I remember one of the first things that attracted me to computers--well, besides Pac-Man--was the blinking lights. Blue, amber, red, or green, I didn‚Äôt really care about the color, rather what they meant. I was curious as to what exactly was that light reporting? I‚Äôm the type of fellow who likes to take things apart to learn what makes them tick.

It started with an altimeter that my father, who was in the Air Force, brought home. I was instantly curious about how it worked. It was a good thing it was a gift and not something that had to be put back on the plane as springs and gears shot out shortly after cracking it open. Like Humpty Dumpty, I was never able to get that altimeter back together again, however it did increase my hunger to understand not just how something works, but how do you understand what‚Äôs going on?

SYSSTAT is a software application comprised of several tools that offers advanced system performance monitoring. It provides the ability to create a measurable baseline of server performance, as well as the capability to formulate, accurately assess and conclude what led up to an issue or unexpected occurrence. In short, it lets you peel back layers of the system to see how it‚Äôs doing... in a way it is the blinking light telling you what is going on, except it blinks to a file. SYSSTAT has broad coverage of performance statistics and will watch the following server elements:

- Input/Output and transfer rate statistics (global, per device, per partition, per network filesystem and per Linux task / PID)
- CPU statistics (global, per CPU and per Linux task / PID), including support for virtualization architectures
- Memory and swap space utilization statistics
- Virtual memory, paging and fault statistics
- Per-task (per-PID) memory and page fault statistics
- Global CPU and page fault statistics for tasks and all their children
- Process creation activity
- Interrupt statistics (global, per CPU and per interrupt, including potential APIC interrupt sources)
- Extensive network statistics: network interface activity (number of packets and kB received and transmitted per second, etc.) including failures from network devices; network traffic statistics for IP, TCP, ICMP and UDP protocols based on SNMPv2 standards.
- NFS server and client activity
- Socket statistics
- Run queue and system load statistics
- Kernel internal tables utilization statistics
- System and per Linux task switching activity

- Swapping statistics
- TTY device activity

(List source - http://pagesperso-orange.fr/sebastien.godard/features.html)

Scope

This article covers a brief overview of how the SYSSTAT utility works, initial configuration, deployment and testing on Linux based servers. It includes an optional system configuration guide for writing SYSSTAT data into a MySQL database. This article is not intended to be an in-depth explanation of the inner workings of SYSSTAT, nor a detailed manual on database storage operations.

Now... on to the interesting parts of SYSSTAT!

Overview

The SYSSTAT software application is composed of several utilities. Each utility has a specific function:

- **iostat** reports CPU statistics and input/output statistics for devices, partitions and network filesystems.
- **mpstat** reports individual or combined processor related statistics.
- **pidstat** reports statistics for Linux tasks (processes) : I/O, CPU, memory, etc.
- **sar** collects, reports and saves system activity information (CPU, memory, disks, interrupts, network interfaces, TTY, kernel tables, NFS, sockets etc.)
- **sadc** is the system activity data collector, used as a backend for sar.
- **sa1** collects and stores binary data in the system activity daily data file. It is a front end to sadc designed to be run from cron.
- **sa2** writes a summarized daily activity report. It is a front end to sar designed to be run from cron.
- **sadf** displays data collected by sar in multiple formats (CSV, XML, etc.) This is useful to load performance data into a database, or import them in a spreadsheet to make graphs.

(List source - http://pagesperso-orange.fr/sebastien.godard/documentation.html)

The four main components used in collection activities are **sar**, **sa1, sa2** and **cron**. **Sar** is the
s
ystem
a
ctivity
r
eporter. This tool will display interpreted results from the collected data. Sar is ran interactively by an administrator via command line. When a sar file is created, it is written into the
/var/log/sa
directory and named
sar##.
The
##
is a numerical value that represents the day of the month (i.e.
sa03
would be the third day of the month). The numerical value changes accordingly without system administrator intervention. There are many option flags to choose from to display data in a sar file to view information about server operations, such as cpu, network activity, NFS and sockets. These options can be viewed by reviewing the man pages of sar.

Sa1 is the internal mechanism that performs the actual statistical collection and writes the data to a binary file at specified times. Information is culled from the **/proc** directory where the Linux kernel writes and maintains pertinent data while the operating system is running. Similar to sar, the binary file is written into
/var/log/sa
and named
sa##
. Again, the
##
represents the day of the month (i.e.
sar03
would be the third day of the month). Once more, the numerical value changes accordingly without system administrator intervention.

Sa2 is responsible for converting the sa1 binary file into a human readable format. Upon successful creation of the binary file **sa##** it becomes necessary to set up a cron task that will call the sa2 libraries to convert the sa1 binary file into the human-readable sar file. SYSSTAT utilizes the scheduled cron command execution to draw and record specified performance data based upon pre-defined parameters. It is not necessary to run the
sa2
cron at the same time or as often as the
sa1

cron. The
sa2
function will create and write the sar file to the /var/log/sa directory.

How often SYSSTAT ‚Äúwakes up‚Äù to record and what data is captured, is determined by your operational needs, regulatory requirements and purposes of the server being monitored. These logs can be rotated to a central logging server and stored for analysis at a later date if desired.

SYSSTAT Configuration

Now that you have the 40,000-foot overview of the components, onward to the nitty gritty of building out your SYSSTAT capabilities. The following is a suggested base configuration. You can tweak for your environment, but I will step through a traditional set up. My testing environment utilized a SUSE 10 Linux server.

As a side bar, every Linux-based server I have come across, installed or worked with has the SYSSTAT package deployed as part of the base server set up at installation. I would suggest however that you always look at the option to upgrade to the latest version of SYSSTAT. It will offer bug correction and more performance monitoring elements such as:

- Autoconf support added.
- Addition of a new command ("pidstat") aimed at displaying statistics for processes, threads and their children (CPU, memory, I/O, task switching activity...)
- Better hotplug CPU support.
- New VM paging metrics added to sar -B.
- Added field tcp-tw (number of sockets in TIME_WAIT state) to sar -n SOCK.
- iostat can now display the registered device name of device-mapper devices.
- Timestamped comments can now be inserted into data files created by sadc.
- XML Schema document added. Useful with sadf -x.
- National Language Support improved: Added Danish, Dutch, Kirghiz, Vietnamese and Brazilian Portuguese translations.
- Options -x and -X removed from sar. You should now use pidstat instead.
- Some obsolete fields (super*, dquot* and rtsig*) were removed from sar -v. Added field pty-nr (number of pseudo-terminals).

(List source - http://pagesperso-orange.fr/sebastien.godard/features.html)

Create Scheduled SYSSTAT Monitoring

First things first--we need to tell our machine to record sar data. The kernel needs to be aware that it is to run SYSSTAT to collect metrics. Depending upon your distribution, at installation it creates a basic cron named **sysstat.cron** in /etc/sysstat/ for SUSE, or in /etc/cron.d/sysstat for Red Hat. If you use SUSE, it is recommended to create SYSSTAT,Äôs cron job as a soft link assignment in /**etc**/**cron.d** pointing to the sysstat.cron in /etc/sysstat/. I would also suggest that several gathering times be created based on times when a server has the potential to be more active. This will ensure collection of accurate statistics. What good is it to see that your server is never busy at 2 in the morning? Data collected during off peak hours would skew later analysis and has the potential to cause erroneous interpretation.

```
#Crontab for sysstat
#Activity reports culled and updated every 10 minutes everyday
-*/10 *   * * *     root  /usr/lib/sa/sa1
#Update log sar reports every day at 2330 hours
30 23 * * *      root  /usr/lib/sa/sa2 ,ÄìA
```

After the softlink has been created, restart the cron daemon to allow it to reload the new assignment:

```
# rccron restart
```

Let's use a script to create a sar backup file and offload to a specified location:

```
#!/bin/bash
# Created 04-AUG-09 / Kryptikos

# Script to create backup and rename with current hostname and date of sar file and offload to storage facility.

# Cycle through directory once looking for pertinent sar files.

ls -1 /var/log/sa/sar* | while read sarname
      do
        mv "$sarname" $(echo "$HOSTNAME"_"$sarname "_`date +"%Y%m%d,Äù`.bkup)
      done

# This section will need to be modified. It is a place holder for code to offload the designated sar backup
# file just created to the localhost, a central logging host or database server. This is dependent upon your ops.
# i.e. scp /var/log/<sarlogname> <username>@<host>:/<desired location>
```

<insert some code to handle the transfer via scp / ssh / mv / nc / or other method>

exit

SYSSTAT will now run and collect the sar log, rename it and then offload it to the location you prefer.

Well that,Äôs great you say, but what if you don,Äôt have time to comb through 30 days worth of sar reports and just need a quick snap shot? Another neat thing you can do with this tool is capture real time statistics of what is going on with your machine. For instance at the command line you can enter:

sar ,Äìn NFS 5 3

That will have SYSSTAT report all NFS activity in five second intervals for 3 times and report it back to the terminal.

root@mymachine # **sar -n NFS 5 3**
Linux 2.6.18 (mymachine) 08/04/2009

	call/s	retrans/s	read/s	write/s	access/s	getatt/s
02:50:39 PM						
02:50:44 PM	0.00	0.00	0.00	0.00	0.00	0.00
02:50:49 PM	0.00	0.00	0.00	0.00	0.00	0.00
02:50:54 PM	0.00	0.00	0.00	0.00	0.00	0.00
Average:	0.00	0.00	0.00	0.00	0.00	0.00

So you have your sar data recording, and you now know how to use it for real-time checking. You are still left with quite a bit of data to comb through. It might be that you don,Äôt need to look through your data until there is a problem and you,Äôd like to track back at what point in time your server started having issues. The next section of the article deals with an advanced configuration for storage of the sar data for later retrieval.

MySQL Database Configuration

If you are running a server that does not carry a heavy user load it is okay to have the sar data stay local on the box. However, with the large volumes of system performance data that will be collected from a Linux server farm running numerous applications, I would suggest establishing a database for storing the relevant SYSSTAT information. By utilizing a MySQL database, customized data may be reviewed at any time and allow for the creation of reports, including charts, that are more granular in nature. It would also allow for analysis of cross sections of pertinent SYSSTAT data from multiple servers at one time. This would alleviate requiring an administrator/engineer reviewing individual sar log files attempting to troubleshoot or identify

issues line by line. The use of a database decreases the time required to locate and diagnose root cause(s) of a server issue. This section covers database creation, setup and methodology to import the recorded logs. It is recommended to install and utilize MySQL version 5.1 or later for utilization of enhanced features and increased performance.

Start the MySQL Daemon

It stands to reason that to run the MySQL daemon you must have already installed MySQL. If you have not installed MySQL now,Äôs the perfect time to pause and grab the latest copy. You should be able to utilize your distribution,Äôs package manager to install the database, however if not, it is just as easy (and the method I typically use) to simply pull a copy from http://dev.mysql.com/downloads/mysql/5.1.html#downloads
. Once you have MySQL installed you can start it with the following command:

```
# <location of mysql>/bin/mysqld_safe --user=mysql --local-infile=1
```

The option **--user=** tells the daemon to run as user **mysql** (must be a local POSIX account). It is **not recommended** to run the MySQL daemon as **root**. The option **--local-infile=1** tells the daemon to enable **LOAD DATA LOCAL INFILE**, which allows pushing tabbed, csv and txt files into a database from a file stored locally on the MySQL server.

Create the SYSSTAT Database

Getting down to business now that the database is up and running, it is time to create the infrastructure we want to hang our sar data upon.

1. Connect to the MySQL server from the command line: # <location of mysql >/bin/ mysql --user=<username> -p Again, the user must exist on the MYSQL server; it is not a POSIX user account. The **,Äìp** prompts for a password to connect to the MYSQL server.
2. From the MySQL prompt check that the database does not already exist: mysql > SHOW databases;
3. Create the database: mysql > CREATE DATABASE <name of database>;
4. Grant privileges on the newly created database to a specified MySQL user account: mysql > GRANT ALL ON <name of database>.* TO ‚Äò<mysql username>,Äô@,Äôlocalhost,Äô;

This grants the MYSQL user specified full control over the database but only when connecting from the localhost the MYSQL daemon is running on. If you prefer to access from alternative locations for administrative purposes, execute the additional command:

mysql > GRANT ALL ON <name of database>.* TO ‚Äò<mysql username>‚Äô@‚Äô%‚Äô

It is possible to control and granulize access via certain networks or domains. For security, if a database is deployed, I would create a ‚Äúworkhorse‚Äù account to perform the upload. This workhorse account would only have **UPDATE** privileges on the desired tables. In my case I chose to name my database ‚Äúsystat_collection‚Äù.

Before:

+------------------------------+

| Database |

+------------------------------+

| information_schema |

| menagerie |

| MySQL |

```
| test                        |

+-----------------------------+
```

4 rows in set (0.00 sec)

After:


+------------------------------+
| Database |

```
+------------------------------+
```

```
| information_schema           |
```

| menagerie |

| MySQL |

| **sysstat_collection** |

| test |

```
+------------------------------+
```

5 rows in set (0.00 sec)

Create the Necessary Tables

We now have our database, but we need to take it one more step. The database has to be ‚Äúmade-ready‚Äù to accept incoming sar data. This is done by building tables. Think of tables as a bookcase. Each block (table) will hold books (sar data). The easiest method to insert tables into your database is to create and utilize sql scripts. These scripts can be quickly invoked by the MySQL daemon and pushed inside the database. Each script should have a unique name that ends with the **.sql** extension. A basic SYSSTAT configuration would require 18 tables. I‚Äôve written an example sql script for you to use:

Example SQL script: create_cpuutilization_table.sql

```
# Created 04-AUG-09 / Kryptikos

 # Drop the CPU UTILIZATION table if it exists, then recreate it.

 DROP TABLE IF EXISTS cpuutilization;

 CREATE TABLE cpuutilization
(
  hostname    VARCHAR(20),
  datestamp   DATE,
  time        VARCHAR(8),
  cpu         VARCHAR(3),
  pct_user    DECIMAL(10,2),
  pct_nice    DECIMAL(10,2),
  pct_system  DECIMAL(10,2),
  pct_iowait  DECIMAL(10,2),
  pct_steal   DECIMAL(10,2),
  pct_idle    DECIMAL(10,2)
);
```

Breaking that down into understandable chunks, CPU utilization is one of the items SYSSTAT will record. SYSSTAT will stamp the kernel, hostname, date, time and then sar value in the string (see the output in the ‚Äúreal-time‚Äù example earlier in the article). I know what kernel version I am using so really all I am interested in is hostname (because I capture multiple servers to this database), date, time and cpu elements. Each sar value has its own elements. The quickest way to view this is to use the man pages of sar to see what values sar records.

Remembering thinking about a database table as a bookshelf, the above values hostname, datestamp, time, cpu, pct_user, pct_nice, etc. are the open shelves. You have to have the shelf

before you can place a book. For each book type (sar element) you have you need a shelf (table point).

Deploy the Tables into the MySQL Database

The great thing about **.sql** scripts is they can be invoked directly by the MySQL daemon. It is not necessary to log into and obtain a MySQL prompt from the server. You can simply feed the script file to the daemon which will parse and execute the commands on your behalf. Lather, rinse and repeat for each table you wish to insert into your database, or create a bash script to feed the .sql scripts all at once. The following is the command structure to execute the table creation script:

/usr/local/ mysql /bin/ mysql -u <user> -p -D <database> < create_cpuutilization_table.sql

Again, the **‚Äìu** tells the daemon to run the script as the specified MySQL user account (not POSIX). The user must have **privileges** on the database to allow modification. The **‚Äìp** again prompts for the user password and the **‚ÄìD** specifies which database you want to execute the script contents upon. The re-direct sign ‚Äú<‚Äù feeds the script to the daemon. You can by-pass the password prompt and stream your password directly in by changing
‚Äìp
to
--password=<passwd value>.

Loading SYSSTAT Logs Into the MySQL Database

Moving right along, once the cron job and backup script has run, it is necessary to format the data from the sar file in order to prepare it for loading into the SYSSTAT database. The elegance of **sar** is it loads the data into tabulated columns in one large file and breaks apart sections by blank lines. By utilizing the stream editor (sed) and translate capabilities we can quickly parse the sar file into bite size pieces ready to load into its respective table. A few things are worthy to note here. I like to recommend processing sar data before 0000 hours (midnight) to maintain time/date integrity. Second, in preparation I like to stash and load sar data from its own directory source, say /var/log/sysstatdbprepare, or from the /tmp directory. In the environment I work in I have numerous servers reporting and prefer one location where logs are stored for sar.

The script listed below is an example of formatting and uploading data into a database. Variables inside the script should be changed to fulfill operational requirements. Spaces are included to increase legibility in this article:

```
 #!/bin/bash
# Created 04-AUG-09 / Kryptikos

# This script will parse the sar file and prepare/format the data to make it
# available to upload into a MySQL database. It will then call and upload the
# data into the selected MySQL database and its respective tables.

# Set miscellaneous variables needed.
DATESTAMP=$(date '+%Y-%m-%d')
WORKDIR=/tmp/sysstatdbprepare
FMATDIR=/tmp/sysstatdbformatted

# Begin main.

# Change into work directory.

cd $WORKDIR

# Start preprocessing formatting.

for file in `dir -d *`;
do

# Prepare and format designated hosts' sar log files to be loaded into MYSQL database:

 sed -n "/proc/,/cswch/ p" $file | sed "$ d" | tr -s [:blank:] | sed -n '1h;2,$H;${g;s/ /,/g;p}' | sed
'/Average:/ d' | sed "s/^/$HOSTNAME,$DATESTAMP,/" | sed '$d' >
"$FMATDIR"/"$file"_taskcreation.csv

 sed -n "/cswch/,/CPU/ p" $file | sed "$ d" | tr -s [:blank:] | sed -n '1h;2,$H;${g;s/ /,/g;p}' | sed
'/Average:/ d' | sed "s/^/$HOSTNAME,$DATESTAMP,/" | sed '$d' >
"$FMATDIR"/"$file"_systemswitchingactivity.csv

 sed -n "/user/,/INTR/ p" $file | sed "$ d" | tr -s [:blank:] | sed -n '1h;2,$H;${g;s/ /,/g;p}' | sed
'/Average:/ d' | sed "s/^/$HOSTNAME,$DATESTAMP,/" | sed '$d' >
"$FMATDIR"/"$file"_cpuutilization.csv

 sed -n "/INTR/,/CPU/ p" $file | sed "$ d" | tr -s [:blank:] | sed -n '1h;2,$H;${g;s/ /,/g;p}' | sed
'/Average:/ d' | sed "s/^/$HOSTNAME,$DATESTAMP,/" | sed '$d' >
"$FMATDIR"/"$file"_irqinterrupts.csv

 sed -n "/i000/,/pswpin/ p" $file | sed "$ d" | tr -s [:blank:] | sed -n '1h;2,$H;${g;s/ /,/g;p}' | sed
'/Average:/ d' | sed "s/^/$HOSTNAME,$DATESTAMP,/" | sed '$d' >
"$FMATDIR"/"$file"_inputactivityperprocperirq.csv

 sed -n "/pswpin/,/tps/ p" $file | sed "$ d" | tr -s [:blank:] | sed -n '1h;2,$H;${g;s/ /,/g;p}' | sed
```

```
'/Average:/ d' | sed "s/^/$HOSTNAME,$DATESTAMP,/" | sed '$d' >
"$FMATDIR"/"$file"_swappingstatistics.csv

 sed -n "/tps/,/frmpg/ p" $file | sed "$ d" | tr -s [:blank:] | sed -n '1h;2,$H;${g;s/ /,/g;p}' | sed
'/Average:/ d' | sed "s/^/$HOSTNAME,$DATESTAMP,/" | sed '$d' >
"$FMATDIR"/"$file"_iotransferrate.csv

 sed -n "/frmpg/,/TTY/ p" $file | sed "$ d" | tr -s [:blank:] | sed -n '1h;2,$H;${g;s/ /,/g;p}' | sed
'/Average:/ d' | sed "s/^/$HOSTNAME,$DATESTAMP,/" | sed '$d' >
"$FMATDIR"/"$file"_memorystatistics.csv

 sed -n "/TTY/,/IFACE/ p" $file | sed "$ d" | tr -s [:blank:] | sed -n '1h;2,$H;${g;s/ /,/g;p}' | sed
'/Average:/ d' | sed "s/^/$HOSTNAME,$DATESTAMP,/" | sed '$d' >
"$FMATDIR"/"$file"_ttydeviceactivity.csv

 sed -n "/IFACE/,/rxerr/ p" $file | sed "$ d" | tr -s [:blank:] | sed -n '1h;2,$H;${g;s/ /,/g;p}' | sed
'/Average:/ d' | sed "s/^/$HOSTNAME,$DATESTAMP,/" | sed '$d' >
"$FMATDIR"/"$file"_networkstatistics.csv

 sed -n "/rxerr/,/call/ p" $file | sed "$ d" | tr -s [:blank:] | sed -n '1h;2,$H;${g;s/ /,/g;p}' | sed
'/Average:/ d' | sed "s/^/$HOSTNAME,$DATESTAMP,/" | sed '$d' >
"$FMATDIR"/"$file"_networkstatisticserrors.csv

 sed -n "/call/,/scall/ p" $file | sed "$ d" | tr -s [:blank:] | sed -n '1h;2,$H;${g;s/ /,/g;p}' | sed
'/Average:/ d' | sed "s/^/$HOSTNAME,$DATESTAMP,/" | sed '$d' >
"$FMATDIR"/"$file"_networkstatisticsnfsclientactvty.csv

 sed -n "/scall/,/pgpgin/ p" $file | sed "$ d" | tr -s [:blank:] | sed -n '1h;2,$H;${g;s/ /,/g;p}' | sed
'/Average:/ d' | sed "s/^/$HOSTNAME,$DATESTAMP,/" | sed '$d' >
"$FMATDIR"/"$file"_networkstatisticsnfsserveractvty.csv

 sed -n "/pgpgin/,/kbmemfree/ p" $file | sed "$ d" | tr -s [:blank:] | sed -n '1h;2,$H;${g;s/ /,/g;p}' |
sed '/Average:/ d' | sed "s/^/$HOSTNAME,$DATESTAMP,/" | sed '$d' >
"$FMATDIR"/"$file"_pagingstatistics.csv

 sed -n "/kbmemfree/,/dentunusd/ p" $file | sed "$ d" | tr -s [:blank:] | sed -n '1h;2,$H;${g;s/
/,/g;p}' | sed '/Average:/ d' | sed "s/^/$HOSTNAME,$DATESTAMP,/" | sed '$d' >
"$FMATDIR"/"$file"_memoryswapspaceutilization.csv

 sed -n "/dentunusd/,/totsck/ p" $file | sed "$ d" | tr -s [:blank:] | sed -n '1h;2,$H;${g;s/ /,/g;p}' |
sed '/Average:/ d' | sed "s/^/$HOSTNAME,$DATESTAMP,/" | sed '$d' >
"$FMATDIR"/"$file"_inodefilekerneltable.csv

 sed -n "/totsck/,/runq-sz/ p" $file | sed "$ d" | tr -s [:blank:] | sed -n '1h;2,$H;${g;s/ /,/g;p}' | sed
'/Average:/ d' | sed "s/^/$HOSTNAME,$DATESTAMP,/" | sed '$d' >
"$FMATDIR"/"$file"_networkstatisticssocket.csv
```

```
 sed -n "/runq-sz/,// p" $file | sed "$ d" | tr -s [:blank:] | sed -n '1h;2,$H;${g;s/ /,/g;p}' | sed
'/Average:/ d' | sed "s/^/$HOSTNAME,$DATESTAMP,/" | sed '$d' >
"$FMATDIR"/"$file"_queuelengthloadavgs.csv

 done

 # Kick off uploading formatted data into MYSQL database:

 # Change into format directory.

 cd $FMATDIR

 # Pushing data into MYSQL via -e flag.

 for file in `dir -d *`;
 do

 /usr/local/MySQL/bin/MySQL -u <MySQLuser> --password=<password> -D <database> -e
"LOAD DATA LOCAL INFILE '/tmp/sysstatdbformatted/${file}' INTO TABLE `echo $file | sed
's/.csv//g' | awk -F_ '{print $2}'` FIELDS TERMINATED BY ',' IGNORE 1 LINES;"
```

Wrap-Up and Overview

That,Äôs it, just a few scripts and a little bit of time to set up automation to inject your data into a database. If everything is correct syntax wise on your scripts, you should now be able to log into the MySQL server and see the data loaded into the tables. I tend to use MySQL Administrator (GUI tool) to log in and look at the databases and tables. It is a bit quicker to use for checking.

The following overview points are suggested recommendations to implement SYSSTAT on your Linux server(s):

- Schedule cron via soft link assignment in **/etc/cron.d** pointing to the sysstat.cron in /etc/sysstat/.
- Schedule multiple **sa1** function crons: record statistics more often during higher utilization times and less during off-peak hours with respect to server operation.
- Utilize MYSQL database to store collected data for minimum of 30 - 45 days before purging records and restarting storage process.
- If database option is not chosen: write **sar** files to a central logging server and rename with hostname and current date values.
- If database option is not chosen: store renamed sar files for 25 - 30 days until purged by central logging server.

SYSSTAT can give you a wealth of information as to what is going on with your server. It gives you the chance to watch a historical trend of when your server is getting utilized, how heavy the use is and a host of other empirical data. It will allow you to focus and determine root-cause analysis if you suddenly find your server having issues. You are only limited to your imagination as to what you could use it for to compliment troubleshooting. If you do end up using a database there are packages out there that will generate pretty graphs for easier interpretation, or you could even scribble up some PHP code and pull up the data via a web browser.

The thing I love about Linux is how I can continue to break things apart, learn how they work and then deploy based on my needs. If you have suggestion as to content, or have questions please feel free to comment. Either way, hope this helped a little bit for your environment.

~Jonathan Peck (Kryptikos) is a Linux Systems Engineer for a major international company and has worked in Linux and Unix platforms since 2001. When not at work you can often find him on the Ubuntu Forums, Linux.com boards, and contributing articles and tips to Novell Cool Solutions. He,Äôs always interested in learning more, helping others as they explore the great world of Linux and marketing the penguin and other open source solutions.