Debugging */ }
```

Die Funktion `show` dient nur dem Anzeigen der geparsten Informationen und soll das Debuggen vereinfachen.

9.5 Erweiterte BOINC-Modifikationen erstellen

Das BOINC-Serversystem besitzt einige Rädchen, an denen gedreht werden darf. In den nachfolgenden Abschnitten beschreibe ich Ihnen, welche Stellen das sind und wie Sie eigene Implementierungen realisieren können.

9.5.1 Datenbank: das Hirn eines BOINC-Projekts

Es kann vorkommen, dass Sie in Ihrem BOINC-Projekt weitere Datenbanktabellen benötigen, um zum Beispiel Ergebnisse abspeichern zu können, was durch den Prozess mit Hilfe eines Assimilierers möglich ist. Das BOINC-Framework besitzt eine Schnittstelle für den Zugriff auf Datenbanktabellen. Dabei wird jede Datenbanktabelle durch eine C++-Klasse beschrieben, welche wiederum eine einheitliche Basisschnittstelle besitzt. Es werden für den Zugriff nur die MySQL-Client-Bibliotheken inklusive Programmierbibliotheken benötigt. Abschnitt 5.3 beschreibt die Installation dieser Pakete für eine Ubuntu-Installation.

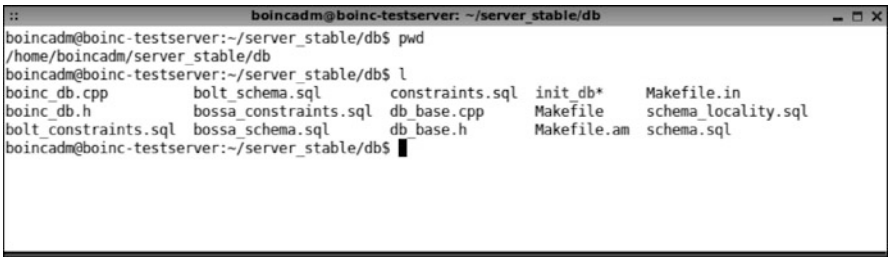


Abb. 9.4 Die BOINC-Quellen für den Zugriff auf die MySQL Datenbanktabellen

Abstraktionsschicht der BOINC-Datenbankstruktur

Die für den aktuellen Abschnitt benötigten Header-Dateien befinden sich im Unterordner *db/* innerhalb der BOINC-Quellen. Abbildung 9.4 zeigt die in diesem Ordner vorliegenden Dateien. Nicht alle Datenbanktabellen sind für die Erstellung von eigenen BOINC-Dämonen oder BOINC-Tasks nötig. Ein Großteil der Datenbanktabellen wird für die Web-2.0 Möglichkeiten genutzt, wie zum Beispiel für die Darstellung von Foren und die Erstellung von Nachrichten auf der Webseite eines BOINC-Projekts (vgl. Abschn. 5.7.1). Abbildung 9.5 zeigt die Hierarchie einiger C++-Klassen, die für den Zugriff auf die Datenbanktabellen vorhanden sind:

DB_PLATFORM Diese Tabelle enthält die Informationen über die generell unterstützten Plattformen einer wissenschaftlichen Applikation. Das Werkzeug `bin/xadd` füllt diese Tabelle mit den Plattforminformationen, welche in der Datei `project.xml` definiert sind. Mit Hilfe des Werkzeugs `bin/appmgr` können einzelne Plattformen auf bequemem Wege aus dieser Tabelle gelöscht werden.

DB_APP Informationen über eine wissenschaftliche Applikation.

DB_APP_VERSION Informationen über die einzelnen Versionen einer wissenschaftlichen Applikation, Unterscheidung zwischen neuen Versionsnummern oder unterschiedlichen unterstützten Plattformen.

DB_USER Alle Teilnehmer eines BOINC-Projekts sind in dieser Tabelle vorzufinden. Jede Zeile beschreibt einen Teilnehmer.

DB_TEAM Jeder Teilnehmer kann Mitglied eines Teams sein, diese Teams werden von einer zentralen Webseite geregelt und die Informationen über alle existierenden Teams müssen von dieser Webseite importiert werden. Daraufhin können sich die Teilnehmer zu einem Team hinzufügen. Bei einer ersten Registrierung an einem BOINC-Projekt mit Hilfe des BOINC-Managers wird automatisch ein Webbrowser gestartet, daraufhin können Sie sich selber einem BOINC-Team hinzufügen.

DB_HOST Beschreibungen der registrierten Rechner eines Teilnehmers sind in dieser Tabelle hinterlegt. Jeder Rechner wird durch eine Datenreihe beschrieben und enthält die Identifikationsnummer eines BOINC-Teilnehmers des Besitzers. Ein Rechner wird durch zahlreiche Informationen beschrieben, u. a. die Prozessoridentifikation, die IP-Adresse der letzten Verbindung zum BOINC-Projekt,

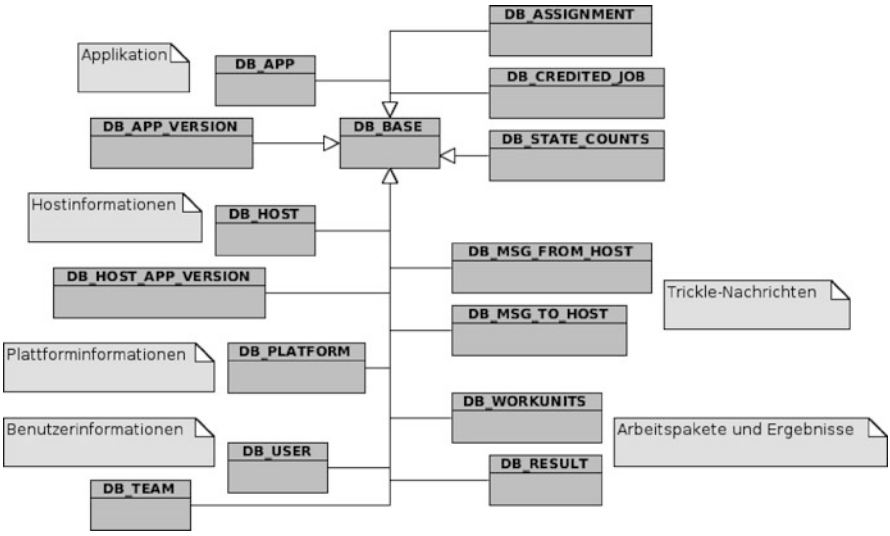


Abb. 9.5 C++-Klassenhierarchie der Datenbank-Abstraktionsschicht von BOINC

wie viel Rechenleistung der Host besitzt, welches Betriebssystem installiert ist, die Menge an Speicher, der für das aktuelle BOINC-Projekt zur Verfügung steht, und noch einige weitere Informationen.

DB_WORKUNIT Beschreibungen eines Arbeitspaketes stehen in dieser Tabelle, zusätzlich auch die Informationen, wo die zugehörigen Eingabedateien herunterzuladen sind. Dafür liest der BOINC-Scheduler die Informationen aus und liefert sie einem BOINC-Teilnehmer. Weiterhin enthält jeder Eintrag Informationen über den aktuellen Berechnungsstatus und in welchem Abarbeitungsschritt sich ein Arbeitspaket befindet, z. B. ob noch validiert werden muss (`need_validate = 1`).

DB_RESULT Ein Arbeitspaket kann gesplittet, an unterschiedliche BOINC-Teilnehmer versendet und bearbeitet werden. Die einzelnen Teile eines Arbeitspaketes werden *Results* genannt. Dies ist u. a. nötig, um eine Validierung der einzelnen Arbeitspakete zu ermöglichen. Alle *Results* werden in einzelnen Datenreihen in der Tabelle `DB_RESULT` abgelegt und im Arbeitsverlauf des Transitioner werden diese mit Hilfe von Standardregeln oder selbst definierten Regeln geprüft. Dadurch werden die Results für die nächsten Abarbeitungsschritte freigegeben oder nicht. Zum Beispiel müssen bei der Konfiguration `min_quorum = 2` mindestens zwei *Results* vorliegen, so dass eine Validierung möglich ist.

DB_CREDITED_JOB Die Zuordnung von einem erfolgreichen Berechnungsergebnis zu einem BOINC-Teilnehmer findet sich in dieser Tabelle. Hieraus berechnen sich die Credits eines Teilnehmers.

DB_HOST_APP_VERSION Dient der Berechnung der Credits, abhängig von der Rechenleistung eines Hosts und der wissenschaftlichen Applikationen einer Plattform.

DB_MSG_FROM_HOST Die zu verarbeitenden Trickle-Messages, welche von einem Host an ein BOINC-Projekt gesendet wurden.

DB_MSG_TO_HOST Die zu versendenden Trickle-Messages, welche einem BOINC-Teilnehmer zugestellt werden, wenn sich dieser wieder bei einem BOINC-Projekt meldet, eventuell Berechnungsergebnisse melden und neue Arbeitspakete herunterladen will.

DB_ASSIGNMENT Diese Tabelle beinhaltet Informationen über einzelne Arbeitspakete, die entweder für einen bestimmten Host, ein bestimmtes Team oder für einen bestimmten BOINC-Teilnehmer gedacht sind [192]. Dieses Feature existiert zwar, allerdings ist es anscheinend nicht sehr gut getestet und sollte nicht ohne weitere persönliche Tests verwendet werden.

Hinzufügen einer BOINC-Datenbanktabelle

In diesem Abschnitt erweitern wir die Datenbank um eine weitere Tabelle. Diese Tabelle soll es ermöglichen, dass Benutzer zu bestimmten Benutzerrollen hinzugefügt werden können. Es kann in eigenen Entwicklungen zum Beispiel wichtig sein, dass nicht alle Mitarbeiter in einem Unternehmen Zugriff auf das BOINC-Projekt erhalten sollen. Es kann in diesem Fall eine Hierarchie für die Zugriffsregelungen errichtet werden und die Entwickler und Administratoren eines BOINC-Projekts können für spezielle Aufgaben definiert werden. In unserem Beispiel hier werden wir vier Gruppen einrichten:

1. Administrator (engl. *administrator*),
2. Entwickler (engl. *developer*),
3. Wissenschaftler (engl. *scientist*) und
4. Tester (engl. *tester*)

Wir listen an dieser Stelle die deutschen und englischen Bezeichnungen auf; grundsätzlich sollten Sie aber nur englische Namen verwenden, um so eine weite Verbreitung Ihrer Implementierungen zu erzielen, wenn Sie Ihre Implementierungen eventuell unter eine Open-Source-Lizenz stellen sollten. Jede Rolle besitzt einen definierten Aufgabenbereich, der ganz nach Ihren Wünschen angepasst werden kann. In diesem Fall definieren wir uns folgende Regelungen:

Administrator Der Administrator hat die Möglichkeit, ein BOINC-Projekt zu starten, zu stoppen und neu zu starten. Weiterhin können Benutzer dieser Rolle Fehlerprotokolle einsehen und Konfigurationen für die Laufzeit modifizieren.

Entwickler Benutzer mit dieser Rolle dürfen neue wissenschaftliche Applikationen erstellen und zum BOINC-Projekt hinzufügen oder auch entfernen. Weiterhin ist es erlaubt, die Konfigurationsdateien der wissenschaftlichen Applikationen zu modifizieren und ggf. zu ersetzen.

Wissenschaftler Alle Benutzer im Besitz dieser Rolle können neue Arbeitspakete erstellen und Ergebnisse einsehen und nutzen. Es dürfen aber keine Modifikationen an den Laufzeiteinstellungen eines BOINC-Projekts vorgenommen werden, die eventuell Einfluss auf das Verhalten haben könnten. Diese

Rolle beschreibt prinzipiell einen rein lesenden Zugriff auf Daten eines BOINC-Projekts.

Tester Diese Rolle ermöglicht vollen Zugriff auf alle Daten, Konfigurationen und Applikationen eines BOINC-Projekts. Diese Rolle sollte nur in Testinstallationen definiert sein und vor einem produktiven Release gelöscht oder deaktiviert werden.

Listing 9.25 zeigt den SQL-Aufruf zum Erstellen der neuen Tabelle, wobei angenommen wird, dass die neue Tabelle in der MySQL-Datenbank *tah* installiert werden soll. *tah* ist der Name unseres Beispiel-BOINC-Projekts aus Abschn. 5.6.1. Natürlich kann dieser Datenbankname durch einen für Sie passenden Namen ersetzt werden.

Listing 9.25 SQL-Aufruf zum Erstellen der Tabelle für die Verwendung von Benutzerrollen

```
CREATE TABLE IF NOT EXISTS `tah`.`roles` (
  `id` INT NOT NULL AUTO_INCREMENT PRIMARY KEY ,
  `userid` INT NOT NULL ,
  `appid` INT NOT NULL ,
  `role` INT( 5 ) NOT NULL ,
  `description` VARCHAR( 254 ) NOT NULL
) ENGINE = MYISAM ;
```

ID ist ein ganzzahliger Zahlenwert, der mit jedem neuen Eintrag um Eins inkrementiert wird. So wird ein eindeutiger Zugriff auf eine Zeile in der Tabelle ermöglicht und dies ermöglicht, direkt und einfach SQL-Anfragen zu erstellen, zumindest wenn die *ID* bekannt ist. *userid* und *appid* sind virtuelle Verweise auf die jeweiligen Einträge in den Tabellen *user* und *app*. *role* beschreibt die Rolle, die ein Benutzer zu einer wissenschaftlichen Applikation besitzt. Alle diese Identifikatoren in einem Datensatz beschreiben eine Zugriffsregelung eines Benutzers zu einer bestimmten wissenschaftlichen Applikation. *description* enthält eine kurze prägnante Beschreibung eines Eintrags; zum Beispiel beschreibt „Benutzer einer Abt. Forschung, Spinhenge, Admin“, dass der Benutzer in der Abteilung für die Arbeit im Spinhenge-Projekt angestellt ist und ein Projekt administrieren kann. Sollte ein Benutzer mehrere Rollen erhalten, so muss jede Zuordnung einzeln vorgenommen werden. Die logische Abarbeitung dieser Definitionen muss in diesem Beispiel von den jeweiligen BOINC-Projekten übernommen werden, und dies erfordert einiges an Implementierungsaufwand.

BOINC-DB-Abstraktionsschicht mit der neuen Tabelle erweitern

Wie in Abb. 9.4 zu sehen, ist die Basis für jede neue C++-Klasse für den Zugriff auf eine BOINC-Datenbanktabelle die C++-Klasse *DB_BASE*. Wir müssen diese Klasse – in der Sprache des Systems-Engineering – spezialisieren. Dies kann durch den C++-Mechanismus der Vererbung geschehen. Weiterhin wird jede Datenbanktabelle durch eine einfache Struktur beschrieben. Eine Struktur besitzt für jede Tabellenspalte einer Datenbank eine Variable mit einem passenden Datentyp. Tabelle 9.1 enthält eine Auflistung der typischen Datenbank-Datentyp-Verhältnisse zwischen den MySQL-Datentypen und denen der Programmiersprache C++. Weiterhin besitzt

Tab. 9.1 Beziehungen zwischen den MySQL- und C++-Datentypen innerhalb des BOINC-Frameworks

MySQL-Datentyp	C++-Datentyp
‘app‘.‘id‘ INT	int id;
‘app‘.‘name‘ VARCHAR(254)	char name[256];
‘app‘.‘deprecated‘ SMALLINT(6)	bool deprecated;
‘app‘.‘weight‘ DOUBLE	double weight;
‘app_version‘.‘xml_doc‘ MEDIUMBLOB	char xml_doc[APP_VERSION_XML_BLOB_SIZE];
‘user‘.‘global_prefs‘ BLOB	char global_prefs[BLOB_SIZE];

eine solche Struktur immer eine Methode *clear()*, mit der die Variablen gelöscht werden können. Die Speicherstellen der Variablen werden in unserem Beispiel in Listing 9.27, Zeile 9 auf Null gesetzt.

Listing 9.26 zeigt die Definitionen der neuen Struktur und die Spezialisierungen der Klassen *DB_BASE* und *ROLES*. Die Konstanten der Rollendefinitionen werden hart-codiert und stellen eine Bitmaske dar, wobei die Angaben in den Zeilen 8–12 die dezimale Darstellung verwenden. Zur Erinnerung, die binäre Form der Konstanten lautet:

$$0_{10} = 0000_2$$
$$1_{10} = 0001_2$$
$$2_{10} = 0010_2$$
$$4_{10} = 0100_2$$
$$8_{10} = 1000_2$$

Dies ermöglicht, bei der Abfrage der Rollenzugehörigkeit eine einfache Bitoperation durchzuführen um zu prüfen, ob ein Benutzer eine bestimmte Rolle besitzt. Die Methode `get_id()` gibt die Identifikationsnummer eines Datensatzes zurück, dieser Wert ist eineindeutig. Mit `db_print()` können die Datenspalten mit zugehörigem Wert abgefragt werden, die Informationen werden im übergebenen Zeichenspeicher *buf* abgespeichert. Der Entwickler trägt Sorge bezüglich der ordentlichen Größe dieses Speichers, da dieser nicht vom System überprüft wird und dadurch eine Sicherheitslücke entsteht oder die Applikation unerwartet abstürzt. Mit `db_parse()` werden die Daten einer Abfrage in den Variablen der Struktur *ROLES* abgespeichert. Diese Methode muss nicht manuell aufgerufen werden, sondern wird durch die gleichnamige virtuelle Funktion aus der Basisstruktur aufgerufen.

Listing 9.26 Implementierung der Header-Datei einer neuen BOINC-Datenbanktabellenabstraktion für die Verwendung von Benutzerrollen

```
#ifndef __DB_ROLES_H__
#define __DB_ROLES_H__

#include <db/db_base.h>
#include <db/boinc_db.h>
```

```
// Roles
#define ROLE_UNKNOWN        0
#define ROLE_ADMINISTRATOR  1
#define ROLE_DEVELOPER      2
#define ROLE_SCIENTIST      4
#define ROLE_TESTER         8

struct ROLES {
  int id;
  int userid;
  int appid;
  int role;
  char description[256];
  void clear();
};

struct DB_ROLES : public DB_BASE, public ROLES {
public:
  DB_ROLES(DB_CONN* p=0);
  int get_id();
  void db_print(char* buf);
  void db_parse(MYSQL_ROW &row);
};

#endif
```

Listing 9.27 enthält die Implementierungen der zuvor beschriebenen Methoden. Beim Erstellen einer neuen Instanz der Struktur *DB_ROLES* kann dem Konstruktor ein optionaler Parameter übergeben werden. Dieser Parameter kann eine vorhandene Datenbankverbindung beschreiben; sollte dieser Wert Null sein, so wird die globale Datenbankverbindung genutzt, die immer vorhanden ist, wenn mindestens ein Dämon gestartet ist. Dies liegt daran, dass in Zeile 12 die globale Variable *boinc_db* verwendet wird, die im Vorfeld vom Entwickler gesetzt werden muss.

Listing 9.27 Implementierung der Quelldatei einer neuen BOINC-Datenbanktabellenabstraktion für die Verwendung für Benutzerrollen

```
#include <lib/str_util.h>
#include <lib/str_replace.h>
#include <db/boinc_db.h>

// @see boinc_db.cpp
#define ESCAPE(x) escape_string(x, sizeof(x))
#define UNESCAPE(x) unescape_string(x, sizeof(x))

void ROLES::clear() { memset(this, 0, sizeof(*this)); }

DB_ROLES::DB_ROLES(DB_CONN* dc) :
  DB_BASE("roles", dc?dc:&boinc_db){}

int DB_ROLES::get_id() { return id; }

void DB_ROLES::db_print(char *buf) {
  ESCAPE(description);
  sprintf(buf,
    "userid=%d, appid=%d, role=%d, description=%s",
    userid, appid, role, description);
  UNESCAPE(description);
}

void DB_ROLES::db_parse(MYSQL_ROW &r) {
  int i=0;
  clear();
  id = atoi(r[i++]);
```

```
  userid = atoi(r[i++]);
  appid = atoi(r[i++]);
  role = atoi(r[i++]);
  strcpy2(description, r[i++]);
}
```

Durch die bis hier implementierten Strukturen und Funktionen reichen im folgenden Beispiel in Listing 9.28 wenige Zeilen aus, um dadurch den gesamten Inhalt dieser Tabelle auszulesen. Zeile 19 öffnet eine Datenbankverbindung, die daraufhin für alle nachfolgenden Routinen und Aufrufe über die globale Variable *boinc_db* zur Verfügung steht. In Zeile 33 wird eine Instanz der Struktur *DB_ROLES* erstellt, mit deren Hilfe die Abfragen durchgeführt werden. Durch die von der Basisstruktur *DB_BASE* zur Verfügung gestellte Methode `count()` kann die Anzahl der Datensätze ermittelt und in Zeile 36 ausgegeben werden. Die while-Schleife in den Zeilen 38–40 enumeriert über jeden einzelnen Datensatz und gibt die Beschreibung jeder einzelnen Benutzerrollendefinition aus. In Zeile 41 wird das Ende der Enumeration signalisiert. Der Anfang der Enumeration wird mit der Methode `enumerate()` begonnen. Sollte beim Aufruf keine bisherige Abfrage erfolgt sein, so wird eine SQL-Anweisung mit Hilfe von *SELECT* durchgeführt, die dafür sorgt, dass alle Datensätze durchwandert werden. Als optionale Parameter können zudem Bedingungen an die Enumeration weitergegeben werden. Zum Beispiel könnten nur die Rollendefinitionen eines bestimmten Benutzers abgefragt werden: `roles.enumerate("WHERE userid=1")`.

Listing 9.28 Beispiel zur Verwendung der neuen Rollentabelle in der BOINC-Projekt-Datenbank

```
// C
#include <stdio.h>
// C++
#include <string>
#include <iostream>
// BOINC
#include <db/boinc_db.h>
#include <sched/sched_config.h>
// Extension
#include <roles_db.h>

int main( int argc, char **argv) {
  int retval = config.parse_file("./");
  if(retval) {
    printf("could not open BOINC configuration\n");
    exit(1);
  }

  retval = boinc_db.open(config.db_name, config.db_host,
                         config.db_user, config.db_passwd);
  if(retval) {
    printf("could not connect to database(%s@%s): %d\n",
           config.db_user, config.db_host, retval);
    exit(1);
  }

  // Values of role definitions:
  std::cout << "UNKNOWN: "       << ROLE_UNKNOWN << std::endl;
  std::cout << "ADMINISTRATOR: " << ROLE_ADMINISTRATOR << std::endl;
  std::cout << "DEVELOPER: "     << ROLE_DEVELOPER << std::endl;
  std::cout << "SCIENTIST: "     << ROLE_SCIENTIST << std::endl;
  std::cout << "TESTER:"         << ROLE_TESTER << std::endl;

  DB_ROLES roles;
```

```
  int roles_count = 0;
  roles.count(roles_count);
  std::cout << "Roles Count: "   << roles_count << std::endl;

  while(roles.enumerate() == 0) {
    std::cout << roles.description << std::endl;
  }
  roles.end_enumerate();

  return 0;
}
```

Eine zielgerichtete Abfrage einer bestimmten Rolle kann wie in Listing 9.29 gezeigt aussehen. Die Funktion erwartet die Beschreibung eines Benutzers *user*, die Datenbankverbindung innerhalb der Datenbankstruktur *DB_ROLES* und die zu prüfende Rollendefinition.

Listing 9.29 Zielgerichtete Abfrage einer Benutzerrolle

```
bool role_user( DB_USER& user, DB_ROLES& role, int role_id ) {
  char q[32] = {'\0'};
  sprintf(q, "where userid=%d AND role=%d", user.id, role_id);
  if(role.lookup(q)) {
    return false;
  }
  return true;
}
```

Kapitel 10
Debugging innerhalb des BOINC-Frameworks

Fehler: Tastatur nicht angeschlossen. Bitte Taste F1 drücken!

Wenn Sie Glück haben, dann stürzen Ihre Anwendungen ab und Sie können sich bewusst darüber werden, dass es Probleme innerhalb der Software gibt. Das andere Verhalten könnte zu gravierenden Fehlern in der Abarbeitung und womöglich zu verfälschten Berechnungen oder zu Fehlern bei der Abspeicherung führen. BOINC unterstützt Sie beim Untersuchen Ihrer Infrastruktur und des Laufzeitverhaltens Ihrer verteilten wissenschaftlichen Applikationen. Lernen Sie in diesem Kapitel, wie Sie sich Informationen beschaffen können, wie diese zu lesen und zu deuten sind.

10.1 Diagnoseoptionen für wissenschaftliche Applikationen

Zu Beginn der Ausführung einer wissenschaftlichen Applikation können Sie mit Hilfe der BOINC-API einige Flags setzen, welche Ihnen das Diagnostizieren ermöglicht, um dadurch Fehlerquellen aufzuspüren. Es gibt die Möglichkeit, Diagnosen für einfache wissenschaftliche Applikationen sowie einen Bildschirmschoner zu erstellen. Das Listing 10.1 enthält die zwei BOINC-Funktionen, um die Diagnose zu modifizieren, denn standardmäßig werden die Diagnose-Flags aus Listing 10.2 eingestellt. Es gilt zusätzlich zu beachten, dass ein jeweiliger Aufruf für die Initialisierung vor der eigentlichen Initialisierung geschieht.

Listing 10.1 Funktion für das Modifizieren der Diagnoseoptionen

```
#include <lib/diagnostics.h>
2
int boinc_init_diagnostics(int flags);
int boinc_init_graphics_diagnostics(int flags);
```

`flags` erwartet einen Integer-Wert, der eine ODER-Verknüpfung der Diagnose-Flags beinhaltet. Die möglichen Werte finden Sie in Listing 10.3, mit dazugehöriger Beschreibung der jeweiligen Option.

C. B. Ries, *BOINC*, Xpert.press,
DOI 10.1007/978-3-642-23382-8_10, © Springer-Verlag Berlin Heidelberg 2012

Listing 10.2 Funktion für das Initialisieren einer BOINC-Grafikanwendung

```
#include <lib/diagnostics.h>

#define BOINC_DIAG_DEFAULTS \
    BOINC_DIAG_DUMPCALLSTACKENABLED | \
    BOINC_DIAG_HEAPCHECKENABLED | \
    BOINC_DIAG_MEMORYLEAKCHECKENABLED | \
    BOINC_DIAG_REDIRECTSTDERR | \
    BOINC_DIAG_TRACETOSTDERR
```

Die einzelnen Werte in Listing 10.3 beschreiben ein Bitfeld und jede Option setzt ein Bit in diesem Bitfeld auf Eins. Mehrere Flags können Sie durch eine ODER-Verknüpfung miteinander kombinieren und den Funktionen in Listing 10.1 übergeben. Sie können die einzelnen gesetzten Flags mit einer UND-Verknüpfung wieder deaktivieren.

Listing 10.3 Mögliche Diagnose-Flags für das Untersuchen der Ausführung einer wissenschaftlichen Applikation

```
#include <lib/diagnostics.h>

#define BOINC_DIAG_DUMPCALLSTACKENABLED       0x00000001L
  // Wenn die Applikation abstuerzt, dann kann der vorherige
  // Verlauf und die Verwendung des Stacks nachverfolgt werden.
#define BOINC_DIAG_HEAPCHECKENABLED           0x00000002L
  // Prueft den Heap auf deren Integritaet.
#define BOINC_DIAG_MEMORYLEAKCHECKENABLED     0x00000004L
  // Nach Beendigung der Ausfuehrung werden alle noch
  // reservierten Speicherstellen nach stderr geschrieben.
#define BOINC_DIAG_ARCHIVESTDERR              0x00000008L
#define BOINC_DIAG_ARCHIVESTDOUT              0x00000010L
  // Bei sequentiellen Start der Applikation werden
  // vorhandene stderr- und stdout-Dateien nach *.old kopiert.
#define BOINC_DIAG_REDIRECTSTDERR             0x00000020L
  // Der Ausgabekanal stderr wird in die gleichnamige Datei
  // und beim Bildschirmschoner in die Datei stderrgfx umgeleitet.
#define BOINC_DIAG_REDIRECTSTDOUT             0x00000040L
  // Der Ausgabekanal stdout wird in die gleichnamige Datei
  // und beim Bildschirmschoner in die Datei stdoutgfx umgeleitet.
#define BOINC_DIAG_REDIRECTSTDERROVERWRITE    0x00000080L
#define BOINC_DIAG_REDIRECTSTDOUTOVERWRITE    0x00000100L
  // Diese beiden Flags haengen die Ausgaben jeweils an die
  // zuvor beschriebenen Ausgabedateien.
#define BOINC_DIAG_TRACETOSTDERR              0x00000200L
#define BOINC_DIAG_TRACETOSTDOUT              0x00000400L
  // Flags fuer Windows, Ausgaben werden in stderr und stdout
  // geschrieben.
#define BOINC_DIAG_HEAPCHECKEVERYALLOC        0x00000800L
  // Prueft die Integritaet des Heaps, wenn Speicher
  // durch den Entwickler alloziiert wird.
#define BOINC_DIAG_BOINCAPPLICATION           0x00001000L
  //
```

Ich musste in meinen bisherigen Entwicklungsarbeiten noch nie die Diagnose-Flags modifizieren, bisher konnte ich durch reines Prüfen und Analysieren der Protokolle alle bisher aufgekommenen Fehler finden. Im Abschn. 10.2 erhalten Sie eine Beschreibung der Protokolle und welche Informationen dort zu finden sind.

10.2 Protokollierungsdateien

Während der Laufzeit erstellt jede Komponente eine Protokollierungsdatei; der Ort der Abspeicherung kann in der `config.xml` eines BOINC-Projekts festgelegt werden. Dies kann für die einzelnen Tasks sowie Dämonen direkt durchgeführt werden. In beiden Fällen muss `<output>...</output>` gesetzt werden; ein Beispiel finden Sie in Listing 10.4. Sie sehen in Zeile 8, dass nur ein Dateiname, allerdings kein Dateipfad definiert ist. Standardmäßig werden diese Ausgabedateien in einem Unterordner Ihres BOINC-Projekts erstellt, welches durch `<log_dir>...</log_dir>` definiert wird; ebenfalls in der `config.xml`.

Listing 10.4 Konfiguration der Ausgabedateien für Protokollierungen

```
<boinc>
  <config>
    ...
  </config>
  <tasks>
    <task>
      <cmd> db_output -d 2 -dump_spec ../db_dump_spec.xml </cmd>
      <output> db_dump.out </output>
    </task>
  </tasks>
</boinc>
```

10.2.1 Protokollierungsfunktionen

Listing 10.5 zeigt einen Auszug der Klassendefinitionen, die beim Erstellen von Protokollierungen hilfreich sind; mehr Funktionsumfang ist in der Regel nicht nötig.

Listing 10.5 Klassendefinitionen für Protokollierungsaufgaben

```
#include <lib/msg_log.h>

class MSG_LOG {
public:
  MSG_LOG(FILE* output);
  void printf(int kind, const char* format, ...);
  void printf_multiline(int kind, const char* str,
                        const char* prefix_format, ...);
  void printf_file(int kind, const char* filename,
                   const char* prefix_format, ...);
  ...
};

#include <sched/sched_msg.h>

class SCHED_MSG_LOG : public MSG_LOG {
public:
  enum { MSG_CRITICAL=1, MSG_NORMAL, MSG_DEBUG };
  SCHED_MSG_LOG(): MSG_LOG(stderr) { debug_level = MSG_NORMAL; }
  void set_debug_level(int new_level) { debug_level = new_level; }
  void set_indent_level(const int new_indent_level);
  ...
};
```

Sie können direkt Ausgaben in einer Protokolldatei erstellen, da es ein globales Klassenobjekt gibt; es heißt `log_messages`. Dazu mehr im nächsten Abschnitt mit Beispielaufrufen der Verwendung dieses Klassenobjekts.

10.2.2 Format der Protokollierungsdateien

Das Format kann durch Sie festgelegt werden, intern wird für alle Protokollierungen entweder die Klasse MSG_LOG oder die spezialisierte Klasse SCHED_MSG_LOG genutzt. Drei unterschiedliche Protokollierungsformate werden in der aktuellen Version von BOINC unterstützt: *MSG_CRITICAL*, *MSG_NORMAL* und *MSG_DEBUG*. Je nach ausgewähltem Typ ändert sich das Ausgabeformat, wobei je nach Typ ein Prefix an die Ausgabetexte angehängt wird. Ein einfaches Beispiel finden Sie im Listing 10.6, mit den entsprechenden Ausgaben in den Protokollierungsdateien für unseren im Kap. 5.2.2 installierten Testserver. Die Beispiele stammen direkt aus den BOINC-Quellen, die entsprechenden Namen der Quelldateien sind mit angegeben.

Listing 10.6 Beispiele für Protokollausgaben

```
//
// in sched/feeder.cpp
//
log_messages.printf(MSG_DEBUG,
  "result [RESULT#%d] already in array\n", wi.res_id
);
log_messages.printf(MSG_DEBUG, "Added %d results to array\n", nadditions);
log_messages.printf(MSG_DEBUG,
  "%d results already in array\n", ncollisions
);
log_messages.printf(MSG_DEBUG,
  "No action; sleeping %.2f sec\n", sleep_interval
);

/* Ausgabe: feeder.log
 * ------------------
 * 2011-10-14 22:25:06.8171 [debug] result [RESULT#388] already in array
 * 2011-10-14 22:25:06.8171 [debug] Added 0 results to array
 * 2011-10-14 22:25:06.8171 [debug] 19 results already in array
 * 2011-10-14 22:25:06.8171 [debug] No action; sleeping 5.00 sec
 */

//
// in sched/sched_main.cpp
//
log_messages.printf(MSG_CRITICAL,
  "Can't attach shmem: %d (feeder not running?)\n",
  retval
);
log_messages.printf(MSG_CRITICAL,
  "uid %d euid %d gid %d eguid%d\n",
  getuid(), geteuid(), getgid(), getegid()
);

/* Ausgabe: scheduler.log
 * ----------------------
 * 2011-10-13 18:06:29.5149 [PID=2528 ] [CRITICAL] Can't attach shmem: -144 (
      feeder not running?)
```

```
 * 2011-10-13 18:06:29.5153 [PID=2528 ] [CRITICAL] uid 33 euid 33 gid 33
      eguid33
 */

//
// in sched/transitioner.cpp
//
log_messages.printf(MSG_NORMAL, "Starting\n");

/* Ausgabe: transitioner.log
 * ------------------------
 * 2011-10-12 10:48:37.4870  Starting
 */
```

Teil IV
Praxis

Kapitel 11
Kreiszahl@home: Monte-Carlo-Algorithmus für die Kreiszahl π

„Any fool can write code that a computer can understand.
Good programmers write code that humans can understand."

Martin Fowler

Dieses Kapitel beschreibt eine wissenschaftliche Applikation zur Annäherung der Kreiszahl π. Diese wird durch eine statistische Berechnung ermittelt, indem ein Monte-Carlo-Algorithmus verwendet wird und wobei die Iterationen für die einzelnen Durchgänge innerhalb des Monte-Carlo-Algorithmus variiert werden. Sie erfahren, wie so eine Applikation implementiert wird, wie Sie die spezifischen Konfigurationen für ein BOINC-Projekt vornehmen und auf welche Aspekte Sie achten müssen. Weiterhin werden die Berechnungsergebnisse für eine spätere Verarbeitung in einer Datenbank gespeichert und durch ein Skript visuell dargestellt.

11.1 Wissenschaftliche Applikation

Bei der Berechnung der Kreiszahl mit Hilfe eines Monte-Carlo-Algorithmus geht es um die nichtlineare Annäherung an die Kreiszahl $\pi \approx 3{,}141592654$. Für diesen Zweck wird ein trivialer Monte-Carlo-Algorithmus angewendet, der es ermöglicht, eine statistische Annäherung an π zu berechnen [183, Kreiszahl]. Dabei werden die Berechnungen oft bis sehr häufig ausgeführt und zusammengeführt. Nach dem *Gesetz der großen Zahlen* sollen die Ergebnisse der Berechnungen ein stabileres Endergebnis mit einer besseren Konvergenz erstellen [183, Gesetz der großen Zahlen]. In diesem Algorithmus werden je Iteration zwei Zufallszahlen zwischen 0–1 erzeugt und geprüft, ob diese in einem Viertelkreis mit dem Radius Eins liegen oder nicht. Abbildung 11.1 zeigt ein beispielhaftes Ergebnis. Die Punkte links vom Kreis werden als innere Punkte angenommen und die Punkte, die rechts vom Kreis liegen, werden als außerhalb des Kreises angenommen. Sollte ein Punkt direkt auf der Kreisbahn liegen, so wird dieser als innerer Punkt gewertet. Der Kreis bildet hierbei einen Einheitskreis und kann durch den *Satz des Pythagoras* errechnet werden. Für diesen Fall müssen nur die zwei Zufallszahlen für die Abszisse (Wert der x-Achse) und die Ordinate (Wert der y-Achse) mit einbezogen werden. Die Hypotenuse – hier ist das der Radius – muss bei der Berechnung Eins ergeben. Gleichung 11.1 zeigt den trivialen mathematischen Ausdruck für die Berechnung eines Einheitskreises.

C. B. Ries, *BOINC*, Xpert.press,
DOI 10.1007/978-3-642-23382-8_11, © Springer-Verlag Berlin Heidelberg 2012

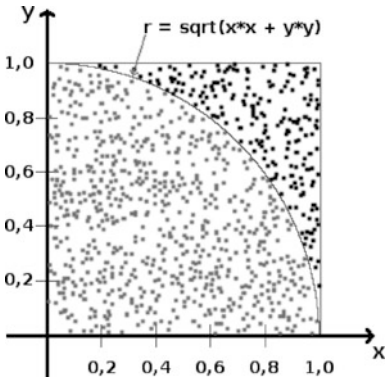


Abb. 11.1 Viertelkreis mit dem Radius Eins, durch den die Kreiszahl π mit Hilfe eines Monte-Carlo-Algorithmus angenähert wird

Daraus kann eine Überprüfung, ob ein Punkt beschrieben aus Wert für die x-Achse und y-Achse innerhalb oder außerhalb des Einheitskreises liegt, in einem Algorithmus beschrieben werden [183, Satz des Pythagoras].

$$r = \sqrt{x^2 + y^2} \tag{11.1}$$

Die Prüfung des Radius auf kleiner, gleich Eins oder größer als Eins gibt uns die Möglichkeit zu sehen, ob die zwei Werte für die Achsen einen Punkt innerhalb oder außerhalb des Viertelkreises beschreiben. Je nach Beschreibung inkrementieren wir den Zähler für innere Punkte oder wir gehen zur nächsten Iteration über. Da diese Zählung bei einem Versuch mit einem Viertelkreis natürlich nur ein Viertel des Endergebnisses liefert, muss dieser Wert am Ende noch mit vier multipliziert werden. Listing 11.1 enthält eine Beispiellösung für 100.000 Berechnungsiterationen.

Listing 11.1 Beispiellösung für zehn Durchgänge mit jeweils **100.000** Iterationen

```
Durchgang: 001 = 3.1401
Durchgang: 002 = 3.1432
Durchgang: 003 = 3.1422
Durchgang: 004 = 3.1398
Durchgang: 005 = 3.1366
Durchgang: 006 = 3.1368
Durchgang: 007 = 3.1373
Durchgang: 008 = 3.1371
Durchgang: 009 = 3.1437
Durchgang: 010 = 3.1360
```

Die Ergebnisse kommen schon sehr gut an den technisch relevanten Wert von π heran und wir sehen, dass das Prinzip der großen Zahlen ein guter Ansatz für die Bestimmung von π ist und in dieser Problemstellung angewendet werden kann. Es sollte nebenbei angemerkt sein, dass es heutzutage natürlich nicht wirklich von Relevanz ist, den Wert π in solch einem Verfahren zu errechnen.

11.1.1 Aufbau und Programmlogik

Eine der Hauptkomponenten dieses Kapitels ist in Listing 11.2 aufgeführt. Es enthält den Quelltext für die wissenschaftliche Applikation, um eine statistische

Berechnung der Zahl π mit Hilfe von BOINC durchführen zu können. Die einzelnen Bereiche der Implementierung werden zwischendurch erläutert. Für diesen Zweck ist das Listing in einzelne Bereiche unterteilt, die jeweils direkt nach einem Bereich eine Erläuterung besitzen.

Listing 11.2 Die wissenschaftliche Applikation für die statistische Berechnung der spezifischen Kreiszahl $\pi \approx 3{,}141592654$

```
// C++
#include <iostream>
#include <string>

// C
#include <stdio.h>
#include <errno.h>
#include <math.h>

// BOINC
#include <boinc_api.h>
#include <filesys.h>
#include <miofile.h>
#include <parse.h>
#include <mfile.h>
#include <util.h>
#include <error_numbers.h>
```

Dieser Bereich inkludiert zwei Standard C++-Header-Dateien. Mit diesen Dateien wird die einfache Ein-/Ausgabe von Informationen innerhalb eines Terminals ermöglicht, und weiterhin wird die C++-Standardklasse `std::string` für das Arbeiten mit Zeichenketten eingebunden. In den nachfolgenden Zeilen werden drei Standard C-Header-Dateien eingebunden. Diese binden Funktionen für die Ein-/Ausgabe, für das Arbeiten mit Fehlermeldungen und mathematische Funktionen ein. Die weiteren Inkludierungen betreffen das BOINC-System. Diese sieben Header-Dateien werden in Kap. 7 behandelt und mit Anwendungsbeispielen erläutert.

```
#define CHECKPOINT_FILE       "checkpoint.xml"
#define CONFIGURATION_FILE    "configuration.xml"
#define RESULT_FILE           "result.xml"
```

Diese Zeilen definieren die Dateinamen, welche später für die Eingabe-, Ergebnisdateien und die Checkpoint-Datei verwendet werden. Der Dateiname einer Checkpoint-Datei, beschrieben durch `CHECKPOINT_FILE`, kann zu einem selbst gewählten Dateinamen geändert werden, ohne dass dies großen Einfluss auf die Laufzeit einer wissenschaftlichen Applikation hätte. Denn eine Checkpoint-Datei wird lokal erzeugt und verbleibt auch auf der lokalen Maschine. Wenn diese Datei fehlen sollte, so wird die Ausführung einer wissenschaftlichen Applikation einfach von Beginn an durchgeführt, ansonsten wird die Berechnung an der Stelle weitergeführt, welche durch die Checkpoint-Datei beschrieben wird. Ein wenig komplizierter sieht es bei den zwei weiteren Dateinamen aus, die durch `CONFIGURATION_FILE` und `RESULT_FILE` beschrieben werden. Jede Änderung einer dieser Dateinamen hat zur Folge, das die Eingabeschablonen (engl. *input template*) und Ergebnisschablonen (engl. *result template*) aus Abschn. 11.2.2 angepasst werden müssen. Sind keine Änderungen in den Schablonen erfolgt, so hätte dies Auswirkungen auf die Ausführung der wissenschaftlichen Applikation. Die wissenschaftliche Applikation könnte in diesem Fall entweder keine Eingabedaten für die Bearbeitung einlesen oder die

Ergebnisse werden in einer Datei gespeichert, die allerdings niemals beim BOINC-Projekt eingehen würde. Dadurch wäre eine effektive und erfolgversprechende Ausführung unmöglich und darüber hinaus auch noch völlig sinnlos.

```
/**
 * @brief Describes configuration values for the runtime.
 */
typedef struct CONFIGURATION_XML {
  int iterations;
  int runs;

  inline void init() {
    this->iterations = 100;
    this->runs = 1;
  }

  /**
   * @param lmBoincXMLFiles Pointer to the open xml file.
   * @return -1 if something went wrong, otherwise 0.
   */
  int parse(FILE *configurationXMLFile) {
    init();

    MIOFILE configurationMIOFile;
    configurationMIOFile.init_file(configurationXMLFile);
    XML_PARSER p(&configurationMIOFile);

    if(!p.parse_start("configuration")) {
      fprintf(stderr, CONFIGURATION_FILE" is not"
                      " formatted correctly\n");
      return ERR_XML_PARSE;
    }

    bool is_tag = false;
    char tag[128] = {'\0'};
    while (!p.get(tag, sizeof(tag), is_tag)) {
      if (!is_tag) {
        fprintf(stderr, "unexpected text in "
                        CONFIGURATION_FILE": %s\n", tag);
        continue;
      }
      if (!strcmp(tag, "/configuration")) return 0;
      if (p.parse_int(tag, "iterations", this->iterations)) continue;
      if (p.parse_int(tag, "runs", this->runs)) continue;
      // add more...
      p.skip_unexpected(tag, true, "configuration");
    }
    return 0;
  }
} CONFIGURATION_XML;
```

In die Struktur `CONFIGURATION_XML` werden vorhandene Informationen aus der Eingabeparameterdatei `CONFIGURATION_FILE` eingelesen und für die spätere Verwendung vorgehalten. Die Memberfunktion `init` setzt die zwei globalen Variablen `iterations` und `run` auf Standardwerte. `parse` erwartet einen Zeiger auf eine geöffnete Konfigurationsdatei. Die `while`-Schleife liest jeden XML-Tag einzeln ein und prüft ihn auf vorhandene Laufzeitparameter. Diese filtert sie gegebenenfalls aus den XML-Tags und ordnet sie entsprechend den Variablen zu.

```
/**
 * @param configuration
 * @return FALSE if something went wrong, otherwise TRUE.
 */
bool initConfigurationXML(ConfigurationXML &configuration)
```

```
{
 std::string xmlFilePath;
 boinc_resolve_filename_s(CONFIGURATION_FILE, xmlFilePath);

 configuration.init();

 FILE *xmlFile = boinc_fopen(xmlFilePath.c_str(), "r");
 if(xmlFile == NULL) {
   return false;
 }
 return (configuration.parse(xmlFile)?false:true);
}
```

Dieser Bereich öffnet die Konfigurationsdatei `CONFIGURATION_FILE`. Mit Hilfe der BOINC-API-Funktion `boinc_resolve_filename_s` wird im ersten Schritt geprüft, ob es sich um einen Soft-Link handelt (vgl. Abschn. 7.3.5), und mit `boinc_fopen` wird die Datei geöffnet. Bei erfolgreicher Öffnung wird die Konfigurationsdatei in Einzelschritten ausgelesen und die Laufzeitparameter aus den XML-Informationen gefiltert.

```
/**
 * @param argc
 * @param argv
 * @return
 */
int main(int argc, char **argv) {

 BOINC_OPTIONS options;
 options.main_program = 1;
 options.send_status_msgs = 1;
 options.handle_process_control = 1;
 options.direct_process_action = 1;

 // Das BOINC-Framework fuer die Ausfuehrung
 // dieser Applikation initialisieren.
 int returnValue = boinc_init_options(&options);
 if (returnValue) {
  char buf[24];
  std::cerr << boinc_msg_prefix(buf)
            << " boinc_init returned "
            << returnValue << std::endl;
  exit(returnValue);
 }
```

Die oben getroffenen Optionseinstellungen sind von vornherein vom BOINC-Framework eingestellt, aber für die persönliche Vollständigkeit selbst definiert und mit angegeben. Daraufhin wird das BOINC-Framework initialisiert, wobei die zuvor gewählten Standardoptionen gesetzt und für die Laufzeit der Applikation übernommen werden. Wenn die Initialisierung misslingt, so wird die Ausführung der wissenschaftlichen Applikation beendet.

```
 CONFIGURATION_XML configuration;
 if(!initConfigurationXML(configuration)) {
  std::cout << "Could not read "CONFIGURATION_FILE"." << std::endl;
  boinc_finish(1);
 }
```

Die oben beschriebene Struktur für das Vorhalten der Laufzeitparameter wird hier erstellt und gefüllt. Sollte das Einlesen nicht gelingen, so wird die Ausführung beendet und der BOINC-Client bekommt mit `boinc_finish(1)` diesen Fehler gemeldet, so dass das BOINC-Projekt eine Meldung über das Fehlverhalten erhalten

kann. Daraufhin entscheidet das BOINC-Projekt wie mit dem Arbeitspaket umzugehen ist, welches bei einer solchen fehlerhaften Ausführung nicht bearbeitet werden konnte. Zum Beispiel könnte das Arbeitspaket erneut dem Teilnehmer mit der fehlerhaften Ausführung zugewiesen oder an weitere Teilnehmer im BOINC-Projekt weitergegeben werden.

```
srand(time(NULL));

int run = 0; // Set to checkpoint value if available!
// Read checkpoint file.
std::string checkpointPath;
if(!boinc_resolve_filename_s(CHECKPOINT_FILE, checkpointPath)) {
 FILE* chk = boinc_fopen(CHECKPOINT_FILE, "r");
 if(chk == NULL) {
 } else {
   char line[1024] = {'\0'};
   if(fgets(line, 1024, chk)) {
     parse_int(line, "<run>", run);
   }
   fclose(chk);
 }
}
```

Zu Beginn der Ausführung wird der Zufallszahlengenerator mit der Anzahl an Sekunden seit der UNIX-Zeitrechnung initialisiert [179]. Daraufhin wird geprüft, ob die wissenschaftliche Applikation schon einmal im Vorfeld ausgeführt wurde. Dies wird durch das Einlesen einer eventuell vorhandenen Checkpoint-Datei realisiert, indem der XML-Tag zur Beschreibung des letzten Durchlaufs geprüft wird. Der XML-Tag besitzt das Format `<run>%d</run>` und durch `%d` wird der Durchlauf beschrieben, der vor der letzten Erstellung der Checkpoint-Datei durchgeführt wurde. Es sei nochmals darauf hingewiesen, dass dieser Wert nicht den letzten Wert der Anzahl an Iterationen des Monte-Carlo-Algorithmus beschreibt, sondern die Anzahl der bis dahin durchgeführten Durchführungen des Monte-Carlo-Algorithmus. Wenn keine Checkpoint-Datei existiert, so wurde die wissenschaftliche Applikation noch nicht im Vorfeld gestartet oder das Intervall für die Erstellung einer Checkpoint-Datei war bis dahin noch nicht überschritten. Bei einer nicht vorhandenen Checkpoint-Datei wird die wissenschaftliche Applikation immer von Null anfangen, sprich eine komplette Monte-Carlo-Simulation durchführen.

```
// Calculation of the Kreiszahl
// @see http://de.wikipedia.org/wiki/Kreiszahl#BBP-Reihen
for( ; run < configuration.runs; run++) {
 double pi = 0.0;
 int inBound = 0;
 int iteration = 0;
 for( ; iteration < configuration.iterations; iteration++) {
   double dotx = rand()/(double)RAND_MAX;
   double doty = rand()/(double)RAND_MAX;
   double radius = sqrt(dotx*dotx + doty*doty);

   if(radius <= 1.0) {
     inBound++;
   } else {
   }

   double fd = (double) iteration /
               (double) configuration.iterations / 100.0;
```

```
    boinc_fraction_done(fd);
    boinc_sleep(0.5);
  }
  pi = 4.0*(double)inBound/(double)iteration;
```

Dieser Bereich beschreibt die zu Grunde liegende statistische Berechnung der Kreiszahl π für den Monte-Carlo-Algorithmus. Hier werden in der `for`-Schleife die einzelnen Durchläufe der Berechnungen mit dem Monte-Carlo-Algorithmus gestartet und durchgeführt. Mit Hilfe der Funktion `boinc_fraction(fd)` senden wir den aktuellen Berechnungsstand an den BOINC-Client, der dann wiederum vom BOINC-Manager angezeigt werden kann. Bei diesem Wert handelt es sich – von der Idee her – um einen prozentualen Wert, welcher allerdings in die Grenzen 0 und 1 skaliert ist. Eine Multiplikation mit 100 würde aus diesem Wert einen wirklichen Prozentwert machen. Dies kann in Abschn. 7.3.9 nachgelesen werden. Die Variable `fd` enthält dabei diesen skalierten Wert, welcher aus dem Wert der aktuellen Iteration und der durchzuführenden Iterationen innerhalb des Monte-Carlo-Algorithmus errechnet wird.

```
150 std::string resultPath;
    if(!boinc_resolve_filename_s(RESULT_FILE, resultPath)) {
     FILE* fresult = boinc_fopen(resultPath.c_str(), "a");
     fprintf(fresult, "<pi run=\"%d\" iterations=\"%d\">%.10f</pi>\n",
                      run, configuration.iterations, pi);
155  fclose(fresult);
     fprintf(stderr, "Iteration: %03d = %0.4f\n", (run+1), pi);
    } else {
     std::cerr << "Could not find "RESULT_FILE"." << std::endl;
     boinc_finish(1);
160 }
```

Das Ergebnis des Monte-Carlo-Algorithmus wird in der Datei `RESULT_FILE` gespeichert. Dieser Bereich zur Abspeicherung und der nachfolgende Bereich zur Erstellung einer Checkpoint-Datei können zu Ergebnisproblemen führen. Eine Ausführung der wissenschaftlichen Applikation kann zu jedem Zeitpunkt unterbrochen werden. Es könnte dann passieren, dass ein Ergebnis gespeichert wurde, aber der Wert der letzten Durchführung keine Angaben dazu macht, so dass mehrere Ergebnisse eines Durchlaufs mit derselben Identifikation abgespeichert werden. Hier wäre es besser, einen C++-Container oder ein Array zu haben, in dem alle Werte seit der letzten Checkpoint-Erstellung zwischengespeichert sind, so dass diese Werte in einem Rutsch – zur selben Zeit wie die Checkpoint-Erstellung – abgespeichert werden. Dies verhindert das Abspeichern doppelter Ergebnisse und sorgt dafür, dass wirklich nur die Anzahl an Ergebnissen abgespeichert werden, die auch gewünscht sind. Es ist sinnvoll, den Prozess für die Ergebnisabspeicherung zum selben Zeitpunkt durchzuführen, in dem auch eine Checkpoint-Datei erstellt wird, weil diese Abspeicherung durch die Funktion `boinc_time_to_checkpoint()` in einem atomaren Bereich ausgeführt wird. Durch die Verwendung eines atomaren Bereichs wird die wissenschaftliche Applikation erst beendet, wenn dieser Bereich wieder verlassen wurde. Im nachfolgenden Abschn. 11.1.2 wird dieser Missstand korrigiert. Diese Problematik wurde aus Gründen der Sensibilisierung aufgeführt, so dass Sie ein Gespür für solche Problematiken entwickeln und diese im besten Fall lösen können.

```
 // Do checkpointing!
 if(boinc_time_to_checkpoint()) {
  std::string checkpointPath;
  if(!boinc_resolve_filename_s(CHECKPOINT_FILE, checkpointPath)) {
    FILE* chk = boinc_fopen(checkpointPath.c_str(), "w");
    if(chk == NULL) {
    } else {
      fprintf(chk, "<run>%d</run>\n", run);
      fclose(chk);
    }
  }
  boinc_checkpoint_completed();
 }

} // run++
/**
 * End of your work!
 */
```

In diesem Fall wird nicht die Checkpoint-Datei ausgelesen, sondern eine erstellt oder überschrieben. Wie oben schon erläutert, beschreibt das zu schreibende XML-Tag `<run>%d</run>`, welcher Durchlauf der Berechnung mit dem Monte-Carlo-Algorithmus bis zu diesem Zeitpunkt beendet wurde. Eine Abspeicherung findet nur statt, wenn das Grenzintervall für das Erstellen einer Checkpoint-Datei überschritten ist. Die Hintergründe dieser Technik und dieser Prüfmethode werden in Abschn. 7.3.11 behandelt.

```
 boinc_fraction_done(1);
 boinc_finish(0);
}

#ifdef _WIN32
int WINAPI WinMain(HINSTANCE hInst,
                   HINSTANCE hPrevInst,
                   LPSTR Args,
                   int WinMode) {
  LPSTR command_line;
  char* argv[100];
  int argc;

  command_line = GetCommandLine();
  argc = parse_command_line( command_line, argv );
  return main(argc, argv);
}
#endif
```

Der Bereich mit den Preprozessor-Direktiven soll eine Kompilierung der wissenschaftlichen Applikation unter dem Betriebssystem Microsoft Windows ermöglichen. Unter Windows sieht die Hauptfunktion zum Starten einer Anwendung ein wenig anders aus. Durch diese Preprozessor-Direktiven ist dieser Bereich für Compiler sichtbar oder unsichtbar.

11.1.2 Verbesserungen erreichen

Wie in Abschn. 11.1 erläutert, kann es vorkommen, dass zu viele Rechenergebnisse in der Ergebnisdatei `RESULT_FILE` abgespeichert werden, da nach der Erstellung eines Ergebnisses eine sofortige Abspeicherung erfolgt. Eine Checkpoint-Datei

wird dagegen erst in einem bestimmten Intervall erstellt und nicht bei jedem Durchlauf des Monte-Carlo-Algorithmus. Weiterhin wird eine vorhandene Checkpoint-Datei bei jedem Schreiben komplett überschrieben. Dadurch sind aus Prinzip schon keine doppelten Eintragungen innerhalb der Checkpoint-Datei möglich.

Doch wie handhaben wir nun das Problem der möglichen doppelten Eintragungen der Ergebnisse? Denn diese werden zu jedem Zeitpunkt abgespeichert und die Ausführung der wissenschaftlichen Applikation kann zu jedem Zeitpunkt abgebrochen werden. Egal in welcher Reihenfolge wir den Bereich für die Checkpoint-Erstellung und der Ergebnisspeicherung schreiben, in beiden Fällen kann es zu Problemen kommen:

- Im ersten Szenario wird ein neuer Wert in die Checkpoint-Datei geschrieben und daraufhin sofort ein Abbruch der Ausführung der wissenschaftlichen Applikation gefordert. In diesem Fall wäre das letzte Ergebnis der Berechnung nicht abgespeichert. Das könnte nun nach jeder Berechnung vorkommen, und wenn die Berechnung länger dauern würde, als das Intervall für die Checkpoint-Erstellung ist, so würden niemals Berechnungsergebnisse abgespeichert werden. Das Arbeitspaket wäre irgendwann fertig markiert, ohne ein Ergebnis zu besitzen.
- Im zweiten Szenario würden erst die Ergebnisse abgespeichert und daraufhin die Checkpoint-Datei erstellt werden. Dies könnte dazu führen, dass die Berechnungen unendlich lange dauern und niemals fertiggestellt würden. Zum Beispiel ist dies der Fall, wenn die wissenschaftliche Applikation abgebrochen und später weitergeführt wird und vorher niemals eine Checkpoint-Datei erstellt wird, um nach der letzten Berechnung einzusteigen.

Die Überlegung an dieser Stelle ist, dass die Ergebnisse eines Durchlaufs des Monte-Carlo-Algorithmus in einem Array oder Vektor-Container der Standardklasse `std::vector` abgespeichert werden. In diesem Fall könnten alle Ergebnisse auf einmal innerhalb der Checkpoint-Erstellung abgespeichert werden, so dass die wissenschaftliche Applikation korrekt funktioniert. Die möglichen Modifikationen finden Sie in Listing 11.3. Nach den jeweiligen Bereichen wird angegeben, an welchen Stellen diese Änderungen in Listing 11.2 vorgenommen werden müssen.

Listing 11.3 Verbesserte wissenschaftliche Applikation für die statistische Berechnung der Kreiszahl π

```
// C++
#include <iostream>
#include <string>
#include <vector>
```

Dieser Teil ersetzt den Bereich um Zeile 3 aus Listing 11.2.

```
typedef struct SimulationData {
  int run;
  double pi;
} SimulationData;

std::vector<SimulationData> results;

bool storeResults(std::vector<SimulationData>& results) {
```

```
  boinc_begin_critical_section();
  std::vector<SimulationData>::iterator dit = results.begin();
  for( ; dit != results.end(); dit++) {
    std::string resultPath;
    if(!boinc_resolve_filename_s(RESULT_FILE, resultPath)) {
      FILE* fresult = boinc_fopen(resultPath.c_str(), "a");
      fprintf(fresult, "<pi run=\"%d\" iterations=\"%d\">%.10f</pi>\n",
                       dit->run, dit->iterations, dit->pi);
      fclose(fresult);
      fprintf(stderr, "Iteration: %03d = %0.4f\n",
                      (dit->run+1), dit->pi);
    } else {
      std::cerr << "Could not find "RESULT_FILE"."
                << std::endl;
      boinc_end_critical_section();
      return false;
    }
  }
  results.clear();
  boinc_end_critical_section();
  return true;
}
```

Der obere Bereich wird als Ganzes direkt nach der Zeile 83 in Listing 11.2 eingefügt.

```
// Calculation of the Kreiszahl
// @see http://de.wikipedia.org/wiki/Kreiszahl#BBP-Reihen
for( ; run < configuration.runs; run++) {
  ...
  ...
pi = 4.0*(double)inBound/(double)iteration;

// Enhance
SimulationData data = {
  run, configuration.iterations, pi
};
results.push_back(data);
```

Die Zeilen 42 bis 46 aus Listing 11.3 werden nach Zeile 149 in Listing 11.2 eingefügt.

```
// Do checkpointing!
bool errorHappend = false;
if(boinc_time_to_checkpoint()) {
  std::string checkpointPath;
  if(!boinc_resolve_filename_s(CHECKPOINT_FILE, checkpointPath)) {
    FILE* chk = boinc_fopen(checkpointPath.c_str(), "w");
    if(chk == NULL) {
    } else {
      fprintf(chk, "<run>%d</run>\n", run);
      fclose(chk);

      // Enhance, store results:
      errorHappend = !storeResults(results);
    }
  }
  boinc_checkpoint_completed();
  if(errorHappend) {
    boinc_finish(1);
  }
 }

} // run++
```

```
70  // Enhance
    if(!storeResults(results)) {
      fprintf(stderr, "Could not store results.\n");
      boinc_finish(1);
    }
```

Listing 11.3, Zeile 59 ersetzt in Listing 11.2 den Bereich um die Zeilen 150 bis 173. Weiterhin werden die Zeilen 71 bis 74 aus Listing 11.3 vor Zeile 179 aus Listing 11.2 hinzugefügt.

Diese wenigen Änderungen sichern unsere wissenschaftliche Applikation gegen das Problem von mehrfach vorhandenen Berechnungsergebnissen ab. Weiterhin werden alle Rechenergebnisse zwischen zwei Checkpoint-Zeiten auf einmal gesichert. Ein Beenden durch den BOINC-Client ist erst nach der Speicherung der Berechnungsergebnisse möglich und sollte immer auf dem gezeigten Weg oder ähnlichen Wegen durchgeführt werden. So kann man zu jedem Zeitpunkt sicher sein, dass die Abspeicherung korrekt ist. In diesem Fall kann man sich auf die Ergebnisse verlassen.

Allerdings beheben diese Änderungen nicht das Problem der unendlich laufenden Berechnungen, wenn die wissenschaftliche Applikation immer vor der Erstellung einer Checkpoint-Datei beendet wird. Dieses Problem kann durch das Ersetzen der BOINC-Checkpoint-Funktionen durch die atomaren BOINC-Funktionen behoben werden (vgl. Abschn. 7.3.10). Das BOINC-System ist allerdings in jedem Fall für solche Fälle gewappnet, da jedes Arbeitspaket eine Zeitmarke erhält. Diese Zeitmarke gibt einem Arbeitspaket eine Lebenszeit. Wenn diese überschritten wird, so wird das Arbeitspaket als verloren angenommen und ein weiterer Projektteilnehmer erhält das Arbeitspaket.

11.1.3 Eingabeparameter und Ergebnisdatei

Für das einfache Berechnen einer Annäherung an die Kreiszahl π müssen keine großen Anforderungen bezüglich der Eingabeparameter beachtet werden. In der heutigen Zeit besitzen die – im privaten und gewerblichen Gebrauch befindlichen – Computer Rechenleistungen von durchschnittlich 3 GHz oder mehr, so dass eine wissenschaftliche Applikation ruhig einige hunderte Durchläufe des Monte-Carlo-Algorithmus für die statistische Berechnung von π durchführen kann. In Listing 11.4 wird die Konfiguration unserer Anwendung aufgeführt. Es werden zwei Konfigurationsparameter definiert:

runs Beschreibt die Anzahl der Durchführungen einer statistischen Berechnung, die bei der Ausführung der wissenschaftlichen Applikation erfolgen soll. Jedes Einzelergebnis wird für eine spätere Verwendung abgespeichert und im Nachgang zum BOINC-Projektserver zurückgesendet.

iterations Dieser Wert beschreibt die Anzahl an Iterationen, die innerhalb des Monte-Carlo-Algorithmus durchgeführt werden sollen.

Die Struktur `CONFIGURATION_XML` übernimmt mit der Member-Funktion `parse()` das Auslesen dieser Konfigurationsdatei. Im späteren Verlauf können diese Werte zu jedem Zeitpunkt aus der Struktur verwendet werden.

Listing 11.4 Eingabedatei für die statische Berechnung der Kreiszahl π

```
<configuration>
  <runs>10</runs>
  <iterations>100</iterations>
</configuration>
```

11.1.4 Kompilieren

Für das Kompilieren unserer wissenschaftlichen Applikation erstellen wir uns ein komfortables Makefile[1] [132]. Listing 11.5 enthält unser Makefile, speziell für das Kompilieren unserer *Kreiszahl*-Applikation. Zeile 1 definiert den Namen der wissenschaftlichen Applikation, hier `kreiszahl`. Die Quelltext-Dateien bestehen nur aus der Datei `main.cpp`. Für das Kompilieren unserer wissenschaftlichen Applikation benötigen wir die in Kap. 5 heruntergeladenen und kompilierten BOINC-Quellen. Es muss ein symbolischer Link zu diesen Quellen gesetzt werden:

```
ln -sf ../../../../server_stable boincsrc
```

Eventuell müssen Sie diesen Befehl an Ihre Dateisystemhierarchie anpassen. Wir verwenden hier einen relativen Pfad. Das Ergebnis dieser Ausführung sollte bei Ihnen so aussehen wie in Abb. 11.2. In dem Makefile werden zudem einige Unterordner aus `boincsrc/` definiert. Diese enthalten benötigte BOINC-Header-Dateien und mitzulinkende BOINC-Bibliotheken. In diesem Makefile sind drei Targets[2] definiert:

all Das standardmäßig verwendete Target wird immer genutzt, wenn kein weiteres Target angegeben wurde.

run Dieses Target führt die wissenschaftliche Applikation im standalone-Betrieb aus.

clean Räumt den aktuellen Dateiordner auf und entfernt dabei alle irrelevanten Dateien.

[1] Ein Makefile dient dem Übersetzen von Programmfragmenten und nutzt dabei Standardeinstellungen. Weiterhin können Tool-Chains für das Kompilieren von größeren Applikationen definiert werden und dadurch Versionsabhängigkeiten überprüft und darauf hingewiesen werden. Es können einzelne Dateien sowie ganze Projekte durch einen Aufruf von `make` kompiliert werden.

[2] Makefile Targets sind Stellen im Makefile, an denen die Ausführung gestartet werden kann. Diese Targets können beim Aufruf von `make` als erster Parameter mit übergeben werden.

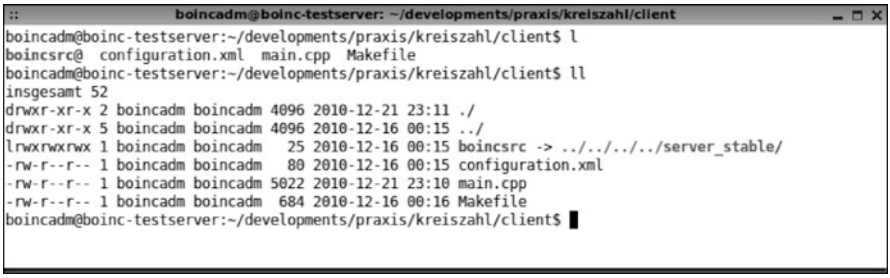


Abb. 11.2 Ansicht aller Dateien der wissenschaftlichen Applikation für die statistische Berechnung der Kreiszahl π. Zu erkennen sind der symbolische Link zu den BOINC-Quellen, eine Beispielkonfiguration für das Testen der Applikation, die Quelldatei und das Makefile zum Kompilieren der Applikation

Listing 11.5 Das Makefile für das Kompilieren der wissenschaftlichen Applikation für die statistische Berechnung der Kreiszahl π. Es werden drei Targets beschrieben: (1) `all` kompiliert die Applikation, (2) `run` führt die Applikation aus und `clean` räumt das Verzeichnis auf und entfernt dabei irrelevante Dateien

```
A=kreiszahl
S+=main.cpp

INCLUDES=-I./boincsrc/api -I./boincsrc/lib
LIBARIES=-L./boincsrc/api \
  -lboinc_api -L./boincsrc/lib -lboinc -pthread

all: ${S}
  g++ -g -O2 -Wall -o ${A} ${S} ${INCLUDES} ${LIBARIES}
  cp -f ${A} ../server/platforms/${A}_0.1_i686-pc-linux-gnu
  cp -f ${A} ../server/platforms/${A}_0.1_x86_64-pc-linux-gnu

run:
  ./${A}

clean:
  rm -rf boinc_finish_called
  rm -rf result.xml
  rm -rf stderr.txt
  rm -rf checkpoint.xml
  rm -rf *~ *.o
  rm -rf ${A}
```

Nach dem Ausführen des Makefile mit dem Befehl `make` haben wir unsere wissenschaftliche Applikation erstellt. Abbildung 11.3 zeigt das Ergebnis des Kompilierungsprozesses. In den ersten Zeilen werden die Befehle zum Kompilieren ausgegeben und im unteren Bereich finden wir unsere erstellte Anwendung `kreiszahl`. Weiterhin ist zu erkennen, dass zwei Kopien der Applikation erstellt wurden, die direkt in den Ordner `../server/platforms` kopiert wurden. Die Kopien erhalten einen neuen Namen, der die Zielplattform für die Ausführung beschreibt. In diesem Fall handelt es sich dabei um Kopien für die Ausführung auf einem Linux-System mit 32-/64-Bit-Unterstützung. Sollte die Kompilierung unter einem 32-Bit-System erfolgen, so ist die Applikation prinzipiell als 32-Bit-Anwendung kompiliert, siehe Abb. 11.4. Die zweite Kopie – für ein 64-Bit-System angedacht – ist diesem Fall auch eine Kopie der 32-Bit-Anwendung. Die zweite Kopie sollte im idealen

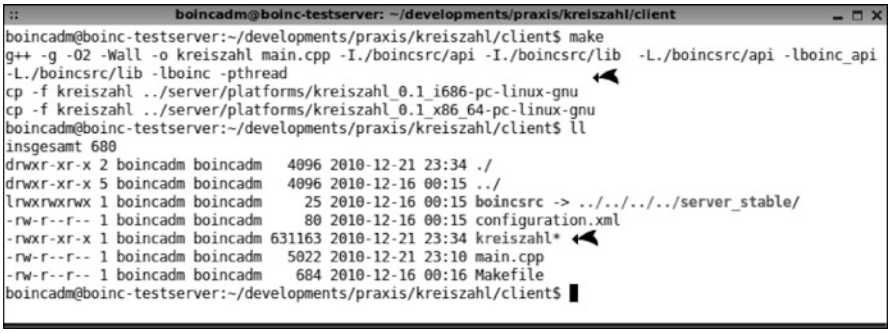


Abb. 11.3 Prozess zum Kompilieren der wissenschaftlichen Applikation für die statistische Berechnung von π. Im *oberen Bereich* wird der Kompilierungsprozess mit Hilfe von `make` gestartet, daraufhin erfolgen Ausgaben, die Informationen über die aktuelle Kompilierung liefern. Das Ergebnis dieses Prozesses ist die Applikation `kreiszahl`

Abb. 11.4 32-Bit-Version der wissenschaftlichen Applikation für die statistische Berechnung der Kreiszahl π

Fall durch eine 64-Bit-Applikation ersetzt werden. Dies kann durch das Kompilieren unter einem 64-Bit-System ermöglicht werden. Prinzipiell ist es möglich, 32-Bit-Anwendungen auf einem 64-Bit-System auszuführen. Für diesen Zweck müssen für Linux alle benötigen 32-Bit-Systembibliotheken installiert sein [115]. Windows unterstützt diese Möglichkeit auch [146]. Mac OS X kann zwar auch 32-Bit-Anwendungen unter einer 64-Bit Installation starten [169], allerdings muss dafür ein Häkchen in den Startereinstellungen für 32-Bit-Unterstützung angeklickt werden. Dadurch kann es in Verbindung mit dem BOINC-Client und BOINC-Manager zu Problemen kommen. In diesem Fall sollte eine richtige 64-Bit-Version für Macintosh OS X erstellt werden.

11.1.5 Eine erste Ausführung

Die wissenschaftliche Applikation `kreiszahl` kann nun ausgeführt werden. Im ersten Durchlauf starten wir die Anwendung im standalone-Modus, schauen uns das Verhalten und das Ergebnis der Ausführung an. Eine Ausführung wird mit

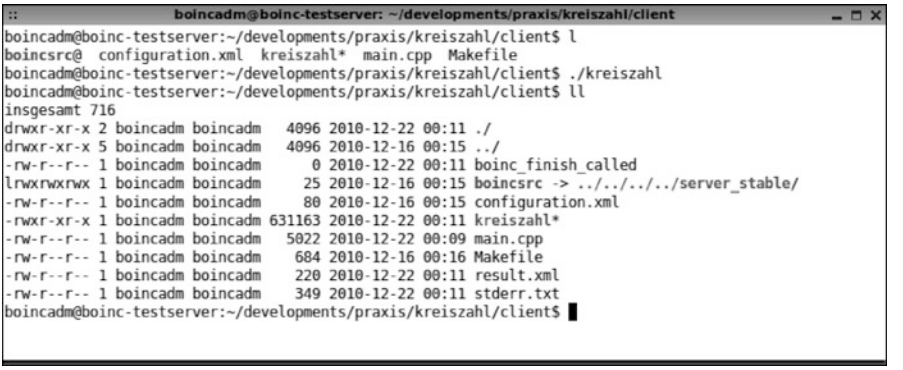


Abb. 11.5 Nach der Ausführung der wissenschaftlichen Applikation für die statistische Berechnung der Kreiszahl π

```
boincadm@boinc-testserver:~/developments/praxis/kreiszahl/client$ \
    ./kreiszahl
```

gestartet. Nach der Ausführung sind drei neue Dateien im Verzeichnis. Abbildung 11.5 zeigt diese Dateien. Die Dateien haben folgenden Zweck und Bedeutung:

boinc_finish_called Diese Datei beschreibt das Ende der Ausführung einer wissenschaftlichen Applikation in Verbindung mit BOINC. Der BOINC-Client prüft kontinuierlich auf diese Datei. Wenn die Applikation die Funktion `boinc_finish()` oder `boinc_exit()` aufruft, wird die Datei erstellt und der BOINC-Client erhält dadurch das Zeichen, dass die Ausführung beendet wurde. Abschnitt 7.3.3 liefert eine detailliertere Beschreibung.

result.xml Diese Datei enthält die Berechnungsergebnisse für die Ausführung aller Durchläufe des Monte-Carlo-Algorithmus. Das längere XML-Tag `<pi run="%d" iterations="%d">%0.10f</pi>` dient dem Beschreiben von Berechnungsergebnissen und wird für das Abspeichern jedes einzelnen Ergebnisses verwendet. Dieser XML-Tag wird beim Erstellen mit passenden Dezimalzahlen für den Durchlauf, die Anzahl der ausgeführten Iterationen und dem Fließkommazahl-Ergebnis gefüllt.

stderr.txt Diese Datei enthält Fehler- und Statusmeldungen, die während der Ausführung der wissenschaftlichen Applikation als Feedback ausgegeben werden. Mit Hilfe dieser Datei kann ein BOINC-Projektadministrator Fehlern auf die Schliche kommen, denn diese Datei wird mit dem Senden eines Ergebnisses auch an ein BOINC-Projekt zurückgesendet.

11.2 Projekteinstellungen

Wie müssen wir unsere wissenschaftliche Applikation zu unserem BOINC-Projekt hinzufügen und welche Einstellungen müssen oder sollten vorgenommen werden?

Diese Problematiken werden in diesem Abschnitt behandelt und Sie werden in die Lage versetzt, die Konfiguration Ihren Bedürfnissen entsprechend anzupassen. In den nachfolgenden Abschnitten erhalten Sie Einblicke in generelle Konfigurationsparameter und in einige eventuell für diese wissenschaftliche Applikation wichtige Einstellungsmöglichkeiten. Vieles kann sehr individuell konfiguriert werden, allerdings besteht darin auch die Krux. Denn durch viele Möglichkeiten kann auch vieles falsch eingestellt werden. **Bevor ein BOINC-Projekt öffentlich gemacht wird, sollten Sie genau prüfen, ob Ihr Projekt auch wirklich so arbeitet, wie Sie es wünschen!**

11.2.1 Konfiguration und Anwendung hinzufügen

Was haben wir bisher? Eine wissenschaftliche Applikation für die Berechnung der Kreiszahl π und eine Konfigurationsdatei, durch die die Anzahl der Berechnungen eines Monte-Carlo-Algorithmus und die Anzahl der Iterationen innerhalb dieses Algorithmus definiert werden können. Bis zu diesem Punkt weiß unser BOINC-Projekt aus Kap. 5 noch nichts davon. In diesem Abschnitt gehen wir davon aus, dass Sie das BOINC-Projekt von der Testinstallation aus Kap. 5 bereinigen. Sie können aber auch ein neues BOINC-Projekt anlegen. Wir gehen nachfolgend davon aus, dass Sie ein leeres Projekt haben, es sind keine wissenschaftlichen Applikationen definiert oder dem Projekt hinzugefügt, die Datenbank wurde geleert und in den Projektunterordnern `download/` und `upload/` sind alle Dateien gelöscht.

Im ersten Schritt muss die Konfigurationsdatei `config.xml` an unsere Bedürfnisse angepasst werden. Die Konfiguration ähnelt sehr der Konfiguration des in Abschn. 5.6.2.5 beschriebenen BOINC-Testprojekts *Test@home*. Es können natürlich individuelle Änderungen vorgenommen werden, allerdings ist es für dieses Praxisprojekt nur erforderlich, den Teil für die Dämonen zu modifizieren. Als erste Änderung muss der Name der Applikation geändert werden, dem Parameter `-app` muss nun der Name der hier beschriebenen wissenschaftlichen Applikation `kreiszahl` angehängt werde. Weiterhin greifen wir Abschn. 11.2.5 vor und stellen einen anderen Assimilierer ein. Abbildung 11.6 zeigt die Änderungen für das aktuelle Projekt.

11.2.2 Eingabe-/Ergebnisschablonen

Die wissenschaftliche Applikation benötigt eine Beschreibung der Laufzeitparameter, die das Verhalten des Monte-Carlo-Algorithmus beschreibt. Diese Beschreibung wird durch die Eingabeparameterdatei bereitgestellt. Der Dateiname dieser Datei wird durch `CONFIGURATION_FILE` beschrieben. Die Ergebnisse der Berechnungen werden in der Ergebnisdatei `RESULT_FILE` abgespeichert. Diese Dateinamen sind in der Implementierung von `kreiszahl` fest verdrahtet.

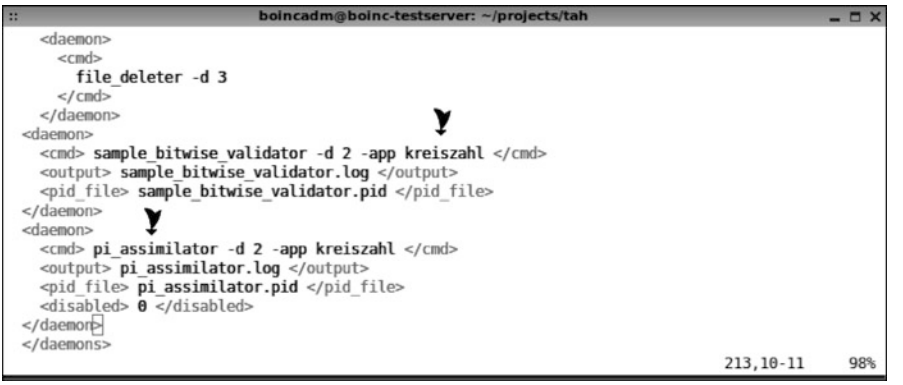


Abb. 11.6 Modifikation der Konfiguration `config.xml` für Kreiszahl@home. Die Dämonen werden für die Verarbeitung von Ergebnissen der wissenschaftlichen Applikation `kreiszahl` eingestellt. Zudem wird ein eigener Assimilierer `pi_assimilator` konfiguriert

Kreiszahl@home benötigt eine Eingabeschablone und eine Ergebnisschablone, um eine Zuordnung der Arbeitspakete zu den fest verdrahteten Dateinamen zu ermöglichen. Listing 11.6 enthält den Inhalt der Eingabeschablone zur Beschreibung der Eingabeparameterdatei, und Listing 11.7 liefert den Inhalt der Ergebnisschablone für die Beschreibung der Ergebnisdatei. Die beiden Schablonen (engl. *templates*) müssen in den Unterordner `templates/` des BOINC-Projekts Kreiszahl@home kopiert werden. **Sie können die Dateien ganz nach Ihren Wünschen benennen, nur beachten Sie, dass die Namen im Listing 11.9 in den Zeilen 44 und 45 angepasst werden müssen!** Abbildung 11.7 verdeutlicht diesen Zusammenhang.

Listing 11.6 Eingabeschablone (engl. *input template*) für die Beschreibung von Eingabeparameterdateien für die statistische Berechnung der Kreiszahl π

```
<file_info>
  <number> 0 </number>
</file_info>
<workunit>
  <file_ref>
    <file_number> 0 </file_number>
    <open_name> configuration.xml </open_name>
  </file_ref>
  <min_quorum> 1 </min_quorum>
  <target_nresults> 1 </target_nresults>
</workunit>
```

Listing 11.7 Ergebnisschablone (engl. *result template*) für die Beschreibung von Ergebnisdateien für die statistische Berechnung der Kreiszahl π

```
<file_info>
  <name> <OUTFILE_0/> </name>
  <generated_locally/>
  <upload_when_present/>
  <max_nbytes> 100000 </max_nbytes>
  <url> <UPLOAD_URL/> </url>
</file_info>
```

```
<result>
  <file_ref>
    <file_name> <OUTFILE_0/> </file_name>
    <open_name> result.xml </open_name>
  </file_ref>
</result>
```

11.2.3 Arbeitspakete erstellen

Die Arbeitspakete für Kreiszahl@home können durch das Programm in Listing 11.8 erstellt werden. Dies ist ein einfaches C/C++-Programm, mit dessen Hilfe Dateien mit eingefügten Parametersätzen erstellt werden. Hierbei erwartet das Programm beim Starten fünf Parameter, welche die Erstellung und den Inhalt der Dateien beschreiben. Ohne Angabe von Parametern wird eine Hilfe und ein Beispiel für den korrekten Programmaufruf ausgegeben. Die Dateien beinhalten – nach Erstellung – die für die Laufzeit der wissenschaftlichen Applikation benötigten Parameter für die Anzahl der Durchläufe `runs` und die Häufigkeit der Iterationen für den Monte-Carlo-Algorithmus `iterations`. Der Inhalt entspricht dabei dem Listing 11.4.

Listing 11.8 Applikation zum Erstellen der Eingabedateien für die Arbeitspakete. Es werden $MAX - ID$ Dateien erstellt

```
// C/C++
#include <string>
#include <iostream>
#include <stdlib.h>
#include <stdio.h>

int main( int argc, char **argv ) {
  if(argc < 6) {
    std::cout << "Usage: " << argv[0]
              << " 4xOptions" << std::endl;
    std::cout << "Options:" << std::endl;
    std::cout << " ID    id to begin with for new file names"
              << std::endl;
    std::cout << " MAX   end id for new file names"
              << std::endl;
    std::cout << " ITER  first iteration value for simulations"
              << std::endl;
    std::cout << " STEP  step between to iteration values"
              << std::endl;
    std::cout << " RUNS  how many runs for each simulation"
              << std::endl;
    std::cout << "Example: ./create_wu_files 1 100 10 10 100"
              << std::endl;
    return 1;
  }

  unsigned int startId   = atoi(argv[1]);
  unsigned int maxId     = atoi(argv[2]);
  unsigned int startIter = atoi(argv[3]);
  unsigned int stepIter  = atoi(argv[4]);
  unsigned int runs      = atoi(argv[5]);

  for( ; startId <= maxId; startId++ ) {
    char filename[1024] = {'\0'};
    sprintf(filename, "workunits/wu_%06d_configuration.xml",
                      startId);
    FILE* f = fopen(filename, "w");
```

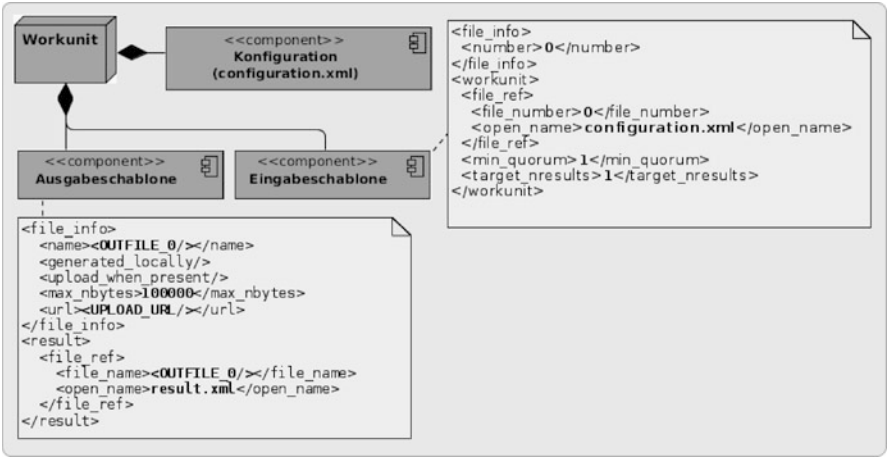


Abb. 11.7 Struktur eines Arbeitspakets für die statistische Berechnung der Kreiszahl π. Ein Arbeitspaket benötigt zwei Schablonen für die Beschreibung der Eingabe-/Ergebnisdateien. `configuration.xml` ist der Name der Eingabedatei, `result.xml` ist die Ergebnisdatei. Es ist erforderlich, dass die Eingabedatei existiert; die Ergebnisdatei muss nicht existieren, diese wird bei der Ausführung der Applikation erstellt

```
    if(f != NULL) {
      fprintf(f, "<configuration>\n");
      fprintf(f, "  <runs>%d</runs>\n", runs);
      fprintf(f, "  <iterations>%d</iterations>\n", startIter);
42    fprintf(f, "</configuration>\n");
      fclose(f);

      startIter += stepIter;
    } else {
47    continue;
    }
  }
}
```

Zeile 22 in Listing 11.8 liefert ein Beispiel `./create_wu_files 1 100 10 10 100` für den richtigen Aufruf des Programms, dabei beschreiben die Zahlenangaben folgenden Sachverhalt:

`startId = 1, maxId = 100`
: Das Programm erstellt Eingabedateien mit dem Dateinamen `wu_%06d_configuration.xml` und `%06d` wird wie in der C-Standardfunktion `printf` – oder vergleichbaren Funktionen – zu einem Zahlenwert umgewandelt, welcher als Parameter übergeben wird. Bei der Umwandlung wird eine Dezimalzahl erwartet. In diesem konkreten Fall ist diese Zahl zwischen `startId=1` und `maxId=100` und erstellt hundert Dateien. Die Namen der Dateien gehen von `wu_000001_configuration.xml` bis `wu_000100_configuration.xml` mit Eins als Inkrement.

`startIter = 10, stepIter = 10`
: Jede dieser erstellten Dateien enthält einen Wert für die Anzahl der Iterationen für den Monte-Carlo-Algorithmus. Die erste Datei enthält den Wert Zehn, in der

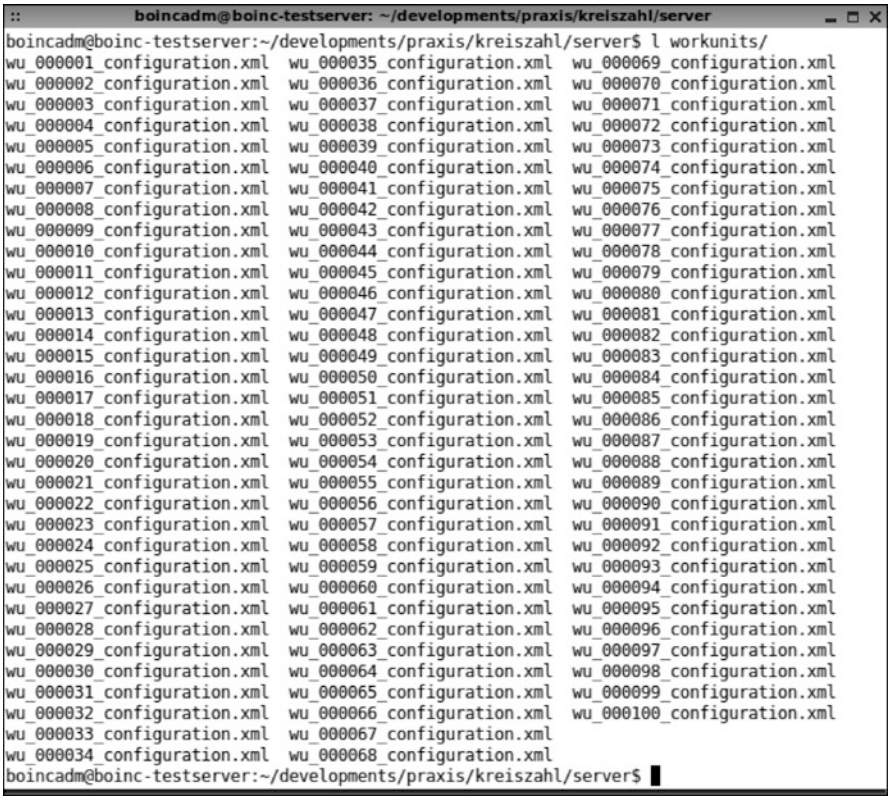


Abb. 11.8 Eingabedateien für die Arbeitspakete

zweiten Datei Zwanzig und so weiter. Mit `startIter` wird der erste Wert und mit `stepIter` wird die Schrittweite zwischen den Einzelwerten definiert. Bei hundert zu erstellenden Dateien würde, in diesem konkreten Fall, die letzte Datei den Wert Tausend enthalten.

`runs = 100`

Es werden in einem Arbeitspaket hundert Berechnungen mit dem Monte-Carlo-Algorithmus durchgeführt. Die Einzelergebnisse dieser Berechnungen werden jeweils in einer Zeile in der Ergebnisdatei abgespeichert.

Nach dem Aufruf `./create_wu_files 1 100 10 10 100` haben wir 100 Dateien im Unterordner `workunits/` erstellt, das Ergebnis wird in Abb. 11.8 gezeigt. Das Skript aus Listing 11.9 ermöglicht das Hinzufügen von Arbeitspaketen. Dabei muss beim Aufruf nur der Projektname und das Verzeichnis der Eingabedateien mit angegeben werden. Das Skript kümmert sich um das Kopieren der Dateien und den Aufruf des BOINC-Skripts `create_work`, welches sich innerhalb eines BOINC-Projekts befindet. Die for-Schleife durchläuft die Liste an Dateien im Unterordner `${DIRECTORY}` und kopiert diese in die Download-Hierarchie von unserem BOINC-Projekt. Im nächsten Schritt wird nur der Basis-

name (engl. *basename*) dieser Datei erwartet, welcher in der Datenbank zuvor gespeichert wurde, wodurch eine Zuordnung ermöglicht wird und dadurch die erforderlichen Informationen für das Arbeitspaket abgespeichert werden können. Als eindeutiger Name für ein Arbeitspaket wird der Dateiname – ohne Dateiendung – der jeweiligen Eingabeparameterdatei verwendet. Die Arbeitspakete benötigen eindeutig Dateinamen. Alle Dateien im Unterorder besitzen schon einen solchen eindeutigen Namen, dies wird durch die Einbindung einer Identifikationsnummer garantiert. Dadurch entfällt die Implementierung einer weiteren Routine für die Namensgebung.

Listing 11.9 Arbeitspakete zum BOINC-Projekt hinzufügen

```
function usage() {
  echo "Usage: ./make_workunits -p 'project name'"
  echo "                        -d 'directory of \
files without trailing slash'"
  exit 1
}

if [ $# -le 3 ]
  then
  usage
fi

COUNT=0

while getopts ":p:m:d:i:" flag
do
  case "$flag" in
    p) PROJECTNAME=$OPTARG ;;
    d) DIRECTORY=$OPTARG ;;
    *) usage ;;
  esac
done

# Root directory where projects should be installed.
ROOTPRE=${HOME}
# Directory of one BOINC-Project.
ROOTPROJECT=${ROOTPRE}/projects/${PROJECTNAME}

for file in ${DIRECTORY}/*
do
  WUNAME=`basename ${file} .xml`

  WUNAME1=${file}
  WUPATH1=`pwd`/${WUNAME1}
  echo "WU1: " ${WUPATH1}

  cd ${ROOTPROJECT}
    pwd
    BASEWUNAME1=`basename ${WUNAME1}`
    cp ${WUPATH1} `bin/dir_hier_path ${BASEWUNAME1}`

    cmd="./bin/create_work -appname kreiszahl \
      -wu_name ${WUNAME} \
      -wu_template templates/tplInput_Kreiszahl \
      -result_template templates/tplResult_Kreiszahl \
      ${BASEWUNAME1}"

    echo $cmd
    $cmd
  cd -
done
```

Der Aufruf `./make_workunits -p tah -d workunits` fügt die Arbeitspakete zum BOINC-Projekt hinzu. Wichtig ist bei dem Aufruf, dass die Angabe des Verzeichnisses keinen Schrägstrich am Ende besitzt, dies würde zu falschen Pfadangaben führen.

11.2.4 Validierung – Prüfung der Ergebnisse

Jedes BOINC-Projekt besitzt zwei Standardvalidierer, die bei jeder Projekterstellung automatisch mit installiert werden. Diese beiden Validierer ermöglichen das Validieren unserer Berechnungsergebnisse und können beide verwendet werden. Allerdings sollte im Vorfeld überlegt werden, welcher von beiden sinnvollerweise verwendet werden sollte. Die Validierer unterscheiden sich sehr stark in der Funktionsweise der Validierung.

Im Fall von Kreiszahl@home hängt die sinnvolle Entscheidung eines Validierers von dem Konfigurationsparameter `min_quorum` für die Arbeitspaketerstellung ab. Listing 11.6 beschreibt einen Wert `min_quorum` von Eins, somit wird nur ein Ergebnis benötigt, um das Arbeitspaket erfolgreich validieren und an den Assimilierer weiterreichen zu können. Wenn dieser Wert höher gesetzt wird, so müssen auch ebenso viele Berechnungen erfolgreich durchgeführt und die Ergebnisse an das Projekt zurückgesendet werden.

Wenn davon ausgegangen werden kann, dass die zu erwartenden Ergebnisse gleich sind, so könnte der Standardvalidierer `sample_bitwise_validator` Verwendung finden. Ist das in diesem Projekt sinnvoll? Werden die Ergebnisse gleich sein? Bedenken Sie, dass die wissenschaftliche Applikation sehr stark mit Zufallszahlen arbeitet. Es ist eher unwahrscheinlich, dass in diesem Fall wirklich äquivalente Ergebnisse berechnet werden!

In diesem Projekt definieren wir uns einfach, dass wir nur ein erfolgreiches Ergebnis benötigen und wir wissen, dass dieses Ergebnis als korrekt angesehen werden kann. In Folge dessen können zwar für ein Arbeitspaket mehrere Berechnungen erfolgen, es muss aber nur eins von diesen beachtet werden. Aufgrund dieser Überlegungen kann auch der `sample_trivial_validator` in Frage kommen.

Abhängig vom gewählten Validierer müssen die Einstellungen für die Arbeitspakete vorgenommen werden. Bei diesem Typ einer wissenschaftlichen Applikation – die Ergebnisse hängen von der Wahrscheinlichkeit ab – kann es bei der Nutzung des Validierers `sample_bitwise_validator` dazu kommen, dass zu wenig bis gar keine Ergebnisse als erfolgreich validiert werden. Die nachfolgenden zwei Beispiele sollen die erwähnte Problematik veranschaulichen und Ihnen helfen, diese in eigenen BOINC-Projekten zu umgehen.

Für die folgenden Beispiele erstellen wir anstatt der hundert nur zehn Arbeitspakete. Diese Anzahl reicht vollkommen aus, um die Problematik zu veranschaulichen. Der Vorteil hierbei ist eine schnellere Ausführung der Berechnungen auf den teilnehmenden Rechnern. Für die Erstellung muss der zweite Parameter für `./create_wu_files` auf zehn gesetzt werden:

```
./create_wu_files 1 10 10 10 100
```

Die Datenbanktabelle `results_pi` sollte nach der Berechnung aller Arbeitspakete 1000 Ergebnisse abgespeichert haben, zehn Arbeitspakete mit je hundert Ausführungen des Monte-Carlo-Algorithmus.

Erstes Beispiel – Wie es nicht sein sollte!

Wir stellen in der Konfiguration `config.xml` den Validierer `sample_bitwise_validator` ein:

Listing 11.10 Erste Beispielkonfiguration `config.xml` mit dem Validierer `sample_bitwise_validator` zur Prüfung auf Gleichheit von zwei Berechnungsergebnissen

```
<daemon>
  <cmd> sample_bitwise_validator -d 3 -app kreiszahl </cmd>
  <output> sample_bitwise_validator.log </output>
  <pid_file> sample_bitwise_validator.pid </pid_file>
  <disabled> 1 </disabled>
</daemon>
```

Weiterhin stellen wir für unsere Eingabeschablone folgende Parameter ein:

Listing 11.11 Unvorteilhafte Konfiguration für die wissenschaftliche Applikation Kreiszahl@home und Vorgabe, mindestens zwei Berechnungsergebnisse für die Validierung zu benötigen

```
<file_info>
  <number>0</number>
</file_info>
<workunit>
  <file_ref>
    <file_number>0</file_number>
    <open_name>configuration.xml</open_name>
  </file_ref>
  <min_quorum> 2 </min_quorum>
  <target_nresults> 5 </target_nresults>
  <max_total_results> 5 </max_total_results>
  <max_success_results> 5 </max_success_results>
</workunit>
```

Was bedeuten diese Einstellungen nun? Es werden in jedem Fall zwei Berechnungsergebnisse für eine erfolgreiche Validierung benötigt, beschrieben durch Zeile 9. Zeile 10 definiert fünf zu Beginn initialisierte Arbeitspakete, von denen nur die zwei zuvor erwähnten nötig sind. Weiterhin können bis zu fünf Arbeitspakete an Teilnehmer vergeben werden; die ersten beiden brauchbaren Ergebnisse finden Verwendung. Durch Zeile 11 wird definiert, dass nur maximal fünfmal der Versuch gestartet wird, ein Ergebnis zu erhalten. Sollte dieser Wert überschritten werden, so wird das Arbeitspaket als Fehler gekennzeichnet und es wird nicht weiter versucht, dieses zu berechnen. Dies kann relevant sein, wenn sich Teilnehmer ein oder mehrere Arbeitspaket(e) heruntergeladen haben, allerdings die Berechnung nicht zu einem gewissen Zeitpunkt zum Ende führen. Daraufhin wird versucht, Ergebnisse durch weitere Teilnehmer zu erhalten, die ein gleiches Arbeitspaket erhalten. Mit dem Wert in Zeile 12 werden die maximal möglichen erfolgreichen Ergebnisse auf fünf gesetzt, ein höherer Wert würde keinen Sinn ergeben, da sowieso nicht mehr

Abb. 11.9 Die Berechnungsergebnisse für das falsch konfigurierte Projekt: Kreiszahl@home. Es sind 109 Einträge in der Datenbanktabelle `results_pi` vorhanden. 100 von diesen sind Ergebnisse von Berechnungen und die restlichen 9 sind Fehlerbeschreibungen für Fehler bei der Bearbeitung der Ergebnisse

Rechnungen durchgeführt werden. Hier wäre es vielleicht sinnvoll, eine Minimierung auf drei zu setzen, so dass nicht unnötigerweise weitere Arbeitspakete verteilt werden, wenn die benötigten zwei Ergebnisse schon vorhanden sind.

Wir erwarten tausend Ergebnisse. Wie Abb. 11.9 zeigt, erhalten wir aber nur etwa hundert! Woran liegt das? Ganz einfach daran, dass wir nur ein Pärchen von zusammengehörenden Arbeitspaketen als valide erkannt haben, wie Abb. 11.10 zeigt. Die Abbildungen zeigen, dass zwar die erstellten fünfzig Arbeitspakete erfolgreich berechnet, allerdings nur zwei von ihnen als valide erkannt wurden, und dies auch nur aus Zufall. Denn die Arbeitspakete wurden alle nur an eine Maschine gesendet und zufällig wurden zwei zugehörige Arbeitspakete zur selben Zeit gestartet, so dass der Zufallszahlengenerator dieselben Zahlenwerte generierte.

An dieser Stelle könnten Sie ein anderes Verfahren für die Initialisierung des Zufallszahlengenerators verwenden, allerdings würden dadurch die Ergebnisse sicherlich noch einmaliger und wir würden kein einziges valides Ergebnis erhalten.

Wären die Pakete mit einem Versatz von einer Sekunde ausgeführt worden, so wären keine Arbeitspakete als valide erkannt worden. Zudem wurden innerhalb des BOINC-Managers zwei Prozessoren für die Berechnungen freigegeben, so dass ein paralleles Starten überhaupt erst möglich ist. Sollte allerdings nur ein Prozessor für die Berechnungen freigegeben sein und dieser Teilnehmer erhält zwei Berechnungen für dasselbe Arbeitspaket, so kann im Vorfeld schon davon ausgegangen werden, dass dieses Arbeitspaket niemals als valide eingestuft wird. Der Zufallszahlengenerator würde mit unterschiedlichen Initialisierungswerten in Betrieb genommen werden. Wundern Sie sich zudem nicht darüber, dass zwei Arbeitspakete erkannt wurden, aber nur rund hundert Ergebnisse in der Datenbank abgespeichert sind. Sie werden in Abschn. 11.2.5 lesen, dass nur eine Ergebnisdatei in der Datenbanktabelle abgespeichert wird. Die restlichen Ergebnisse werden verworfen und nicht beachtet.

Wir können das Fazit ziehen, dass es wenig sinnvoll ist, eine Prüfung auf gleiche Ergebnisse durchzuführen, wenn die Basis der Berechnungen auf Zufallszahlen beruht. Eine Möglichkeit, solche Probleme zu handhaben, wäre die Implementierung einer Toleranzprüfung. Wenn abgeschätzt werden kann, in welchem Bereich sich die Ergebnisse bewegen müssen, so kann durch die Toleranz eine erfolgreiche Validierung durchgeführt werden. In diesem Fall der statistischen Berechnung ist das vielleicht noch denkbar. Aber kann bei physikalischen Modellen eine Eingren-

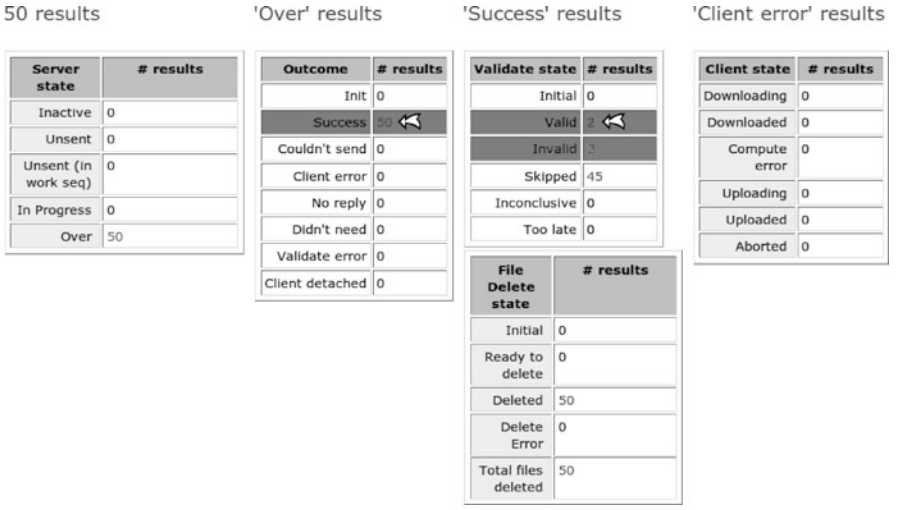


Abb. 11.10 Es sind alle **50** Arbeitspakete erfolgreich abgearbeitet worden, allerdings nur ein Ergebnispärchen als valide eingestuft

zung gemacht werden? Kann es nicht auch sein, dass die Ergebnisse wirklich die Naturgesetze wiedergeben?

Ein Beispiel für diesen Fall sind Resonanzfrequenzen. Diese Frequenzen lassen ein System quasi „explodieren“ [183, Tacoma-Narrows-Brücke] und dabei kann alles mögliche als Ergebnis herauskommen. Kann solch eine Berechnung durch Toleranzen validiert werden? Es ist von Fall zu Fall zu unterscheiden; wenn im Vorfeld keine Entscheidung getroffen werden kann, so sollte die im nächsten Teilabschnitt beschriebene Methodik verwendet werden.

Zweites Beispiel – Eine bessere Konfiguration!

Eine bessere Konfiguration, im Vergleich zu der im vorherigen Abschnitt, wird in diesem Teilabschnitt beschrieben und für diesen Typ einer wissenschaftlichen Applikation empfohlen. Der wichtigste Unterschied zur vorherigen Konfiguration ist die Verwendung eines anderen Validierers. Das nachfolgende Listing 11.12 zeigt die neue Konfiguration für den Validierer in der Datei `config.xml`.

Listing 11.12 Zweite Beispielkonfiguration `config.xml` für den Validierer `sample_trivial_validator` mit einer simplen Prüfung auf Validierung, ob eine vorgegebene Anzahl an Berechnungszyklen überschritten wurde oder nicht

```
<daemon>
  <cmd> sample_trivial_validator -d 3 -app kreiszahl </cmd>
  <output> sample_trivial_validator.log </output>
  <pid_file> sample_trivial_validator.pid </pid_file>
  <disabled> 0 </disabled>
</daemon>
```

Abb. 11.11 Die Berechnungsergebnisse für das ordentlich konfigurierte Projekt: Kreiszahl@home. Es sind erfolgreich **1000** Berechnungsergebnisse abgespeichert und für die spätere Nutzung bereitgestellt

In dieser Konfiguration wird der triviale Validierer `sample_trivial_validator` aus den Standardvalidierern von BOINC eingestellt. Dieser prüft nur, ob die Anzahl der benötigten Rechenzyklen überschritten wurde. Ist dieser Wert nicht überschritten, so wird das Arbeitspaket als valide eingestuft, ansonsten als nicht valide.

Die Eingabeschablone kann aus dem ersten Beispiel übernommen werden. Es können weiterhin mindestens zwei Arbeitspakete erforderlich sein, um eine Validierung durchzuführen. Es sei an dieser Stelle darauf hingewiesen, dass dies nicht nötig ist und der Wert ruhig auf Eins gesetzt werden kann. Wozu sollten zwei Ergebnisse benötigt werden, wenn diese sowieso nicht gegeneinander geprüft werden? Diese Entscheidung wird Ihnen überlassen. Zusätzlich sei noch erwähnt, dass es zu Problemen führen kann, wenn mindestens zwei Ergebnisse erforderlich sein sollten. Wenn Sie Pech haben, dann kann es bei fünf zu verteilenden Arbeitspaketen – an fünf verschiedene Teilnehmer oder an einen Teilnehmer – dazu kommen, dass das gesamte Arbeitspaket als fehlerhaft markiert wird. Dieses Verhalten tritt dann auf, wenn dieser/diese Teilnehmer das Arbeitspaket nicht bis zum Ende bearbeiten. Dann haben Sie eventuell nur ein Ergebnis von einem Teilnehmer erhalten und warten umsonst auf ein zweites, was Sie ja im Grunde gar nicht benötigen. Diese Einstellung hängt wieder von den Kriterien einer wissenschaftlichen Applikation ab.

Wie im ersten Beispiel erwarten wir – nach der Abarbeitung aller Arbeitspakete – 1000 Berechnungsergebnisse in der Datenbank. Abbildung 11.11 verdeutlicht, dass dies **mit Erfolg geklappt** hat! Abbildung 11.12 zeigt weiter, dass 50 Arbeitspakete berechnet und erfolgreich validiert wurden. Es wurden pro Arbeitspaket fünf Berechnungen definiert, bei der Assimilation wird allerdings nur eines von diesen fünf abgespeichert. Es könnten rein theoretisch auch alle fünf abgespeichert werden, dies würde nur bedeuten, dass wir möglicherweise ein stabileres Ergebnis für unseren Monte-Carlo-Algorithmus erhalten. Dies liegt aktuell nicht im Fokus unserer Betrachtungen!

11.2.5 Assimilation – Ergebnisse abspeichern

Wie eingangs erwähnt werden die einzelnen Ergebnisse der jeweiligen Ausführungen des Monte-Carlo-Algorithmus nacheinander in einer Ergebnisdatei abgespeichert. Diese Datei wird vom Assimilierer ausgelesen und der Wert der Ite-

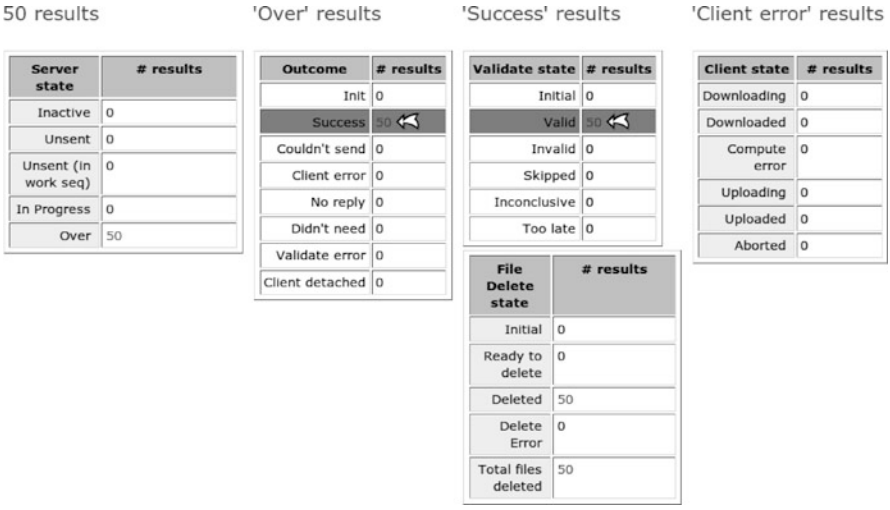


Abb. 11.12 Es sind alle **50** Arbeitspakete mit Erfolg bearbeitet und alle erfolgreich validiert. Dementsprechend hat keine Berechnung die Grenze der erlaubten Rechenzyklen überschritten

Abb. 11.13 Dateien für die Implementierung eines Assimilierers für Kreiszahl@home. Die Dateien `assimilate_handler.h`, `assimilate.cpp` und `validate_util.cpp` entstammen den BOINC-Quellen und `pi_assimilator.cpp`, `results_pi.cpp` und `results_pi.h` sind für Kreiszahl@home erstellt worden und enthalten spezifische Implementierungen

rationen und das Berechnungsergebnis in einer Datenbank abgespeichert. Abbildung 11.13 zeigt die Dateien für die Implementierung des hier verwendeten Assimilierers. Die Dateien `assimilate_handler.h`, `assimilator.cpp` und `validate_util.cpp` sind aus dem Unterordner `sched/` der BOINC-Quellen kopiert. Diese Dateien beinhalten ein Grundgerüst (engl. *framework*) zur Implementierung eines eigenen Assimilierers. Die weiteren Dateien wurden für den Assimilierer erstellt. Sie werden in den nachfolgenden Abschnitten besprochen und erläutert.

Vorbereitungen für die Datenbank

Die Berechnungsergebnisse liefern mehrere Zahlenwerte, die in einer Datei gespeichert und dann zum Projektserver übertragen werden. Nach erfolgreicher Validierung liegen diese Dateien im Kreiszahl@home-Unterordner `upload/`. Diese Daten können direkt durch eine Applikation verwendet werden, um so zum Beispiel eine grafische Lösung zu erhalten. Dadurch wird das Verstehen der Ergebnisse eventuell erleichtert und es kann abgelesen werden, welcher Wert für die Anzahl an Iterationen einer Berechnung ausreichend ist. Durch ein direktes Bearbeiten der Ergebnisse durch externe Anwendungen kann es schwierig und dadurch sehr aufwendig sein, eine effektive Verarbeitung zu realisieren. Jede Datei müsste in diesem Fall einzeln auf Existenz geprüft, geöffnet und zeilenweise ausgelesen werden. Zudem kann es sein, dass die externen Anwendungen nicht auf derselben Maschine, wo die Ergebnisdateien vorliegen, installiert sind. In diesem Fall müssten die Dateien kompliziert zur Verfügung gestellt werden.

Hier empfiehlt es sich, die Rechenergebnisse in einer Datenbank zu speichern. So können die Rechenergebnisse durch triviale SQL-Anweisungen selektiert und ausgelesen werden. Dieses Verfahren erhöht den Komfort beim Umgang mit den Ergebnissen und ermöglicht eine schnellere Implementierung von Verarbeitungsroutinen.

Datenbanktabelle

Das Rechenergebnis besteht aus einer Liste von Zahlenwerten, die das Ergebnis eines Berechnungsdurchlaufs mit dem Monte-Carlo-Algorithmus beschreiben. Was uns interessiert, ist die Anzahl der Iterationen, die für eine Berechnung eines Ergebnisses verwendet wurden, und natürlich interessiert uns auch das Ergebnis. Entsprechend werden zwei Tabellenspalten in der neuen Datenbanktabelle benötigt. Nun kann es auch vorkommen, dass eine Berechnung fehlschlägt, aus welchem Grund auch immer, weshalb wir eine Tabellenspalte für die Fehlerbeschreibung hinzufügen. *Es wird definiert, wenn das Beschreibungsfeld eines Fehlers gesetzt ist, dann sind die beiden anderen Fehler auf* -1 *gesetzt.* Zusätzlich wird eine Tabellenspalte erstellt, die für jedes Rechenergebnis eine eindeutige Identifikation in Form eines inkrementierenden Zahlenwerts speichert. Listing 11.13 enthält das SQL-Statement zur Erstellung dieser Datenbanktabelle.

Listing 11.13 SQL-Statement zur Erstellung der Datenbanktabelle für die Abspeicherung der Ergebnisse im Kreiszahl@home-Projekt

```
CREATE TABLE `tah`.`results_pi` (
  `id` INT NOT NULL AUTO_INCREMENT ,
  `iterations` INT NOT NULL ,
  `value` DOUBLE NOT NULL ,
  `description` BLOB ,
  PRIMARY KEY ( `id` )
) ENGINE = MYISAM ;
```

Abstraktionsschicht für den Zugriff auf die Datenbank

Abschnitt 9.5.1 beschreibt, wie eine Abstraktionsschicht für Datenbanktabellen in BOINC erstellt und verwendet werden kann. Dieses Vorgehen wird in Listing 11.14 und 11.15 angewendet. Es wird eine Struktur `RESULTS_PI` erstellt, die die einzelnen Werte einer Datenreihe aus der Datenbank aufnehmen kann und eine Member-Funktion `clear()` besitzt, um die Member-Variablen zurückzusetzen. Weiterhin wird eine Struktur `DB_RESULTS_PI` definiert, die alle Member-Funktionen und Member-Variablen der Strukturen `DB_BASE` und `RESULTS_PI` erbt. Durch die Vererbung ist es möglich, die Methoden der Basisstruktur `DB_BASE` zu verwenden, die unter anderem die Möglichkeit zum Hinzufügen von neuen Datenreihen besitzen. Für diesen Zweck muss nur eine Instanz dieser Struktur erstellt werden, den Member-Variablen müssen Werte zugewiesen und dann die Member-Funktion `insert()` aufgerufen werden. Die Member-Funktion `insert()` erstellt ein syntaktisch korrektes SQL-Statement und führt es aus. Dieses Prinzip wird in Listing 11.16 weiter verdeutlicht. Weiterhin ist zu beachten, dass aus Gründen der Konformität die Strukturen denselben Namen wie die Datenbanktabelle erhalten sollten. Hier ist das die Datenbanktabelle `results_pi`, und die Struktur für die Abstraktion sollte dementsprechend `RESULTS_PI` und die Struktur für die Abstraktion des Datenbankzugriffs `DB_RESULTS_PI` heißen.

Listing 11.14 Header-Datei `results_pi.h` für die Abstraktion des Datenbankzugriffs auf die Datenbanktabelle `results_pi`

```
#ifndef __RESULTS_PI__
#define __RESULTS_PI__

#include <db_base.h>
#include <boinc_db.h>

struct RESULTS_PI {
  int id;
  int iterations;
  double value;
  char description[BLOB_SIZE];
  void clear();
};

struct DB_RESULTS_PI : public DB_BASE, public RESULTS_PI {
  DB_RESULTS_PI(DB_CONN* p=0);
  int get_id();
  void db_print(char*);
  void db_parse(MYSQL_ROW &row);
};

#endif // __RESULTS_PI__
```

Listing 11.15 enthält die Implementierungen der definierten Methoden. Zeile 18 selektiert dabei die Datenbanktabelle `results_pi`, in der die Ergebnisse gespeichert werden sollen.

Listing 11.15 Quelltextdatei `results_pi.cpp` für die Abstraktion des Datenbankzugriffs auf die Datenbanktabelle `results_pi`

```
// BOINC
#include <boinc_db.h>
#include <db_base.h>
#include <str_util.h>
#include <str_replace.h>

#include <results_pi.h>

// @see boinc_db.cpp
#define ESCAPE(x) escape_string(x, sizeof(x))
#define UNESCAPE(x) unescape_string(x, sizeof(x))

void RESULTS_PI::clear() {
  memset(this, 0, sizeof(*this));
}

DB_RESULTS_PI::DB_RESULTS_PI(DB_CONN* dc) :
  DB_BASE("results_pi", dc?dc:&boinc_db){}

int DB_RESULTS_PI::get_id() {
  return id;
}

void DB_RESULTS_PI::db_print(char *buf) {
  ESCAPE(description);
  sprintf(buf, "iterations=%d, value=%f, "
               "description='%s'",
        iterations, value, description);
  UNESCAPE(description);
}

void DB_RESULTS_PI::db_parse(MYSQL_ROW &r) {
  int i=0;
  clear();
  id=atoi(r[i++]);
  iterations=atoi(r[i++]);
  value=atof(r[i++]);
  strcpy2(description, r[i++]);
}
```

Die Zeile 25 ist dafür verantwortlich, entsprechende Zeichen in der Zeichenkette `description` zu markieren, welche eventuell Einfluss auf das SQL-Statement haben könnten. Zeile 29 entfernt diese Markierungen wieder. Beide Aufrufe sind mitgelieferte BOINC-Framework-Funktionalitäten und entsprechend aus den BOINC-Quellen kopiert und in dieses Beispiel eingefügt. In den BOINC-Quellen sind diese Aufrufe nicht in Header-Dateien, sondern in Quelltextdateien definiert. Daher können diese Aufrufe nicht inkludiert werden und sind aus diesem Grund als Kopie mit in die Implementierung übernommen worden.

Implementierung

Der Assimilierer (engl. *assimilator*) übernimmt das Abspeichern der einzelnen Berechnungsergebnisse. Innerhalb von Kreiszahl@home werden die Ergebnisse in

Form von einzelnen Datenreihen in der zuvor beschriebenen Datenbanktabelle eingefügt. Im BOINC-Framework werden mehrere Assimilierer als Beispiele mitgeliefert. Auf Basis des Assimilierers `sample_assimilator` ist der in Listing 11.16 vorgestellte Assimilierer implementiert. Im ersten Schritt werden alle Routinen für das Kopieren oder Abspeichern von Dateien entfernt, so dass ein Rahmen für den neuen Assimilierer zur Verfügung steht

Listing 11.16 Quelltextdatei `pi_assimilator.cpp` für den Assimilierer, der für Kreiszahl@home verwendet wird

```
#include <vector>
#include <string>
#include <cstdlib>

#include "boinc_db.h"
#include "error_numbers.h"
#include "filesys.h"
#include "sched_msgs.h"
#include "validate_util.h"
#include "sched_config.h"

#include <str_util.h>
#include <str_replace.h>

// Kreiszahl
#include <results_pi.h>

using std::vector;
using std::string;
```

Im oberen Bereich wird die Header-Datei `results_pi.h` inkludiert, so dass die Abstraktionsschicht für den Datenbankzugriff zur Verfügung steht.

```
struct XML_RESULTS {
  std::vector<int> runs;
  std::vector<int> iterations;
  std::vector<double> values;

  XML_RESULTS() {
    init();
  }

  inline void init() {
    runs.clear();
    iterations.clear();
    values.clear();
  }

  void get(int id, int& run,
           int& iteration, double& value) {
    run = runs[id];
    iteration = iterations[id];
    value = values[id];
  }

  void parse(FILE *resultFile) {
    MIOFILE xmlMIOFile;
    xmlMIOFile.init_file(resultFile);
    XML_PARSER p(&xmlMIOFile);

    bool is_tag = false;
    char tag[128] = {'\0'};
    char attrs[256] = {'\0'};
    while( !p.get(tag, sizeof(tag), is_tag,
                  attrs, sizeof(attrs))) {
```

```
      if(!is_tag) continue;
      double value = 0.0;
      if(p.parse_double(tag, "pi", value)) {
        this->values.push_back(value);

        char attrs_runs[128] = {'\0'};
        parse_attr(attrs, "run", attrs_runs,
                           sizeof(attrs_runs));
        this->runs.push_back(atoi(attrs_runs));

        char attrs_iterations[128] = {'\0'};
        parse_attr(attrs, "iterations", attrs_iterations,
                           sizeof(attrs_iterations));
        this->iterations.push_back(atof(attrs_iterations));
      }
    }
  }
};
```

Die Struktur `XML_RESULTS` ist für das Auslesen der Ergebnisdateien zuständig. Dabei ist es unwichtig, wie viele Ergebnisse in der Ergebnisdatei vorhanden sind; dies ist auch nicht relevant, da jedes Ergebnis für sich einen Durchlauf einer Berechnung mit dem Monte-Carlo-Algorithmus repräsentiert. Die Member-Funktion `parse()` enthält die Logik oder Intelligenz dieser Struktur. In der while-Schleife werden die einzelnen XML-Tags aus der Datei ausgelesen und die erforderlichen Informationen gefiltert. Wenn in den nachfolgenden Zeilen das XML-Tag `<pi ...> ... </pi>` gefunden wird, so werden die Attribute und das Berechnungsergebnis eines XML-Tags in die dafür vorgesehenen Member-Variablen abgespeichert. Diese Member-Variablen in den Zeilen 21 bis 23 sind C++-Container und speichern die Ergebnisse seit Start der Ausführung oder seit der letzten Checkpoint-Erstellung ab.

```
int write_error(char* p) {
  DB_RESULTS_PI dbres;
  dbres.iterations = -1;
  dbres.value = -1.0;
  strcpy2(dbres.description, p);
  if(dbres.insert() < 0) {
    return ERR_DB_CANT_CONNECT;
  }
  return 0;
}
```

Falls während der Verarbeitung Fehler auftreten, so werden diese auch in der Ergebnistabelle abgespeichert. Dabei werden die restlichen Datenspalten – nicht die Identifikationsnummer – auf -1 gesetzt. Diese Funktion zeigt schon die einfache Handhabung der in den Listings 11.14 und 11.15 erstellten Datenbankabstraktion. Drei Schritte sind ausreichend, um eine neue Datenreihe in der Datenbanktabelle abzuspeichern:

1. Eine Variable der Datenbankstruktur erstellen,
2. den Member-Variablen Werte zuweisen und
3. im letzten Schritte die Member-Funktion `insert()` aufrufen.

```
int assimilate_handler(
    WORKUNIT& wu,
    vector<RESULT>& /*results*/,
    RESULT& canonical_result
```

```
) {
    unsigned int i;

    if (wu.canonical_resultid) {
        vector<FILE_INFO> output_files;
      get_output_file_infos(canonical_result,
                            output_files);
      unsigned int n = output_files.size();
      for (i=0; i<n; i++) {
        FILE_INFO& fi = output_files[i];

        FILE* f = boinc_fopen(fi.path.c_str(), "r");
        if(f != NULL) {
          XML_RESULTS xmlResults;
          xmlResults.parse(f);
          fclose(f);

          for(unsigned int j=0;
              j<xmlResults.runs.size(); j++) {
            DB_RESULTS_PI dbres;
            dbres.iterations = xmlResults.iterations[j];
            dbres.value = xmlResults.values[j];
            dbres.insert();
          }
        }
      }
    } else {
      char buf[1024] = {'\0'};
      sprintf(buf, "%s: 0x%x\n", wu.name, wu.error_mask);
      return write_error(buf);
    }
    return 0;
}
```

Die Funktion `assimilate_handler()` wird vom Basis-Framework für die Assimilierung aufgerufen. Alle erforderlichen Daten werden als Referenzen mit übergeben und können verarbeitet werden. Diese Funktion wird für jedes zu verarbeitende Arbeitspaket aufgerufen. In Zeile 92 werden alle Ergebnisdateien eines Arbeitspakets sequentiell abgearbeitet. Jede Datei wird geöffnet und es wird versucht, sie durch die vorher implementierte Struktur zum Parsen der Ergebnisdateien zu verarbeiten. Wenn eine Ergebnisdatei verarbeitet wurde, so ist die Anzahl der vorhandenen Elemente in den jeweiligen Container-Variablen größer als Null. Dieser Wert gibt Auskunft über die Anzahl der abzuspeichernden Ergebnisse. In Zeile 101 werden die einzelnen Ergebnisse durchlaufen und daraufhin in der Datenbank abgespeichert.

Die Berechnungsergebnisse können auch in einer externen Datenbank (sog. *Science Database*) abgespeichert werden. Für diesen Fall muss eine weitere Datenbankverbindung aufgebaut und dem Konstruktor von der Struktur `DB_RESULTS_PI` übergeben werden. Listing 11.17 zeigt exemplarisch, wie solch eine *Science Database* verwendet werden kann.

Listing 11.17 Anwendungsbeispiel für die Nutzung einer *Science Database*

```
#include <db/boinc_db.h>
...
DB_CONN science_db;
int retval = science_db.open(
        config.db_name, config.db_host,
        config.db_user, config.db_passwd
);
```

```
8   if (retval) {
      printf("science_db.open: %d\n", retval);
    } else {
      DB_RESULTS_PI dbres(science_db);
      ...
13    ...
    }
```

Makefile

Das Makefile erstellt den Assimilierer. Bei diesem Prozess werden die Dateien beachtet, die der Variablen `S` zugewiesen sind. Die von BOINC mitgelieferte Datei `assimilator.cpp` enthält das Basis-Framework eines Assimilierers, das die zuvor erwähnte Handler-Funktion für jedes Arbeitspaket aufruft. Im Verzeichnis des Makefiles muss zudem ein Ordner mit den BOINC-Quellen oder ein symbolischer Link zu diesen mit dem Namen `boincsrc/` vorhanden sein. Erst dadurch ist ein erfolgreiches Kompilieren möglich. Ein Assimilierer ist abhängig von den MySQL-Bibliotheken und zahlreichen BOINC-spezifischen Header-Dateien und Bibliotheken, die in den BOINC-Quellen zur Verfügung stehen. Nach dem erfolgreichen Kompilieren kann die Applikation `pi_assimilator` in den Unterordner `bin/` des BOINC-Projekts kopiert werden. Dieser Assimilierer wurde zuvor schon in Abschn. 11.2.1 konfiguriert.

Listing 11.18 Makefile für den Assimilierer `pi_assimilator` des Kreiszahl@home-Projekts

```
1   A=pi_assimilator
    S+=assimilator.cpp \
      pi_assimilator.cpp \
      validate_util.cpp \
      results_pi.cpp
6   I+= -I./ `mysql_config --cflags`
    L+= -pthread \
      `mysql_config --libs`

    INCLUDES=\
11    -I./boincsrc \
      -I./boincsrc/db \
      -I./boincsrc/sched \
      -I./boincsrc/lib \
      ${I}
16
    LIBARIES=\
      -L./boincsrc/sched -lsched \
      -L./boincsrc/api -lboinc_api \
      -L./boincsrc/lib -lboinc \
21    ${L}

    all: ${S}
      g++ -O2 -Wall -o ${A} ${S} ${INCLUDES} ${LIBARIES}

26  run:
      ./${A}

    clean:
      rm -rf *~ *.o
31    rm -rf ${A}
```

11.3 Arbeitspakete verteilen und rechnen lassen

Wenn die vorherigen Abschnitte durchgearbeitet sind, dann kann Kreiszahl@home gestartet werden. Der Aufruf

```
boincadm@boinc-testserver:~/projects/tah$ ./bin/start
Entering ENABLED mode
Starting daemons
  Starting daemon: feeder -d 3
  Starting daemon: transitioner -d 3
  Starting daemon: file_deleter -d 3
  Starting daemon: sample_bitwise_validator -d 2 -app kreiszahl
  Starting daemon: pi_assimilator -d 2 -app kreiszahl
```

startet Kreiszahl@home. Die letzten beiden Zeilen zeigen die zu startenden Dämonen, die nur für diese aktuelle wissenschaftliche Applikation arbeiten. Das Debug-Level wurde bei den ersten drei Dämonen auf dem Standardwert Drei und bei den beiden anderen Dämonen auf Zwei belassen. Mit dem Aufruf von `./bin/status` kann eine Prüfung der aktuellen Dämonen durchgeführt werden. Dabei sollte die Ausgabe zeigen, dass alle Dämonen gestartet sind, und in der Spalte *status* sollten die einzelnen Zeilen jeweils *running* enthalten.

Sie können sich nun mit dem BOINC-Manager zum Projekt verbinden. Abbildung 11.14 zeigt die Ansicht der Bearbeitung, wenn eine erfolgreiche Registrierung beziehungsweise Authentifizierung erfolgte, die wissenschaftliche Applikation und Arbeitspakete auf den Rechner heruntergeladen wurden und ausgeführt werden. Die Abb. 11.15 und 11.16 zeigen die Ansicht der Datenbanktabelle `results_ip`, in der die Berechnungsergebnisse gespeichert sind. Zu diesem Zeitpunkt sind in der Tabelle schon 2800 Ergebnisse eingetragen. Die zweite Abbildung liefert einen Auszug der Datenreihen; hier sind beispielsweise 130 Iterationen für die ersten Durchläufe des Monte-Carlo-Algorithmus verwendet worden und das vierte Berechnungsergebnis hat den Wert 3.2. Es sei an dieser Stelle darauf hingewiesen, dass die Arbeitspakete nicht sequenziell bearbeitet werden. Sie haben sicherlich bemerkt, dass der Wert der Iterationen bei den ersten Ergebnissen viel zu hoch ist. Wir erwarten an der Stelle den Wert 10. Dies Beispiel zeigt, dass eine zufällige Abarbeitungsfolge stattfindet. Abbildung 11.17 zeigt den aktuellen Status der Bearbeitung dieses Projekts.

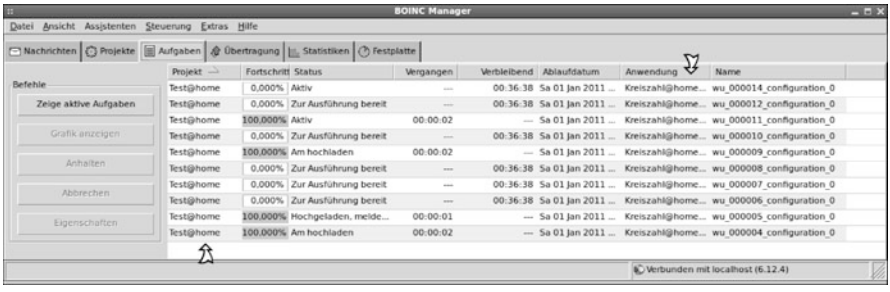
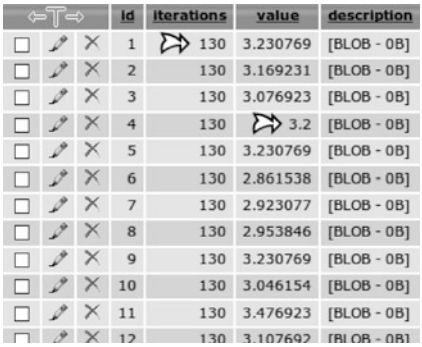


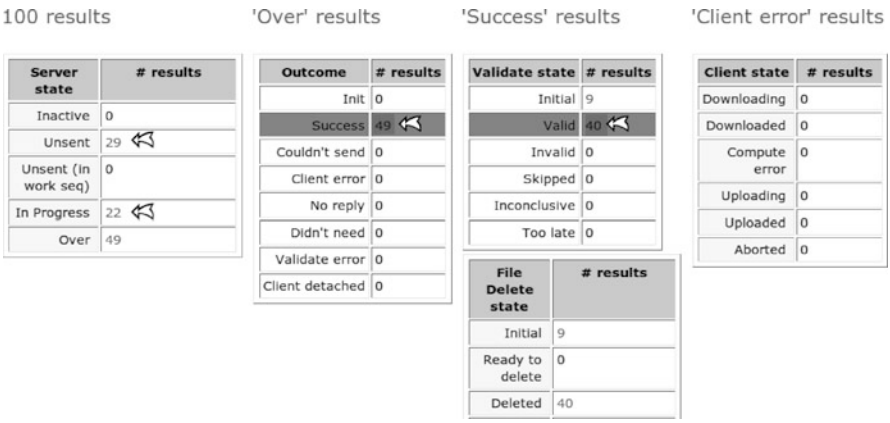
Abb. 11.14 Bearbeitung der Arbeitspakete durch Kreiszahl@home

Abb. 11.15 Ansicht der Datenbanktabelle `results_pi` mit bisher **2800** abgespeicherten Berechnungsergebnissen



	id	iterations	value	description
	1	130	3.230769	[BLOB - 0B]
	2	130	3.169231	[BLOB - 0B]
	3	130	3.076923	[BLOB - 0B]
	4	130	3.2	[BLOB - 0B]
	5	130	3.230769	[BLOB - 0B]
	6	130	2.861538	[BLOB - 0B]
	7	130	2.923077	[BLOB - 0B]
	8	130	2.953846	[BLOB - 0B]
	9	130	3.230769	[BLOB - 0B]
	10	130	3.046154	[BLOB - 0B]
	11	130	3.476923	[BLOB - 0B]
	12	130	3.107692	[BLOB - 0B]

Abb. 11.16 Ansicht der Datenbanktabelle `results_pi` mit der Auflistung der einzelnen Iterationen und den dazugehörigen Berechnungsergebnissen. Beispielsweise hat das Berechnungsergebnis eins **130** Iterationen verwendet, und die vierte Berechnung hat **3.2** als Ergebnis der Monte-Carlo-Simulation



100 results

Server state	# results
Inactive	0
Unsent	29
Unsent (in work seq)	0
In Progress	22
Over	49

'Over' results

Outcome	# results
Init	0
Success	49
Couldn't send	0
Client error	0
No reply	0
Didn't need	0
Validate error	0
Client detached	0

'Success' results

Validate state	# results
Initial	9
Valid	40
Invalid	0
Skipped	0
Inconclusive	0
Too late	0

File Delete state	# results
Initial	9
Ready to delete	0
Deleted	40

'Client error' results

Client state	# results
Downloading	0
Downloaded	0
Compute error	0
Uploading	0
Uploaded	0
Aborted	0

Abb. 11.17 Während der Arbeitspaketverteilung an die Teilnehmer eines Projekts kann über die Administrationsseite zu jedem Zeitpunkt der aktuelle Bearbeitungszustand ermittelt werden. Folgender Status wird gezeigt: **29** Arbeitspakete wurden noch nicht an Teilnehmer versendet, **22** befinden sich zur Zeit in Bearbeitung, **49** wurden bisher an den Projektserver zurückgesendet und **40** wurden schon validiert

11.4 Visualisierung der Ergebnisse

Die im vorherigen Abschnitt berechneten Arbeitspakete befinden sich nun in der Datenbanktabelle `results_pi`. Zu diesem Zeitpunkt sind 10000 Ergebnisse in dieser Tabelle abgespeichert, welche in den nächsten Abschnitten ausgelesen und visualisiert werden.

11.4.1 Ergebnisse aufarbeiten

Die 10000 Einzelergebnisse können mit Hilfe vieler verschiedener Programmiersprachen abgefragt werden; einzig eine Abfrage an die MySQL-Datenbank muss irgendwie möglich sein. In diesem Beispiel wird die Skript-Sprache PHP (PHP: Hypertext Preprocessor) genutzt. Mit Hilfe des Skripts aus Listing 11.19 lesen wir die Ergebnisse aus und erstellen darauf eine CSV-Datei (Comma-separated values, CSV). Zu Beginn wird eine Verbindung zum MySQL-Server aufgebaut und die relevante Datenbank für die Bearbeitung ausgewählt. In der `while`-Schleife werden die einzelnen Iterationswerte erfragt. Dabei wird jeder Iterationswert nur einmal ermittelt, so dass wir eine Liste der unterschiedlichen Iterationswerte erhalten. Diese Iterationswerte werden in der foreach-Schleife der Reihe nach abgearbeitet und für jeden einzelnen Iterationswert werden die Summe, die Anzahl der Einträge, das Maximum und das Minimum der Berechnungsergebnisse ermittelt.

Listing 11.19 PHP-Skript zum Auslesen der Berechnungsergebnisse von Kreiszahl@home

```
#!/usr/bin/php
<?php
$dbRes = mysql_connect(
  "localhost",
  "root",
  "boincadm"
);
mysql_select_db("tah");

$sql = "SELECT DISTINCT iterations FROM results_pi
          ORDER BY iterations ASC";
$result = mysql_query($sql);
$iterations = array();
while($row = mysql_fetch_assoc($result)) {
  $iterations[] = $row['iterations'];
}

echo "Iterationen;Durchschnitt;Max;Min;\n";
foreach($iterations AS $it) {
  $sql =
    "SELECT
      SUM(value) AS sum_values,
      COUNT(value) as count_values,
      MAX(value) AS max_values,
      MIN(value) AS min_values FROM
        results_pi WHERE iterations=".$it;
  $result = mysql_query($sql);
  $row = mysql_fetch_assoc($result);
  echo  $it.";".
       ($row['sum_values']/$row['count_values']).";".
```

```
        $row['max_values'].";".$row['min_values'].";\n";
}

mysql_close($dbRes);
?>
```

Diese Werte werden direkt in einem Terminal ausgeben und in eine CSV-Datei umgeleitet. Listing 11.20 listet eine beispielhafte CSV-Datei auf, die das Ergebnis nach der Ausführung mit dem PHP-Skript enthält. Die erste Zeile enthält dabei die Beschreibung beziehungsweise Namen der jeweiligen Werte in den Spalten, getrennt durch Semikolons. Die darauf folgenden Zeilen enthalten die jeweiligen Ergebnisse der SQL-Abfrage aus dem vorherigen Listing 11.19.

Listing 11.20 Inhalt der CSV-Datei für das Importieren in eine externe Applikation

```
Iterationen;Durchschnitt;Max;Min;
10;3.252;4;2;
20;3.11;3.8;2.4;
30;3.12933333;3.6;2.533333;
40;3.191;3.7;2.7;
50;3.1984;3.68;2.56;
60;3.15933333;3.666667;2.666667;
70;3.16571424;3.542857;2.571429;
80;3.133;3.6;2.65;
90;3.13955558;3.6;2.533333;
100;3.1436;3.6;2.76;
...
...
```

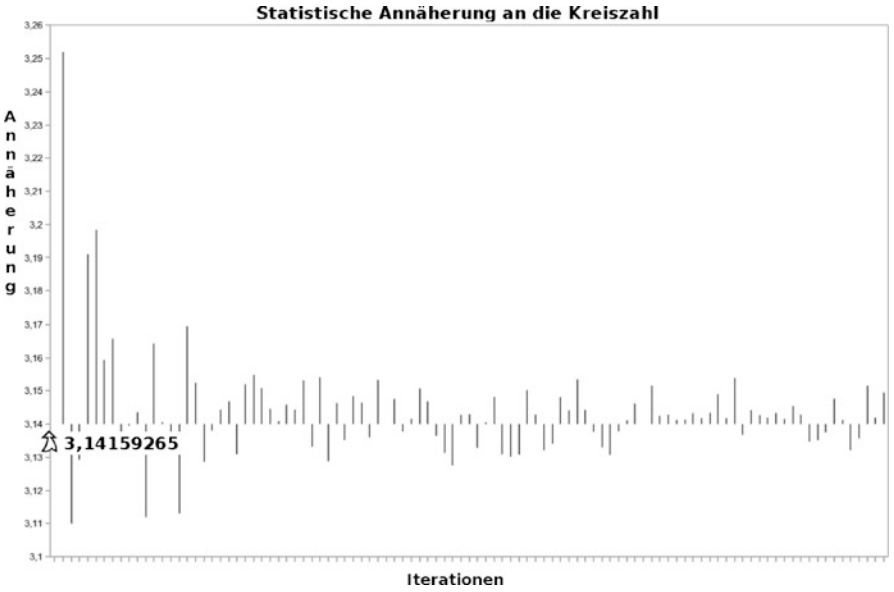


Abb. 11.18 Ergebnisvisualisierung der statistischen Annäherung an die Kreiszahl π. Die Abszisse beschreibt die Anzahl der Iterationen von 10 bis 1000 je Durchführung des Monte-Carlo-Algorithmus, und die Ordinate liefert den Wert der statistischen Annäherung an den Wert π

Diese Datei kann daraufhin in andere Applikationen importiert werden. Wir verwenden in diesem Beispiel OpenOffice [156] und erstellen uns aus den Daten ein anschauliches Diagramm, wie es Abb. 11.18 enthält.

11.4.2 Ergebnisse deuten und verstehen

Abbildung 11.18 visualisiert die berechneten statistischen Ergebnisse der Durchführung des Monte-Carlo-Algorithmus mit anwachsenden Iterationen. Es ist anscheinend so, dass ein höherer Wert der Iterationen auch eine bessere statistische Annäherung an π liefert. Zu Beginn werden nur zehn Iterationen verwendet, was eine deutliche Abweichung von dem Wert von π zur Folge hat. In den nachfolgenden Iterationen wird dieser Wert jeweils um 10 inkrementiert und es zeigt sich, dass die einzelnen Ergebnisse sich etwa zwischen $\approx 3{,}136$ und $\approx 3{,}151$ einpendeln.

Kapitel 12
Filmsequenzen bearbeiten

Action!

Wenn Sie sich Erleichterung bei der Erstellung eines BOINC-Projekts wünschen, so hilft Ihnen dieses Kapitel weiter. Wir beschreiben ein Starter-Paket, mit dem es jedem ermöglicht wird, schnell und einfach ein voll funktionsfähiges BOINC-Projekt zu erstellen, inklusive einer wissenschaftlichen Applikation. Diese Applikation dient der Bearbeitung von Filmsequenzen, wobei verschiedene Algorithmen der Bildbearbeitung zum Einsatz kommen. Sie müssen nichts weiter machen, als ein paar einfache Befehle auszuführen, und können schnell Ihre Urlaubsvideos in einen wie mit Ölfarben gemalten Film verwandeln.

12.1 Wissenschaftliche Applikation

Dieses Kapitel beschreibt ein BOINC-Projekt zur Modifikation einer frei gewählten Filmsequenz. Die Ausführung für die Modifikation kann durch die Verwendung von unterschiedlichen Eingabeparametern an die eigenen Bedürfnisse angepasst werden. Für den einfachen Fall stehen fünf Arten der Modifikation zur Verfügung. Diese werden im nachfolgenden Unterabschnitt, an der entsprechenden Stelle innerhalb des Abschnitts zum Quelltext der wissenschaftlichen Applikation, erläutert. Es handelt sich dabei um Standardmethoden der Bildverarbeitung. Der Ablauf der wissenschaftlichen Applikation erfolgt in der folgenden Reihenfolge (Kurzfassung): *die Applikation wird gestartet, die Laufzeitparameter und ein ZIP-Archiv mit den zu bearbeitenden Filmsequenzen werden geöffnet und die einzelnen Filmsequenzen nacheinander aus dem Archiv extrahiert und durch die Bildbearbeitungsmethoden modifiziert. Daraufhin werden die Ergebnisse in einem dafür vorgesehenen ZIP-Archiv gespeichert und die Ausführung der wissenschaftlichen Applikation beendet.* Die Laufzeitparameter werden durch eine XML-Datei beschrieben (s. Listing 12.2). Das komplette Starter-Paket, inklusive aller Skripts und der Quellen der wissenschaftlichen Applikation, kann auf der Webseite zu diesem Buch heruntergeladen werden [93]. Nach dem Entpacken sollten Sie die Ansicht aus Abb. 12.1 vorfinden.

C. B. Ries, *BOINC*, Xpert.press,
DOI 10.1007/978-3-642-23382-8_12, © Springer-Verlag Berlin Heidelberg 2012

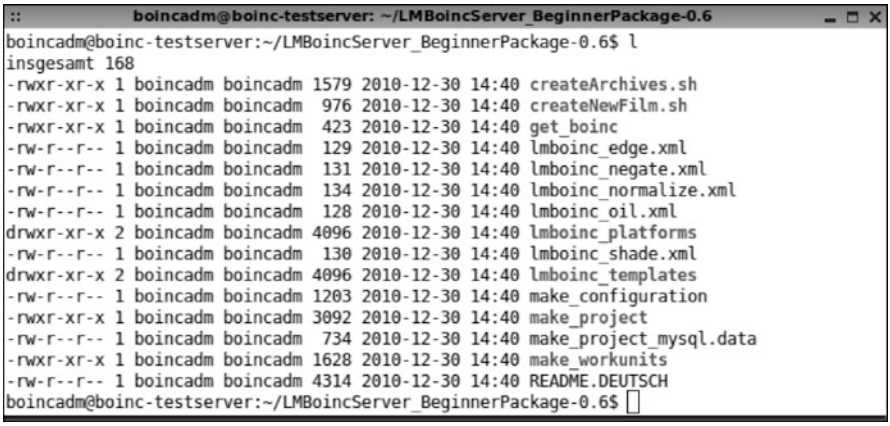


Abb. 12.1 Dateien aus dem Starterpaket des BOINC-Projekts *lmboinc*. Das Paket beinhaltet Skripts für das Erstellen eines neues BOINC-Projekts; diese übernehmen das Einstellen wichtiger Konfigurationen, um das BOINC-Projekt direkt starten zu können. Die erforderlichen Schritte für den Umgang mit dem Starterpaket sind in der `README.DEUTSCH` aufgelistet

12.1.1 Glossar für lmboinc

Zu Beginn definieren wir uns ein Glossar, um ein Nachschlagewerk für bestimmte Begrifflichkeiten und Wortverwendungen in diesem Kapitel zu besitzen.

Projekt Es ist das noch zu erstellende oder erstellte BOINC-Projekt gemeint, welches in diesem Kapitel beschrieben wird.

Quellversion Beschreibt den C/C++-Quelltext der wissenschaftlichen Applikation, welcher in Listing 12.1 aufgeführt ist.

lmboinc Beschreibt die ausführbare Version der Quellversion, dabei wird kein Unterschied zwischen verschiedenen Plattformen oder Betriebssystemen gemacht. Die Verwendung ist stets *kursiv*.

Starterpaket Dieser Begriff beschreibt das Paket mit allen darin enthaltenen Skripts, Konfigurationsdateien und vorkompilierten Versionen von *lmboinc*.

Eingabearchiv Dabei handelt es sich um ein ZIP-Archiv, welches die einzelnen Filmsequenzen enthält. In diesem Archiv werden die Rohdaten vorgehalten, die zu den einzelnen Teilnehmern am Projekt gesendet, dort durch die Bildverarbeitungsroutinen bearbeitet und dann zurück an das Projekt gesendet werden. Innerhalb der Quellversion wird dieses Archiv durch die Konstante `INPUT_FILE` beschrieben; physikalisch hat es den Archivnamen `archive_in.zip`.

Ergebnisarchiv Es handelt sich dabei um ein ZIP-Archiv, das die Ergebnisse der durch die Bildverarbeitungsroutinen modifizierten Filmsequenzen enthält. Innerhalb der Quellversion wird dieses Archiv durch die Konstante `OUTPUT_FILE` beschrieben; physikalisch hat es den Archivnamen `archive_out.zip`.

Bildverarbeitung Meint die digitale Bildverarbeitung in Bezug auf zweidimensionale Bilder [39].

12.1.2 Aufbau und Programmlogik

lmboinc nutzt die Programmbibliothek `libzip` [116] für Linux, leider steht diese nicht als Windows-Portierung zur Verfügung. Aus diesem Grund fokussiert sich dieses Kapitel hauptsächlich auf die Nutzung einer Linux-Installation, der Linux-Systemeigenschaften und Systemtechniken. Für Entwickler und Programmierer, die sich unter Windows zu Hause fühlen, sollte es möglich sein, eine Portierung der hier gezeigten Implementierung umzusetzen [107, 175].

Die Dateien in dem Eingabearchiv werden nacheinander abgearbeitet, dabei entspricht die Reihenfolge der alphabetischen Ordnung der Dateinamen innerhalb des ZIP-Archivs. Beim Bearbeiten der Bilder werden diese extrahiert und auf die Festplatte geschrieben. Die nun physikalisch vorhandenen Dateien werden im nächsten Schritt mit Bildverarbeitungsroutinen modifiziert und der Name der jeweiligen Datei in einem C++-Container abgelegt. Bei der Erstellung einer Checkpoint-Datei werden die physikalisch vorhandenen Dateien zum Ergebnisarchiv hinzugefügt und die nun noch auf der Festplatte vorhandenen Dateien, mit Hilfe der im C++-Container gespeicherten Dateinamen, von der Festplatte gelöscht. So ist die Erstellung einer Checkpoint-Datei und das Hinzufügen von weiteren Ergebnisbildern zum Ergebnisarchiv in einem Arbeitsgang möglich. Dieses Vorgehen entspricht dem Vorschlag aus Abschn. 11.1.2 und wurde auch in *lmboinc* konsequent umgesetzt.

Das nachfolgende Listing 12.1 enthält die Quellversion. Einzelne Bereiche werden gesondert beschrieben. Besonderer Wert wird auf die Parametrisierung des Laufzeitverhaltens und das Erstellen der Checkpoint-Datei gelegt.

Listing 12.1 Quellversion zur Modifikation von Filmsequenzen mit Standardmethoden der digitalen Bildverarbeitung [39]

```
#define CHECKPOINT_FILE       "lmboinc.chk"
#define CONFIGURATION_FILE    "lmboinc.xml"
#define INPUT_FILE            "archive_in.zip"
#define OUTPUT_FILE           "archive_out.zip"
```

lmboinc benötigt mindestens zwei Dateien, um die Ausführung zu beginnen: (1) `CONFIGURATION_FILE` und (2) `INPUT_FILE`. In der ersten Datei – der Konfigurationsdatei – stehen die Laufzeitparameter; die zweite Datei enthält die zu bearbeitenden Filmsequenzen. Nach erfolgreicher Ausführung sollte mindestens eine weitere Datei, nämlich `OUTPUT_FILE`, erstellt worden sein. Diese Datei enthält dann das Ergebnis der Bildverarbeitung. Je nach Dauer der Ausführung wird zudem die Checkpoint-Datei `CHECKPOINT_FILE` erstellt. Diese Checkpoint-Datei dient dem Abspeichern einer Startmarke, so dass bei einem Neustart von *lmboinc* nicht alle Berechnungen von vorne beginnen zu müssen, falls die Ausführung zwischenzeitlich pausiert oder beendet wurde. Die Konfigurationsdatei muss nicht alle möglichen Laufzeitparameter enthalten. Wir sehen später, dass bei jedem Starten von *lmboinc* immer eine Standardkonfiguration erstellt wird, falls einige Parameter nicht gesetzt sind. Die Eingabe- und Ergebnisdateien sollen ZIP-Archive sein, d. h. zum Öffnen und Erstellen muss eine Programmbibliothek für das Arbeiten mit ZIP-Archiven verwendet werden [116]. Wichtig ist, dass der Typ der Dateien ZIP-

Dateien charakterisiert. Es reicht nicht, wenn die Dateien eine ZIP-Endung (*.zip) besitzen. Ein solches Vorgehen würde einen Fehler bei der Verarbeitung verursachen!

```
typedef std::vector<std::string> FileList;

typedef struct LMBoincXML {
  /* (1) */ std::string mode;
  /* (2) */ std::string copyright;
  /* (3) */ std::string size;
  /* (4) */ std::string type;
```

Die Struktur `LMBoincXML` ist für das Einlesen der Laufzeitparameter verantwortlich. In der Quellversion sind vier Parameter definiert, um **(1)** den Modus der Bildverarbeitung zu definieren, **(2)** eine Copyright-Nachricht in die Filmsequenzen zu schreiben, **(3)** die Bildgröße der Ergebnisse der Filmsequenzen zu definieren und **(4)** festzulegen, von welchem Bildtyp die Ergebnisse der Filmsequenzen sein sollen. Das Durchführen der jeweiligen Bildverarbeitungen wird mit der Programmbibliothek *ImageMagick Magick++* durchgeführt [73, 138]. In diesem Release der Quellversion werden fünf Modi für die Bildverarbeitung unterstützt:

edge Es wird eine Kantendetektion durchgeführt. Die Kantendetektion erfolgt durch folgenden Kern [139]:

$$\begin{bmatrix} -1 & 0 & -1 \\ 0 & W & 0 \\ -1 & 0 & -1 \end{bmatrix}$$

Die Verwendung von solchen Kernen ist quasi eine Standardtechnik bei der digitalen Bildverarbeitung. Dabei wird ein Kern gewählt, hier ein 3×3-Kern, der auf jedes einzelne Pixel eines Quellbildes gelegt wird. Das aktuell ausgewählte Pixel wird dabei vom Arbeitspunkt (hier W) eines Kerns überdeckt. Daraufhin wird eine Faltung durchgeführt und der Wert im Ergebnisbild an denselben Koordinaten, die auch im Eingangsbild verwendet werden, eingetragen [21]. Das Ergebnisbild ist unabhängig vom Eingangsbild, das heißt die Faltungen werden auf dem Eingangsbild durchgeführt, das Ergebnisbild erhält nur die Ergebnisse der einzelnen Faltungen. Für den äußeren Rand wird eine virtuelle Verlängerung zu der gegenüberliegenden Kante vorgenommen, dadurch kann auch der äußere Rand eines Eingangsbilds in die Bearbeitung einfließen.

oil Das Bild erhält den Eindruck eines Ölgemäldes.

negate Das Bild erhält den Effekt eines Negatives.

shape Das Bild wird durch eine Gauß-Filterung geschärft. In diesem Verfahren wird wie bei der Kantendetektion ein bestimmter Filterkern über das Bild gefahren und die einzelnen Ergebnisse der Faltung im Ergebnisbild abgespeichert.

normalize Das Farbbild wird normalisiert, sprich alle Pixel werden skaliert, so dass diese passend zwischen den minimalen und maximalen Farbbildpunkten des Eingangsbildes liegen. Dieser Modus lässt das Ergebnisbild im Vergleich zum Eingangsbild farblich intensiver aussehen.

Ergebnisbilder einiger Eingangsbilder, die durch drei dieser Modi modifiziert wurden, finden Sie in Abb. 12.4.

```
  LMBoincXML() {
    init();
  }

  inline void init() {
    this->mode = "edge";
    this->copyright = "Christian Benjamin Ries";
    this->size = "320x240";
    this->type = "RGBA";
  }
```

Der Struktur-Konstruktor ruft die Member-Funktion zum Initialisieren der Member-Variablen auf, so dass zu Beginn der Ausführung von *lmboinc* eine Standardkonfiguration erstellt wird. In diesem Fall wird eine Kantendetektion als Standard festgelegt, die Ergebnisbildgröße auf eine Weite von 320 Pixel und die Höhe auf 240 Pixel gesetzt. Das Ausgangsbildformat ist RGBA (Rot, Grün, Blau, Alpha), das heißt, das Ergebnisbild besitzt vier Kanäle, um die Farbinformationen abzuspeichern [183, RGBA color space].

```
  int parse(FILE *lmBoincXMLFile) {
    init();

    MIOFILE xmlMIOFile;
    xmlMIOFile.init_file(lmBoincXMLFile);
    XML_PARSER p(&xmlMIOFile);

    if(!p.parse_start("lmboinc")) {
      fprintf(stderr, "lmboinc.xml is not formatted correctly\n");
      return ERR_XML_PARSE;
    }

    bool is_tag = false;
    char tag[128] = {'\0'};
    while (!p.get(tag, sizeof(tag), is_tag)) {
      if (!is_tag) {
        fprintf(stderr, "unexpected text in lmboinc.xml: %s\n", tag);
        continue;
      }
      if (!strcmp(tag, "/lmboinc")) return 0;
      if (p.parse_string(tag, "mode", this->mode)) continue;
      if (p.parse_string(tag, "copyright", this->copyright)) continue;
      if (p.parse_string(tag, "size", this->size)) continue;
      if (p.parse_string(tag, "type", this->type)) continue;
    }
    return 0;
  }
} LMBoincXML;
```

Die Member-Funktion `parse()` dient dem Einlesen und dem späteren Bereitstellen der Laufzeitparameter. Für die Bereitstellung werden die oben erläuterten öffentlichen Member-Variablen genutzt. Die Konfiguration wird in einem XML-Baum bereitgestellt und die Informationen können mit Funktionen aus der BOINC-API aus Abschn. 7.3.8 verarbeitet werden, wie dies hier exemplarisch durchgeführt wird.

```
bool initLMBoincXML(LMBoincXML &configuration) {
  std::string xmlFilePath;
  boinc_resolve_filename_s(CONFIGURATION_FILE, xmlFilePath);

  FILE *xmlFile = boinc_fopen(xmlFilePath.c_str(), "r");
```

```
  if(xmlFile == NULL) {
    return false;
  }
  return (configuration.parse(xmlFile)?false:true);
}
```

Diese Funktion öffnet die Konfigurationsdatei und liest, mit Hilfe der zuvor erwähnten Member-Funktion, die Laufzeitinformationen ein.

```
void removeOldFiles(FileList *files) {
  if(files != NULL) {
    FileList::iterator it = files->begin();
    for( ; it != files->end(); it++ ) {
      boinc_delete_file(it->c_str());
    }
    files->clear();
  }
}
```

Diese Funktion erwartet einen Zeiger auf einen Container mit darin abgelegten Dateinamen von Dateien einzelner Filmsequenzen und löscht die physikalischen Dateien von der Festplatte. Der Container wird nach der Ausführung geleert. Bei den Dateien handelt es sich um die Ergebnisbilder der Bildverarbeitung, die nach dem Hinzufügen zum Ergebnisarchiv nicht mehr benötigt werden.

```
void doCheckpoint(FileList *files,
                  int countFilesDone,
                  int check=true) {
  if(check) {
    if (!boinc_time_to_checkpoint()) return;
  }

  std::string checkpointPath;
  FILE *checkpointFile = NULL;
  std::string outputArchivePath;
  struct zip *zipFileOut = NULL;
  char zipError[512] = {'\0'};
  int returnValue = 0;

  // Den Pfad im Dateisystem zur Archiv-Datei ermitteln und oeffnen.
  boinc_resolve_filename_s(OUTPUT_FILE, outputArchivePath);
  zipFileOut = zip_open(outputArchivePath.c_str(), ZIP_CREATE, NULL);
  if (zipFileOut == NULL) {
    zip_error_to_str(zipError, 512, 0, errno);
    std::cerr << "zip failed: " << zipError << std::endl;
    boinc_checkpoint_completed();
    boinc_finish(-1);
  }

  // Die Dateien seit dem Start der Anwendung oder seit dem
  // letzten Erstellen einer checkpoint-Datei zum Archiv
  // hinzufuegen.
  FileList::iterator it = files->begin();
  for( ; it != files->end(); it++ ) {
    struct zip_source *s = NULL;
    if( (s=zip_source_file(zipFileOut, it->c_str(), 0, 0)) == NULL
        || zip_add(zipFileOut, it->c_str(), s) < 0)
    {
      zip_source_free(s);
      std::cout << "zip failed: error adding file "
                << zip_strerror(zipFileOut) << std::endl;
    }
  }
  zip_close(zipFileOut);
```

```
  // Den Pfad im Dateisystem zur checkpoint-Datei ermitteln und
  // die aktuellen Anzahl der bisher bearbeiteten Dateien fuer
  // eine spaetere Verwendung speichern.
  boinc_resolve_filename_s(CHECKPOINT_FILE, checkpointPath);
  checkpointFile = boinc_fopen(checkpointPath.c_str(), "w+");
  if (checkpointFile != NULL) {
    returnValue = fprintf(checkpointFile, "%d\n", countFilesDone);
  }
  fclose(checkpointFile);

  // Die hinzugefuegten Dateien von der Festplatte loeschen.
  removeOldFiles(files);

  boinc_checkpoint_completed();
}
```

Die Funktion `doCheckpoint()` erstellt eine Checkpoint-Datei, es wird ein einziger Zahlenwert abgespeichert. Dieser Zahlenwert beschreibt die Anzahl der bisher bearbeiteten Filmsequenzen. `doCheckpoint()` sollte in einem atomaren Bereich ausgeführt werden und dagegen abgesichert werden, dass eventuell ein Zahlenwert abgespeichert wird, der nicht der realen Anzahl der Bildverarbeitungen entspricht. Dieses Problem wurde bereits in Abschn. 11.1.2 behandelt. In eine Checkpoint-Datei gehören nur die nötigsten Informationen, um aus diesen Informationen zu jedem Zeitpunkt der Berechnung eine Fortsetzung zu ermöglichen. Zeile 85 öffnet das ZIP-Archiv für die Ergebnisabspeicherung; wenn diese Datei nicht existiert, wird sie neu erstellt und verwendet. Zeile 97 durchläuft die Liste der bisher bearbeiteten Filmsequenzen und fügt die Dateinamen der physikalisch vorhandenen Dateien zum Ergebnisarchiv hinzu. In Zeile 112 wird letztendlich die Anzahl der bisher abgearbeiteten Filmsequenzen in der Checkpoint-Datei abgespeichert. Falls noch keine Filmsequenzen bearbeitet wurden, so wird die vorher erwähnte for-Schleife nicht verarbeitet und es wird der Wert 0 in die Checkpoint-Datei hinzugefügt.

```
int main(int argc, char **argv) {
    // Magick++
    Magick::InitializeMagick(*argv);
```

Der Start von *lmboinc* und die Initialisierung des ImageMagick-Frameworks.

```
    // Das BOINC-Framework fuer die Ausfuehrung
    // dieser Applikation initialisieren.
    int returnValue = boinc_init();
    if (returnValue) {
      std::cerr << boinc_msg_prefix()
                << " boinc_init returned "
                << returnValue << std::endl;
      exit(returnValue);
    }
```

Die Initialisierung des BOINC-Frameworks erfolgt ohne Umschweife und mit den Standardlaufzeitoptionen für BOINC, die innerhalb des BOINC-Frameworks definiert sind (s. Abschn. 7.3.2). Bei einem Fehler beenden wir die Ausführung von *lmboinc*.

```
    LMBoincXML lmBoincConfiguration;
    if(!initLMBoincXML(lmBoincConfiguration)) {
      // no message
    }
```

lmboinc kann ein vom Entwickler definiertes Ausführungsverhalten erhalten. Hier werden die spezifischen Konfigurationsparameter eingelesen und für eine spätere Verwendung vorgehalten. An dieser Stelle kann die Ausführung beendet werden, wenn zum Beispiel keine Parameterdatei vorhanden ist. Die Struktur wird in diesem Beispiel allerdings mit Standardwerten gesetzt und daher muss *lmboinc* nicht zwangsweise beendet werden. Die Konfigurationsparameter sind nach diesem Schritt in der Strukturvariablen `lmBoincConfiguration` zur späteren Verwendung abgespeichert.

```
    // Dateinamen durch den BOINC-Client ausloesen und
    // das Archiv zum Lesen oeffnen.
    std::string inputArchivePath;
    struct zip *zipFileIn = NULL;
    char zipError[512] = {'\0'};

    boinc_resolve_filename_s(INPUT_FILE, inputArchivePath);
    zipFileIn = zip_open(inputArchivePath.c_str(),
                         ZIP_CHECKCONS, NULL);
    if (zipFileIn == NULL) {
      zip_error_to_str(zipError, 512, 0, errno);
      std::cerr << "zip failed: " << zipError << std::endl;
      boinc_finish(-1);
    }
```

Der Dateipfad zur Eingabedatei wird ermittelt und daraufhin wird versucht, das ZIP-Archiv zu öffnen. Bei einem Fehler wird die Ausführung von *lmboinc* abgebrochen. Es ist zwingend erforderlich, dass eine Eingabedatei vorhanden ist, denn ohne Filmsequenzen ergibt es keinen Sinn, Berechnungen durchzuführen.

```
    // Anzahl der im Archiv enthaltenen Dateien ermitteln, wichtig
    // fuer die Berechnung der aktuellen Bearbeitung in Prozent.
    int countFilesIn = zip_get_num_files(zipFileIn);
    int countFilesDone = 0;
```

Es wird die Anzahl der zu bearbeitenden Filmsequenzen – die sich in der ZIP-Datei befinden – ermittelt, zudem wird ein Zähler gesetzt, um die Anzahl der schon bearbeiteten Filmsequenzen während der Verarbeitung jederzeit griffbereit zu haben.

```
    // Eventuell vorhandene checkpoint-Datei oeffnen und
    // den enthaltenen Zahlenwert auslesen. Dieser Zahlenwert
    // gibt Auskunft ueber die bisher bearbeiteten Bilddateien aus
    // dem Archiv.
    std::string checkpointPath;
    FILE *checkpointFile = NULL;
    boinc_resolve_filename_s(CHECKPOINT_FILE, checkpointPath);
    checkpointFile = boinc_fopen(checkpointPath.c_str(), "r");
    if (checkpointFile != NULL) {
      returnValue = fscanf(checkpointFile, "%d", &countFilesDone);
      if (returnValue <= 0) {
         std::cerr << "checkpoint failed: Could not read in "
                   << "current check point." << std::endl;
         countFilesDone = 0;
      }
      fclose(checkpointFile);
    }

    if (countFilesDone == countFilesIn) {
      boinc_fraction_done(1);
      boinc_finish(BOINC_SUCCESS);
    }
```

Eine eventuell vorhandene Checkpoint-Datei wird ausgelesen und der Wert beschreibt bisher zum aktuellen Zeitpunkt bearbeitete Filmsequenzen. Der Zahlenwert wird in `countFilesDone` vorgehalten. Sollte dieser Wert gleich der Anzahl der Filmsequenzen im ZIP-Archiv sein, so sind keine weiteren Filmsequenzen für die Bearbeitung vorhanden und das Ende der Ausführung ist erreicht. In diesem Fall wird die Ausführung beendet und mit dem Funktionsaufruf `boinc_fraction_done(1)` wird dem BOINC-Client eine fertige Ausführung signalisiert, so dass der BOINC-Teilnehmer dies im BOINC-Manager – wenn dieser gestartet ist – sehen kann. Daraufhin wird *lmboinc* mit dem Wert der Beendigung `BOINC_SUCCESS` abgeschlossen.

```
180     /*
         * Hier faengt die Magie 'Magick++' an :-)
         */
        FileList fileList;

185     // Alle Dateien aus dem Archiv lesen.
        while (countFilesDone < countFilesIn) {

          // Eine Datei aus dem Archiv selektieren.
          struct zip_file *zipReader = zip_fopen_index(zipFileIn,
190                                 countFilesDone, ZIP_FL_UNCHANGED);
          if (zipReader == NULL) {
            zip_error_to_str(zipError, 512, 0, errno);
            std::cerr << "zip failed: " << zipError << std::endl;
            boinc_finish(-1);
195       }

          // Informationen ueber eine Datei im zip-Archiv ermitteln.
          struct zip_stat zipFileInformation;
          zip_stat_index(zipFileIn, countFilesDone,
200                       ZIP_FL_UNCHANGED, &zipFileInformation);

          // Datei aus dem Archiv auslesen.
          char *imageBuffer = new char[zipFileInformation.size];
          zip_fread(zipReader, imageBuffer, zipFileInformation.size);
205
          // Die Datei in ein Magick-Objekt umwandeln
          // und die Bilddaten manipulieren.
          Magick::Blob blob((void*)imageBuffer, zipFileInformation.size);
          Magick::Image img;
210       img.size(lmBoincConfiguration.size);
          img.magick(lmBoincConfiguration.type);
          img.read(blob);
          if (lmBoincConfiguration.mode == "edge") {
            img.edge();
215       } else if (lmBoincConfiguration.mode == "oil") {
            img.oilPaint();
          } else if (lmBoincConfiguration.mode == "shade") {
            img.shade();
          } else if (lmBoincConfiguration.mode == "negate") {
220         img.negate();
          } else if (lmBoincConfiguration.mode == "normalize") {
            img.normalize();
          } else {
            ;
225       }
          // Text zum Bild hinzufuegen.
          img.fillColor("white"); // Fill color
          img.fontPointsize(20);
          img.draw( Magick::DrawableText(5, 20,
230                 lmBoincConfiguration.copyright) );
```

```
    // Datei auf die Festplatte schreiben.
    img.write(zipFileInformation.name);
```

Zwischen den Zeilen 208 und 232 befindet sich der Zauber der Bildmanipulation mit Hilfe von unterschiedlichen Algorithmen der digitalen Bildverarbeitung. Zu Beginn wird ein Speicherbereich `imageBuffer` für jeweils eine Filmsequenz erstellt, der direkt mit den Rohdaten einer Filmsequenz gefüllt wird. Das ImageMagick-Framework stellt die Klasse `Magick::Image` für die einfache Bearbeitung von Bilddaten zur Verfügung [145]. Diese Klasse bietet einfache Methodenaufrufe, die wir in den `if-else`-Verzweigungen unterscheiden. Für das einfachere Lesen des oberen Bereichs wurden die Modus-Werte nach den Methodenaufrufen benannt. Das heißt, wenn die Konfigurationsdatei `lmboinc.xml` für den Parameter `mode` den Wert `normalize` enthält, so ist dies auch der Methodenaufruf, um diese Bildverarbeitung durch die Klasse `Magick::Image` aufzurufen.

An dieser Stelle könnten Sie annehmen, dass es vielleicht sinnvoll ist, eine direkte Verknüpfung oder ein dynamisches Binden der Konfiguration zum Methodenaufruf zu implementieren – fühlen Sie sich frei, dies umzusetzen. Dadurch würde eine Entscheidungsroutine in der Quellversion entfallen, allerdings hätten Sie dann mit Problemen der dynamischen Bindung zu kämpfen, was sich eventuell durch eine Implementierung einer Abstraktionsschicht lösen ließe. Dabei wäre die Schnittstelle zum Aufruf der Bildverarbeitung immer gleich, die darunter liegende logische Implementierung allerdings für die jeweilige Anwendung maßgeschneidert.

Der untere Bereich im oberen Listing-Bereich schreibt unseren Copyright-Text in die obere linke Ecke einer Filmsequenz, und mit `write()` wird die modifizierte Filmsequenz auf der Festplatte abgelegt. Der Dateiname der abgelegten Datei entspricht dem Namen der Filmsequenz aus dem Eingabearchiv. Dies ist im Nachhinein nicht weiter zu beachten, es sei allerdings angemerkt, dass an dieser Stelle ein wahlfreier Name beziehungsweise eine Umbenennung möglich ist.

```
    // Namen merken und bei der Erstellung des
    // Checkpunktes oder beim Ende der Programmausfuehrung
235 // die aufgefuehrten Dateien von der Festplatte loeschen.
    fileList.push_back(zipFileInformation.name);

    if (imageBuffer != NULL) {
      delete imageBuffer; imageBuffer = NULL;
240 }

    zip_fclose(zipReader);

    countFilesDone++;
245
    // Berechnung der bisher geloesten Aufgaben und
    // Uebertragung des Zahlenwertes an den BOINC-Client.
    double done = countFilesDone/(double)countFilesIn;
    boinc_fraction_done(done);
```

Im nächsten Schritt wird der Dateiname der auf der Festplatte abgespeicherten Filmsequenz in den C++-Container `fileList` abgelegt, so dass diese Filmsequenz nach einer Checkpoint-Erstellung von der Festplatte gelöscht werden kann. Das ZIP-Archiv wird nach jedem einzelnen Bearbeitungsschritt geschlossen und bei einem weiteren Durchlauf wieder geöffnet. Die Berechnung am Ende des Listing-

Bereichs erstellt einen neuen Fraction-Done-Wert, der nachfolgend zum BOINC-Client gesendet wird.

```
250         doCheckpoint(&fileList, countFilesDone);
        }
        doCheckpoint(&fileList, countFilesDone, false);
        /*
         * Och, die Zaubershow ist schon vorbei?
255      */

        boinc_fraction_done(1);
        boinc_finish(0);
}
```

Am Ende eines Schleifendurchlaufs in Zeile 250 wird versucht, eine Checkpoint-Datei zu erstellen. Sollte das Zeitintervall für die Erstellung noch nicht überschritten sein, so wird die Ausführung am Anfang der Schleife fortgesetzt. Wenn keine weiteren Filmsequenzen für eine Bearbeitung vorliegen, wird in jedem Fall eine Checkpoint-Datei und damit einhergehend alle Ergebnisfilmsequenzen im Ergebnisarchiv abgespeichert. Dies ermöglicht der dritte Parameter der Funktion `doCheckpoint()` in Zeile 252. Durch *false* wird angegeben, dass keine Prüfung des Checkpoint-Intervalls ausgeführt werden soll und daher eine Ausführung der kompletten Routine zum Erstellen eines Checkpoints erzwungen wird.

12.1.3 Eingabeparameter und Ergebnisdatei

Die Parameterdatei aus Listing 12.2 kann um weitere Parameter erweitert werden, zudem ist es erforderlich, dass `lmboinc` in einem solchen Fall an neuere Umstände angepasst wird. Es können in der aktuellen Version vier Einstellungen vorgenommen werden:

1. Definition von fünf verschiedenen Algorithmen der digitalen Bildverarbeitung,
2. Angabe der Pixelwerte für die Bildweite und Bildhöhe,
3. Definition des Ausgabeformats der einzelnen Filmsequenzen (z. B. PNG, JPEG) und
4. eine Zeichenkette, die oben links zu einer Filmsequenz hinzugefügt wird.

Die Werte für den Modus, die Bildgröße und der Dateityp einer Filmsequenz sind durch die Möglichkeiten von ImageMagick vorgegeben [145]. Die Bildgröße kann durch eine Zeichenkette beschrieben werden:

```
<width>x<height>{+-}<xoffset>{+-}<yoffset>{\%}{!}{<}{>}
```

Die einzelnen Platzhalter <XXX> müssen dabei durch eine Reihe von Ganzzahlen ersetzt werden. In der ImageMagick API wird von der Verwendung dieser Beschreibung abgeraten, denn diese kann einen Programmabsturz zur Folge haben. Durch die Angabe des Dateityps einer Filmsequenz kann das Abspeicherungsformat definiert werden, zum Beispiel GIF [129] oder PNG [61] oder RGBA, das vom Bitmap-Format verwendet wird [63].

Listing 12.2 XML-Definition für die Beschreibung der Laufzeitparameter der Bildverarbeitungsfunktionen. Die Beschreibung erfolgt größtenteils in der EBNF-Notation [50]

```
MODE : edge | negate | normalize | oil | shade ;
SIZE : IMAGE_MAGICK_GEOMETRY ;
TYPE : ID ; <-- zum Beispiel 'RGBA'
// Es koennen noch mehr Zeichen verwendet werden,
// diese sind nur ein Beispiel. Es sind keine
// Vokale erlaubt.
TEXT : ('a'..'z'|'A'..'Z'|'0'..'9'|' '|'&'|';')* ;

<lmboinc>
  <mode> MODE </mode>
  <size> SIZE </size>
  <type> TYPE </type>
  <copyright> TEXT </copyright>
</lmboinc>
```

12.1.4 Eine erste Ausführung

Abbildung 12.1 zeigt die im Starterpaket enthaltenen Dateien. Weiterhin sind zwei Dateiordner `lmboinc_platforms/` und `lmboinc_templates/` zu erkennen. Der erste Ordner enthält zwei vorkompilierte Versionen der wissenschaftlichen Applikation, folglich *lmboinc*, jeweils eine Version für eine 32-Bit- oder 64-Bit-Plattform mit einer Linux-Installation. Für das Starten einer dieser Versionen müssen im Vorfeld einige kleinere Befehle durchgeführt werden. Dieser Abschnitt behandelt alle Arbeitsschritte, die zur Erstellung der benötigten Konfigurationen und Arbeitspakete durchzuführen sind, und wir werden diese einmal manuell ausführen; später werden für diese Arbeitsschritte vorgefertigte Skripts genutzt (ebenfalls im Starterpaket enthalten). Ein einfacher Versuch zum Ausführen von *lmboinc*

```
boincadm@boinc-testserver:~/LMBoincServer_BeginnerPackage-0.6/
    lmboinc_platforms$ ./lmboinc_0.1_i686-pc-linux-gnu
```

liefert uns nicht das gewünschte Ergebnis. Nach der Ausführung befinden sich zwei neue Dateien im aktuellen Ordner:

`init_data.xml`
: Standardeinstellungen und Informationen für die Verwendung des BOINC-Frameworks sind in dieser Datei enthalten. Bei der Ausführung einer wissenschaftlichen Applikation durch den BOINC-Client werden die Informationen durch den BOINC-Client überschrieben. Dieser Umstand wird protokolliert und ist in der ersten Zeile in Listing 12.3 aufgeführt.

`stderr.txt`
: Enthält Meldungen über die Ausführung, im aktuellen Fall enthält die Datei vier Zeilen. Diese weisen darauf hin, dass es während der Ausführung Probleme beim Öffnen von zwei Dateien gab. Listing 12.3 enthält die vier Meldungen.

Eingabeparameter erstellen

Die Protokollierung zeigt, dass die Applikation im standalone-Betrieb ausgeführt wird und dass die zwei benötigten Dateien `lmboinc.xml` sowie ein ZIP-Archiv

nicht geöffnet werden konnten. An dieser Stelle ist zudem eine Möglichkeit für Verbesserungen zu sehen. Die Meldung für ein fehlendes ZIP-Archiv sollte ausdrucksstärker sein und eine präzisere Beschreibung liefern. Im Fall des fehlenden ZIP-Archivs fehlt die entsprechende ZIP-Datei mit dem Dateinamen `INPUT_FILE` auf dem aktuellen Datenträger.

Listing 12.3 Fehlerprotokollierung bei der Ausführung der wissenschaftlichen Applikation `lmboinc`, wenn diese direkt gestartet wird, ohne Kommunikation zum BOINC-Client

```
16:24:04 (2532): Can't open init data file - running in standalone mode
16:24:04 (2532): could not open configuration xml file lmboinc.xml
zip failed: No error
16:24:04 (2532): called boinc_finish
```

Die fehlenden Dateien können manuell erzeugt und für die Ausführung hinzugefügt werden. Im Oberordner befinden sich vorbereitete XML-Dateien, welche die Laufzeitparameter für die Ausführung von *lmboinc* bereithalten. Der Befehl

```
$ ln -s ../lmboinc_oil.xml lmboinc.xml
```

erstellt einen symbolischen Link zur Konfigurationsdatei, mit der die Filmsequenzen durch *lmboinc* den Effekt eines Ölgemäldes erhalten. Nun fehlt uns noch ein ZIP-Archiv, das Bildsequenzen für die Eingabe bereithält. Das Big Buck Bunny-Projekt stellt einen kurzen Trailer Ihres Videos bereit, dies ist perfekt für unsere Zwecke [74]. Erstellen Sie einen Unterordner `video/` und laden Sie den Trailer in diesen Ordner:

```
$ mkdir videos/
$ wget http://video.blendertestbuilds.de/download.blender.org/peach/
    trailer_400p.ogg \
    -O videos/trailer_400p.ogg
```

Im nächsten Schritt erstellen wir die physikalischen ZIP-Archive, die für ein Arbeitspaket benötigt werden und entsprechend als Eingabe dienen. Für die nächsten Arbeitsschritte benötigen wir einen weiteren Unterordner `sequences/`:

```
$ mkdir sequences/
```

Der Befehl

```
$ ffmpeg -i videos/trailer_400p.ogg sequences/trailer_image%06d.png
```

erstellt aus dem Trailer einzelne Filmsequenzen. `ffmpeg` ist ein Werkzeug zum Aufnehmen, Konvertieren und Senden/Empfangen von Audio- und Videodaten [126]. Dabei werden die einzelnen Filmsequenzen mit einer eindeutigen Identifikationsnummer (ID) benannt, wofür die Formatierungssequenz `%06d` verantwortlich ist. In diesem Fall kann die maximale ID 999999 sein, dies hängt von der Zahl in der Formatierungssequenz ab. Je nach Länge eines Videos muss diese Beschreibung angepasst werden, sprich bei längeren Videodateien sollte ein größerer Wert als 6 gewählt werden. Die Dateiendung beschreibt das Dateiformat der Ergebnisfilmsequenz, in diesem Fall werden Filmsequenzen im PNG-Format [61] erstellt. Bei dem zuvor heruntergeladenen Video werden etwa 800 Filmsequenzen erstellt. Im nächsten Schritt müssen einzelne Filmsequenzen zu einem ZIP-Archiv hinzugefügt werden. Für unsere Tests benötigen wir nur ein einziges ZIP-Archiv und wir fügen diesem 50 Filmsequenzen hinzu:

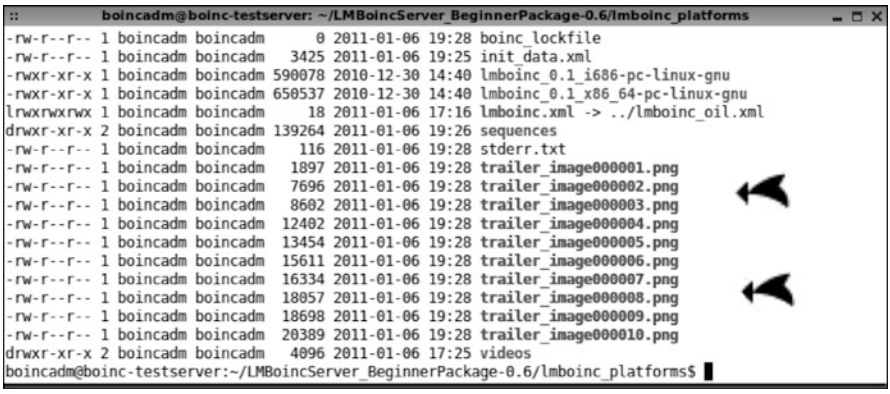


Abb. 12.2 Zwischenstand der Bearbeitung der Filmsequenzen eines Arbeitspakets im standalone-Betrieb

```
$ cd sequences/
$ zip archive_in.zip trailer_image000{001..050}.png
$ mv archive_in.zip ; cd ..
```

Erfolgreiches Ausführen von lmboinc

Nach dem Hinzufügen der zwei fehlenden Eingabedateien können wir einen weiteren Ausführungsversuch unternehmen. Wieder wird *lmboinc* durch einen einfachen Aufruf gestartet:

```
$ ./lmboinc_0.1_i686-pc-linux-gnu &
```

Das kaufmännische *Und*, am Ende der Eingabe, lässt den Prozess im Hintergrund ablaufen. Während der Ausführung werden die Bildsequenzen aus dem ZIP-Archiv entpackt. Sie sind daraufhin direkt verwendbar und liegen physikalisch auf der Festplatte vor. Abbildung 12.2 zeigt einen Snapshot, nachdem *lmboinc* gestartet wurde. Zu erkennen sind 10 bearbeitete Filmsequenzen mit den Namen `trailer_image000001.png` bis `trailer_image000010.png`. Nach dem erfolgreichen Ausführen ist ein weiteres ZIP-Archiv mit dem Namen `archive_out.zip` vorhanden. Dieses ZIP-Archiv erhält die Ergebnisse der jeweiligen Bildverarbeitungen der Filmsequenzen.

12.2 Projekteinstellungen

Das Skript zur Erstellung eines Projekts `make_project` unterstützt Sie bei den notwendigen Projekteinstellungen. Dabei übernimmt das Skript das Einstellen der erforderlichen Zugriffsrechte relevanter und für eine korrekte Ausführung benötigter Dateiordner:

```
chmod 02770 upload
chmod 02770 html/cache
chmod 02770 html/inc
chmod 02770 html/languages
chmod 02770 html/languages/compiled
chmod 02770 html/user_profile
```

Weiterhin werden Dateiordner für das Abspeichern der Berechnungsergebnisse erstellt:

```
mkdir -p sample_results/errors
```

Webadministration aktivieren

Ein BOINC-Projekt besitzt unterschiedliche Webseiten für das Bereitstellen von Informationen und die Administration eines Projekts. Die Hauptadresse unseres Projekts ist gleichzeitig die Webseite, um Informationen einzuholen. Diese Adressen müssen nicht gleich sein und können individuell eingestellt werden. Die Webseiten können individuelle Modifikationen, ganz nach den eigenen Bedürfnissen, erhalten [99]. In den aktuellen Betrachtungen werden keine Designfragen behandelt und die Webseite wird in dem Format übernommen, wie sie auch direkt aus den BOINC-Quellen geliefert wird. In diesem Abschnitt legen wir unser Augenmerk auf die Administrationsseite, da dort vor der ersten Nutzung einige Bereiche freigeschaltet werden müssen, ansonsten wird ganz stur eine Zugriffsverweigerung auftreten.

> Unable to handle request
> You must protect the admin interface with either a .htaccess file or an auto_ops() function.
> See how here [79]

Diese Problematik kann ganz individuell an die eigenen Bedürfnisse angepasst werden. Generell werden zwei Verfahren empfohlen, (1) die Sicherung der Administrationsseite durch eine `.htaccess`-Datei (engl. *hypertext access* „Hypertext-Zugriff“) [70, 71] oder (2) durch eine PHP-Funktion in der Datei `html/project/project.inc`. Im ersten Fall müssen zwei Dateien im `html/ops/`-Ordner erstellt werden:

`.htaccess`
: Beschreibt die Zugriffsmethode, definiert den Meldungstext beim Öffnen einer abgesicherten Webseite und enthält den Pfad zur Passwortdatei `.htpasswd`. Listing 12.4 liefert ein anwendbares Beispiel.

`.htpasswd`
: Enthält eine Liste von Benutzernamen mit zugehörigem Passwort, die in richtiger Kombination einen Zugriff auf eine gesperrte Webseite freigeben. Diese Datei kann mit einem vom Apache-Webserver mitgelieferten Tool erstellt werden:
`$ htpasswd -bc html/ops/.htpasswd BOINC_USERNAME BOINC_PASSWORD`.

Listing 12.4 ...

```
AuthName "lmboinc"
AuthType Basic
AuthUserFile /home/boincadm/projects/lmboinc//html/ops/.htpasswd
require valid-user
```

Die Möglichkeit mit `.htaccess` kann um weitere Zugriffstechniken wie LDAP (Lightweight Directory Access Protocol) [183, LDAP] oder eine MySQL-Datenbank-Anbindung erweitert werden [151]. Es sei erwähnt, dass die oben vorgestellte Methode eine triviale Umsetzung bedeutet und schnell Erfolg verspricht.

Wenn komplexere Sicherheitsabfragen nötig sind, so können diese ganz individuell für die eigenen Bedürfnisse umgesetzt werden. Für diesen Fall müssen Modifikationen an den PHP-Quellen der jeweiligen Webseiten vorgenommen werden, im Speziellen handelt es sich bei der Zugriffsprüfung um die Datei `html/project/project.inc`. Die Datei enthält die Funktion `auth_ops_example()`, in Revision 22890 der BOINC-Quellen ist die Implementierung aus Listing 12.5 enthalten.

Listing 12.5 Ausgangsversion zur Abfrage des Benutzerstatus mit zwei vordefinierten Abfrageroutinen, beide sind deaktiviert. `auth_ops_userid()` erwartet ein Array mit Identifikationsnummern der Projektteilnehmer, die Administrationsrechte besitzen. Diese Routine erfordert das direkte Editieren der PHP-Skripts, gilt allerdings als sehr robust, da keine externen Skripts benutzt werden müssen. Die zweite Routine `auth_ops_privilege()` fragt die Datenbanktabelle `forum_preferences` und dort die Datenspalte `special_user` ab und überprüft, ob der aktuelle Projektteilnehmer das Administrator- oder Entwicklerflag besitzt. Wenn keine der beiden Routinen implementiert sind oder der Zugriff auf die angefragten Ressourcen verweigert werden, so wird dies durch eine entsprechende Webseite signalisiert

```
function auth_ops_example() {
    // if running from cmdline, skip checks
    //
    if (!$_SERVER['REMOTE_ADDR']) {
        return;
    }

    // example: require login as a particular user (id 1 in this case)
    if (0) {
        auth_ops_userid(array(1));
        return;
    }

    // example: require that logged-in user have ADMIN or DEV flags
    // set in their forum_prefs.privilege
    //
    if (0) {
        auth_ops_privilege();
        return;
    }
}
```

Eine automatische Modifikation kann durch die Hilfe des Text-Tools *sed* (Stream Editor) [163, 164] vorgenommen werden. Dieses Tool ermöglicht das on-the-fly-Modifizieren einer Textzeile oder eines Textbereichs, durch optionale Angabe eines Text-Pattern, der Zeilen-/Zeichennummern und der Art der Modifikation (Entfernen, Ersetzen, Hinzufügen). Mit

```
$ sed -i".bak" '/^function\ auth_ops_example/,/^\}/d' project.inc
```

wird die entsprechende Funktion gelöscht. Im nächsten Schritt kann automatisch eine eigene Implementierung hinzugefügt werden. Listing 12.6 fügt eine Standardroutine aus dem vorherigen Listing 12.5 hinzu, die dem BOINC-Teilnehmer mit der Identifikationsnummer 1 Zugriff erlaubt.

Listing 12.6 Skript zum Ersetzen der Standardroutinen für die Zugriffskontrolle zur Administrationswebseite in der PHP-Datei `project.inc`. Entfernt die komplette Funktion `auth_ops_example()` und fügt eine neue Implementierung am Ende der Datei hinzu

```
sed -i".bak" '/^function\ auth_ops_example/,/^\}/d' project.inc
sed -i".bak" 's/?>//' project.inc

cat > ./.project.inc.tmp <<DELIM
// Authorize access to administrative pages.
// You can check for login, IP address, or whatever you want.
//
function auth_ops_example() {
  // example: require login as a particular user (id 1 in this case)
  auth_ops_userid(array(1));
}

?>
DELIM

cat ./.project.inc.tmp >> project.inc
rm ./.project.inc.tmp
```

Im ersten Schritt wird aus der Datei die alte Funktion und nachfolgend der schließende PHP-Tag am Ende der Datei entfernt. Als Ersatz wird am Ende die neue Funktion hinzugefügt, wobei die Implementierung die Anmeldung über die Benutzeridentifikation 1 ermöglicht, sprich in diesem konkreten Fall wird nur einem Benutzer der Zugang zur Administrationsseite gewährt. Sie können an dieser Stelle selbst entscheiden, ob Sie überhaupt Sicherheitsvorkehrungen treffen möchten, und wenn ja, welchen logischen Ablauf diese Vorkehrungen besitzen sollen. Das im Starterpaket enthaltene Skript richtet die Möglichkeit zur Anmeldung des Benutzers mit der Identifikation 1 ein und ermöglicht dadurch eine triviale und schnell umgesetzte Lösung für einen Einstieg in die Administration unseres BOINC-Projekts.

12.2.1 Konfiguration und Anwendung hinzufügen

Das Projekt benötigt die Linux-Plattformdefinitionen für die 32-Bit- und 64-Bit-Systemplattformen. Weiterhin werden die Standardschritte zum Hinzufügen von *lmboinc* durchgeführt, wie schon in Abschn. 5.6.2.1 beschrieben. Das heißt, *lmboinc* wird mit den entsprechenden Namensgebungen in den Unterordner `apps/` kopiert:

```
boincadm@boinc-testserver:~/projects/lmboinc$ tree apps/
apps/
`-- lmboinc
    |-- lmboinc_0.1_i686-pc-linux-gnu
    `-- lmboinc_0.1_x86_64-pc-linux-gnu
```

Weiterhin muss ein Assimilierer und ein Validierer zu der Projektkonfiguration hinzugefügt werden. Diese Arbeit übernimmt wieder der Stream-Editor *sed*. Die einzel-

nen Aufrufe von *sed* befinden sich im Skript zur Erstellung dieses BOINC-Projekts (siehe nachfolgendes Listing):

Listing 12.7 Fügt Standardvalidierer und -assimilierer zur Projektkonfiguration hinzu

```
sed -i".bak" '200i'"<daemon>" config.xml
sed -i".bak" '201i'"<cmd> sample_bitwise_validator -d 2 -app lmboinc </cmd>"
    config.xml
sed -i".bak" '202i'"<output> sample_bitwise_validator.log </output>" config.xml
sed -i".bak" '203i'"<pid_file> sample_bitwise_validator.pid </pid_file>" config
    .xml
sed -i".bak" '204i'"</daemon>" config.xml
sed -i".bak" '205i'"<daemon>" config.xml
sed -i".bak" '206i'"<cmd> sample_assimilator -d 2 -app lmboinc </cmd>" config.
    xml
sed -i".bak" '207i'"<output> sample_assimilator.log </output>" config.xml
sed -i".bak" '208i'"<pid_file> sample_assimilator.pid </pid_file>" config.xml
sed -i".bak" '209i'"</daemon>" config.xml
```

Es wird ein Validierer mit bitweiser Prüfung hinzugefügt und ein Assimilierer, der die Ergebnisse im Dateiordner `sample_results/` und fehlerhafte Berechnungsergebnisse in `sample_results/errors` abspeichert.

12.2.2 Templates erstellen

Für das korrekte Hinzufügen von Arbeitspaketen müssen im aktuellen Projekt zwei Template-Dateien erstellt und hinzugefügt werden. Diese hier verwendeten Template-Dateien sind in Listing 12.8 und 12.9 aufgeführt. Abbildung 12.3 zeigt den Zusammenhang der Eingabedateien und wie diese in den Template-Dateien definiert sein müssen, so dass sie korrekt durch die wissenschaftliche Applikation verwendet werden können. Es ist darauf zu achten, dass die Dateien beim Erstellen eines Arbeitspakets entsprechend der Definition in der Template-Eingabedatei angegeben werden. Im vorherigen Abschnitt wurde ein Bash-Skript zur automatischen Erstellung von Arbeitspaketen, für alle Dateien im Unterordner `archives/`, vorgestellt. Die `for`-Schleife durchläuft den Ordner und liest sequentiell alle Dateinamen ein. Daraufhin werden die Dateien in die Download-Hierarchie des Projekts `lmboinc` kopiert, dabei hilft das BOINC-Werkzeug `dir_hier_path` (s. Abschn. 5.6.2.4). Schließlich wird das BOINC-Werkzeug zur Erstellung eines BOINC-Arbeitspakets aufgerufen, das die vorher zum Projekt hinzugefügten Template-Dateien verwendet und als letzte Parameter die Namen der Template-Dateien erwartet. Bei der Verwendung ist zu beachten, dass die Pfade und Dateinamen der Parameterdateien relativ zum Download-Ordner `download/` – innerhalb des BOINC-Projektes – sind. Es ist daher zu empfehlen, dass das Skript `create_work` im obersten Ordner von `lmboinc` ausgeführt wird. Listing 12.8 enthält die Beschreibung der Parameterdateien. Dabei muss im ersten Schritt die Konfigurationsdatei `lmboinc.xml` für die Laufzeitparameter von *lmboinc* und im zweiten Schritt das ZIP-Archiv `archive_in.zip` mit den Film-Sequenzen angegeben werden.

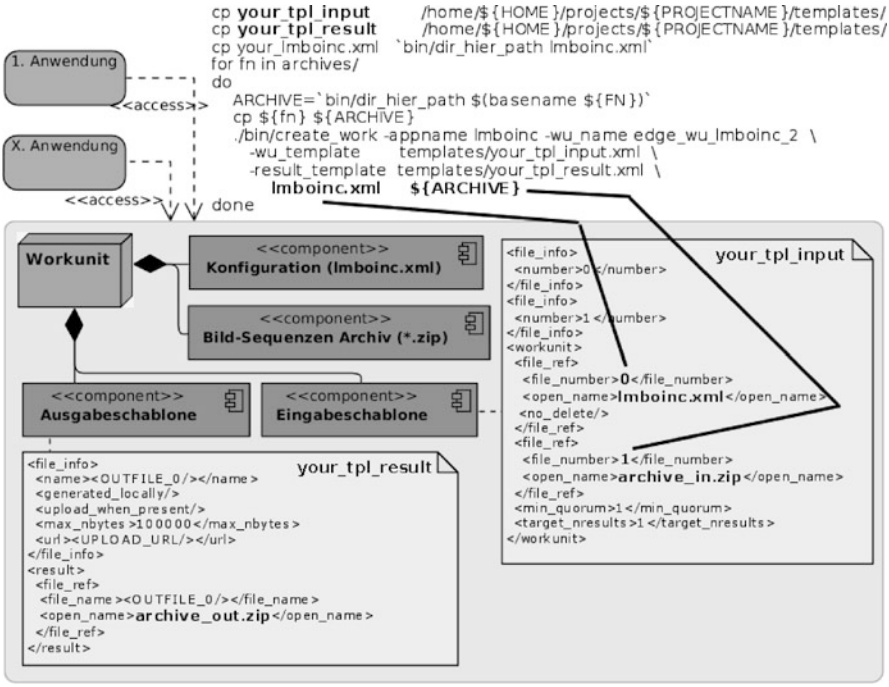


Abb. 12.3 Komponenten einer Workunit, die sich in einem BOINC-Slot befinden

Listing 12.8 Eingabeschablone zur Beschreibung von zwei benötigten Dateien, um die Ausführung von `lmboinc` zu starten. Die erste Datei beschreibt das Laufzeitverhalten, mit der zweiten Datei werden die Daten für die Bearbeitung bereitgestellt

```
<file_info>
  <number>0</number>
</file_info>
<file_info>
  <number>1</number>
</file_info>
<workunit>
  <file_ref>
    <file_number>0</file_number>
    <open_name>lmboinc.xml</open_name>
  </file_ref>
  <file_ref>
    <file_number>1</file_number>
    <open_name>archive_in.zip</open_name>
  </file_ref>
  <min_quorum>1</min_quorum>
  <target_nresults>1</target_nresults>
</workunit>
```

Wir sehen in diesem Template, dass für eine erfolgreiche Validierung nur eine erfolgreiche Abarbeitung eines Arbeitspakets erforderlich ist: `min_quorum = 1`. Abschnitt 11.2.2 beschreibt diese Parameter ausführlicher und beschreibt weitere Konfigurationsbeispiele. Listing 12.9 enthält die erforderliche Beschreibung, um die Ergebnisse von *lmboinc* abspeichern zu können. Die Dateigröße des Ergebnisar-

chivs ist auf die Dateigröße 10.000.000.000 ≈ 10 GB begrenzt. Zudem wird das Ergebnisarchiv lokal erstellt und beim Hochladen an das Projekt wird die durch <UPLOAD_URL/> beschriebene Webadresse verwendet.

Listing 12.9 Ergebnisschablone für das Erstellen des Ergebnisarchivs und Abspeichern der einzelnen Film-Sequenzen

```
<file_info>
  <name><OUTFILE_0/></name>
  <generated_locally/>
  <upload_when_present/>
  <max_nbytes>10000000000</max_nbytes>
  <url><UPLOAD_URL/></url>
</file_info>
<result>
  <file_ref>
    <file_name><OUTFILE_0/></file_name>
    <open_name>archive_out.zip</open_name>
  </file_ref>
</result>
```

12.2.3 Arbeitspakete erstellen

Für diesen Arbeitsschritt stehen zwei Skripts zur Verfügung:

`createArchives.sh`
: Erstellt die ZIP-Archive, welche die einzelnen Bild-Sequenzen enthalten.

`make_workunits`
: Fügt die zuvor erstellten ZIP-Archive als „offizielle“ Arbeitspakete zu unserem Projekt hinzu.

Abbildung 12.3 zeigt im oberen Abschnitt einen Teil des Skripts zum Hinzufügen der benötigten Arbeitspakete. Im ersten Schritt muss ein Video im Unterordner `videos/` bereitgestellt werden; dies wird im Abschn. 12.1.4 behandelt. Listing 12.10 beschreibt die erforderlichen Schritte in einem kurzen Beispiel und macht dabei vom Skript `createArchives.sh` Gebrauch. Die manuellen Schritte aus Abschn. 12.1.4 werden weitgehend durch eine Automatisierung erleichtert.

Listing 12.10 Drei Schritte zur Erstellung der ZIP-Dateien für die einzelnen Arbeitspakete: (1) Dateiordner für das Video erstellen (optional), (2) Bereitstellung des zu verarbeitenden Videos und (3) Erstellung der einzelnen Film-Sequenzen und Hinzufügen zu ZIP-Archiven

```
$ mkdir videos/
$ wget http://ADDRESSE_ZUR_VIDEO_DATEI -O videos/film.ogg
$ ./createArchives.sh -c 50 -v videos/film.ogg -m edge
```

Das Skript `make_workunits` erwartet drei Parameter:

1. Den Namen des BOINC-Projekts, für das die Arbeitspakete verwendet werden sollen,
2. den Modus der Bildbearbeitung, z. B. Kantendetektion oder Ölgemälde, und
3. das Verzeichnis, in dem sich die Eingabearchive der Arbeitspakete befinden.

Sie können das Skript ohne Parameter aufrufen und erhalten eine Beschreibung zum erfolgreichen Ausführen, wie dies in Listing 12.11 zu sehen ist. Das Listing enthält einen weiteren – diesmal vollständigen – Aufruf des Skripts und wir sehen daraufhin die Ausgaben von erfolgreich hinzugefügten Arbeitspaketen.

Listing 12.11 Auflistung der erforderlichen Parameter für das Erstellen und Hinzufügen von Arbeitspaketen zu *lmboinc*

```
boincadm@boinc-testserver:~/LMBoincServer_BeginnerPackage-0.6-done$ ./
    make_workunits
Usage: ./make_workunits -p 'project name'
                        -m (edge | normalize | oil | shade | negate)
                        -d 'directory of files without trailing slash'
                       [-i optional start index, default: 1]

boincadm@boinc-testserver:~/LMBoincServer_BeginnerPackage-0.6-done$ \
   ./make_workunits -p lmboinc -m edge -d archives
WU1:   /home/boincadm/LMBoincServer_BeginnerPackage-0.6-done/lmboinc_edge.xml
WU2:   /home/boincadm/LMBoincServer_BeginnerPackage-0.6-done/archives/
    edge_image000001.png.zip
/home/boincadm/projects/lmboinc
./bin/create_work -appname lmboinc -wu_name edge_wu_lmboinc_1 -wu_template
    templates/tpl_input_FILMNAME.xml -result_template templates/
    tpl_result_FILMNAME.xml lmboinc_edge.xml edge_image000001.png.zip
/home/boincadm/LMBoincServer_BeginnerPackage-0.6-done
...
...
...
WU1:   /home/boincadm/LMBoincServer_BeginnerPackage-0.6-done/lmboinc_edge.xml
WU2:   /home/boincadm/LMBoincServer_BeginnerPackage-0.6-done/archives/
    edge_image000016.png.zip
/home/boincadm/projects/lmboinc
./bin/create_work -appname lmboinc -wu_name edge_wu_lmboinc_16 -wu_template
    templates/tpl_input_FILMNAME.xml -result_template templates/
    tpl_result_FILMNAME.xml lmboinc_edge.xml edge_image000016.png.zip
/home/boincadm/LMBoincServer_BeginnerPackage-0.6-done
```

In diesem konkreten Beispiel wurden die Eingabearchive im Unterordner `archives/`:

Listing 12.12 Die ZIP-Archive für den Bildbearbeitungsmodus der Kantendetektion

```
boincadm@boinc-testserver:~/LMBoincServer_BeginnerPackage-0.6-done$ tree
    archives/
archives/
|-- edge_image000001.png.zip
|-- edge_image000002.png.zip
|-- edge_image000003.png.zip
|-- edge_image000004.png.zip
|-- edge_image000005.png.zip
|-- edge_image000006.png.zip
...
...
|-- edge_image000012.png.zip
|-- edge_image000013.png.zip
|-- edge_image000014.png.zip
|-- edge_image000015.png.zip
`-- edge_image000016.png.zip
```

zur Download-Hierarchie von `lmboinc` hinzugefügt:

Listing 12.13 Die vom Projekt `lmboinc` zur Verfügung gestellten Dateien; diese beinhalten die verschiedenen Versionen der wissenschaftlichen Applikation, die Konfiguration der Laufzeitparameter und die anzugebenden ZIP-Archive mit den jeweiligen Film-Sequenzen. Die Ordnernamen haben eine hexadezimale Kodierung; dieser Wert ist auf maximal `0xfff` beschränkt und kann dadurch 1024 Unterordner beschreiben. Es können mehrere Dateien in einem Unterordner vorhanden sein

```
boincadm@boinc-testserver:~/projects/lmboinc$ tree download/
download/
├── 15a
│   └── edge_image000001.png.zip
├── 193
│   └── edge_image000005.png.zip
├── 1c3
│   └── edge_image000009.png.zip
├── 1da
│   └── edge_image000013.png.zip
├── 211
│   └── edge_image000010.png.zip
├── 213
│   └── edge_image000004.png.zip
├── 250
│   └── lmboinc_edge.xml
├── 257
│   └── edge_image000007.png.zip
├── 2d0
│   └── edge_image000006.png.zip
├── 2df
│   └── edge_image000012.png.zip
├── 2e9
│   └── edge_image000002.png.zip
├── 32f
│   └── edge_image000014.png.zip
├── 346
│   └── edge_image000016.png.zip
├── 350
│   └── edge_image000008.png.zip
├── 36b
│   └── edge_image000015.png.zip
├── 3c
│   └── edge_image000011.png.zip
├── c9
│   └── edge_image000003.png.zip
├── lmboinc_0.1_i686-pc-linux-gnu
└── lmboinc_0.1_x86_64-pc-linux-gnu

17 directories, 19 files
```

12.3 Arbeitspakete verteilen und rechnen lassen

An dieser Stelle kann `lmboinc` anfangen, seine Arbeit aufzunehmen. Der Start des Projekts erfolgt durch die Eingabe von `./bin/start` im Root-Ordner von *lmboinc*. Daraufhin kann der Status aller konfigurierten Dämonen mit `./bin/status` ausgegeben werden und mit Hilfe von `./bin/stop` kann die Ausführung von *lmboinc* gestoppt werden. Falls noch Arbeitspakete auf den teilnehmenden Rechnern bearbeitet werden, so werden diese zu Ende ausgeführt und zu *lmboinc* hochgeladen, sobald die dafür relevanten Dämonen wieder gestartet sind. Andernfalls verbleiben die Ergebnisse so lange auf den teilnehmenden Rechnern und bei Überschreitung der Deadline – der jeweiligen Arbeitspakete – müssen die bis dahin überfälligen Arbeitspakete neu berechnet werden.

12.4 Ergebnisverarbeitung

Der Assimilierer dieses Projekts speichert die Ergebnisse im Dateiordner `sample_results/`, fehlerhafte Ergebnisse werden im Unterordner `sample_results/errors` abgespeichert und können später diagnostiziert werden. Die Ergebnisdateien besitzen die Namen der Arbeitspakete, daher kann aus den Dateinamen nicht

auf den Typ der Datei geschlossen werden. In diesem Projekt sind die Ergebnisdateien ZIP-Archive, die in der Regel die Dateiendung *.zip* tragen. Eine Prüfung mit

```
boincadm@boinc-testserver:~/projects/lmboinc$ file sample_results/
    edge_wu_lmboinc_1
sample_results/edge_wu_lmboinc_1: Zip archive data, at least v2.0 to extract
```

liefert uns die Bestätigung, dass es sich um ZIP-Archive handelt. In diesem Fall sollte mindestens ein ZIP-Programm mit dem Packalgorithmus in der Version 2.0 oder höher installiert sein. Die Archive enthalten die modifizierten Bild-Sequenzen, wie die folgende Ausgabe zeigt:

```
boincadm@boinc-testserver:~/projects/lmboinc/sample_results$ unzip -l
    edge_wu_lmboinc_1
Archive:  edge_wu_lmboinc_1
  Length      Date    Time    Name
---------  ---------- -----   ----
     2085  2011-01-02 17:38   edge_image000001.png
    28634  2011-01-02 17:38   edge_image000002.png
    ....
    ....
    59530  2011-01-02 17:38   edge_image000049.png
    59880  2011-01-02 17:38   edge_image000050.png
---------                     -------
  2865132                     50 files
boincadm@boinc-testserver:~/projects/lmboinc/sample_results$
```

Je nach vorher definierter Anzahl an Bild-Sequenzen in den ZIP-Archiven und der Größe eines Videos können sich zahlreiche Dateien im Ergebnisordner befinden. Die Dateien mit gleichem Anfang und unterschiedlichen Nummern gehören zusammen zu einem Film. Es ist zwingend erforderlich, dass die Archive eine aufsteigende Reihenfolge besitzen, so dass die einzelnen Bild-Sequenzen in der richtigen Reihenfolge zusammengesetzt werden können.

Für das Zusammensetzen wird ein einfaches Linux-Shell-Skript verwendet. Listing 12.14 zeigt die wenigen Zeilen, um alle ZIP-Archive zu entpacken und die einzelnen Bild-Sequenzen zu einem Film zusammenfügen. Das Skript hat eine triviale Logik, zu Beginn werden die übergebenen Parameter überprüft. Bei den Parametern handelt es sich um den Namen des BOINC-Projekts, in dem sich der Unterordner `sample_results/` mit den Ergebnissen befindet – in diesem Beispiel `lmboinc`. Der zweite Parameter beschreibt die Ergebnisbilder eines bestimmten Algorithmus. Da die ZIP-Archive nach dem Muster `MODUS-ID.zip` aufgebaut sind, können dadurch mehrere Ergebnisarchive unterschieden werden. Das Muster wird in der Konfigurationsdatei `make_configuration` beschrieben. Mit dem dritten Parameter wird der Speicherort und der Name des zu erstellenden Videos definiert. Durch den Video-Namen kann der Video-Codec des neuen Videos definiert werden, zum Beispiel ein OGG-Video [176] durch die Dateiendung `.ogg`. Die einzelnen ZIP-Archive werden in den Dateiordner `sequences/` entpackt und mit dem Werkzeug `mogrify` [137] auf ein einheitliches Bilddatei-Format gebracht; hier entspricht das dem JPEG-Bildformat [140] mit einer Qualität von 85 % des Eingangsbildes. `ffmpeg` [126] erstellt im letzten Schritt das neue Video.

Listing 12.14 `createNewFilm.sh`

```
function usage() {
  echo "Usage: $0 -p PROJECTNAME -m MODE -n NEWFILE"
  echo "Description:"
  echo "      -p    Name of your project."
  echo "      -m    edge | normalize | oil | shade | negate"
  echo "      -n    File name of the video output."
  exit 1
}

if [ $# -le 5 ]
  then
  usage
fi

while getopts "p:m:n:" flag
do
  case "$flag" in
    p) PROJECTNAME=$OPTARG
    ;;
    m) MODE=$OPTARG
    ;;
    n) NAME=$OPTARG
    ;;
    *) usage ;;
  esac
done

source make_configuration

rm -r sequence_${MODE} ; mkdir sequence_${MODE}/
cd sequence_${MODE}/
for file in ${ROOTPROJECT}/sample_results/*${MODE}*
do
  echo ${file}
  unzip ${file}
done
cd ..

cd sequence_${MODE}/
mogrify -format jpg -quality 85 *.png
cd -
ffmpeg -i sequence_${MODE}/${PREFORMAT}.jpg ${NAME}
```

Abbildung 12.4 zeigt oben links die Original-Bild-Sequenz und drei Ergebnisse mit dem Verwenden unterschiedlicher Modifikationsalgorithmen.

12.4.1 Validierung – Prüfung der Ergebnisse

Eine wissenschaftliche Applikation wie in diesem Fall kann durch die zwei mitgelieferten Validierer geprüft werden. Im Listing 12.8 haben wir `min_quorum` auf Eins gesetzt; die erfolgreiche Validierung setzt ein berechnetes Arbeitspaket voraus. Wir gehen davon aus, dass die Bildbearbeitungsalgorithmen immer gleich arbeiten und ein Arbeitspaket – wenn es öfter verteilt ist und berechnet wurde – immer ein identisches Ergebnis hat. Diese Voraussetzungen ermöglicht den `sample_bitwise_validator` zu verwenden.

Abb. 12.4 Ein Bild aus der Original-Filmsequenz (*oben links*) und die modifizierten Bilder nach der Bearbeitung mit der Beispielanwendung; Kantendetektion *rechts oben*, Ölgemälde *unten links* und Negativbild *unten rechts*. www.bigbuckbunny.org, Blender Foundation | www.blender.org

12.4.2 Assimilation – Ergebnisse abspeichern

Die validierten Arbeitspakete werden im Unterverzeichnis `sample_results/` für eine spätere Verwendung abgelegt. Es sollte darauf geachtet werden, dass die Arbeitspakete alle einen eindeutigen Namen besitzen, da vorhandene Arbeitspakete sonst überschrieben werden. Dies kann vorkommen, wenn Sie einen neuen Film zum Bearbeiten hinzugefügt, aber die Benennung nicht variiert haben. In diesem Fall würden eventuell zwei Filme miteinander vermischt werden. Daher ist es erstrebenswert, dass Sie vor der Erstellung einer neuen Berechnung die Ergebnisse an einem anderen Speicherplatz abgelegen, oder die Ergebnisse vorher zu einem Endergebnis verarbeiten und daraufhin die bestehenden Ergebnisse löschen, wenn diese nicht für weitere Schritte benötigt werden.

Das eingangs erwähnte Projekt *LMBoinc* [143] mit allen Informationen und Arbeitsschritten dieses Projekts enthält ein Skript zur Verarbeitung der Ergebnisse. Mit Hilfe dieses Skripts kann aus den Ergebnissen ein neuer Film erstellt werden. Die Verwendung ermöglicht das Definieren eines Speicherplatzes für den zu erstellenden Film. Nach dem Aufruf haben wir ein absolutes Ergebnis und die Berechnungsergebnisse werden nicht mehr benötigt und haben für uns keinen Nutzen mehr. Falls die Ergebnisse doch noch einmal benötigt werden sollten, so können die einzelnen Film-Sequenzen aus dem erstellten Video extrahiert werden.

Kapitel 13
Verwendung von ISV-Applikationen

‚Putting some spin on it ...‘

Prof. Dr. rer. nat Christian Schröder

13.1 Wissenschaftliche Applikation

Das Praxisbeispiel aus Kap. 14 beschreibt das Verwenden einer sogenannten ISV-Applikation[1] (Independed Software Vendor) (nachfolgend *ISVA* genannt). Dies ist oft der Fall und eine Modifikation der Laufzeitumgebung ist oft nicht möglich und meist sogar durch die Softwarehersteller nicht gewünscht, u. a. weil Modifikationen dazu führen, ohne Support leben zu müssen. Wenn Sie als Nutzer schon eine in einem speziellen Bereich nicht ersetzbare Software angeschafft haben, so wollen Sie sicherlich auch den angebotenen Support nutzen.

Ebenso kann es sein, dass Sie eine selbst entwickelte Software besitzen und diese nun mit BOINC kombinieren wollen. Vielleicht planen Sie auch eine Anschaffung eines Hochleistungsclusters, oder Sie denken darüber nach, Ihre Berechnungen durch das sog. Cloud Computing zu lösen, fürchten allerdings Sicherheitsprobleme. Mit Hilfe von BOINC können Sie im Vorfeld prüfen, ob sich überhaupt ein Speedup[2] ergibt, und dafür können Sie direkt die schon vorhandene Infrastruktur in Ihrer Firma nutzen.

Wenn Sie nun eine ISVA haben und einen signifikanten Speedup erreichen wollen, so könnten Sie das mit BOINC realisieren. Natürlich müssen die Prämissen aus Abschn. 2.1 erfüllt sein, Ihre Problemstellungen müssen also eine Granularität besitzen und möglichst ohne Kommunikation zwischen den Rechnern auskommen.

[1] Dies sind im Arbeitsumfeld einer Firma oder im privaten Bereich genutzte Applikationen, welche schon länger im Einsatz sind und bei denen die Quelltexte der Applikationen in den meisten Fällen nicht vorhanden sind.

[2] Der Speedup ist definiert als die Division der Zeit der sequenziellen Ausführung durch die Zeit der parallelen Ausführung mit p Prozessoren: $S_p = T_1/T_p$.

C. B. Ries, *BOINC*, Xpert.press,
DOI 10.1007/978-3-642-23382-8_13, © Springer-Verlag Berlin Heidelberg 2012

13.1.1 Problemansatz

Generell haben Sie von Haus aus die Möglichkeit, einen Wrapper zu nutzen, eine Version wird bei BOINC mitgeliefert und unterstützt Windows, Macintosh und Linux ohne weitere Modifikationen. Der Wrapper arbeitet eine Liste von Applikationen ab und beendet sich dann. Ein Arbeitspaket kann aus mehreren einzelnen Aufgaben bestehen und diese müssen nichts miteinander zu tun haben. Die Wrapper-Implementierung sieht vor, dass bei einer Liste von abzuarbeitenden bzw. zu startenden Applikationen jeweils nach der Beendigung einer Applikation ein Checkpoint erstellt werden kann, so dass bei einem Stopp und einem Neustarten vorherige Ausführungen nicht wiederholt werden müssen.

Abbildung 13.1 zeigt den Verlauf der Ausführung von drei wissenschaftlichen Applikationen und wann Checkpoints erstellt werden. Links bei S1 startet der Wrapper die Ausführung der ersten Applikation. Es wurde bis dahin noch kein Checkpoint erstellt, und wenn der BOINC-Teilnehmer die Nachricht „Pause1“ sendet, dann wird die Ausführung beendet. Wenn die Ausführung nun wieder gestartet wird, beginnt die Arbeit wieder bei S1.

Anders ist es, wenn der BOINC-Teilnehmer erst die Nachricht „Pause3“ sendet, wenn die dritte Applikation läuft. Bis zu diesem Zeitpunkt wurden schon zwei Checkpoints erstellt. In diesem Fall ist der Checkpoint2 relevant und in der zugehörigen Checkpoint-Datei stehen zwei Zahlenwerte: 2 und 7432.1. Der erste Wert gibt die Anzahl der bisher ausgeführten Aufgaben an und der zweite Wert enthält die bisher genutzte CPU-Zeit für die bisherigen Aufgaben. Aus der Anzahl der bisher ausgeführten Applikationen und einer in Abschn. 13.3.2 vorgestellten Gewichtung kann der Fraction-Done-Wert errechnet und dem BOINC-Client mitgeteilt werden. So kann man zumindest den bisherigen Verlauf und noch ausstehende Arbeitszeit abschätzen. Wenn der Wrapper nun pausieren und daraufhin die Arbeit wieder aufnehmen soll, so wird die Checkpoint-Datei eingelesen und bei S3 weitergemacht, allerdings muss der gesamte Prozess von der dritten Applikation neu durchgeführt werden, wenn die ISVA keine weiteren Mechanismen für das Wiederaufnehmen der Arbeit besitzt. Diese Umsetzung kann zu einer endlos ausgeführten Ausführung führen oder zumindest eine Abarbeitung bis Übertritt der Deadline haben. Stellen Sie sich vor, die Applikation hat eine 90-Prozent-Gewichtung, die beiden vorherigen Applikationen wurden relativ schnell abgearbeitet und die dritte Applikation benötigt etwa einen Tag moderne Prozessoren. Wenn der Rechner in dieser Zeit mehrmals ausgeschaltet wird, so kann das Arbeitspaket niemals erfolgreich beendet werden. Die Nutzung eines solchen Wrappers ist demnach gefährlich und kann dazu führen, dass kein einziges von Ihnen erstelltes Arbeitspaket jemals erfolgreich abgearbeitet wird.

Man kann ein Workaround für dieses Problem erstellen. Ich werde dies in Abschn. 13.3.2 diskutieren.

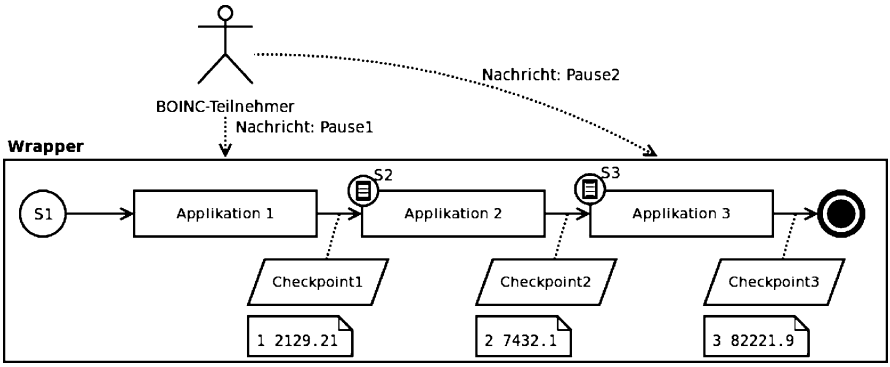


Abb. 13.1 Checkpoint-Mechanismus bei BOINCs mitgeliefertem Wrapper

13.1.2 Aufbau und Programmlogik

Soll eine ISVA als wissenschaftliche Applikation genutzt werden, so kann im Vorfeld eine Checkliste abgearbeitet werden, um die Entscheidung bezüglich des zu verwendenden Ansatzes und der zu verwendenden Technologien zu überprüfen.

1. Sind Sie der Besitzer Ihrer ISVA oder sind Sie Lizenznehmer? Im ersten Fall haben Sie sicherlich auch die Quellen der Applikation und können diese dann zu einer reinen wissenschaftlichen Applikation für BOINC weiterentwickeln, oder den in diesem Kapitel behandelten Wrapper verwenden. Sind Sie Lizenznehmer, so müssen Sie darauf achten, wie Sie die ISVA starten dürfen, evtl. haben Sie nur eine Einzelplatzlizenz oder Lizenz-Bundles für x Rechner in Ihrem Unternehmen.
2. Wenn Sie eine Ausführung der ISVA auf unterschiedlichen Zielplattformen anstreben, so stellt sich die Frage, ob dies von der ISVA unterstützt wird. Wenn die ISVA von vornherein nur ausgewählte Zielplattformen abdeckt, dann schließt sich das Behandeln weiterer Plattformen von vornherein aus.
3. Kann der Ausführungsstatus zu jedem Zeitpunkt ermittelt werden? Ist es Ihnen möglich, während der Laufzeit der ISVA folgende Informationen zu erfragen:

    a. Kann eine individuelle Ausführung der ISVA durchgeführt werden? Dies bedeutet: Ist es möglich, parametrisierte Ausführungen zu definieren, mit den passenden Beschreibungen der Berechnungsroutinen, der mathematischen Modelle bzw. der Simulationsmodelle? Wenn diese Fragestellung nicht bejaht werden kann, dann ist schon jetzt abzusehen, dass viel Aufwand für eine Umsetzung nötig ist. Wenn dieser Fall auf Sie zutrifft und Sie nicht die Quellen der Anwendung besitzen, so kann keine hundertprozentige Aussage über den Status der ISVA-Ausführung gemacht werden.
    b. Können Sie die Ausführung der ISVA zu jedem Zeitpunkt pausieren und zu einem wahlfreien anderen Zeitpunkt weiterführen?

   c. Können Sie ermitteln, zu wie viel Prozent die Ausführung abgeschlossen ist?

4. Besitzen Sie die Quellen der ISVA? Wenn ja, so haben Sie die Möglichkeit, Anpassungen an der ISVA durchzuführen. Ein automatisierter Prozess für die Berechnung von Simulationen benötigt z. B. keine grafische Oberfläche und für einen solchen Fall reicht es, wenn nur die Kernkomponente zum Lösen der Problemstellung zur Verfügung steht. Sie können somit diese Komponente mit Support der BOINC-Infrastruktur umsetzen. Allerdings kommt diese Lösung einer eigenen wissenschaftlichen Applikation für BOINC gleich und wird nicht mehr als ISVA gesehen.

13.1.3 Beschreibung der verwendeten ISV-Applikation

Wir verwenden für dieses Kapitel CINOLA (Classical Spin Dynamics for Core/Shell Particles) [31] als ISVA, für die – zumindest fiktiv – keine Quellen zur Verfügung stehen. Es handelt sich dabei um eine Eigenentwicklung, die seit über einem Jahrzehnt kontinuierlich weiterentwickelt wird, an der zahlreiche verschiedene Arbeitsgruppen (Professoren, wissenschaftliche Mitarbeiter, Studierende) Arbeiten eingepflegt haben und dies noch heute stetig tun. CINOLA ist historisch gewachsen und enthält teilweise sehr komplexe logische Strukturen von durchaus sehr komplizierten Algorithmen und Datenstrukturen. Weiterhin besitzt CINOLA Implementierungen, um u. a. das Message Passing Interface (MPI) und die Berechnung mit Hilfe der GPU durchzuführen. Der Aufwand, diese ISVA für eine volle BOINC-Unterstützung zu modifizieren, ist zu hoch und kommt daher nicht in Betracht.

CINOLA erwartet nur eine Eingabedatei `mm.dat`, um ordnungsgemäß zu starten. Die Datei hat ein bestimmtes zeilenorientiertes Format, um die physikalischen Eigenschaften der magnetischen Wechselwirkungen zwischen zwei oder mehreren Spins von Quanten zu beschreiben, so dass darauf aufbauend eine Simulationsberechnung durchgeführt werden kann. Die `mm.dat` enthält u. a. die Auswahl zwischen verschiedenen Simulationsmodellen (Monte-Carlo-Simulation und stochastische Simulation), wie die Viskosität sein soll, wie viele Spins in der Simulation vorhanden sind, die Anzahl der Iterationen für ein Simulationsmodell und noch einiges mehr. An der Anzahl der möglichen Einstellungen können Sie sicherlich schon erahnen, dass eine schier unendliche Anzahl an verschiedenen Parametersätzen möglich ist und in einzelne Arbeitspakete gepackt werden kann. Es sei noch erwähnt, dass der Dateiname `mm.dat` in CINOLA hardcodiert ist.

Dabei enthält eine `mm.dat` immer nur einen Parametersatz und beim Start von CINOLA werden zwei Dateien generiert:

```
-rw-r–r– boincadm ... 13:20 FCC_SPH_R_1.250000_N_38.1.nn
-rw-r–r– boincadm ... 13:20 FCC_SPH_R_1.250000_N_38.1.nn.jj
```

Diese sind für die Simulation wichtig, werden nach der Generierung eingelesen und sind die eigentlichen Parametersätze eines Arbeitspakets. Dennoch wird zur Startzeit nur die `mm.dat` benötigt und muss – bei Nutzung mit BOINC – durch ein BOINC-Projekt zur Verfügung gestellt werden.

Nach der Beendigung von CINOLA wurden fünf neue Dateien erstellt. Diese Dateien enthalten ein lokales Ergebnis der Berechnung und werden für das Erstellen eines globalen Ergebnisses benötigt:

```
-rw-r–r– boincadm ... fcc_nanoparticle_win_x64_H3.000000_averaged.res
-rw-r–r– boincadm ... fcc_nanoparticle_win_x64_H3.000000_run_1_procs_1.res
-rw-r–r– boincadm ... fcc_nanoparticle_win_x64_T0.100000_averaged.res
-rw-r–r– boincadm ... fcc_nanoparticle_win_x64_T0.100000_H3.000000.dat
-rw-r–r– boincadm ... fcc_nanoparticle_win_x64_T0.100000_run_1.res
```

13.2 BOINC-Wrapper

Der BOINC-Wrapper unterstützt Windows-, Macintosh- und Linux-Plattformen; vorkompilierte Versionen finden Sie im BOINC-Wiki [96]. Eine Wrapper-Installation kann sofort durchgeführt werden, solange es sich bei den Zielplattformen um Windows 32-Bit, 64-Bit, sowie Macintosh PowerPC-Prozessoren (PPC) und Macintosh Intel-Prozessoren handelt. Sie ersparen sich dann den Umstand, evtl. die einzelnen Betriebssysteme plus die Entwicklungsumgebungen zu installieren, womöglich vorher noch kaufen zu müssen.

Für dieses Kapitel kompilieren wir uns eine Linux-Version. Dafür wechseln wir innerhalb der BOINC-Quellen in den Ordner des Wrappers und führen für die Erstellung einer 32-Bit-Version folgende Kommandos aus:

```
boincadm@boinc-testserver:~/server_stable/samples/wrapper$ make
g++ -g -I../.. -I../../lib -I../../api -L../../api -L../../lib -L.   -c -o
    wrapper.o wrapper.cpp
g++ -g -I../.. -I../../lib -I../../api -L../../api -L../../lib -L. -o wrapper
    wrapper.o libstdc++.a -pthread -lboinc_api -lboinc
boincadm@boinc-testserver:~/server_stable/samples/wrapper$ cp wrapper wrapper_0
    .1_i686-pc-linux-gnu
```

Der letzte Befehl erstellt uns eine Kopie des Wrappers, in der von BOINC erwarteten Namenskonvention für 32-Bit-Intel-Prozessoren.

13.2.1 Laufzeitverhalten

Der prinzipielle Ablauf des BOINC-Wrappers wird vereinfacht in Abb. 13.2 gezeigt, wobei oben rechts die Ausführung beginnt, erst einen Verlauf nach links, dann nach rechts nimmt und sich dann beendet. Zu Beginn wird die `job.xml` eingelesen (vgl. Abschn. 13.3.2), danach wird eine evtl. vorhandene Checkpoint-Datei eingelesen; der hardcodierte Dateiname „wrapper_checkpoint.txt“ wird verwendet, woraufhin

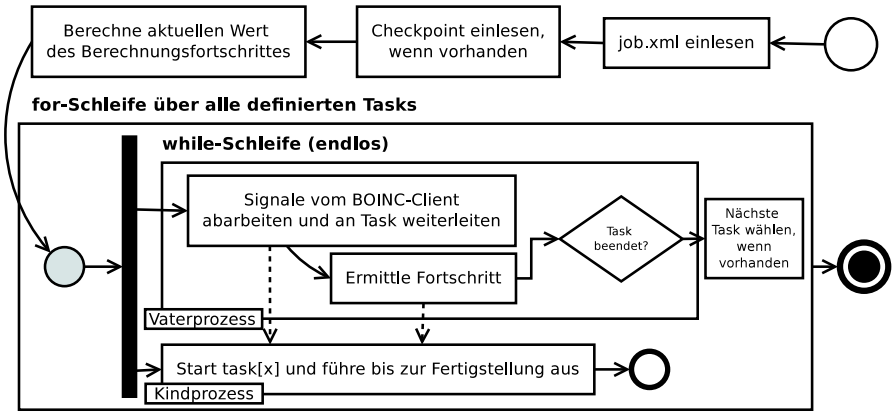


Abb. 13.2 Laufzeitverhalten des BOINC-Wrappers

der aktuelle Fortschritt errechnet wird. Der Startwert der for-Schleife wird auf den Task gesetzt, der auf den Index aus der Checkpoint-Datei folgt; wenn kein Checkpoint vorhanden ist, so wird der erste Task in der `job.xml` ausgewählt.

Nach diesen Vorarbeiten wird der Prozess dupliziert, der Vaterprozess übernimmt die Verwaltung des Gesamtprozesses und die Kommunikation zum BOINC-Client, im Kindprozess wird der Task gestartet und ausgeführt; dabei wird der duplizierte Prozess überschrieben und es existiert nur noch die Applikation des Tasks im Speicher. Der Vaterprozess wechselt nun in eine while-Schleife und wird so lange ausgeführt, wie der Kindprozess sich nicht beendet hat oder die Ausführung vom BOINC-Client nicht gestoppt wird. Für das Abfangen von solchen Kommandos wird kontinuierlich geprüft, ob Nachrichten vom BOINC-Client vorliegen. Diese Nachrichten werden auf Befehle für das Pausieren, Stoppen und Neustarten des Tasks untersucht und ggf. wird daraufhin ein entsprechendes Signal an den Task gesendet.

Wie wir bereits in Abschn. 7.3.2 diskutiert haben, werden Nachrichten über den Shared-Memory zwischen dem BOINC-Client und einer wissenschaftlichen Applikation ausgetauscht; diese dienen der Prozesssteuerung der wissenschaftlichen Applikation. Nach dem Abarbeiten der Nachrichten wird der aktuelle Fortschritt ermittelt und wenn der Kindprozess beendet ist, so wird in der for-Schleife der nächste Task ausgewählt und gestartet bis alle Tasks abgearbeitet sind.

13.2.2 Möglichkeiten eines Tasks

Es ist natürlich nicht ausgeschlossen, dass der jeweils ausgeführte Task nicht selbst einen Checkpoint-Mechanismus besitzt. Auch wenn der BOINC-Wrapper diese Möglichkeit ohne großen Aufwand unterstützt, so könnte eine zusätzliche Lösung gewünscht sein, wenn Sie dies nicht ohnehin schon in Ihrer eigenen ISVA vorliegen haben.

Für diesen Fall können Sie einen weiteren Parameter in der `job.xml` verwenden:

```
<fraction_done_filename>cinola.frac_done</fraction_done_filename>
```

Mit dieser Angabe wird kontinuierlich die Datei `cinola.frac_done` eingelesen und der vorliegende Fließkomma-Wert an den BOINC-Client gesendet. Es ist natürlich nötig, dass in diesem Fall CINOLA einen passenden Wert in diese Datei schreibt.

13.3 Projekteinstellungen

Das Hinzufügen der Projekteinstellungen scheint trivial, allerdings müssen Sie bei der Verwendung des BOINC-Wrappers nun nicht nur Ihre ISVA hinzufügen, sondern natürlich auch den BOINC-Wrapper und evtl. weitere Anwendungen, um Vorarbeiten oder Nacharbeiten durchführen zu können. Letzteres ist in diesem Kapitel gewünscht. Sie haben in einem vorherigen Abschnitt gesehen, dass CINOLA fünf Dateien erstellt, allerdings ist es einfacher, wenn nur eine Datei zum BOINC-Projekt zurückgesendet wird. Sicherlich ist es kein Problem, alle Dateien einzeln zu versenden, die Problematik ist an dieser Stelle aber, dass nicht der Aufwand getrieben werden soll, zu schauen, wie eine Datei später heißt. Der Name der Ergebnisdateien ist nämlich nicht nach jeder Ausführung gleich.

Der Name der Ausgabedateien ist von einer Variablen in der `mm.dat` abhängig. Es ist nicht geschickt, wenn wir bei jeder Erstellung eines Arbeitspakets mehrere Stellen ändern müssen, nämlich die `mm.dat` und die Ergebnisschablone. Die Ergebnisschablone müsste mindestens so viele Einträge erhalten, wie Dateien erstellt werden, zusätzlich müssten die logischen Namen mit den physikalischen Dateinamen übereinstimmen. Das zu verwalten kann recht aufwendig sein, wenn Sie nur ein paar wenige Änderungen in der `mm.dat`-Datei durchführen und daraufhin immer prüfen müssten, ob die Ergebnisschablone noch passend ist oder nicht. Dieser Prozess ist sehr fehleranfällig und ich gehe mit Ihnen jede Wette ein, dass Sie nach zehn Änderungen meinem Rat in Abschn. 13.3.2 folgen werden; wir erstellen einfach ein ZIP-Archiv und packen alle Dateien mit einer bestimmten Dateiendung dort hinein.

13.3.1 Konfiguration und Anwendung hinzufügen

Das Vornehmen der Projekteinstellungen scheint trivial; wir nehmen uns ein vorhandenes BOINC-Projekt und fügen in der Projektdatei `project.xml` die Zeilen aus Listing 13.1 hinzu.

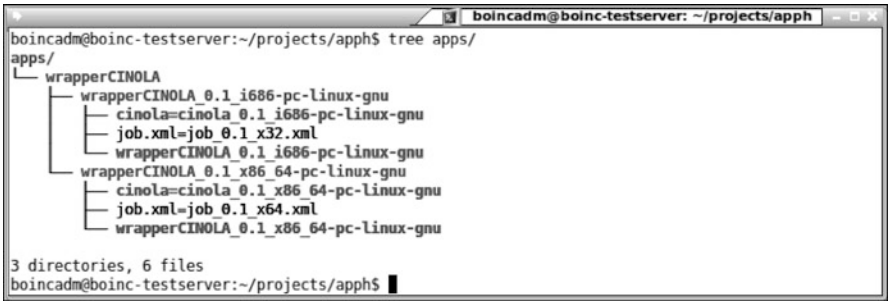


Abb. 13.3 Ordner-/Dateistruktur für CINOLA in Kombination mit BOINC

Listing 13.1 Beschreibung der ISV-Applikation CINOLA in der `project.xml` eines BOINC-Projektes

```
<boinc>
  ...
  <app>
    <name>wrapperCINOLA</name>
    <user_friendly_name>CINOLA@home - Spin me up!</user_friendly_name>
  </app>
</boinc>
```

Abbildung 13.3 zeigt die Ordner-/Dateistruktur unterhalb des `apps/wrapperCINOLA`-Ordners. Wie wir wissen, hängt der Name von den Einstellungen in der `project.xml` ab und unterhalb dieses Ordners werden die Applikationen hinzugefügt, die von den BOINC-Teilnehmern herunterzuladen sind. Zu sehen sind u. a. vier Dateien der Form *NAME=APPLIKATION* oder *NAME=KONFIGURATION*, es handelt sich dabei um ein Mapping einer physikalisch vorhandenen Datei zu einem logischen Namen. Beim Aufruf des Skripts `./bin/update_versions` wird die Form der Dateien erkannt und beim Hinzufügen der Applikation in die Datenbank entsprechend angepasst, so dass der BOINC-Teilnehmer beim späteren Herunterladen auch alle zugehörigen Dateien mit lädt.

Neuere Applikationen werden der Datenbanktabelle `app` hinzugefügt und die Beschreibungen einer bestimmten Version – hier z. B. Linux 32-Bit und 64-Bit mit der 0.1 Version – in die Datenbanktabelle `app_version` eingetragen. `app_version` besitzt das Feld `xml_doc` und dort werden die Informationen der zusätzlichen Dateien im richtigen Format eingegliedert. Die Form der erstellten Information für eine Applikationsversion finden Sie in Listing 13.2. Das Format ist ähnlich der Eingabe-/Ergebnisschablonen von Arbeitspaketen. Der obere Teil zwischen den Zeilen 1 und 26 beschreibt die einzelnen zugehörigen Dateien einer wissenschaftlichen Applikation und wo diese heruntergeladen werden müssen. Sie sehen auch bei den ersten beiden Dateien das XML-Tag `<executable/>`, dies beschreibt ausführbare Dateien, die `job.xml` ist nicht ausführbar und daher ist das XML-Tag auch nicht zugefügt. Das nachfolgende XML-Tag enthält die Signatur einer Datei, so dass diese nicht beim BOINC-Client durch externe Zugriffe modifiziert werden kann; man könnte, aber das Berechnungsergebnis würde als fehlerhaft deklariert werden. Zuletzt wird die Dateigröße angegeben.

Listing 13.2 app_version.xml_doc einer Applikation mit drei abhängigen Dateien

```
<file_info>
  <name>wrapperCINOLA_0.1_x86_64-pc-linux-gnu</name>
  <url>http://192.168.1.100/apph/download/wrapperCINOLA_0.1_x86_64-pc-linux-gnu
      </url>
  <executable/>
  <file_signature>
  ...
  </file_signature>
  <nbytes>818875.000000</nbytes>
</file_info>
<file_info>
  <name>cinola_0.1_x86_64-pc-linux-gnu</name>
  <url>http://192.168.1.100/apph/download/cinola_0.1_x86_64-pc-linux-gnu</url>
  <executable/>
  <file_signature>
  ...
  </file_signature>
  <nbytes>464631.000000</nbytes>
</file_info>
<file_info>
  <name>job_0.1_x64.xml</name>
  <url>http://192.168.1.100/apph/download/job_0.1_x64.xml</url>
  <file_signature>
  ...
  </file_signature>
  <nbytes>577.000000</nbytes>
</file_info>
<app_version>
  <app_name>wrapperCINOLA</app_name>
  <version_num>1</version_num>
  <api_version>6.11.7</api_version>
  <file_ref>
    <file_name>wrapperCINOLA_0.1_x86_64-pc-linux-gnu</file_name>
    <main_program/>
  </file_ref>
  <file_ref>
    <file_name>cinola_0.1_x86_64-pc-linux-gnu</file_name>
    <open_name>cinola</open_name>
  </file_ref>
  <file_ref>
    <file_name>job_0.1_x64.xml</file_name>
    <open_name>job.xml</open_name>
  </file_ref>
</app_version>
```

Die Versionsbeschreibung einer Applikation ist zwischen den Zeilen 27 bis 43 zugefügt. Es wird die erste Version mit der Nutzung der BOINC-API-Version 6.11.7 und drei Dateien hinzugefügt. Die Dateireferenzen (engl. file references) beschreiben das Mapping zwischen der physikalischen Datei und einem logischen Namen, so dass diese Dateien zu einem Slot hinzugefügt werden können. Es ist darauf zu achten, dass die logischen Namen mit den Dateinamen in der `job.xml` übereinstimmen. Die zweite Datei in diesem Bereich in Zeile 36, mit dem logischen Namen `cinola`, wird durch die `job.xml` für den Wrapper zum Öffnen definiert. Das Mapping in dem Dateiordner der hinzuzufügenden Applikationen muss mit der Beschreibung der auszuführenden Tasks unbedingt übereinstimmen.

13.3.2 CINOLAs job.xml

Der BOINC-Wrapper arbeitet sequenziell alle Tasks ab, die in der `job.xml` vorhanden sind. Die Reihenfolge der Tasks ist wichtig und muss der Reihenfolge entsprechen, wie die einzelnen Tasks ausgeführt werden sollen. Wenn das Ergebnis aus einem vorherigen Task benötigt wird, so muss dieser auch im Vorfeld ausgeführt werden. Das Listing 13.3 enthält solch ein Beispiel, da der zweite Task für die Erstellung eines ZIP-Archivs erst dann durchgeführt wird bzw. sinnvoll erscheint, wenn Ergebnisdateien von CINOLA erstellt wurden. Weiterhin besitzt die Konfiguration für beide Tasks drei Standardeinstellungen, betreffend die Standardein-/ausgabedateien. Sie sollten immer alle drei Angaben treffen, denn in Tests hat sich gezeigt, dass eine fehlende Angabe durchaus dazu führt, dass der BOINC-Wrapper nicht ordnungsgemäß funktioniert. Die Interna des BOINC-Wrappers prüft, ob die definierten Dateien geöffnet werden können, wenn nicht, so wird der Kindprozess des BOINC-Wrappers beendet, ohne dass ein Task gestartet wird.

CINOLA schreibt den aktuellen Berechnungsfortschritt in die Datei `cinola.frac_done`, und in der `job.xml` wird spezifiziert, dass diese Datei zusätzlich für die Berechnung des BOINC-Fraction-Done-Wertes einbezogen werden soll. Dafür muss das XML-Tag `<fraction_done>` mit der Datei hinzugefügt werden. Im zweiten Task wird ein Standardwerkzeug innerhalb einer Linux-Systemumgebung spezifiziert, welches in der Regel immer vorhanden ist. Sie sehen, dass zahlreiche ISVA Verwendung finden können, so dass keine Arbeit in die Implementierung gesteckt werden muss. Da ZIP ohne Parameter nicht viel tut, außer sich wieder zu beenden, geben wir durch das XML-Tag `<command_line>` dem Werkzeug einige Startparameter mit auf den Weg. Der erste Parameter ist das zu erstellende ZIP-Archiv mit dem Namen `results.zip`, welches auch in der Ergebnisschablone in Listing 13.5 erwartet wird. Die zwei nachfolgenden Parameter sind ein Wildcast und packen alle Dateien in das ZIP-Archiv, die durch CINOLA für die Ergebniserstellung angefertigt werden (vgl. Abschn. 13.1.3).

Als letzte Konfiguration – zumindest in diesem konkreten Beispiel – wird noch eine Gewichtung der einzelnen Tasks angegeben, so dass die einzelnen Berechnungsfortschritte ungefähr in ein passendes Verhältnis gebracht werden. Es ist offensichtlich, dass die Berechnung durch CINOLA eine höhere Gewichtung erhalten sollte als das Erstellen eines ZIP-Archivs mit einfachen Textdateien, in diesem Fall etwa 600 Bytes; diese paar Bytes werden mit einem Wimpernschlag in ein ZIP-Archiv übernommen. Sie müssen sich bei der Angabe der Gewichtung nicht an ein bestimmtes Schema halten, sprich es gibt keine Grenzwerte, welche Sie einhalten müssen. Eine Gewichtung sollte allerdings möglichst klar erkennbar sein. In diesem Beispiel fließt die Fortschrittsberechnung von CINOLA in den Endwert zu 200 Teilen mehr in das Endergebnis ein als das Erstellen des Ergebnis-ZIP-Archivs.[3]

[3] Eine Gewichtung von jeweils 50 würde dazu führen, dass die Fortschrittsanzeige z. B. im BOINC-Manager für CINOLA noch ordnungsgemäß funktioniert und für die Erstellung des ZIP-Archivs nicht. CINOLA schreibt den eignen Fraction-Done-Wert in eine Datei, die für die Berechnung einbezogen wird, das ZIP-Werkzeug ist allerdings stumm. Die Fortschrittsanzeige im BOINC-

Listing 13.3 job.xml für CINOLA@home

```
<job_desc>
  <task>
    <application>cinola</application>
    <stdin_filename>dummy</stdin_filename>
    <stdout_filename>stdout</stdout_filename>
    <stderr_filename>stderr</stderr_filename>
    <fraction_done_filename>cinola.frac_done</fraction_done_filename>
    <weight>200</weight>
  </task>
  <task>
    <application>zip</application>
    <stdin_filename>dummy</stdin_filename>
    <stdout_filename>stdout</stdout_filename>
    <stderr_filename>stderr</stderr_filename>
    <command_line>results.zip fcc*.res fcc*.dat</command_line>
    <weight>1</weight>
  </task>
</job_desc>
```

13.3.3 Templates erstellen

Die Eingabeschablone in Listing 13.4 muss an die CINOLA-Gegebenheiten angepasst werden, denn CINOLA erwartet eine Datei mit dem Namen `mm.dat` und verwendet keine BOINC-Funktion, um diese Datei zu öffnen. Daraus folgt, dass symbolische Links nicht funktionieren und die Datei in die jeweiligen Slots hineinkopiert werden muss. Dies geschieht durch die Angabe des XML-Tag `<copy_file/>`. Der Rest der Schablone hat das aus anderen Kapiteln bekannte Format.

In den Zeilen 26 bis 30 werden Angaben über die Anzahl der Berechnungen gemacht:

- `<max_error_results>`: wie oft keine erfolgreiche Bearbeitung stattfinden darf, bevor das komplette Arbeitspaket als fehlerhaft deklariert wird,
- `<max_total_results>`: wie viele Ergebnisse maximal nötig sein sollen, um ein Arbeitspaket an die weiteren Prozesse wie Validierer und Assimilierer weiterzuleiten und
- `<max_success_results>`: ab wann ein Arbeitspaket das erste Mal validiert werden kann, nämlich wenn der durch diesen XML-Tag angegebene Zahlenwert erreicht ist.

Das Listing enthält schon einige Informationen darüber, wie in Abschn. 13.4.2 Arbeitspakete hinzugefügt werden. Die XML-Tags `rboinc` beschreiben, wie ein Aufruf von *rBOINC* auszusehen hat, so dass über eine entfernte Verbindung und durch einen einfachen Aufruf bei einem jeweiligen Benutzer Arbeitspakete automatisch, ohne weiteres Hintergrundwissen für die Bearbeitung im BOINC-Projekt eingegliedert werden.

Manager würde den Fortschritt bis 50 % anzeigen und nach Fertigstellung von ZIP direkt zu 100 % springen.

Listing 13.4 Eingabeschablone für CINOLA@home

```
<rboinc application="wrapperCINOLA"
        description="Remote job's submission for CINOLA."/>
<file_info>
  <number>0</number>
</file_info>
<file_info>
  <number>1</number>
</file_info>
<workunit>
  <file_ref>
    <file_number>0</file_number>
    <open_name>mm.dat</open_name>
    <copy_file/>
    <rboinc parameter_name="mmDat"
            parameter_description="Input file with one parameter set for CINOLA
                .">
  </file_ref>

  <file_ref>
    <file_number>1</file_number>
    <open_name>dummy</open_name>
    <rboinc parameter_name="dummy"
            parameter_description="Do not set this variable."
            optional="true"/>
  </file_ref>

  <min_quorum> 2 </min_quorum>
  <target_nresults> 2 </target_nresults>
  <max_error_results> 1 </max_error_results>
  <max_total_results> 3 </max_total_results>
  <max_success_results> 2 </max_success_results>

  <rsc_fpops_bound>1000000000000</rsc_fpops_bound>
  <rsc_fpops_est>1000000000000</rsc_fpops_est>
</workunit>
```

In den vorherigen Abschnitten wird erwähnt, dass die Ergebnisdateien von CINOLA in ein ZIP-Archiv überführt werden. Die Ergebnisschablone in Listing 13.5 führt drei Dateien auf, wobei die letzte Datei dieses ZIP-Archiv beschreibt, zum BOINC-Projekt hochgeladen wird, sobald diese Datei erzeugt wurde, und maximal 8 MByte groß sein darf. Die ersten beiden Dateien sind die in der `job.xml` spezifizierten Dateinamen für die zwei Standardausgabekanäl. Sie werden für später evtl. notwendige Debugging-Arbeiten an den Projektserver übertragen und durch einen angepassten Assimilierer entsprechend abgespeichert. Diese beiden Ausgabedateien dürfen maximal 4 MByte groß sein.

Listing 13.5 Ergebnisschablone für CINOLA@home

```
<file_info>
  <name><OUTFILE_0/></name>
  <generated_locally/>
  <upload_when_present/>
  <max_nbytes>4096</max_nbytes>
  <url><UPLOAD_URL/></url>
</file_info>
<file_info>
  <name><OUTFILE_1/></name>
  <generated_locally/>
  <upload_when_present/>
  <max_nbytes>4096</max_nbytes>
  <url><UPLOAD_URL/></url>
```

```
</file_info>
<file_info>
  <name><OUTFILE_2/></name>
  <generated_locally/>
  <upload_when_present/>
  <max_nbytes>8192</max_nbytes>
  <url><UPLOAD_URL/></url>
</file_info>
<result>
  <file_ref>
    <file_name><OUTFILE_0/></file_name>
    <open_name>stdout</open_name>
  </file_ref>
  <file_ref>
    <file_name><OUTFILE_1/></file_name>
    <open_name>stderr</open_name>
  </file_ref>
  <file_ref>
    <file_name><OUTFILE_2/></file_name>
    <open_name>results.zip</open_name>
  </file_ref>
</result>
```

13.4 Arbeitspakete

13.4.1 Arbeitspakete erstellen

Es gibt nun wiederum die schon bekannten zwei Möglichkeiten, durch ein Skript oder eine eigene Implementierung Arbeitspakete für CINOLA bereitzustellen. Listing 13.6 enthält ein Skript zum Erstellen von Arbeitspaketen mit einer sequenziell hochgezählten Identifikationsnummer, innerhalb der for-Schleife wird diese genutzt, um eindeutig identifizierbare Arbeitspakete zu erhalten. In diesem trivialen Beispiel wird bei jedem Durchlauf die gleiche `mm.dat` verwendet, wodurch alle Arbeitspakete gleich arbeiten. Es steht Ihnen frei, dies für Ihre Bedürfnisse anzupassen.

Listing 13.6 Arbeitspakete für CINOLA@home erstellen

```
ROOTPROJECT=${HOME}/projects/apph
DUMMY=`pwd`/mm.dat

for i in {121..130}
do
  cd ${ROOTPROJECT}
    cp ${DUMMY} `bin/dir_hier_path mm.dat`
    cmd="./bin/create_work --appname wrapperCINOLA \
      --wu_name cinola_wu_${i} \
      --wu_template templates/CINOLA-input.tpl \
      --result_template templates/CINOLA-result.tpl \
      mm.dat dummy"
    echo $cmd
    $cmd
  cd -
  echo ""
done
```

13.4.2 Arbeitspakete mit rBOINC erstellen

In Abschn. 9.2.3.1 haben ich Ihnen rBOINC vorgestellt, so dass Sie Arbeitspakete aus der Ferne zu einem BOINC-Projekt hinzufügen können. Die Eingabeschablone aus Listing 13.4 enthält schon die notwendigen XML-Zeilen, um rBOINC für CINOLA nutzen zu können. Sie können nun im Vorfeld abfragen, welche Variablen für das Erstellen eines Arbeitspakets gesetzt sein müssen:

```
cr@visualgrid-4:~/rboinc$ ./boinc_submit.pl -url http://boinc-testserver/
    tah_rboinc -app_name wrapperCINOLA -help_parameters

Remote application queue 'wrapperCINOLA'
Description: Remote job's submission for CINOLA.
Application on server: 'wrapperCINOLA'

Options defined for this application queue:
   -dummy    (optional) Do not set this variable.
   -mmDat    Input file with one parameter set for CINOLA.

cr@visualgrid-4:~/rboinc$
```

Workarounds für rBOINC

rBOINC ist noch keine vollkommen stabile Software und es müssen einige Dinge beachtet werden. Im Folgenden finden Sie einige Workarounds, die Sie bei der Installation und Nutzung sicherlich benötigen.

Array-Fehler Im oberen Beispiel wird ein Dummy `dummy` verlangt, dies ist der in Abschn. 9.2.3.1 beschriebene Workaround, weil rBOINC mit Arrays arbeitet und dies bei nur einer angegebenen Eingabedatei zu einem Fehler führt. Außerdem ist es leider so, dass die Datei nicht optional ist und angegeben werden muss; eine leere Datei mit einem wahlfreien Namen reicht aus.

$delay_bound Es könnte passieren, dass Sie in Ihrem Apache-HTTP-Server Logfiles folgende Meldung stehen haben:

```
Use of uninitialized value \$delay_bound in concatenation (.) or string
    at /home/boincadm/projects/apph/cgi-bin/boinc_submit_server.pl line
    450.
```

In einem solchen Fall hilft es, wenn Sie in der Datei `boinc_submit_server.pl` die Variable direkt mit einem Wert initialisieren. Ein Hinzufügen des XML-Tags `<delay_bound>` in der Eingabeschablone ist leider nicht von Erfolg gekrönt. Setzen Sie den Wert einfach auf eine Woche `my $delay_bound=3600 * 24 * 7;`.

/bin/bash: process Eine solche Meldung kann bei dem Versuch erscheinen, ein Arbeitspaket hinzuzufügen:

```
Stderr
------

/bin/bash: process: No such file or directory
Detailed error information follows.
<Return DumpedCore="0"
        ExitCode="127"
        SignalNum="0"
        StdErr="/bin/bash: process: No such file or directory "
        StdOut="" />
```

In diesem Fall müssen Sie die Datei boinc_submit_server.pl modifizieren. Suchen Sie den Aufruf von `create_process_sh()` und kommentieren Sie die if-Bedingung um diesen Aufruf aus. Nun kann die Datei durch das Skript erstellt werden und steht dann zur Verfügung.

Virtueller Name Wenn Sie meinen, eine Dummy-Datei benötigt keinen virtuellen Namen, dann irren Sie sich. Wenn der XML-Tag `<open_name>` nicht vohanden ist, so wird folgende Meldung ausgegeben:

```
Stderr
------
No open name found process_wu_template: -112 create_work: -112
Detailed error information follows.
<Return DumpedCore="0"
        ExitCode="1"
        SignalNum="0"
        StdErr="No open name found process_wu_template: -112 create_work:
             -112 "
        StdOut="Processing input 'start myJob01' to wu 'myJob01-CR_Ries-0-1-
             RND6049' APPNAME: wrapperCINOLA , Submit failure for start: 1 "
             />
```

SQL-Injection Fügen Sie in Ihre beschreibenden Texte keine Hochkommata ein, dies führt unweigerlich zu einem Syntaxfehler in der SQL-Anweisung zum Hinzufügen eines Arbeitspakets in die Datenbank.

Wenn Sie das alles beachtet haben, dann antwortet Ihnen die Eingabe von:

```
cr@visualgrid-4:~/rboinc$ ./boinc_submit.pl -url http://boinc-testserver/
    apph_rboinc \
   -app_name wrapperCINOLA \
   -mmDat mm-iter1280.dat \
   -dummy dummy -group Ries -name myJob02
```

vielleicht mit:

```
Submission complete.

Stdout
------
Processing input 'start myJob02' to wu 'myJob02-CR_Ries-0-1-RND6400' Submitted
     myJob02-CR_Ries-0-1-RND6400

Stderr
------

cr@visualgrid-4:~/rboinc$
```

Diese ganzen Workarounds zeigen, was BOINC ist: ein Projekt von vielen kleinen Tüftlern, bei denen ein Proof-of-Concept im Vordergrund steht. Viele Lösungen sind speziell für ein BOINC-Projekt erstellt und darauf angepasst und können in den meisten Fällen nicht direkt für weitere Projekte installiert, konfiguriert und in Betrieb genommen werden. Seien Sie immer vorsichtig, wenn Sie Entwicklungsarbeiten zu Ihrem Projekt hinzufügen wollen, die Sie nicht auf Herz und Nieren geprüft haben.

13.5 Ergebnisverarbeitung

13.5.1 Validierung – Prüfung der Ergebnisse

CINOLA ist ein Werkzeug zur Berechnung von mathematischen Modellen, und die Basis sind unterschiedliche Zufallszahlengeneratoren. Es kann im Vorfeld keine Aussage über die Ergebnisse getroffen werden. Zudem ist das nicht Vorhersagbare auch das eigentliche Ziel von CINOLA, da neue Erkenntnisse gesucht und daraufhin tiefer erforscht werden sollen. Es ergibt sich, dass neue – zumindest in der Theorie – Erkenntnisse über die magnetische Wechselwirkung von Molekülen gefunden werden [22–24].

13.5.2 Assimilation – Ergebnisse abspeichern

Die Ergebnisse werden bei der Nutzung von rBOINC nicht nach der BOINC-Philosophie assimiliert. Es ist eher so, dass die Ergebnisse temporär zwischengespeichert werden und durch einen einfachen Befehl von jedem Wissenschaftler oder Simulanten auf den lokalen Rechner geladen werden können. Der Befehl lautet

```
cr@visualgrid-4:~/rboinc$ ./boinc_retrieve.pl \
    -url http://boinc-testserver/apph_rboinc \
    -app_name wrapperCINOLA -name myJob02 -group Ries
```

Die Ergebnisdateien werden auf dem Projektserver in der Standardeinstellung gelöscht. Wenn dies nicht gewünscht ist, so müssen Sie `-keep` als Parameter angeben. Weiterhin können Sie mit `-into` einen Ordner zum Abspeichern der Ergebnisse angeben.

Kapitel 14
ComsolGrid: COMSOL Multiphysics und BOINC

There is no alternative.

Margaret Thatcher, First woman prime minister of Great Britain and Northern Ireland

Dieses Kapitel gibt einen Überblick über eine junge Entwicklung der Arbeitsgruppe *Computational Materials Science & Engineering* (CMSE) [103] der Fachhochschule Bielefeld. Innerhalb der Arbeitsgruppe wird COMSOL Multiphysics für die Erstellung und Berechnung von physikalischen Simulationen verwendet [108]. COMSOL Multiphysics wird in unterschiedlichen Industriebereichen genutzt:

- Bei der NASA für die Simulation des anisotropen Dielektrikum und Ermittlung der Effekte in Metamaterialien in bestimmten Konstellationen [25],
- in der Chemieforschung für die Berechnung von chemischen Reaktionen [11],
- im Bereich der Informationstechnik und der Bearbeitung von Fragestellungen bzgl. der Optimierungsmöglichkeiten [20].

COMSOL Multiphysics beherrscht das Lösen von gekoppelten Differentialgleichungssystemen (DGL) mit Hilfe der Finite-Element-Methode (FEM). Mit der FEM-Methode wird ein ein-, zwei- oder drei-dimensionales Modell mit einem sog. Mesh (zu Deutsch „Gitter“) überzogen. Jeder Quadrant dieses Gitters kann für sich alleine berechnet werden, die Einzelergebnisse werden am Ende zu einem Gesamtergebnis zusammengeführt. Je größer das Mesh eines erstellten Modells definiert ist, umso länger kann die Berechnung dauern. Ein recht komplexes Modell kann schon mal einige tausende bis Millionen Quadranten besitzen. Es hängt u. a. von der gewählten Anzahl ab, wie lange das Lösen eines solchen FEM-Modells benötigt, dies können Sekunden, Minuten oder auch Tage sein.

Was wir benötigen, ist ein FEM-Modell, das durch eine externe Beschreibung variiert werden kann. COMSOL Multiphysics unterstützt diese Möglichkeit und wir können über grafische Editoren bestimmen, welche Elemente innerhalb des FEM-Modells variiert werden dürfen. Daraufhin kann COMSOL Multiphysics innerhalb einer Eingabeaufforderung oder eines Terminals mit dem erstellten FEM-Modell gestartet werden, und zusätzlich kann der Aufruf soweit modifiziert werden, dass die Modellparameter für die Laufzeit spezifiziert werden können.

Diese Möglichkeit erlaubt uns das Kombinieren von COMSOL Multiphysics mit BOINC, um so hochskalierbare sog. Parameterstudien zu erstellen und zu starten. Der Wunsch war, dass dadurch ein optimaler Parametersatz für ein FEM-Modell

C. B. Ries, *BOINC*, Xpert.press,
DOI 10.1007/978-3-642-23382-8_14, © Springer-Verlag Berlin Heidelberg 2012

ermittelt werden kann, und das möglichst schnell [16, 20]. Mit der Hilfe von BOINC können wir nun einzelne Parametersätze in Arbeitspakete packen und an unsere BOINC-Teilnehmer verschicken, um sie dort lösen zu lassen.

Es werden nicht alle Aspekte von ComsolGrid behandelt, denn viele Techniken und zu bedenkende Aspekte wurden schon in den vorherigen Kapiteln erwähnt und diskutiert. Es werden nur als wichtig eingestufte Techniken und Implementierungsdetails beschrieben. Die Idee und die Umsetzungen haben wir ***ComsolGrid*** getauf; dies kann von einer Sourceforge-Seite heruntergeladen werden [105].[1]

Desktop Grid für COMSOL Multiphysics

Eine große Problematik ist zu lösen – bedenken Sie, dass die kleinste Installation von COMSOL Multiphysics rund 1,4 Giga-Byte beträgt. Wir können es den BOINC-Teilnehmern nicht zumuten, sich das für die Teilnahme von unserem BOINC-Projektserver herunterzuladen; ebenso wäre dies für unsere Performance ein Problem, wenn auf einen Schlag hunderte von Anfragen eintreffen. Unsere Server wären rasch nicht mehr verfügbar.

ComsolGrid eignet sich u. a. wegen der genannten Problematik hervorragend für die Erstellung eines sog. Desktop-Grid[2]. Solche eine Installation hat den Vorteil, dass wir die einzelnen teilnehmenden Rechner konfigurieren können und auf jedem Rechner eine COMSOL Multiphysics Installation vornehmen können, das Herunterladen dieser Anwendung würde dann entfallen. Ein weiterer Ansatz wäre, dass wir nur eine Installation vornehmen und den Installationsordner über das Netzwerk auf jedem Host einbinden. Der Vorteil dieser Lösung wäre das Einsparen von Wartungsarbeiten, da nur ein Installationsort gepflegt werden müsste. Eine solche Verteilung ist u. a. mit dem Network File System (NFS) möglich und wird auch in diesem Praxisbeispiel genutzt.

Eine weitere Problematik ist die nicht vorhandene Schnittstelle zwischen COMSOL Multiphysics zu BOINC. Wir könnten den Wrapper aus Kap. 13 konfigurieren und nutzen, allerdings unterstützt der Wrapper nicht alle Anforderungen aus dem Abschn. 14.2.

14.1 Grundidee

Abbildung 14.1 zeigt die Idee einer COMSOL-Multiphysics- und BOINC-Konstellation. Wir sehen hier, dass ComsolGrid die BOINC-Anwendungen nutzt und zusätzlich eine *ComsolGridFCGI (CGFCGI)-Schnittstelle*, eine grafische Oberflä-

[1] http://comsolgrid.sourceforge.net.

[2] Ein Desktop Grid verwendet nur die Rechner, die sich im Netz einer Domäne befinden. In der Regel werden diese Rechner auf der Softwareseite für die Nutzung von speziellen Anwendungen konfiguriert.

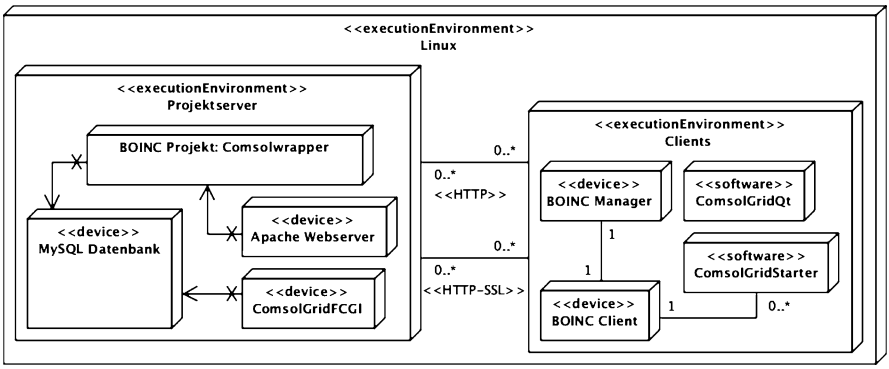


Abb. 14.1 Komponenten eines ComsolGrid-Projekts

che *ComsolGridQt* und eine Wrapper-Implementierung ComsolGridStarter besitzt. Die ersten beiden ComsolGrid-Anwendungen werde ich nicht im Detail vorstellen, da sie spezielle Lösungen innerhalb der Infrastruktur unserer Arbeitsgruppe darstellen. Einige Worte kann ich allerdings darüber verlieren:

ComsolGridFCGI Im Abschn. 9.5.1 haben wir diskutiert, wie man eigene Datenbanktabellen zu einem BOINC-Projekt hinzufügen kann. Die Authentifizierung von Anwendern mit bestimmten Rollendefinitionen wurde umgesetzt; dies ist ein Teil von CGFCGI. Mit Hilfe dieser Schnittstelle können Anwender neue Arbeitspakete zu ComsolGrid hinzufügen.

ComsolGridQt Eine Qt [110]-basierende grafische Oberfläche, der zweite Verbindungspartner in der Kommunikation zur CGFCGISchnittstelle. Innerhalb dieser Anwendung können die im FEM-Modell definierten Variablen spezifiziert werden, es wird eine Verbindung mit der CGFCGI-Schnittstelle aufgebaut und die Parametersätze der Arbeitspakete übertragen. Daraufhin erstellt die serverseitige Schnittstelle mit Hilfe der in Abschn. 9.2.2 erwähnten Funktionen neue Arbeitspakete für das ComsolGrid-Projekt. Die Kommunikation zwischen diesen beiden Anwendungen wird durch das HTTP-SSL (Hypertext Transfer Protocol with Secure Socket Layer)-Protokoll verschlüsselt.

14.2 Anforderungen an ComsolGrid

Wir könnten COMSOL Multiphysics einfach installieren, den Wrapper aus Kap. 13 konfigurieren und das Ganze laufen lassen. Es würde einfach nicht funktionieren und Sie hätte keine Freunde an diesem sehr trivialen Satz. Einige Probleme würden Sie in kürzester Zeit ausmachen:

- Der Status der aktuellen Berechnung würde nicht angezeigt werden.
- Es gibt keine Möglichkeit, um die Ausführung von COMSOL Multiphysics zu pausieren,

- entsprechend können Sie eine Berechnung dann auch nicht weiterführen.
- Das Beenden oder Abbrechen einer Berechnung klappt nicht, im Hintergrund wird die Berechnung weitergeführt.

Diese kleine Auflistung liefert die Hauptgründe dafür, dass eine eigene Implementierung nötig ist, um eine voll funktionsfähige COMSOL-Multiphysics-Berechnung durchzuführen.

14.3 ComsolGridStarter

Die im vorherigen Abschnitt genannten Probleme können durch eine eigene Implementierung eines Wrappers für COMSOL Multiphysics gelöst werden; dieser wird hier als ComsolGridStarter vorgestellt und es wird erläutert, auf welche Probleme Sie achten müssen. Die in diesem Abschnitt erläuterten Techniken und Ideen können Sie für die Implementierung eigener Wrapper nutzen, so dass Sie evtl. eigene entwickelte Applikationen ohne Modifikation nutzen können, wenn der mitgelieferte BOINC-Wrapper nicht für Ihre Applikation funktioniert.

14.3.1 Prozesskontrolle

Die Abb. 14.2 enthält das abstrakte Aktivitätendiagramm der ComsolGridStarter-Implementierung und gibt einen Überblick über die Komplexität der darunterliegenden Architektur. Die Architektur kann grob in drei Bereiche unterteilt werden:

1. Den BOINC-Manager, der das Starten und Steuern von ComsolGridStarter übernimmt,
2. die Implementierung des Wrapper „ComsolGridStarter“ selbst und
3. die COMSOL-Multiphysics-Simulationssoftware und die damit einhergehende virtuelle Java-Maschine[3].

COMSOL Multiphysics ist durch die Nutzung von Java unabhängig von der Plattform ausführbar. Diese Unabhängigkeit wird durch das Verwenden einer virtuellen Maschine erreicht, die zwischen den Programmbefehlen der Ausführung und der Rechnerarchitektur gestartet wird und die Umsetzung der Programmbefehle zu den spezifischen Instruktionen einer Rechnerarchitektur übernimmt. Das ist auch schon das Problem der eingangs erwähnten Wrapper-Implementierungen aus Abschn. 14.2, mit denen die COMSOL-Multiphysics-Simulationssoftware nicht steuerbar ist.

[3] Die COMSOL-Multiphysics-Simulationssoftware ist in Java programmiert.

Listing 14.1 zeigt die Prozessstruktur einer COMSOL-Multiphysics-Ausführung der Major-Release Version 3.5x, und das Listing 14.2 zeigt die Prozessstruktur eines Major-Release in Version 4.0a. Die Zahlen in den Klammern sind die Prozessidentifikationsnummern (PID). Es ist zu sehen, dass COMSOL mit der PID(30521) läuft und als Kindprozess eine Java-Instanz mit der PID(30674) besitzt. Diese Java-Instanz besitzt weitere Java-Kindprozesse. Die Implementierung aus Kap. 13 arbeitet mit der PID(30521), um das Programm zu steuern. Wenn der Prozess mit der PID(30521) das Signal `SIGSTOP` empfängt, würde dieser Prozess stoppen, aber der Java-Prozess mit der PID(30674) und die Kindprozesse arbeiten in diesem Fall weiter.

Listing 14.1 Prozessstruktur von COMSOL Multiphysics 3.5x nach dem Start

```
su(30443)---bash(30452)---comsol(30521)---java(30674)-+-{java}(30675)
                                                      |-{java}(30676)
                                                      `-...
```

Für das weitere Verständnis ist es wichtig zu wissen, dass das Signal `SIGSTOP` nicht wirklich stoppt; im Sinne von beenden, was man annehmen könnte. Dieses Signal pausiert die Ausführung, die zu einem späteren Zeitpunkt wieder aufgenommen werden kann. Der pausierte Prozess verbleibt während dieser Zeit im Speicher.

Listing 14.2 Prozessstruktur von COMSOL Multiphysics 4.0a und höher nach dem Start

```
...-+-comsol(15895)---comsollauncher(15919)-+-{comsollauncher}
    |                                       |-{comsollauncher}
    |                                       |-{comsollauncher}
    |                                       `-...
```

Bei der COMSOL Multiphysics Version 4.0a und neueren verhält es sich gleich. Ein `SIGSTOP` für den Prozess PID(15895) stoppt nicht den gesamten Prozess, es wird die PID(15919) benötigt. Der COMSOL-Multiphysics-Prozess ist somit nicht auf diesem Wege steuerbar. Wenn allerdings der Java-Prozess mit der PID(30674) ein `SIGSTOP` empfängt, dann wird der Prozess inklusive der Kindprozesse gestoppt.

An dieser Problematik setzt die Implementierung von ComsolGridStarter an, ermittelt nach dem Starten von COMSOL Multiphysics die relevante PID der Java-Prozesse und verwendet diese PIDs für die Prozesssteuerung. Zu diesem Zweck wurde die Klasse `ComsolGrid::Task` zur Steuerung des Prozesses entwickelt.

Anwendungen, die durch den BOINC-Client (nachfolgend „BC“ genannt) gestartet werden, stehen mit dem BC in Kontakt. Der BC steuert das gestartete Programm durch Statusnachrichten. Dafür ist ein Kommunikationskanal vorhanden, der zwischen dem BC und dem Wrapper Daten austauscht und dafür Shared-Memory (SHM) (vgl. Abschn. 7.3.14) verwendet. Diese Informationen können kontinuierlich abgefragt werden, wenn vorher konfiguriert wurde, dass diese explizit vom Anwendungsentwickler zu handhaben sind.

Die bis hier erwähnten Charakteristika über die Prozessstruktur und implementierten BOINC-Funktionalitäten werden in Listing 14.3 angewendet. Das Listing 14.3 enthält eine gekürzte Form der ComsolGridStarter-Implementierung. In der `main`-Funktion werden im ersten Abschnitt die BOINC-Optionen gesetzt, so dass der Wrapper die Prozesskontrolle übernehmen kann (Zeilen 3–15). Ab Zeile 17 beginnt die Funktionalität von ComsolGridStarter. Es wird eine Instanz der Pro-

zesskontrollklasse erstellt, die Konfiguration von ComsolGridStarter in der Struktur `cxml` übergeben, der Pfad und die Parameter für die Ausführung von COMSOL Multiphysics angegeben und die Prozesspriorität[4] gesetzt.

In Zeile 22 wird COMSOL Multiphysics gestartet und ab Zeile 24 kontinuierlich geschaut, wie weit der Fortschritt der Simulation (Zeile 28 und 29) ist und welche Steuerungsbefehle vom BC geschickt werden (Zeile 31). Die Funktion aus Zeile 31 ist ab Zeile 37 aufgeführt. Sie liest den Status aus, prüft die Werte in der Status-Struktur und reagiert in den Fallunterscheidungen passend auf die einzelnen Werte. Die Instanz `task` zur Prozesskontrolle besitzt Methoden, um den Java-Prozess zu beenden (`SIGKILL`, `SIGINT`), zu stoppen (`SIGSTOP`) und weiterarbeiten (`SIGCONT`) zu lassen.

Listing 14.3 Aktivierung der implementierten Prozesskontrollmechanismen

```
1  int main(int argc, char **argv){
     ...
     BOINC_OPTIONS options;
     memset(&options, 0, sizeof(options));
     // Es handelt sich um die Hauptanwendung.
6    options.main_program = true;
     // Pruefen ob Verbindung zum Client besteht.
     options.check_heartbeat = true;
     // Der Entwickler handhabt die Programmsteuerung.
     options.handle_process_control = true;
11   // Status Nachrichten senden, {\zb} fraction_done.
     options.send_status_msgs = true;
     // Damit die oberen Optionen greifen!
     options.direct_process_action = false;
     boinc_init_options(&options);
16   ...
     comsolTask = new ComsolGrid::Task();
     comsolTask->setComsolXML(&cxml);
     comsolTask->setComsolBinary(comsolBinary, &args);
     comsolTask->setPriority(PROCESS_IDLE_PRIORITY);
21
     if(!comsolTask->run()) { /* Error handling */ }

     while(comsolTask->isRunning()) {
       // read out the fraction done -> BOINC client
26     fractionDone = comsolTask->getFractionDone();

       boinc_fraction_done(fractionDone);
       boinc_report_app_status( 0.0, 0.0, fractionDone );

31     poll_boinc_messages(comsolTask);

       boinc_sleep(1.0);
     }
   }
36
   void poll_boinc_messages(ComsolGrid::Task *task) {
     ...
     BOINC_STATUS status;
     boinc_get_status(&status);
41
     if(status.no_heartbeat)  { task->kill(); exit(0); }
     if(status.quit_request)  { task->kill(); exit(0); }
     if(status.abort_request) { task->kill(); exit(0); }
```

[4] Nähere Informationen zu diesem Parameter sind in den Manpages der Funktionen `getpriority` und `setpriority` zu finden, `$ man 2 setpriority`.

```
46    // COMSOL Multiphysics pausieren oder
      // weiterlaufen lassen.
      if(status.suspended) {
            if(!task->isSuspended()) {
                task->stop();
51          }
        } else {
            if(task->isSuspended()) {
                task->resume();
            }
56      }
}
```

Das Aktivitätendiagramm in Abb. 14.2 verdeutlicht den Verlauf und die Prinzipien der Implementierung von ComsolGridStarter. Mit dem BOINC-Manager können Befehle über den BC an den ComsolGridStarter geschickt werden. Es wird kontinuierlich geprüft, ob Arbeitspakete (engl. *Workunits*) vorhanden sind; falls ja, wird geschaut, wo auf dem System die COMSOL-Multiphysics-Simulationssoftware installiert ist, und diese dann ausgeführt. Der ComsolGridStarter erstellt einen zweiten Prozess. Dieser führt die COMSOL-Multiphysics-Simulationssoftware aus. COMSOL Multiphysics startet eine virtuelle Java-Maschine, um die Simulation durchzuführen (oberer Kasten „InitializeComsol()“). Der Speicherbereich dieser Startfunktion wird durch COMSOL Multiphysics ersetzt. Die PID des Prozes-

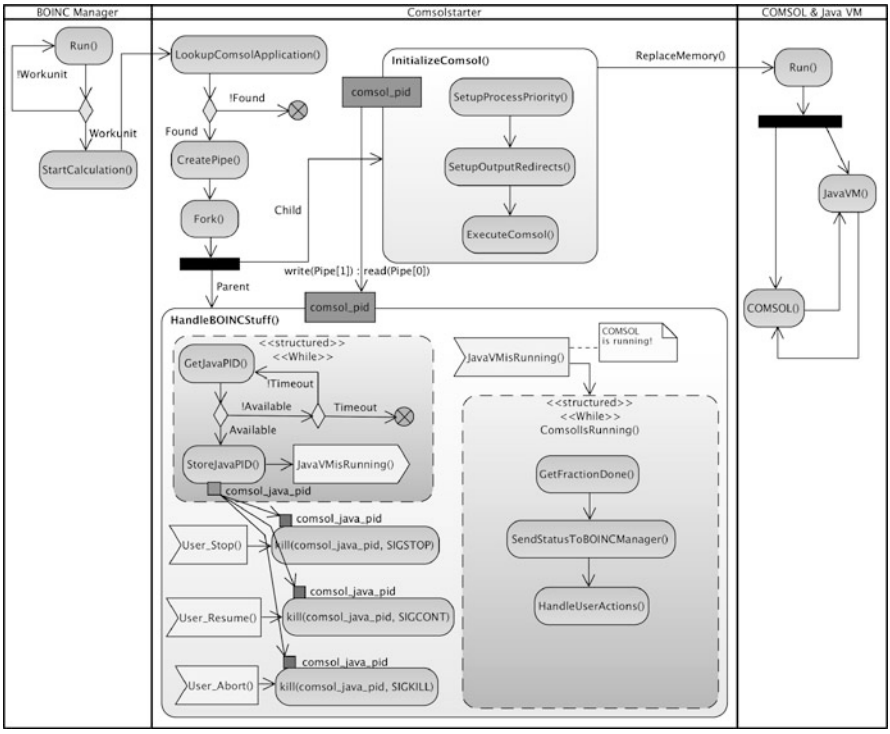


Abb. 14.2 Ablaufdiagramm der Implementierung von *ComsolGridStarter*

ses von COMSOL Multiphysics wird an die „HandleBOINCStuff()“-Darstellung weitergegeben. Diese sucht im Prozessbaum nach der PID der virtuellen Java-Maschine und hält diese für spätere Aktivitäten des Benutzers bereit. Wenn der Benutzer die Durchführung stoppen (Aktivität „User_Stop()“), weiterführen (Aktivität „User_Resume()“) oder beenden (Aktivität: „User_Abort()“) will, wird diese PID verwendet und das passende Signal an den richtigen Prozess geschickt – eine volle Kontrolle der Durchführung der COMSOL-Multiphysics-Simulationssoftware ist möglich. Weiterhin ist eine manuelle Steuerung der *ComsolGridStarter*-Applikation möglich, wenn diese nicht in Verbindung mit dem BC ausgeführt werden soll. Dafür musste das Signal `SIGSTOP` umdefiniert werden. Das Signal kann nicht mit einem Signal-Handler abgefangen werden, so dass die Ausführung einer Applikation immer sofort zur Beendigung führt. Wenn die Applikation *ComsolGridStarter* pausiert werden soll, muss nach Definition das Signal `SIGUSR1` verwendet werden. Das Signal zum Fortführen der Ausführung `SIGCONT` ist ohne Probleme anwendbar. Die Implementierung des Signal-Handlers für die manuelle Verwendung ist in Listing 14.4 nachzulesen.

Listing 14.4 Signalhandler für die manuelle Signalverarbeitung

```
void signal_handler(int signr) {
  if(comsolTask != NULL) {
    if(signr == SIGUSR1 ||
       signr == SIGCONT ||
       signr == SIGSTOP ) {
         comsolTask->signalHandler(signr);
    }
  }
}
```

14.3.2 Start- und Simulationsparameter

COMSOL Multiphysics kann unter verschiedenen Betriebssystemen und Systemarchitekturen gestartet werden. Es werden von der Firma COMSOL hauptsächlich die Betriebssysteme Linux, Windows und Mac OS unterstützt, jeweils in der 32-Bit- und 64-Bit-Version. Die Systemarchitektur kann beim Starten von COMSOL Multiphysics angegeben werden. Dies wird empfohlen, um eine höhere Ausführungsgeschwindigkeit zu erzielen. Bei der Ausführung von ComsolGridStarter wird die Datei `comsol.xml` eingelesen und die darin enthaltenen Werte verarbeitet, in Listing 14.5 ist ein Beispiel dieser Datei aufgelistet.

Listing 14.5 Extended Backus–Naur Form (EBNF) [50] Struktur und Beispiel einer `comsol.xml`-Konfigurationsdatei

```
OUTPUT:
  <output>%s</output>

PARAMETER:
  <param>%s</param>
  [PARAMETER]

GLOBALS:
```

```
  <globals>
  [PARAMETER]
  </globals>

XML_comsol_cfg:
  <comsol>
    <version>3.5a | 4.0 | 4.0a</version>
    <32bit/> | <64bit/>          <!-- Beispiel -->
    <ckl/>
    <cores>6</cores>
    <model>comsol.mph</model>    <!-- Beispiel -->
    [OUTPUT]?
    <parameters>
      <file>comsol.txt</file>    <!-- Beispiel -->
      <param>maxT</param>        <!-- Beispiel -->
      <param>objWidth</param>    <!-- Beispiel -->
      <param>objHeight</param>   <!-- Beispiel -->
      [PARAMETER]*
    </parameters>
    [GLOBALS]?
    <!-- Beispiele -->
    <logfile>comsol.log</logfile>
    <stdout_filename>comsol_stdout.log</stdout_filename>
    <stderr_filename>comsol_stderr.log</stderr_filename>
</comsol>
```

Mit dem XML-Tag `<version>` ermittelt ComsolGridStarter die zur Simulationsdatei `<model>` passende COMSOL Multiphysics Major-Version, z. B. Version $\geq$ 4.x. Das XML-Tag `<ckl/>` ist bei der Anwendung in der Fachhochschule Bielefeld von Bedeutung, da dort *Floating-Licenses*[5] verwendet werden. Der XML-Tag `<cores>` soll dazu dienen, n-Prozessoren für eine ComsolGridStarter-Ausführung zu reservieren. Das Reservieren von Prozessoren wird von dem BOINC-Framework in der verwendeten Revision 22328 noch nicht unterstützt. Durch Erstellung einer weiteren Abstraktionsschicht oder Modifikation des BOINC-Frameworks könnte diese Funktion implementiert werden, es müssen sich nur Freiwillige melden. Mit dem XML-Tag `<parameters>` wird ein Bereich eröffnet, der die Simulationsparameter eines FEM-Modells beschreibt. Die Werte der variablen Parameter eines Simulationsmodells sind in der durch `<file>` beschriebenen Eingabedatei enthalten.

Das Format der Eingabedatei ist wichtig. Jede Zeile beschreibt die Simulationsparameter für einen Simulationslauf. Die Anzahl der zu variierenden Simulationsparameter ist gleich der Anzahl an Spalten in dieser Datei, d. h. jede Spalte symbolisiert einen Simulationsparameter. Diese Simulationsparameter sind jeweils mit einem Tabulatorsprung voneinander getrennt, es können mehrere, aber mindestens einer verwendet werden. Es darf keine Leerzeile am Ende vorkommen. Im Abschn. 14.5.1 wird näher auf diese Datei eingegangen.

Alle weiteren Parameter werden beim Aufrufen von COMSOL Multiphysics verwendet. `<logfile>` gibt die Datei an, in die Protokolldaten geschrieben werden. Die XML-Tags `<(stdout, stderr)_filename>` definieren die Abspeicherungsorte für Standard- und Fehlerausgaben während der Ausführung der Berech-

[5] Im Netzwerk ist ein Lizenzserver mit einer Anzahl N vorhandener Lizenzen installiert. Beim Starten einer lizenzpflichtigen Anwendung wird eine Verbindung zu diesem Server aufgebaut und geschaut, ob eine passende Lizenz vorliegt oder noch frei ist.

nung eines FEM-Modells. Die Log- und die Standardfehlerausgabedatei sind nicht weiter beachtenswert, bei der Standardausgabedatei ist der Fall anders. Darauf wird in Abschn. 14.3.3 näher eingegangen.
Aus den in Listing 14.5 aufgeführten Parametern mit der Textmarkierung *Beispiel* wird folgender Befehl erstellt, der direkt für die Ausführung verwendet werden kann und eine Berechnung startet:

```
/usr/local/comsol35a/bin/comsol -ckl -64 batch -input comsol.mph -paramfile comsol.txt
"maxT,objWidth,objHeight" -logfile comsol.log
```

Es wird das FEM-Modell in der Datei `comsol.mph` mit den Parametersätzen aus der Datei `comsol.txt` im sogenannten *Batch-Modus* berechnet.[6]

14.3.3 Prozessfortschritt

Die Standardausgabedatei `<stdout_filename>` aus dem Listing 14.5 wird für die Ermittlung des Berechnungsfortschritts verwendet. Eine Beispielausgabe von COMSOL Multiphysics ist in Listing 14.6 aufgeführt.

Listing 14.6 Standardausgabe bei der Ausführung von COMSOL Multiphysics im *Batch-Modus*

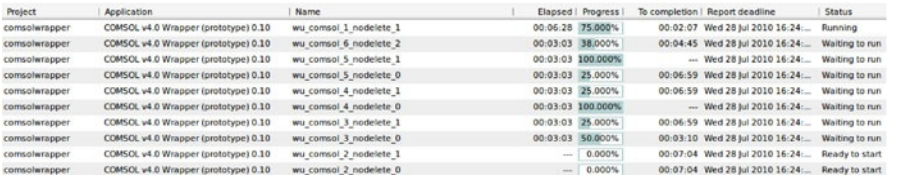
```
COMSOL Batch (64-bit)
2 ------------------------------------------------------------
  Version: COMSOL 3.5a (COMSOL 3.5.0.603)

  Patent pending.
  Copyright (c) 1998-2008 by COMSOL AB.
7 All rights reserved.
------------------------------------------------------------

  Starting batch job.
------------------------------------------------------------
12 Solve Problem
  Current Progress:          0 %
  Updating extended mesh
  Current Progress:          0 %
  ...
17 ...
  Matrix factorization
  Current Progress:          1 %
     20      2.0858     0.36232       36    9   36    2    1    0
     21      2.4482     0.36232       37    9   37    2    1    0
22 ...
```

Der Funktionsaufruf `comsolTask->getFractionDone()` in Listing 14.3, Zeile 26, öffnet diese Datei, springt an das Ende, liest rückwärts x Bytes und sucht nach der Zeichenkette `Current Progress:`. Bei einer gefundenen Zeichenkette wird der Prozentwert ausgelesen und an die aufrufende Funktion zurückgegeben. Dieser Wert wird mit einer Funktion des BOINC-Frameworks `boinc_report_app_status()` an den BOINC-Client gesendet. Der Wert wird mit dem

[6] Der Batch-Modus bei COMSOL Multiphysics arbeitet sequenziell die Parametersätze aus einer übergebenen Datei ab [16, 106].



Project	Application	Name	Elapsed	Progress	To completion	Report deadline	Status
comsolwrapper	COMSOL v4.0 Wrapper (prototype) 0.10	wu_comsol_1_nodelete_1	00:06:28	75.000%	00:02:07	Wed 28 Jul 2010 16:24:...	Running
comsolwrapper	COMSOL v4.0 Wrapper (prototype) 0.10	wu_comsol_6_nodelete_2	00:03:03	38.000%	00:04:45	Wed 28 Jul 2010 16:24:...	Waiting to run
comsolwrapper	COMSOL v4.0 Wrapper (prototype) 0.10	wu_comsol_5_nodelete_1	00:03:03	100.000%	---	Wed 28 Jul 2010 16:24:...	Waiting to run
comsolwrapper	COMSOL v4.0 Wrapper (prototype) 0.10	wu_comsol_5_nodelete_0	00:03:03	25.000%	00:06:59	Wed 28 Jul 2010 16:24:...	Waiting to run
comsolwrapper	COMSOL v4.0 Wrapper (prototype) 0.10	wu_comsol_4_nodelete_1	00:03:03	25.000%	00:06:59	Wed 28 Jul 2010 16:24:...	Waiting to run
comsolwrapper	COMSOL v4.0 Wrapper (prototype) 0.10	wu_comsol_4_nodelete_0	00:03:03	100.000%	---	Wed 28 Jul 2010 16:24:...	Waiting to run
comsolwrapper	COMSOL v4.0 Wrapper (prototype) 0.10	wu_comsol_3_nodelete_1	00:03:03	25.000%	00:06:59	Wed 28 Jul 2010 16:24:...	Waiting to run
comsolwrapper	COMSOL v4.0 Wrapper (prototype) 0.10	wu_comsol_3_nodelete_0	00:03:03	50.000%	00:03:10	Wed 28 Jul 2010 16:24:...	Waiting to run
comsolwrapper	COMSOL v4.0 Wrapper (prototype) 0.10	wu_comsol_2_nodelete_1	---	0.000%	00:07:04	Wed 28 Jul 2010 16:24:...	Ready to start
comsolwrapper	COMSOL v4.0 Wrapper (prototype) 0.10	wu_comsol_2_nodelete_0	---	0.000%	00:07:04	Wed 28 Jul 2010 16:24:...	Ready to start

Abb. 14.3 BOINC-Manager mit Darstellung des *fraction_done*-Wertes von *ComsolGridStarter*

BOINC-Manager ausgetauscht und dann dort in der grafischen Ausgabe angezeigt (vgl. Abb. 14.3).

14.4 BOINC-Manager Modifikationen

Ein ComsolGrid-Projekt kann verschiedene Parameterstudien besitzen, z. B. eine Studie für einen Temperaturübergang durch Glas, jeweils in einem Modell eines COMSOL Multiphysics 3.5 oder 4.0 Major-Release. Die Modelle sind untereinander nicht kompatibel und es muss für die jeweilige Datei die richtige COMSOL-Version installiert sein. Der Benutzer hat eventuell nur eine von beiden installiert und kann von daher die Simulation nicht durchführen. Es ist unsinnig, dass dieser Benutzer Arbeitspakete vom Projektserver erhält, wenn er diese nicht bearbeiten kann.

Um dieses Problem handhaben zu können, wurde der BOINC-Manager modifiziert. Die BOINC-Quellen enthalten im Unterordner `client/` u. a. die Quellen des BOINC-Managers. In der Datei `./cs_platform.cpp` werden die Informationen der verwendeten bzw. unterstützten Systemplattformen ermittelt, z. B. ob es sich um ein Linux- oder Windows-Betriebssystem handelt, auf dem der BOINC-Manager installiert ist. Diese Funktion wurde um den Teil in Listing 14.7 erweitert. Dieser kurze Teil durchläuft die gefundenen Plattformdefinitionen (gespeichert in einem C++-Container `platforms` [105]) und fügt jeweils die gefundenen COMSOL-Multiphysics-Versionen hinzu.

Listing 14.7 Modifikation am BOINC-Manager zur Ermittlung der COMSOL-Versionen

```
//
// Add COMSOL versions to founded platforms.
//
ComsolGrid::VersionsList versions = ComsolGrid::lookup();

int s = platforms.size();

for(unsigned int i=0; i<s; i++) {
  std::string pname = platforms.at(i).name;
  for(unsigned int j=0; j<versions.size(); j++) {
    std::string pname_comsol = pname;
    pname_comsol += "_comsol";
    std::string major = versions.at(j).getMajor();
    // Removes the '.' character, not supported by BOINC.
    major.erase(1,1);
    pname_comsol += major;
```

```
      fprintf(stdout, "Add platform: %s\n", pname_comsol.c_str());
18    add_platform(pname_comsol.c_str());
    }
  }
```

Werden z. B. zwei Plattformen mit den nachfolgenden Definitionen gefunden,

```
Linux running on an Intel x86-compatible CPU
  -> i686-pc-linux-gnu

Linux running on an AMD x86_64 or Intel EM64T CPU
  -> x86_64-pc-linux-gnu
```

so werden diese jeweils um eine Definition erweitert; wenn z. B. die COMSOL Multiphysics Version 3.5a installiert ist, dann wird jeweils die Zeichenkette `_comsol35a` angehängt:

```
...x86-compatible CPU with COMSOL v3.5a
  -> i686-pc-linux-gnu_comsol35a

...x86_64 or Intel EM64T CPU with COMSOL v3.5a
  -> x86_64-pc-linux-gnu_comsol35a
```

Die wissenschaftlichen Applikationen müssen im Projekt mit eben diesen Plattformdefinitionen und in der Datenbank des Projekts hinzugefügt sein. In den Namen darf kein Punkt vorhanden sein, dieser wird von dem BOINC-Framework als Trennzeichen erkannt. Aus diesem Grund wird im Listing 14.7 dieser Punkt mit `major.erase(1,1)` entfernt. Das Vorgehen zum Hinzufügen von Plattformdefinitionen innerhalb eines BOINC-Projekts geschieht im Projektverzeichnis durch den Befehl `$./bin/xadd` (vgl. Abschn. 5.6.1.3) und wird hier nicht weiter vertieft.

In der Datei für die Protokollierung `stdoutdae.txt`[7] des BOINC-Managers wird die Meldung aus Listing 14.8 ausgegeben, um zu zeigen, welche COMSOL-Multiphysics-Versionen gefunden wurden.

Listing 14.8 Beispiel der durch den BOINC-Manager hinzugefügten Plattformdefinitionen

```
Add platform: x86_64-pc-linux-gnu_comsol35a
Add platform: x86_64-pc-linux-gnu_comsol40
Add platform: x86_64-pc-linux-gnu_comsol40a
Add platform: i686-pc-linux-gnu_comsol35a
5 Add platform: i686-pc-linux-gnu_comsol40
Add platform: i686-pc-linux-gnu_comsol40a
```

14.5 Arbeitspakete

14.5.1 Erstellung von Arbeitspaketen

Wir wissen schon durch den Abschn. 9.2, wie Arbeitspakete für ein BOINC-Projekt erstellt und hinzugefügt werden können. ComsolGrid verwendet eine angepasste C/C++-Lösung für das Erstellen von Arbeitspaketen, um so eine bessere Anpassung an die Systemumgebung zu ermöglichen. Arbeitspakete werden nicht durch

[7] Diese Datei wird im Ordner des BOINC-Managers erstellt.

einen manuellen Aufruf auf der BOINC-Serverseite erstellt, sondern indirekt durch das Verwenden der schon erwähnten ComsolGridQt-Benutzeroberfläche. Erstellte Arbeitspaketdefinitionen werden an die ComsolGridFCGI-Schnittstelle übertragen, und dort werden die Arbeitspakete entsprechend erstellt und ComsolGrid hinzugefügt.

Ein Arbeitspaket bei BOINC besteht aus ein oder mehreren Simulationsdateien und entsprechenden Parameter-Eingabedateien. Diese sind für die Simulation elementar und im Falle von *ComsolGrid* entsprechen diese Dateien einer COMSOL-Multiphysics-Simulation und einer zusätzlichen Datei für die Parametersätze eines FEM-Modells. Abbildung 14.4 enthält die Beschreibung eines beispielhaften Arbeitspakets. Das Arbeitspaket hängt in diesem Fall von vier Dateien ab:

1. Zwei Schablonendateien (s. Anhang 15.3),
2. einem FEM-Modell und
3. einer einfachen Textdatei, die die Parameter für das FEM-Modell enthält.

Bei der Erstellung eines Arbeitspakets ist die Reihenfolge der anzugebenden Dateien sehr wichtig! In dem unteren, linken Kommentar in Abb. 14.4 ist eine Beispieldarstellung einer Eingabeschablonendatei enthalten. Dort sind zwei Dateien angegeben, die jeweils durch den XML-Tag `<number>` einen Index besitzen. Die Datei `comsol.mph` hat die Indexnummer Null und `comsol.txt` die Indexnummer 1. Das heißt für die Arbeitspaketerstellung, dass erst die `comsol.mph`- und dann die `comsol.txt`-Datei angegeben werden muss. Im ComsolGridStarter werden die Dateinamen `comsol.mph` und `comsol.txt` zum Öffnen der Dateien angenommen. Dies sind logische Namen im BOINC-Framework und werden durch die schon bekannte Funktion `boinc_resolve_file_s()` in die physikalischen Pfade aufgelöst. Angenommen, es befinden sich eine Simulationsdatei `edge2d.mph` und eine Parameterdatei `wu291_values` in dem Unterordner `workunits2add/`, dann würde die Befehlsreihenfolge in Listing 14.9 ein Arbeitspaket erstellen; die Daten sind angelehnt an die Namen in Abb. 14.4.

Listing 14.9 Manuelle Erstellung eines Arbeitspakets für ComsolGrid

```
$ cp edge2d.mph `bin/dir_hier_path edge2d.mph`
$ cp workunits2add/wu291_values `bin/dir_hier_path wu291_values`
$ ./bin/create_work -appname comsolstarter \
    -wu_name wu291 \
    -wu_template ./Eingabenschablone \
    -result_template ./Ergebnisschablone \
    edge2d.mph wu291_values
```

Die erste und zweite Zeile kopieren jeweils die Simulationsdateien in die Download-Hierarchie vom ComsolGrid. Weiterhin wird in der Datenbank eine Referenz eingetragen, die die Zuordnung zwischen den Dateinamen in der Datei der Eingabeschablone und der physikalischen Ordnerstruktur ermöglicht. Die dritte Zeile erstellt das Arbeitspaket, zu beachten ist die Reihenfolge der Dateien in der letzten Zeile. `edge2d.mph` wird von ComsolGridStarter mit `comsol.mph` geöffnet und `wu291_values` mit dem Dateinamen `comsol.txt`. Aber das kennen Sie alles schon aus den Grundlagen.

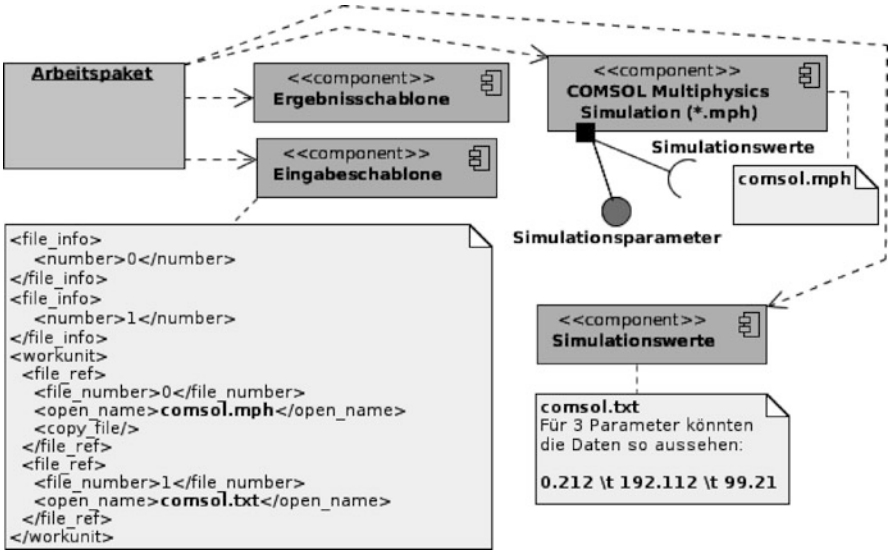


Abb. 14.4 Beispiel eines Komponentendiagramm für ein Arbeitspaket im ComsolGrid-Framework

Das Kommunikationsprotokoll für das Übertragen der Parametersätze zwischen den zwei Anwendungen ComsolGridQt und ComsolGridFCGI basiert auf XML und muss einige Hürden nehmen, um erfolgreich verwendet werden zu können. Wichtigste Hürde ist, dass ein FEM-Modell in Base64-Kodierung [57] vorliegen muss, so dass dieses in XML eingebettet werden kann.

14.5.2 Definition der Parameterstudienwerte

Die Parametersätze haben folgendes Format: `START:STOP:STEP:DEFAULT`. Das Listing 14.10 enthält ein Beispiel eines Parametersatzes mit den drei Parametern `maxT`, `objWidth` und `objHeight`. In den Zeilen 1–3 wird ein Parametersatz definiert, das Format eines Parameters wurde in Abschn. 14.3.2 vorgestellt.

Bei der Erstellung der Parameterdateien werden alle drei Parameter einzeln für sich durchlaufen. Das bedeutet, im ersten Schritt wird der Parameter `maxT` variiert und von 0.0 bis 1.0 hochgezählt, als Schrittweite wird 0.1 nach jedem Schritt hinzuaddiert.

Listing 14.10 Beispiel eines `Range` mit drei Parametersätzen

```
struct ComsolGrid::Parameter p1={"maxT",       0.0,  1.0, 0.1,  0.5 };
struct ComsolGrid::Parameter p2={"objWidth", 10.0, 20.0, 1.0, 15.0 };
struct ComsolGrid::Parameter p3={"objHeight",33.5, 34.0, 0.1, 33.5 };

ComsolGrid::Range r1;
r1.push_back(p1);
r1.push_back(p2);
```

```
r1.push_back(p3);

ComsolGrid::Ranges ranges;
ranges.push_back(r1);
```

Bei dieser Durchführung werden die anderen Parameter auf deren Standardwert gesetzt und für jeden Durchlauf verwendet. Für den Parameter `objWidth`, wird der Wert 15.0 und für den Parameter `objHeight` wird 33.5 als Wert in die Simulation mit einbezogen. Das bedeutet für den ersten Fall, dass 11 Parametersätze erstellt werden, die in Listing 14.11 exemplarisch aufgezeigt sind.

Listing 14.11 Beispiel einer generierten Parameterdatei

```
0.0     15.0    33.5
0.1     15.0    33.5
0.2     15.0    33.5
0.3     15.0    33.5
0.4     15.0    33.5
0.5     15.0    33.5
0.6     15.0    33.5
0.7     15.0    33.5
0.8     15.0    33.5
0.9     15.0    33.5
1.0     15.0    33.5
```

In dem Beispiel aus Listing 14.10 werden 28 Variationen erstellt.

14.6 Ergebnisverarbeitung

14.6.1 Validierer für ComsolGrid

In der Testversion von ComsolGrid werden zwei Standardvalidierer verwendet. Diese wurden direkt aus den BOINC-Quellen und den beigefügten Beispielanwendungen übernommen (s. Abschn. 9.3.1). Es wurden der `trivial_validator` (A1) und der `bitwise_validator` (A2) hinzugefügt.

A1 wurde umbenannt zu `comsol_trivial_validator` und A2 wurde zu `comsol_bitwise_validator` umbenannt, damit eine spätere Modifizierung ermöglicht wird und nicht die in den Konfigurationen des BOINC-Projektservers eingegriffen werden muss. So können neuere Versionen an die Stelle der alten Versionen kopiert und der BOINC-Projektserver einfach neu gestartet werden, so dass die Änderungen greifen. Wir kennen die beiden mitgelieferten Validierer bereits aus vorherigen Kapiteln und geben dazu keine weiteren Informationen, da nicht mehr als die Umbenennung nötig war.

14.6.2 Assimilierer für ComsolGrid

Der Assimilierer im ComsolGrid wurde wie der Validierer direkt aus den BOINC-Quellen übernommen. Der dort beigefügte Beispielassimilierer `copydir_assi-`

`milator` wurde zu `comsol_copydir_assimilator` umbenannt und dem Entwicklungszweig von ComsolGrid hinzugefügt. Ebenso wie bei den Validierern können Änderungen durch einfaches Ersetzen der alten Anwendung und Neustarten des BOINC-Projektservers übernommen werden. Das valide Ergebnis aus der Validierung wird in einen Dateiordner kopiert, die Ergebnisse werden durchgehend mit einer Identifikation ausgestattet. Dies ist der einzige Unterschied zum BOINC mitgelieferten Assimilierer. Im Original werden nur Identifikationsnummern verwendet, wenn mehr als ein Arbeitspaket mit demselben Namen vorhanden ist.

Teil V
Anhang

Kapitel 15
BOINC

15.1 Fehlernummern und Fehlermeldungen

Eine Auflistung aller in BOINC definierten Fehlernummern finden Sie in der Datei `lib/error_numbers.h`. Eine Beschreibung für die jeweilige Fehlernummer finden Sie in der Datei `lib/str_util.cpp`, innerhalb der Funktion `boinc_error()`. Erwarten Sie keine besonderen Fehlerbeschreibungen, wenn Sie schon Erfahrungen mit der C-Funktion `perror()` haben, so kommen Ihnen die BOINC-Beschreibungen bekannt vor. Die Beschreibungen spiegeln quasi den Namen der Konstante einer Fehlernummer wieder. Einige Beispiele:

ERR_LISTEN -> „listen() failed“
ERR_ABORTED_VIA_GUI -> „result aborted via GUI“
ERR_BAD_PASSWORD -> „bad password“
ERR_SYMLINK_FAILED -> „symlink() failed“

Sie sehen, wie einfach einige Beschreibungen gehalten sind. Es gibt aber auch einige, welche eventuell nicht trivial zu erkennen sind:

ERR_IDLE_PERIOD -> „user preferences say can not start work“
ERR_SCHED_SHMEM -> „scheduler shared memory contents bad“
ERR_WU_USER_RULE -> „user already did result for this workunit“
ERR_SHMEN_NAME -> „can not get shared mem segment name“
ERR_ATTACH_FAIL_INIT -> „Could not start master page download“

Weiterhin finden Sie im unteren Bereich hardcodierte Umwandlungen von HTTP-Codes:

404 -> „HTTP file not found“
407 -> „HTTP proxy authentication failure“
416 -> „HTTP range request error“
500 -> „HTTP internal server error“
501 -> „HTTP not implemented“
502 -> „HTTP bad gateway“
503 -> „HTTP service unavailable“
504 -> „HTTP gateway timeout“

C. B. Ries, *BOINC*, Xpert.press,
DOI 10.1007/978-3-642-23382-8_15, © Springer-Verlag Berlin Heidelberg 2012

Weitere Informationen zu den HTTP-Codes finden Sie in der aktuellen Version des Request for Comments (RFC) 2616 [59]. Sie sollten keine neuen Fehlernummern definieren, welche diese HTTP-Codes überschreiben würden.

15.2 Laufzeitfehler

15.2.1 State file error: missing file

Sollten Sie im Nachrichtenfenster des BOINC-Managers eine Meldung erhalten, die in etwa so lautet:

[error] State file error: missing file
[error] Can't handle task_chatathome_wu_30_1 in scheduler reply

So haben Sie eventuell vergessen, in Ihrer Ergebnisschablone mindestens eine Datei zu beschreiben. Ich selber hatte auch diesen Fehler bei dem BOINC-Projekt *Chat@home*, da ich dabei keine Ergebnisse erzeuge und zurücksenden muss, aber BOINC erwartet zumindest eine Beschreibung einer solchen Datei. Für mich reichte dann ein einfaches Hinzufügen eines Dummy-Eintrages, und das Problem wurde erfolgreich gelöst.

15.2.2 OpenGL-Fehler

Dieser Fehler kann vorkommen, wenn freeglut [128] in der Version 2.6.0 genutzt wird. *freeglut* kann eventuell kein Fenster mit Unterstützung von Transparenz initialisieren und bricht die Ausführung mit der Fehlermeldung in Listing 15.1 ab. Es kann helfen, wenn Sie `GLUT_ALPHA` aus der Parameterliste von `glutInitDisplayMode` auskommentieren. Im Abschn. 8.2.1.1 wird das exemplarisch gezeigt.

Listing 15.1 Funktion für das Initialisieren einer BOINC-Grafikanwendung

```
freeglut (./screensaver): ERROR: Internal error <FBConfig with necessary
    capabilities not found> in function fgOpenWindow
X Error of failed request: BadWindow (invalid Window parameter)
  Major opcode of failed request: 4 (X_DestroyWindow)
  Resource id in failed request: 0x0
  Serial number of failed request: 26
  Current serial number in output stream: 29
```

15.3 Konfigurationsdateien

15.3.1 Plattformen

Die Plattformen aus der Tab. 15.1 werden standardmäßig vom BOINC-Client unterstützt. Sie können diese Tabelle manuell an Ihre eigenen Bedürfnisse anpassen, wie ich das für das Praxisbeispiel in Kap. 14 getan habe, allerdings müssen Sie dann auch den BOINC-Client und optional den BOINC-Manager anpassen.

15.3.2 Projekteinstellungen in der config.xml

Innerhalb des Bereiches `<config>...</config>` können zahlreiche Einstellungen vorgenommen werden, welche in diesem Abschnitt aufgelistet sind, größtenteils auf den Informationen im BOINC-Wiki [90] basieren und durch weitere Informationen aus den BOINC-Quellen verständlich gemacht werden. Viele Einstellungsmöglichkeit sind meiner persönlichen Ansicht nach nicht unbedingt erforderlich und teilweise veraltet, wurden seit Jahren nicht mehr gepflegt oder ziehen zahlreiche weitere administrative Tätigkeiten hinter sich her, weil die Features nicht vollkommen implementiert wurden. Aus diesen genannten Gründen finden Sie nur ausgewählte Konfigurationselemente, welche ich persönlich schon getestet habe und von denen ich weiß, dass sie funktionieren.

Listing 15.2 zeigt den Aufbau der `config.xml`, die innerhalb jedes BOINC-Projektes existiert und die Konfiguration für alle zu startenden Tasks und Dämonen beinhaltet.

Tab. 15.1 Die in `bin/appmgr` und der `project.xml` definierten Standardplattformen

MySQL-Datentyp	C++-Datentyp
windows_intelx86	Microsoft Windows (98 or later) running on an Intel x86-compatible CPU
windows_x86_64	Microsoft Windows running on an AMD x86_64 or Intel EM64T CPU
i686-pc-linux-gnu	Linux running on an Intel x86-compatible CPU
x86_64-pc-linux-gnu	Linux running on an AMD x86_64 or Intel EM64T CPU
powerpc-apple-darwin	Mac OS X 10.3 or later running on Motorola PowerPC
i686-apple-darwin	Mac OS 10.4 or later running on Intel
x86_64-apple-darwin	Intel 64-bit Mac OS 10.5 or later
sparc-sun-solaris2.7	Solaris 2.7 running on a SPARC-compatible CPU
sparc-sun-solaris	Solaris 2.8 or later running on a SPARC-compatible CPU
sparc64-sun-solaris	Solaris 2.8 or later running on a SPARC 64-bit CPU
powerpc64-ps3-linux-gnu	Sony Playstation 3 running Linux
anonymous	anonymous

Listing 15.2 BOINC-Konfigurationsdatei `config.xml` [98]

```
<boinc>
  <config>
    [ configuration options ]
  </config>
  <tasks>
    TASK*
  </tasks>
  <daemons>
    DAEMON*
  </daemons>
</boinc>
```

Dieser Teil beschreibt die allgemeinen Einstellungen und die individuellen Tasks und Dämonen für ein BOINC-Projekt. Die Angabe von Tasks und Dämonen ist optional, es müssen zwar für die erfolgreiche Abarbeitung ein paar wenige Dämonen konfiguriert sein (vgl. Abschn. 4.1), allerdings wird das nicht vorgeschrieben.

```
PERIOD:
  seconds | minutes | hours | days

TASK:
  <task>
    <cmd> COMMAND </cmd>
    <output> FILEPATH </output>
    <period> INTEGER PERIOD </period>
    [ <host> FQN | IP </host>          ]
    [ <disabled>  0|1 </disabled>      ]
    [ <always_run> 0|1  </always_run> ]
  </task>

DAEMON:
  <daemon>
    <cmd> COMMAND </cmd>
    [ <host> FQN | IP </host> ]
    [ <disabled> 0|1 </disabled> ]
    [ <output> FILEPATH </output> ]
    [ <pid_file> FILEPATH </pid_file> ]
  </daemon>
```

Die beiden Konfigurationsabschnitte TASK und DAEMON sind ähnlich strukturiert, in beiden Abschnitten muss mindestens ein auszuführender Befehl definiert werden. Für einen Task kann eine Ausgabedatei angegeben werden, so dass alle Ausgaben in dieser Datei landen. Sie haben in Abschn. 5.6.1.2 gesehen, dass Sie einen Crontab-Eintrag erstellen müssen, welcher standardmäßig alle fünf Minuten ausgeführt wird. Während jeder Ausführung wird geprüft, ob der durch `<period>` definierte Wert seit der letzten Ausführung überschritten ist, so dass entsprechend ein Task ausgeführt wird oder nicht. Ein Zeitintervall kann in Sekunden, Minuten, Stunden oder Tagen angegeben werden, z. B. `10 minutes` oder `1 days`. Ist `<always_run>` gesetzt, so wird die Zeitangabe ignoriert, standardmäßig ist dies deaktiviert, so dass die Zeitangaben verwendet werden.

Mit der Angabe eines Wertes für host können die einzelnen Tasks und Dämonen auf unterschiedlichen Hosts ausgeführt werden (vgl. Abschn. 5.8). Beim Starten eines BOINC-Projektes wird der Hostname eines Hosts mit dem hier definierten Wert geprüft. Stimmen diese überein, so wird der entsprechende Befehl ausgeführt.

Mit `<disabled>` kann eine Ausführung deaktiviert werden, standardmäßig ist der Wert Null. `<pid_file>` beinhaltet die PID eines Prozesses, so dass diese PID

beim Stoppen eines BOINC-Projektes ausgelesen wird und die BOINC-Skripte die jeweiligen Prozesse wieder stoppen können.

Allgemein

```
<ban_cpu> REGEXP </ban_cpu>
<ban_os> REGEXP </ban_os>
```

REXEXP sollen reguläre Ausdrücke im Format des IEEE POSIX Standard 1003.2 sein. `ban_cpu` überprüft die Werte in

- `host.p_vendor` (zu Deutsch Hersteller),
- `host.p_model`

und schickt den Hosts, die mit diesen regulären Ausdrücken übereinstimmen, keine neuen Arbeitspakete zu. Tabelle 15.2 zeigt einige Beispiele für den Inhalt dieser Datenbankfelder [185]. Diese Werte werden nicht direkt durch BOINC ermittelt. Für das ordentliche Sammeln dieser Informationen werden Betriebssystemfunktionen genutzt, so dass die Informationen direkt aus den Prozessoren kommen. Unter Linux können Sie dies zum Beispiel durch den Aufruf `cat /proc/cpuinfo` selber einmal praktizieren: Sie können dann den Hersteller und die Prozessoridentifikation Ihres Prozessors einsehen und erkennen womöglich eine Übereinstimmung mit einem der Werte aus Tab. 15.2. `ban_os` wird für das Überprüfen der Datenbankfelder

- `host.os_name`,
- `host.os_version`

verwendet. Einige Beispiel für deren Inhalt finden Sie in Tab. 15.3.

Ebenso wie bei `ban_cpu`, werden an alle übereinstimmenden Hosts keine neuen Arbeitspakete geschickt.

```
<homogeneous_redundancy> N </homogeneous_redundancy>
```

Tab. 15.2 Auswahl von möglichen Werten der Datenbankfelder `host.p_vendor` und `host.p_model`

host.p_vendor	host.p_model
AuthenticAMD	AMD Sempron(tm) 2400+
AuthenticAMD	AMD Sempron(tm) 2600+ [x86 Family 6 Model 8 Stepping 1]
AuthenticAMD	AMD Athlon(tm) 64 Processor 3000+
AuthenticAMD	AMD Athlon(tm) 64 FX-60 Dual Core Processor [x86 Family 15 Model 35 Stepping 2] [fpu tsc pae nx sse sse2 3dnow mmx]
GenuineIntel	x86 Family 6 Model 8 Stepping 3
GenuineIntel	Intel(R) Pentium(R) M processor 1.86GHz
GenuineIntel	Intel(R) Pentium(R) 4 CPU 2.40GHz
GenuineIntel	Intel(R) Pentium(R) 4 CPU 2.66GHz

Tab. 15.3 Auswahl von möglichen Werten der Datenbankfelder `host.p_name` und `host.p_version`

host.os_name	host.os_version
Microsoft Windows XP	Home Edition, Service Pack 2, (05.01.2600.00)
Microsoft Windows XP	Professional Edition, Service Pack 2, (05.01.2600.00)
Microsoft Windows 2003	Enterprise Server Edition, (05.02.3790.00)
Microsoft Windows Server 2003	Standard Server Edition, Service Pack 2, (05.02.3790.00)

Bestimmt, wie die Prüfung der Homogenität von unterschiedlichen Betriebssystemen und Prozessortypen durchgeführt werden soll. `N` kann einen Wert zwischen Null und Zwei haben. Null steht für keine Prüfung der Homogenität, Eins schaltet die feinkörnige Prüfung[1] an und Zwei aktiviert die grobkörnige Prüfung[2]. Im Abschn. 4.2.3 wird die `hr_class` eines Arbeitspakets vorgestellt.

```
<ignore_delay_bound/>
```

Die Angabe einer Deadline eines Arbeitspakets wird nicht beachtet.

```
<multiple_clients_per_host> 0|1 </multiple_clients_per_host>
```

Ist dieser Wert gesetzt, so wird keine Prüfung der IP-Adresse oder des Hostnamens durchgeführt. Dadurch kann ein Host mehrere BOINC-Clients starten und Arbeitspakete erhalten, auch wenn `<one_result_per_host_per_wu>` gesetzt ist. Sinnvoll kann dies sein, wenn der Host aus mehreren virtuellen Maschinen besteht, bei der die jeweilige Maschine eine CPU zum Arbeiten besitzt.

```
<nowork_skip> 0|1 </nowork_skip>
```

Wenn der Scheduler keine neuen Arbeitspakete hat, so werden keine Datenbankabfragen durchgeführt. Sinnvoll ist dies, wenn der BOINC-Server stark ausgelastet ist, allerdings werden durch diese Konfiguration keine vom BOINC-Teilnehmer eingestellten Konfigurationen übernommen, z. B. maximale Anzahl an zur Verfügung gestellten Prozessorkernen.

```
<prefer_primary_platform> 0|1 </prefer_primary_platform>
```

Senden immer die passendste Version einer wissenschaftlichen Applikation, z. B. eine 64-Bit-Version für eine 64-Bit-Plattform, auch wenn eine 32-Bit-Version vorhanden ist. Das heißt, wenn eine Applikation noch nicht in der idealen Version vorliegt, so wird erst eine ähnliche genommen, und sobald sich das ändert, wird auf die neue Version aktualisiert.

```
<report_grace_period> x </report_grace_period>
```

[1] Die Prüfung von Betriebssystem- und Prozessortyp.

[2] Die Prüfung vom Betriebssystemtyp allein.

Eine Deadline kann um den Wert x in Sekunden erweitert werden, so dass die einzelnen – womöglich gerade noch fertigzustellenden – Arbeitspakete noch eine allerletzte Chance erhalten und nicht direkt ignoriert werden.

Limitierungen für Arbeitspakete

Begrenzungen der Arbeit für die BOINC-Teilnehmer.

```
<one_result_per_user_per_wu/>
<one_result_per_host_per_wu/>
```

Die erste Konfiguration setzt fest, dass nur ein Teilergebnis für die komplette Berechnung eines Arbeitspaketes durch einen BOINC-Teilnehmer bearbeitet werden soll. Der zweite Konfigurationssatz ist ähnlich, allerdings werden die erforderlichen Berechnungen für Teilergebnisse nur zu einzelnen Hosts geschickt, wodurch ein Benutzer auch das gesamte Arbeitspaket berechnen kann, wenn dieser genügend Hosts beim BOINC-Projekt registriert hat.

```
<max_wus_in_progress> N </max_wus_in_progress>
<max_wus_in_progress_gpu> M </max_wus_in_progress_gpu>
```

Limitiert die Anzahl von zu verarbeitenden Arbeitspaketen eines BOINC-Teilnehmers und eines einzelnen Hosts. Der BOINC-Client teilt dem BOINC-Scheduler mit, wie viele Ressourcen schon durch Arbeitspakete belegt sind und noch zur Verfügung stehen. Die maximale Anzahl der zu verarbeitenden Arbeitspakete ist $N \cdot NCPUS$ für einfache Prozessoren (siehe weiter unten <max_ncpus>) und $M \cdot NGPUS$ für Grafikprozessoren mit Berechnungskernen. Die allgemeine Formel für die Berechnung der maximal zu nutzenden Ressourcen ist die Summe beider Teilergebnisse. Wir verwenden diese Konfigurationsparameter unter anderem in Abschn. 9.4.

```
<gpu_multiplier> GM </gpu_multiplier>
<max_wus_to_send> NSEND </max_wus_to_send>
<max_ncpus> NCPUS </max_ncpus>
```

Wenn Sie eine wissenschaftliche Applikation mit GPU-Unterstützung haben, so wird mit <gpu_multiplier> ein Verhältnis der Rechenleistung zur Leistung einer CPU angegeben: $N \cdot (NCPUS + GM \cdot NGPUS)$. Mit NSEND wird die maximale Anzahl an möglichen Arbeitspaketen für einen Host begrenzt: $NSEND \cdot (NCPUS + GM \cdot NGPUS)$, der Standardwert ist 10.

Zwischenspeicherung von Arbeitspaketen

Arbeitspakete für eine schnelle Ausführung vorhalten.

```
<shmem_work_items> N </shmem_work_items>
```

N gibt an, wie viele Teile der Arbeitspakete sich im Shared-Memory für den Feeder befinden [193, Bruce 2 Feb 2005]. Die aktuell im Shared-Memory befindlichen Teile der Arbeitspakete können mit dem Tool `./bin/show_shmem` ermittelt werden.

```
<feeder_query_size> N </feeder_query_size>
```

Der Wert N limitiert die Anzahl an abzufragenden noch zu verarbeitenden Arbeitspaketen, wenn der Feeder sie einem BOINC-Teilnehmer bekanntmachen will.

Verteilung von Daten auf der Welt

Eine Erfindung des Einstein@home-Projekts gibt Ihnen die Möglichkeit, Ihre Arbeitspakete und Applikationen auf Servern auf der ganzen Welt zu verteilen. Dies ermöglicht, die Netzwerk- und Serverlasten zu verteilen. Dabei wird die URL zum Herunterladen der Daten durch eine Adresse eines Servers in der nächstliegenden Zeitzone des anfragenden Benutzers ersetzt. Die Lasten können dadurch auf mehrere Server verteilt werden. Dies ist allerdings kein Garant dafür, dass die Anfragen auch schneller bearbeitet werden; eine Prüfung der Antwortzeiten findet nicht statt.

```
<replace_download_url_by_timezone> URL </replace_download_url_by_timezone>
<cache_md5_info> 0|1 </cache_md5_info>
```

Der zweite Konfigurationswert speichert die Checksummen der Dateien zwischen; dies kann sinnvoll sein, wenn sich Dateien nicht oft ändern und evtl. permanent vorliegen. Wenn Sie u. a. einen Prozess besitzen, der dynamisch Arbeitspakete erstellt, und Sie wissen, dass ein großer Teil der Eingabedateien identisch ist und diese womöglich sehr groß sind, so ist es sinnvoll, die Prüfsummen zwischenzuspeichern.

Debbugging und Protokollierungen

Die nachfolgenden Konfigurationen ermöglichen das Nachverfolgen von sehr vielen Aktionen innerhalb des BOINC-Frameworks. Sie sollten diese Konfigurationen auf jeden Fall wieder entfernen, wenn Sie einen Fehler gefunden und korrigiert, so dass Sie nicht unnötig viel Speicherkapazitäten Ihrer Festplatten verbrauchen. Die Namensgebung der einzelnen Konfigurationen gibt in der Regel Aufschluss über deren Funktionalität. Tabelle 15.4 listet zudem deren Zweck auf.

BOINC-Client vorkonditionieren

Einstellungen für das Verhalten der BOINC-Clients.

```
<next_rpc_delay> x </next_rpc_delay>
```

x ist ein Zeitwert in Minuten und beschreibt, wann die nächste Aktualisierungsanfrage an ein BOINC-Projekt gesendet wird.

```
<min_core_client_version> N </min_core_client_version>
<min_core_client_version_announced> NANNOUNCE </
     min_core_client_version_announced>
<min_core_client_upgrade_deadline> NDEAD </min_core_client_upgrade_deadline>
```

Tab. 15.4 Konfiguration der zu protokollierenden Meldungen während der Abarbeitung eines BOINC-Projektes

Konfiguration	Beschreibung
<debug_assignment/>	Beschreibt das Senden von markierten Arbeitspaketen.
<debug_handle_results/>	Protokolliert das Erarbeiten von einzelnen Berechnungsergebnissen.
<debug_prefs/>	Zeigt das Nutzen der globalen Konfigurationsparameter.
<debug_request_details />	Zeigt Informationen über Anfragen von BOINC-Teilnehmern.
<debug_request_headers/>	Zeigt HTTP-Informationen von Anfragen der BOINC-Teilnehmer.
<debug_send/>	Liefert Beschreibungen der gesendeten Arbeitspakete.
<debug_user_messages/>	Zeigt von BOINC-Teilnehmern versendete Nachrichten.
<debug_version_select/>	Zeigt, welche Version einer wissenschaftlichen Applikation von BOINC-Teilnehmern verwendet wird.
<sched_debug_level> N </sched_debug_level>	N kann folgende Werte besitzen: *1:=minimal*, *2:=normal* (Standard aller genannten Konfigurationsparameter), *3:=debug*.
<fuh_debug_level> N </fuh_debug_level>	`fuh` steht für *file upload handler* und N kann folgende Werte besitzen *1:=minimal*, *2:=normal* (Standard aller genannten Konfigurationsparameter), *3:=debug*.

Wenn Sie als Projektadministrationen bestimmte Funktionalitäten auf der Clientseite benötigen, die erst ab einer bestimmten BOINC-Client-Version vorhanden sind, so können Sie dies den BOINC-Teilnehmern mitteilen. N ist eine minimale BOINC-Client-Version, der zweite Konfigurationssatz in der Form *NANNOUNCE* $= 10000 \cdot major + 100 \cdot minor + release$ beschreibt eine in Zukunft zu nutzende Version und mit $NDEAD$ gibt man einen Zeitwert an, der die Zeit in Sekunden seit dem 1. Januar 1970 beschreibt. Spätestens dann muss der BOINC-Client erneuert werden.

```
<msg_to_host/>
```

Aktiviert das Verarbeiten von Trickle-Messages auf der Seite des BOINC-Servers.

```
<non_cpu_intensive> 0|1 </non_cpu_intensive>
```

Ist diese Konfiguration angegeben, so wird vom BOINC-Client immer nur ein Teil eines Arbeitspaketes heruntergeladen und immer ausgeführt, wenn Berechnungen möglich sind.

Standardcredits

Einstellungen für die Vergabe von Berechnungspunkten (sog. Credits) bei der erfolgreichen Verarbeitung von Arbeitspaketen.

```
<granted_credit_weight> X </granted_credit_weight>
<granted_credit_ramp_up> N </granted_credit_ramp_up>
<fp_benchmark_weight> X </fp_benchmark_weight>
<granted_credit_weight> X </granted_credit_weight>
```

Webserver- und Werkzeugkonfiguration

Einstellungen für den Apache-HTTP-Server.

```
<httpd_user> Benutzer </httpd_user>
```

Der Benutzer, der den Apache-HTTP-Server startet. Dieser Benutzer sollte nur die nötigsten Rechte erhalten, z. B. Zugriffsrechte auf die Ordner zum Lesen der Eingabedateien, so dass diese von BOINC-Teilnehmern heruntergeladen werden können. Weiterhin sollte die Möglichkeit zum Erstellen und Verarbeiten der Eingabedateien gegeben sein.

```
<www_host> Hostname </www_host>
```

Der Hostname des Hosts, auf dem der Apache-HTTP-Server gestartet ist.

```
<sched_host> Hostname </sched_host>
```

Der Hostname des Hosts, auf dem der Scheduler gestartet ist.

```
<uldl_host> Hostname </uldl_host>
<uldl_pid> Pfad </uldl_pid>
```

Der Hostname des Webservers, von dem Dateien heruntergeladen werden sollen und zu dem Berechnungsergebnisse hochgeladen werden können. Letztere Konfiguration dient der Angabe einer Datei, in der die PID des Apache-HTTP-Servers abgespeichert ist.

Hosts, Verzeichnisse und URLs

Erweiterte Einstellungen für die Konfiguration von verschiedenen Hosts und der Laufzeitumgebung.

```
<master_url> URL </master_url>
<cgi_url> CGIURL </cgi_url>
```

Master enthält die Hauptadresse eines BOINC-Projekts, und `CGIURL` ist der Pfad zum BOINC-Scheduler.

```
<download_url> URL </download_url>
<download_dir> Pfad </download_dir>
```

Webadresse und Pfad innerhalb eines BOINC-Projekts, um Dateien herunterladen zu können.

```
<upload_url> URL </upload_url>
<upload_dir> Pfad </upload_dir>
```

Webadresse und Pfad innerhalb eines BOINC-Projekts, um Dateien hochzuladen.

```
<uldl_dir_fanout> N </uldl_dir_fanout>
```

`N` ist standardmäßig auf 1024 eingestellt, wobei dieser Wert beschreibt, wie viele Unterverzeichnisse die Ordner der herunterzuladenden Dateien besitzt. Dies erhöht sehr die Zugriffsgeschwindigkeit auf die Ordner, weil nicht mehr alle Dateien vorher geprüft werden müssen, wenn zum Beispiel eine Datei mit dem Anfangsbuchstaben

‚z' gesucht wird, allerdings noch 10.000 Dateien mit den Anfangsbuchstaben ‚a' bis ‚y' vorhanden sind und geprüft werden müssen. Die 1024 Unterordner sind durch einen hexadezimalen Wert beschrieben, wobei die Werte zwischen *0x00* und *0x3ff* liegen. In Abschn. 5.6.2.4 wird diese Konfiguration vorgestellt.

```
<host> Hostname </host>
<long_name> Name </long_name>
```

Der Hostname ist der Haupthost eines BOINC-Projekts, auf dem standardmäßig alle Tasks und Dämonen gestartet werden. Die zweite Konfiguration soll eine Beschreibung eines BOINC-Projekts beinhalten, z. B. Spinhenge@home.

```
<shmem_key> Shared-Memory-Schluessel </shmem_key>
```

Ein eindeutiger Wert, mit dem ein Shared-Memory-Bereich zum Austauschen von Daten zwischen den BOINC-Applikationen verwendet wird, u. a. sind hier die Informationen über Arbeitspakete beim Scheduler oder Feeder vorgehalten.

```
<log_dir> Pfad </log_dir>
<bin_dir> Relativer-Pfad </bin_dir>
<cgi_bin_dir> Relativer-Pfad </cgi_dir>
<sched_lockfile_dir> Pfad </sched_lockfile_dir>
```

Vier Angaben von Verzeichnissen, um zur Laufzeit erstellte Dateien abzulegen und vorzuhalten.

Webseitenparameter

Möglichkeiten, die Webseitendarstellung zu konfigurieren.

```
<profile_screening/>
```

Ist dieser XML-Tag vorhanden, so werden auf der Projektwebseite Profile der Teilnehmer angezeigt und können durchsucht werden. Sie können das Verhalten bestimmen, wie die Teilnehmer aus der Datenbank ermittelt werden sollen. Für diesen Fall müssen Sie in der `html/project/project.xml` eine oder beide Funktionen `uotd_candidates_query()` und `profile_screen_query()` implementieren, beide Funktionen müssen SQL-Anfragen zurückgeben. Auf der Startseite kann ein Benutzer als „Tagesbester" oder nach anderen Strategien angezeigt werden, die erste Funktion erledigt diese Aufgabe und liefert den sogenannten User of the Day (UOTD) [89].

```
<show_results/>
<dont_suppress_pending/>
```

Es werden die Berechnungsergebnisse und noch offenen Arbeitspakete angezeigt.

```
<no_web_account_creation/>
```

Es können sich keine neuen Benutzer über eine Webanfrage registrieren (vgl. Abschn. 4.3.1.1), d. h. diese müssen entweder schon vorhanden sein oder manuell hinzugefügt werden.

```
<akismet_key> 1234567890ab </akismet_key>
```

Ein individueller Akismet-Schlüssel für die Prävention gegen Spam auf Webseiten [65], in diesem Fall für ungewünschte Einträge im BOINC-Forum.

```
<users_per_page> N </users_per_page>
<teams_per_page> N </teams_per_page>
<hosts_per_page> N </hosts_per_page>
```

N ist jeweils die Anzahl der maximal anzuzeigenden Datensätze auf einer der jeweiligen Webseiten für die Anzeige von BOINC-Teilnehmern, BOINC-Teams und registrierten Hosts.

```
<profile_min_credit> X </profile_min_credit>
```

X ist die Anzahl der Credits eines BOINC-Teilnehmers, welche dieser mindestens zugeteilt haben muss, so dass sein Profil angezeigt wird.

Miscellaneous/Vermischtes

Konfigurationsmöglichkeiten ohne spezielle Kategorieeinordnung.

```
<ended> 0|1 </ended>
```

Die aktuell noch versendeten Arbeitspakete werden noch abgewartet, aber es werden keine neuen Arbeitspakete mehr an die BOINC-Teilnehmer verteilt.

```
<disable_account_creation/>
```

Neuen Benutzern ist nicht erlaubt, sich an dem BOINC-Projekt zu registrieren. Es können sich nur schon vorhandene Benutzer anmelden und am Projekt teilnehmen.

```
<min_passwd_length> N </min_passwd_length>
```

N ist die Anzahl an Zeichen eines Benutzerpassworts, die es mindestens haben muss.

```
<dont_store_success_stderr/>
```

Die Fehlerausgaben während der Ausführung der wissenschaftlichen Applikation werden nicht in der Datenbank abgespeichert, wenn die Abarbeitung erfolgreich beendet wurde.

Datenbank

Listing 15.3 enthält die Authentifizierungsdaten zu einem Datenbankserver, der für ein BOINC-Projekt genutzt werden soll. Wenn ein Replikationsserver vorhanden ist, so werden die oberen Authentifizierungsdaten verwendet, wenn keine anderen angegeben sind, in diesem Fall muss allerdings mindestens der Hostname des Replikationsservers spezifiziert werden.

Listing 15.3 BOINC-Konfigurationsdatei: Datenbank

```
<db_name>name</db_name>
<db_user> database_user_name </db_user>
[ <db_host> hostname</db_host> ]
```

```
[ <db_passwd> database_password </db_passwd> ]

[ <replica_db_host> hostname </replica_db_host> ]
[ <replica_db_user> database_user_name </replica_db_user> ]
[ <replica_db_host> hostname </replica_db_host> ]
[ <replica_db_passwd> database_password </replica_db_passwd> ]
```

15.3.3 Eingabe-/Ergebnisschablonen für Arbeitspakete

Listing 15.4 und 15.5 enthalten die Formatbeschreibungen für die Eingabe- und Ergebnisschablone von Arbeitspaketen. Die Beschreibung erfolgt in der EBNF-Syntax.

Listing 15.4 Format der Arbeitspaket-Eingabeschablone [95]

```
<input_template>
  FILEINFO*
  <workunit>
    FILEREF*
    [ <rsc_fpops_est> FLOAT </rsc_fpops_est> ]
```

Ein geschätzter Wert für die Abarbeitung eines Arbeitspaketes.

```
    [ <rsc_fpops_bound> FLOAT </rsc_fpops_bound> ]
```

Der maximale Grenzwert der Anzahl an Fließkommaoperationen, um eine Abarbeitung durchzuführen. Sollte dieser Wert überschritten werden, so wird die Abarbeitung abgebrochen und als fehlerhaft markiert.

```
    [ <rsc_memory_bound> FLOAT </rsc_memory_bound> ]
```

Definition der maximal zulässigen Nutzung von RAM. Weiterhin werden Arbeitspakete nur an Hosts gesendet, welche mindestens so viel RAM zur Verfügung stellen können.

```
    [ <rsc_disk_bound> FLOAT </rsc_disk_bound> ]
```

Ähnlich wie `<rsc_memory_bound>`, allerdings für die Nutzung der Festplattenkapazität.

```
    [ <delay_bound> FLOAT </delay_bound> ]
```

Der Wert gibt, an wie lange ein Arbeitspaket von einem BOINC-Teilnehmer bearbeitet werden darf, umgangssprachlich können wir diesen Wert die Deadline nenne. Der Standardwert ist eine Woche, entsprechend müssten Sie 604.800 (für Sekunden) eintragen.

```
    [ <min_quorum> INTEGER </min_quorum> ]
```

Die Anzahl, wie oft ein Arbeitspaket berechnet werden muss, bevor eine Validierung stattfinden kann.

```
    [ <target_nresults> INTEGER </target_nresults> ]
```

Dieser Wert beschreibt, wie oft ein Arbeitspaket für die Bearbeitung verteilt werden kann, der Standardwert ist Fünf.

```
    [ <max_error_results> INTEGER </max_error_results> ]
    [ <max_total_results> INTEGER </max_total_results> ]
    [ <max_success_results> INTEGER </max_success_results> ]
```

Definitionen von Grenzwerten, welche bei Überschreitung ein Arbeitspaket als fehlerhaft markieren oder keine weiteren Berechnungen mehr erforderlich machen, wenn das Arbeitspaket schon validiert werden kann. <max_total_results> sollte immer größer sein als <min_quorum> und <max_success_results>, denn dieser Wert beinhaltet erfolgreiche sowie fehlerhafte Teilergebnisse eines Arbeitspaketes.

```
  </workunit>
</input_template>

FILEINFO:
  <file_info>
    <number> INTEGER </number>
```

Die Indexnummer einer Eingabedatei, die im Bereich <file_ref> näher beschrieben werden muss.

```
    [ <sticky/> ]
```

Die Datei verbleibt beim BOINC-Teilnehmer und wird nicht gelöscht, so lange noch Arbeitspakete für das entsprechende BOINC-Projekt bearbeitet werden sollen. Dies ist sinnvoll bei großen Dateien, so dass diese nicht immer wieder neu heruntergeladen werden müssen.

```
    [ <nodelete/> ]
```

Die Eingabedatei wird auf der Serverseite nicht gelöscht.

```
    [ <report_on_rpc/> ]
```

Die Datei wird bei jeder Anfrage zum BOINC-Projekt gemeldet, wenn diese als <sticky/> markiert ist.

```
    [ <url> URL </url> ]
```

Webadresse zum Herunterladen dieser Eingabedatei eines Arbeitspaketes.

```
    [ <md5_cksum> Checksumme </md5_cksum> ]
```

Die MD5-Checksumme dieser Datei eines Arbeitspaketes.

```
    [ <nbytes> INTEGER </nbytes> ]
```

Die Dateigröße in Bytes dieser Eingabedatei.

```
  </file_info>

FILEREF:
  <file_ref>
    <file_number> INTEGER </file_number>
```

Gibt an, an welcher Stelle die Datei bei der Erstellung von Arbeitspaketen angegeben werden muss. Null ist die erste Stelle, Eins die zweite Stelle, usw. Wie Sie Arbeitspakete erstellen können, wird in Abschn. 9.2 diskutiert.

```
    <open_name> Dateiname </open_name>
```

Virtueller Dateinamen, so dass eine physikalische Datei in einen Slot verlinkt werden kann.

```
    [ <copy_file/> ]
```

Die Eingabedatei wird in einen jeweiligen Slot für die Ausführung physikalisch kopiert, d. h. die Datei kann direkt geöffnet werden und enthält den Originalinhalt anstatt eines symbolischen Link.

```
  </file_ref>
```

Listing 15.5 Format der Arbeitspaket-Ergebnisschablone [95]

```
<output_template>
  <file_info>
    <name> <OUTFILE_0/> </name>
```

Ein automatisch generierter Name für die jeweiligen Ergebnisdateien, Sie müssen sich über diesen Namen keine Gedanken machen.

```
    <generated_locally/>
```

Kennzeichnet eine Datei, die lokal bei den BOINC-Teilnehmern erstellt wurde. Dieses XML-Tag sollte immer verwendet werden.

```
    <upload_when_present/>
```

Die Datei wird zum BOINC-Projekt hochgeladen, wenn diese existiert.

```
    <max_nbytes> INTEGER </max_nbytes>
```

Maximale Dateigröße einer Ergebnisdatei.

```
    <url> <UPLOAD_URL/> </url>
```

Enthält die URL zu der die Ergebnisse hochgeladen werden sollen. Dieser Wert hängt von der BOINC-Projektkonfiguration ab und wird durch die Scheduler-URL ersetzt.

```
  </file_info>
  <result>
    <file_ref>
      <file_name> <OUTFILE_0/> </file_name>
```

Dateiname der Ergebnisdatei, welcher mit `<OUTFILE_N/>` generiert wird. Sie müssen nichts weiter über diesen Namen wissen, weil Sie innerhalb Ihrer Anwendungen den virtuellen Dateinamen nutzen sollten.

```
      <open_name> Dateiname </open_name>
```

Dieser Parameter hat dieselbe Funktionalität wie in Listing 15.4 und beschreibt den virtuellen Dateinamen einer Datei.

```
      [ <copy_file> 0|1 </copy_file> ]
```

Die Ergebnisdatei wird im Slot erstellt und nach der Bearbeitung zum Projektordner kopiert.

```
      [ <optional> 0|1 </optional> ]
```

Bei Eins ist die Datei optional, ansonsten muss die Datei in jedem Fall erstellt werden, sonst ist das Ergebnis fehlerhaft.

```
      [ <no_validate> 0|1 </no_validate> ]
      [ <no_delete> 0|1 </no_delete> ]
```

Diese Datei wird (nicht) validiert und gelöscht, wenn die Bearbeitung stattgefunden hat.

```
    </file_ref>
    [ <report_immediately/> ]
```

Die Arbeitspakete werden nach dem Hochladen der Ergebnisdateien sofort gemeldet, ansonsten kann es sein, dass bis zu einem Tag gewartet wird.

```
  </result>
</output_template>
```

Kapitel 16
Lizenzen

Das BOINC-Framework steht unter der GNU General Public License (GNU GPL)-Lizenz. Dieses verlangt bei der Verwendung von entsprechenden Quelltexten, dass die Lizenz als Kopie mitgeliefert wird.

16.1 GNU Free Documentation License

Version 1.3, 3 November 2008, Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc. <http://fsf.org/>, Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

C. B. Ries, *BOINC*, Xpert.press,
DOI 10.1007/978-3-642-23382-8_16, © Springer-Verlag Berlin Heidelberg 2012

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
H. Include an unaltered copy of this License.
I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements”.

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s

users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

16.2 SGI free Software License B

SGI FREE SOFTWARE LICENSE B (Version 2.0, Sept. 18, 2008)

Copyright (C) [dates of first publication] Silicon Graphics, Inc. All Rights Reserved.

> Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute,

Literaturverzeichnis

Das nachfolgende Literaturverzeichnis enthält alle Quellen über die in diesem Buch abgehandelten Techniken und Verfahren. Es wird empfohlen, auf diese Liste zurückzugreifen, falls der Bedarf an weiteren Recherchearbeiten zu dem ein oder anderen Thema besteht. BOINC ist ein Framework, das sich im stetigen Umbruch befindet, bei dem Wissenschaft und Praxis „Hand in Hand gehen“. Sie müssen kein Profi in allen Bereichen sein, doch Wissen über eine breite Palette von aktuellen Softwareprodukten und Verfahren ermöglicht es, dass Sie BOINC durch Ihre Arbeiten eine persönliche Note verleihen.

Berichte und wissenschaftliche Arbeiten

1. Anderson, D.P., Reed, K.: Celebrating Diversity in Volunteer Computing. In: Proc. HICSS, 2009, 5–8 January
2. Anderson, D.P., Christensen, C., Allen, B.: Designing a Runtime System for Volunteer Computing, UC Berkeley Space Sciences Laboratory, Dept. of Physics, University of Oxford, and Physics Dept., University of Wisconsin, Milwaukee (2006)
3. Anderson, D.P., Fedak, G.: The Computational and Storage Potential of Volunteer Computing. In: Proc. IEEE/ACM International Symposium on Cluster Computing and the Grid, May 2006
4. Anderson, D.P.: BOINC. A System for Public-Resource Computing and Storage. In: Proc. 5th IEEE/ACM International Workshop on Grid Computing, November 2004
5. Dorigo, M., Di Caro, G., T. Stützle, et al.: Special issue on Ant Algorithms. Future Generation Computer Systems, vol. 16, Number 8, 2000
6. Englisch, T.: High Performance Computing, Energietag, Fachhochschule Bielefeld, 2010
7. Foster, I.: IFIP International Conference on Network and Parallel Computing, LNCS 3779, S. 2–13 Springer, New York (2006)
8. Giorgino, T., Harvey, M.J., De Fabritiis, G.: Distributed computing as a virtual supercomputer: Tools to run and manage large-scale BOINC simulations, Comp. Phys. Commun. **181**, 1402 (2010)
9. Giorgino, T., Harvey, M.J., De Fabritiis, G.: Distributed computing as a virtual supercomputer: Tools to run and manage large-scale BOINC simulations, Comp. Phys. Commun. **181**, 1402 (2010)

C. B. Ries, *BOINC*, Xpert.press,
DOI 10.1007/978-3-642-23382-8, © Springer-Verlag Berlin Heidelberg 2012

10. Kornmayer, H., Polak, M., Thandavan, A., Tsouloupas, G., Wolniewicz, P.: g-Eclipse Project. In: Proc. 2nd Austrian Grid Symposium, September 21–23, 2006
11. Iyoha, O.U.: H_2 Production in Palladium, Palladium-Copper Membrane Reactors at 1173 K in the Presence of H_2S. Ph.D. thesis, University of Pittsburgh, 2007
12. Moore, G.E.: Cramming more components onto integrated circuits. Electronics **19**(3), 114–117 (1965)
13. Mounsey, S.: Looking for the silver lining. Sci. Comput. World, **112**, 35–38 (2010)
14. Mudgal, K.R.: A Multi-platform Job Submission System for Epidemiological Simulation Models. Master thesis, Virginia Polytechnic Institute and State University, Virginia, 2011
15. Ríos, G., Fonseca, P., Díaz, O.: Legion: An extensible lightweight web framework for easy BOINC task submission, monitoring and result retrieval using web services, 2011
16. Ries, C.B.: ComsolGrid – Konzeption, Entwicklung und Implementierung eines Frameworks zur Kopplung von COMSOL Multiphysics und BOINC um hoch-skalierbare Parameterstudien zu erstellen. M.Sc., Fachhochschule Bielefeld, Deutschland, 2010
17. Ries, C.B., Schröder, C., Grout, V.: A UML Profile for Berkeley Open Infrastructure for Network Computing (BOINC). In: Proc. ICCAIE, 2011
18. Ries, C.B., Schröder, C., Grout, V.: Generation of an Integrated Development Environment (IDE) for Berkeley Open Infrastructure for Network Computing (BOINC). In: Proc. SEIN, 2011
19. Ries, C.B., Hilbig, T., Schröder, C.: A Modeling Language Approach for the Abstraction of the Berkeley Open Infrastructure for Network Computing (BOINC) Framework. In: Proc. IEEE-IMCSIT, 2010, pp. 663–670
20. Ries, C.B., Schröder, C.: ComsolGrid – A framework for performing large-scale parameter studies using Comsol Multiphysics and Berkeley Open Infrastructure for Network Computing (BOINC). In: Proc. COMSOL Multiphysics Conference, 2010
21. Ries, C.B.: Optische Systeme: Bildvorverarbeitung, Fachhochschule Bielefeld, Deutschland, 13. April 2010
22. Schröder, C., Prozorov, R., Kögerler, P., Vannette, M.D., Fang, X., Luban, M., Matsuo, A., Kindo, K., Müller, A., Todea, A.M.: Multiple nearest-neighbor exchange model for the frustrated Keplerate magnetic molecules {Mo72Fe30} and {Mo72Cr30}. Phys. Rev. B **77**, 224409, 2008
23. Schröder, C., Schmidt, H.J., Schnack, J., Luban, M.: Metamagnetic phase transition of the antiferromagnetic Heisenberg icosahedron. Phys. Rev. Lett. **94**, 207203, 2005
24. Schröder, C., Nojiri, H., Schnack, J., Hage, P., Luban, M., Kögerler, P.: Competing Spin Phases in Geometrically Frustrated Magnetic Molecules. Phys. Rev. Lett. **94**, 017205, 2005
25. Starinshak, D.P., Smith, N.D., and Wilson, J.D.: Using COMSOL Multiphysics Software to Model Anisotropic Dielectric and Metamaterial Effects in Folded-Waveguide Traveling-Wave Tube Slow-Wave Circuits. In: Proc. IVEC, 2008
26. Silva, L., Fedak, G., Kelley, I.: Optimizing the Data Distribution Layer of BOINC with BitTorrent. CoreGRID Technical Report Number TR-0139, June, 2008
27. Somers, M.F.: Leiden Grid Infrastructure, v1.31, 19. September 2011
28. Taufer, M., Anderson, D., Cicotti, P., Brooks, C.L.: Homogeneous redundancy: a technique to ensure integrity of molecular simulation results using public computing. In: Proc. HCW, 2005
29. Vafopoulos, M., Gravvanis, G., Platis, A.: New directions in Computing on Demand (CoD). In: Proc. HERCMA, 2007

Bücher

30. Bauke, H., Mertens, S.: Cluster Computing. Springer, Berlin Heidelberg (2006)
31. Engelhardt, L., Schröder, C.: Simulating computationally complex magnetic molecules in Molecular Cluster Magnets. Winpenny, R.E.P., Ed. World Scientific, Singapore (2011)

32. Freitag, S.: Rechnen lassen – Rechnen im Netz: Grid versus Cloud, S. 94–97. Heise, Hannover (2011)
33. Gulbins, J., Obermayr, K.: Linux – Konzepte, Kommandos, Oberflächen. Springer, (2003)
34. Hager, G., Wellein, G.: Introduction to High Performance Computing for Scientists and Engineers. CRC Press,
35. Herold, H.: Linux-Unix-Systemprogrammierung. Nachdr. der 2. Aufl. Addison-Wesley, (1999)
36. Kirk, D., Hwu, W.M.W.: Programming Massively Parallel Processors: A Hands-On Approach (Applications of GPU Computing Series). Morgan Kaufman, Burlington, MA (2011)
37. Lee, J.: Beginning Perl. Springer, New York (2010)
38. F. Magoulès (Editor): Fundamentals of Grid Computing – Theory, Algorithms and Technologies. Chapman & Hall/CRC, Boca Raton, FL (2011)
39. Jähne, B.: Digitale Bildverarbeitung. Springer, Berlin Heidelberg (2005)
40. Papula, L.: Mathematische Formelsammlung für Ingenieure und Naturwissenschaftler. Vieweg/GWV Fachverlage, Wiesbaden (2003)
41. Rauber, T., Rünger, G.: Parallele Programmierung. Springer, Heidelberg (2007)
42. Stevens, W.R.: Programmieren von UNIX-Netzwerken. Hanser, München Wien (2000)
43. Tanenbaum, A.S.: Modern Operating Systems. Prentice Hall, (2008)
44. Tanenbaum, A.S.: Distributed operating systems. Prentice Hall, (1995)
45. Westermann, T.: Mathematik für Ingenieure mit Maple, Bd. 1. Springer, Berlin Heidelberg New York (1996)
46. Wilkinson, B.: Grid Computing – Techniques and Applications. Chapman & Hall/CRC, Boca Raton, FL (2011)

Spezifikationen

47. DIN 44302. Informationsverarbeitung – Datenübertragung, Datenübermittlung – Begriffe., 1987
48. GLX & GLU Libraries. Internet: www.opengl.org/documentation/specs/glu/glu1_3.pdf [6. Oktober 2011]
49. GLUT 3 Specification. Internet: www.opengl.org/documentation/specs/glut/spec3/spec3.html [6. Oktober 2011]
50. ISO/IEC 14977: 1996(E). Internet: www.cl.cam.ac.uk/~mgk25/iso-14977.pdf
51. NVIDIA. NVIDIA CUDA C Programming Guide, Version 3.2 [22. October 2010]
52. Object Management Group. UML Version 2.4 - Beta 2, Superstructure specification, ptc/2010-11-14
53. Object Management Group. UML Version 2.4 - Beta 2, Infrastructure specification, ptc/2010-11-16
54. Object Management Group. MOF 2.0/XMI Mapping, XMI specification, formal/2007-12-01
55. OpenMP. OpenMP Specification, Version 3.0, Mai 2008
56. RFC. RFC 5531 – RPC: Remote Procedure Call Protocol Specification Version 2. Internet: tools.ietf.org/html/rfc5531
57. RFC. RFC 4648 – The Base16, Base32, and Base64 Data Encodings, Internet: tools.ietf.org/html/rfc4648
58. RFC. RFC 4510 – Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map. Internet: tools.ietf.org/html/rfc4510

59. RFC. RFC 2616 – Hypertext Transfer Protocol – HTTP/1.1. Internet: www.w3.org/Protocols/rfc2616/rfc2616.html
60. RFC. RFC 2518 – HTTP Extensions for Distributed Authoring – WEBDAV.
61. RFC. RFC 2083 – PNG (Portable Network Graphics) Specification Version. Internet: www.faqs.org/rfcs/rfc2083.html
62. RFC. RFC 1831 – RPC: Remote Procedure Call Protocol Specification Version 2. Internet: tools.ietf.org/html/rfc1831
63. RFC. RFC 797 – FORMAT FOR BITMAP FILES. Internet: www.faqs.org/rfcs/rfc797.html
64. The BitTorrent Protocol Specification. Internet: www.bittorrent.org/beps/bep_0003.html [Version 11031]

Webseiten

65. Akismet. Comment spam prevention for your blog. Internet: akismet.com
66. Amazon. Amazon Web Services Cloud. Internet: aws.amazon.com
67. Amazon. Amazon Web Services Cloud, Service Health Dashboard, Internet: status.aws.amazon.com
68. AnandTech. ARM's Mali-400 MP4 is the Fastest Smartphone GPU...for Now. Internet: www.anandtech.com/show/4760/arms-mali400-mp4-is-the-fastest-smartphone-gpufor-now [24. Oktober 2011]
69. Apache. Apache Core Features. Internet: httpd.apache.org/docs/2.0/mod/core.html
70. Apache. Apache Security Tips. Internet: httpd.apache.org/docs/current/misc/security_tips.html
71. Apache. Apache Tutorial: .htaccess files. Internet: httpd.apache.org/docs/current/howto/htaccess.html
72. Astronews. PC-User suchen jetzt auch nach Radiopulsaren, Internet: www.astronews.com/news/artikel/2009/03/0903-034.shtml
73. Bablok, B.: Zauberlehrling – Automatische Bildbearbeitung mit Imagemagick. Internet: www.linux-magazin.de/Heft-Abo/Ausgaben/2008/08/Zauberlehrling [August 2008]
74. Big Buck Bunny. Internet: www.bigbuckbunny.org
75. Biochemical Library. Internet: boinc.vanderbilt.edu/bcl
76. boingboing. Stolen laptop recovered thanks to SETI@home software. Internet: boingboing.net/2007/02/22/stolen-laptop-recove.html
77. BOINC. Internet: boinc.berkeley.edu
78. BOINC. Download. Internet: boinc.berkeley.edu/download.php
79. BOINC. Administrative web interface. Internet: boinc.berkeley.edu/trac/wiki/HtmlOps [Version 6]
80. BOINC. Application graphics. Internet: boinc.berkeley.edu/trac/wiki/GraphicsApi [7. Oktober 2011]
81. BOINC. API for remote job submission. Internet: boinc.berkeley.edu/trac/wiki/RemoteJobs [Version 23]
82. BOINC. Assigned work. Internet: boinc.berkeley.edu/trac/wiki/AssignedWork [Version 11]
83. BOINC. Backend program logic. Internet: boinc.berkeley.edu/trac/wiki/BackendLogic [Version 2]
84. BOINC. Data distribution. Internet: boinc.berkeley.edu/trac/wiki/ProjectOptions#Datadistribution [Version 114]
85. BOINC. Dealing with numerical discrepancies. Internet: boinc.berkeley.edu/trac/wiki/HomogeneousRedundancy [Version 15]
86. BOINC. Hierarchical upload/download directories. Internet: boinc.berkeley.edu/trac/wiki/DirHierarchy [Version 3]
87. BOINC. Low-level validator framework. Internet: boinc.berkeley.edu/trac/wiki/ValidationLowLevel [Version 7]

88. BOINC. Prerequesite Software. Internet: boinc.berkeley.edu/trac/wiki/SoftwarePrereqsUnix
89. BOINC. Profile screening and UOTD selection. Internet: boinc.berkeley.edu/trac/wiki/ProfileScreen [Version 2]
90. BOINC. Project configuration.
Internet: boinc.berkeley.edu/trac/wiki/ProjectOptions [Version 114]
91. BOINC. Project web site.
Internet: boinc.berkeley.edu/trac/wiki/ProjectMain#Projectwebsite [Version 80]
92. BOINC. Project list. Internet: boinc.berkeley.edu/wiki/Project_list
93. BOINC. Referenz – Praktische Einführung in das Hochleistungsrechnen mit BOINC.
Internet: www.christianbenjaminries.de/boinc_reference
94. BOINC. Setting up a BOINC server.
Internet: boinc.berkeley.edu/trac/wiki/ServerIntro [Version 64]
95. BOINC. Submitting jobs.
Internet: boinc.berkeley.edu/trac/wiki/JobSubmission [Version 19]
96. BOINC. The BOINC Wrapper.
Internet: boinc.berkeley.edu/trac/wiki/WrapperApp [Version 51]
97. BOINC. The old BOINC graphics API.
Internet: boinc.berkeley.edu/trac/wiki/GraphicsApiOld [7. Oktober 2011]
98. BOINC. The project configuration file.
Internet: boinc.berkeley.edu/trac/wiki/ProjectConfigFile [Version 3]
99. BOINC. Web site customization.
Internet: boinc.berkeley.edu/trac/wiki/WebConfig [Version 19]
100. BURP. Internet: burp.renderfarming.net
101. Chess@home. Internet: chessathome.org
102. Chip.de. Datenverlust & Server-Ausfall. Internet: business.chip.de/artikel/Sicherheitstipps-fuer-Cloud-Computing-3_41255852.html [7. Februar 2010]
103. Computational Materials Science & Engineering – CMSE. Internet: www.fh-bielefeld.de/cmse
104. Computerwoche.de. Wie deppert muss man sein? Internet: www.computerwoche.de/management/cloud-computing/1908076/ [16. Oktober 2009]
105. ComsolGrid. ComsolGrid. Internet: comsolgrid.sourceforge.net
106. COMSOL. Installing and running COMSOL 4.2 on a Linux cluster, Internet: www.comsol.it/support/knowledgebase/1001/files/3789/cluster_install_linux_42.pdf
107. Code Project. Gerald Gibson Jr. Compress Zip files with Windows Shell API and C#. Internet: www.codeproject.com/KB/cs/compress\T1\ss-withwinshellapics.aspx [24. Oktober 2005]
108. COMSOL Multiphysics. Internet: www.comsol.com
109. Constellation. Internet: aerospaceresearch.net/constellation/
110. Cross-platform application and UI framework. Internet: qt.nokia.com
111. cURL. Internet: http://curl.haxx.se
112. Climateprecition. Internet: climateprediction.net
113. CycleComputing. Internet: www.cyclecomputing.com
114. Datenrettung-Fakten.de. Daten in der Cloud - Amazons Datenverlust Disaster. Internet: www.datenrettung-fakten.de/Daten-in-der-Cloud-Amazons-Datenverlust-Disaster.html [3. Mai 2011]
115. Debian. Running 32-bit Applications on 64-bit Debian GNU/Linux,
Internet: www.debian-administration.org/articles/534 [18. Juni 2007]
116. die.net. libzip(3) – Linux man page. Internet: linux.die.net/man/3/libzip
117. DrugDiscovery@Home. Internet: boinc.drugdiscoveryathome.com
118. Duden – Bibliographisches Institut GmbH. Internet: www.duden.de
119. Earth Simulator. Internet: www.jamstec.go.jp/esc/index.en.html
120. Einstein@Home. About. Internet: einstein.phys.uwm.edu/einsteinathome/about/index.html
121. Einstein@Home. Pulsar Discoveries in Parkes Multibeam Suvey Data. Internet: einstein.phys.uwm.edu/radiopulsar/html/PMPS_discoveries

122. Einstein@home. Teilnehmer finden Pulsar. Internet: www.heise.de/newsticker/meldung/Einstein-home-Teilnehmer-finden-Pulsar-1058162.html
123. Einstein@home. What is a radio pulsar?. Internet: einstein.phys.uwm.edu/radiopulsar/html/topic1.php [27. September 2011]
124. Enigma@Home. Internet: www.enigmaathome.net
125. FastCGI. Internet: www.fastcgi.com/drupal
126. FFmpeg. Internet: hwww.ffmpeg.org
127. Fokus.com. Verteiltes Rechnen – Aliensuche hilft bei Verbrecherjagd. Internet: www.focus.de/digital/multimedia/glasers_modernste_zeiten/verteiltes-rechnen_aid_53668.html
128. freeglut. Internet: freeglut.sourceforge.net
129. GIF Graphics Interchange Format. Internet: www.digitalpreservation.gov/formats/fdd/fdd000133.shtml
130. gLite – Lightweight Middleware for Grid Computing. Internet: glite.cern.ch
131. GLUT – The OpenGL Utility Toolkit. Internet: www.opengl.org/resources/libraries/glut [6. Dezember 2010]
132. GNU Make. Internet: www.gnu.org/software/make/
133. Golem.de. Made in Japan. Internet: www.golem.de/1106/84353-2.html
134. GPU Computing – CUDA. Internet: www.cst.com/Content/Products/MWS/GPU.aspx [16. Oktober 2011]
135. GPUGRID.net. Internet: www.gpugrid.net
136. Rational Rhapsody. Internet: www-01.ibm.com/software/awdtools/rhapsody
137. ImageMagick. Internet: www.imagemagick.org
138. ImageMagick. Magick++ API. Internet: www.imagemagick.org/Magick++
139. ImageMagick. Image Effects Routines. Internet: www.usf.uni-osnabrueck.de/infoservice/doc/localhtml/ImageMagick/www/api/effects.html
140. JPEG. Internet: www.jpeg.org/jpeg/index.html
141. LHC@home. Internet: lhcathome.cern.ch/lhcathome/
142. LINPACK. Internet: www.netlib.org/linpack/
143. LMBoinc. Internet: lmboinc.sourceforge.net
144. M23. Internet: m23.sourceforge.net
145. Magick::Image Class. Internet: www.imagemagick.org/Magick++/Image.html
146. Microsoft. 32-bit and 64-bit Windows: frequently asked questions. Internet: windows.microsoft.com/en-US/windows-vista/32-bit-and-64-bit-Windows-frequently-asked-questions
147. Microsoft. Farbmodelle. Internet: msdn.microsoft.com/de-de/library/cc295135.aspx
148. Microsoft. SetThreadPriority function. Internet: msdn.microsoft.com/en-us/library/ms686277 [30. November 2010]
149. Microsoft. Interprocess Communications. Internet: msdn.microsoft.com/en-us/library/aa365574 [11. April 2010]
150. MilkyWay@Home. Internet: milkyway.cs.rpi.edu/milkyway
151. Mod Auth MySQL Under Apache 2 and Debian. Internet: www.howtoforge.com/mod_auth_mysql_apache2_debian
152. National Astronomy and Ionosphere Center – Arecibo Observatory. Internet: www.naic.edu
153. Netcraft. August 2009 Web Server Survey. Internet: news.netcraft.com/archives/2009/08/31/august_2009_web_server_survey.html
154. NVIDIA. Developer Zone. Internet: developer.nvidia.com/nvidia-graphics-sdk-11
155. NVIDIA. Developer Zone, CUDA Download. Internet: developer.nvidia.com/cuda-downloads
156. OpenOffice.org. OpenOffice.org – The Free and Open Productivity Suite. Internet: www.openoffice.org
157. Oracle. grid Engine. Internet: www.oracle.com/us/products/tools/oracle-grid-engine-075549.html
158. Programming Escape Characters. Internet: www.wilsonmar.com/1eschars.htm
159. phpMyAdmin. Internet: www.phpmyadmin.net

160. Red Hat. Red Hat Documentation, Kapitel 18.3.5, Automating the Installation with Kickstart, Installation Guide. Internet: docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/index.html [18. Oktober 2011]
161. Red Hat. Red Hat Documentation, Chapter 31. Kickstart Installations. Internet: docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/5/html/Installation_Guide/ch-kickstart2.html
162. Rosetta@home. Internet: boinc.bakerlab.org/rosetta/
163. Sed. An Introduction and Tutorial. Internet: www.grymoire.com/Unix/Sed.html
164. Sed. The Sed FAQ. Internet: sed.sourceforge.net/sedfaq.html
165. Selfhtml. Die CGI-Schnittstelle. Internet: de.selfhtml.org/intro/schnittstellen/cgi.htm [16. Oktober 2011]
166. SETI@home. Internet: setiathome.berkeley.edu/
167. SETI@home. SETI@home wird fündig – Gestohlenes Notebook meldet sich via SETI@home. Internet: www.golem.de/0702/50680.html
168. SETI@home. Forumeintrag vom 25.08.2010. Internet: www.seti-germany.de/forum/das-hauptforum/4706-computerdiebstahl.html
169. Simple Help, How to run OS X programs in 32-bit mode. Internet: www.simplehelp.net/2009/09/15/how-to-run-os-x-programs-in-32-bit-mode/ [15. September 2009]
170. SOCI. The C++ Database Access Library. Internet: soci.sourceforge.net/doc/index.html
171. Spinhenge@home. Internet: spin.fh-bielefeld.de
172. Star Trek Borg. Internet: www.startrek.com/database_article/borg
173. The Open Group Base Specifications Issue 6 IEEE Std 1003.1, 2004 Edition. Internet: pubs.opengroup.org/onlinepubs/009695399/mindex.html
174. The software Rogue. Internet: hthesoftwarerogue.blogspot.com/2010/05/porting-of-libcurl-to-android-os-using.html [20. Oktober 2011]
175. The Zip, GZip, BZip2 and Tar Implementation For .NET. Internet: www.icsharpcode.net/OpenSource/SharpZipLib
176. Theora video compression. Internet: www.theora.org
177. Ubuntu. Internet: https://help.ubuntu.com/10.04/installation-guide/i386/automatic-install.html
178. Ubuntu. Ubuntu Server. Internet: www.ubuntu.com/server
179. UNIX Timestamp. Internet: www.unixtimestamp.com
180. University of Westminster. New DIY supercomputer saves £1,000s. Internet: www.westminster.ac.uk/about/news-and-events/news/2011/university-of-westminster-launches-new-diy-supercomputer-saving-hundreds-of-thousands-of-pounds [29. März 2011]
181. VirtualBox. Internet: https://www.virtualbox.org
182. VirtualBox. VirtualBox Manual. Internet: www.virtualbox.org/manual/ch01.html
183. Wikipedia. Internet: de.wikipedia.org/wiki
184. Wikipedia. Asynchrone Kommunikation. Internet: de.wikipedia.org/wiki/Asynchrone_Kommunikation [4. September 2010]
185. Wikipedia. CPUID. Internet: en.wikipedia.org/wiki/CPUID [2. November 2011]
186. Wikipedia. Einstein@home. Internet: en.wikipedia.org/wiki/Einstein@Home
187. Wikipedia. M23. Internet: en.wikipedia.org/wiki/M23_software_distribution_system
188. Microsoft. Windows Azure, Cloud Services-Platform. Internet: www.microsoft.com/de-de/azure/
189. WirtschaftsWoche. Cloud Computing im Alltag. Internet: www.wiwo.de/technik-wissen/galerien/cloud-computing-im-alltag-1423/1/alles-im-netz.html [23. Oktober 2011]
190. WSJ: GPUs Speed Up Risk Computations 40x at JP Morgan, Internet: insidehpc.com/2011/08/04/wsj-gpus-speed-up-risk-computations-40x-at-jp-morgan/ [8. April 2011]
191. WUProb@home. Internet: wuprop.boinc-af.org

Vermischtes

192. BOINC-Quellen. Revision 22328, checking_notes_2008
193. BOINC-Quellen. Revision 22328, checking_notes_2005
194. European Information Technology Observatory, EITO Country Report Germany 2011. Internet: www.eito.eu/EITO-Country-Report-Germany-2011 [22. Juni 2011]
195. Leider ist die Quelle unbekannt, sollten Sie diese kennen, so teilen Sie uns diese bitte mit.

Sachverzeichnis

X

Z