Paper 010-2007

Building Your Own Real-Time SAS® Server Monitor under Unix

Houliang Li, Frederick, MD

ABSTRACT

No matter how abundant your production or development hardware resources may be, you will run into problems once in a while, whether it is CPU, memory, or disk space. If such problems occur too frequently, you need to take some preventive measures, such as getting an advance warning system. You can have devoted administrators watching over your Unix SAS servers around the clock with commercial monitoring tools, or you can build one from scratch. Base SAS has provided all the tools you need to construct your own real-time SAS server monitoring and reporting system, and this paper will show you how to put them together. The benefits to your organization are endless, and you get to play The Big Brother for your SAS users. Trust me, they will love you.

INTRODUCTION

One trend that seems to accelerate by the day is the growth of business data. Most companies are forced to play the game of catching up and are constantly investing in their hardware resources by buying more and faster CPUs, disks, and memory, sometimes every year but at least every other year. Yet for practical purposes, almost no company can afford to put the resources in place that will cover all the extreme cases in the near future. Inevitably, system overload happens once in a while. Or once in a couple of days! It could be the CPUs running at 100% for a sustained period of time, or the file systems are filled up, or the memory is fully utilized. Sometimes, that extreme case comes from an unusual spike in data volume, but more often, it originates from a careless mistake by a developer, or a curious business user experimenting with his newly acquired technique. The end result is always the same: production jobs and / or development efforts suffer, and the whole organization pays a price.

One solution to this problem is dedicated server administrators and commercial monitoring tools. Depending on how much the company is willing to pay and how capable the tools are, you may or may not get the best results for your investment. One frequent deficiency with this approach is the communication gaps between the server administrators and the SAS users. Another issue is the potential misfit between the commercial tools and the SAS users' needs.

Despite all the potential drawbacks with this approach, consider yourself lucky if you can afford this solution. Most small and medium-sized companies do not have this option at all due to its relatively high cost. Very often, one or a few power users are responsible for the daily health of the SAS server and act as virtual administrators. If this description fits you, you know the hassles of dealing with the consequences of an overloaded system. Welcome to the real world!

THE HIDDEN (AND FREE) SOLUTION

Base SAS provides the familiar DATA step, the Macro facility, and the built-in capability to interact with the operating system. Surprisingly, these are all you need to construct your real-time SAS server monitor. There is no need to buy anything extra from anyone, including additional SAS products. Simply combine the three elements above with your basic understanding of the Unix operating system and you have just promoted yourself to the position of The Big Brother, in the benign sense of George Orwell's ***Nineteen Eighty Four*** character. It is nice to have some limited administrator privileges on the SAS server, but that is not absolutely necessary.

Three simple steps will take you to your first real-time SAS server monitor. The first step is creating a macro program with an infinite loop in it. No, this is not a typo. We want a loop that will run forever, until either the SAS server is brought down for whatever reason or the monitoring program is manually terminated. However, we do not want the program to run continuously. That in itself creates a big burden on the CPU, contrary to our goal of maintaining a healthy server. The trick is having the monitoring program do its work periodically, for example, once every minute, and remain asleep during most of the period. Your past experiences play a key role in selecting the optimal interval.

Let's say you want to monitor the disk space usage. If your operations can write large volumes of data to the disk very quickly, you probably want to pick a small interval, for example, one minute. On the other hand, if the data growth is more gradual, you can choose a longer interval, say five or even ten minutes. How much free disk space you typically have should also inform you on the choice of the interval. In addition, how fast you normally respond to the notifications generated by the monitoring program should also be factored in. The goal here is to make sure that you are adequately warned before the file system is actually filled up by any fast-writing process. In a word, pick an interval that you can comfortably afford to ignore a few times without any adverse consequences.

The next step is deciding on what resources to monitor and setting the thresholds. You should already know the usual bottlenecks in your company's server resources. The culprit is either the CPU, or the memory, or the disk

space, or the I/O system, but it could be more than one. After identifying your target(s), you need to determine the thresholds at which certain preventive actions are automatically taken.

Let's continue with the disk space as an example. Suppose we want to monitor the Unix partition used exclusively for SAS temporary libraries. If its total space is 200GB, we may want to set five thresholds. The first can be a warning threshold, at 150GB. The SAS server administrator will be sent an email every time this amount, or more, of disk space is used by user sessions. A second threshold can be set at 180GB, but this time, not only the SAS server administrator will be notified, but the top three users with the biggest temporary libraries will be notified as well. The next threshold can be set at 190GB, at which point the process with the biggest temporary library will be killed and the library deleted. Again, both the owner of the process and the SAS server administrator will be notified. A separate threshold can be set up as the size limit on each SAS session's temporary library, for example 100GB. Whenever any user's single SAS session reaches the limit, the SAS process will be killed, but the library can be preserved if the first three thresholds have not been breached. This gives the owner a chance to use whatever results have been produced so far. But prior to the termination of the owner's process, a lower limit, say 80GB, is used to start the dissemination of reminders to the process owner in the hope that he or she can voluntarily terminate the session or delete unnecessary files from the session.

The third step is choosing the frequencies at which the notifications are sent as well as the contents of the notifications. The easiest solution is picking a short interval in the first step, as discussed above, so that there is a relatively high frequency to begin with. Suppose the interval is set at one minute. Again we use the disk space as our example. We can choose to send out a warning at or above 150GB, but only once every five minutes, thus avoiding inundating the SAS server administrator's inbox. At 180GB or above, we can send out the messages once every minute. The messages should contain detailed instructions about how to terminate the offending processes and cleaning up the temporary libraries. Once the 190GB threshold is reached, only one message is necessary, i.e., to notify the user and administrator of the session termination. It is appropriate to urge the offending user to seek help in order to avoid a repetition of the disaster. Similarly, a different frequency can be chosen for the warnings to the user with an individual session approaching the 100GB limit. The most frequent notifications may be appropriate, i.e., once every minute. This will continue even after the session was terminated – until the user deletes the dead session's temporary library.

It is important to try to strike a balance between bugging the users with too many messages and providing users with too few warnings or reminders. Too many messages pose the risk of alienating the users and thus hurting future cooperation, while too few notifications increase the danger of missed communication and delayed response. Supplying helpful information in the messages will definitely improve user responsiveness. Finally, you need to determine how frequently messages should be sent out after hours and during the weekends and holidays.

IMPLEMENTATION

The Unix operating system provides many built-in commands and other utilities that can help with our real-time monitoring system. To learn about them, you can pick up an introduction to the Unix system, use the **man** utility at the command line, or search on the Internet. You can certainly ask your Unix administrators for help. In order to kill another user's SAS process and delete its temporary library, you will need certain administrator privileges, such as using the **sudo** command. Otherwise, the success of your monitoring system depends entirely on timely, human / manual reaction to the notifications.

The sample real-time monitoring system we will be building shortly assumes that your company does not have 24/7 server support, even though you may be able to intervene after hours if the SAS server administrator has remote access to the machine. The emphasis here is automated monitoring and corrective actions. With careful planning and extensive consultation with SAS users, the real-time SAS server monitor should behave just like a dedicated and experienced professional server administrator. It should do everything that a human administrator will do in the same situation, except more faithfully and precisely and, above all, at no cost.

Because a SAS macro does not allow inline data in a DATA step, we will build a user email list outside the macro first, like this:

```
libname monitor "/my/monitor/directory";

DATA monitor.emailaddresses;
   infile datalines truncover;
   input username : $8. emailaddress : $50.;

   datalines;
   sandlera        Adam.Sandler@movies.com
   stillerb        Ben.Stiller@movies.com
   kidmann         Nicole.Kidman@movies.com
   ;
RUN;
```

Whenever new users are added to the server, the email list needs to be updated so that notifications can always reach their proper destinations. The monitor should be designed in such a way that neither terminating it nor restarting it at any time creates negative side effects.

The macro is defined with only one parameter, the target partition name. Obviously, you can add as many parameters as you need.

```
%macro server_monitor (partition= );

%do %while (1);

   <action statements>

   %sysexec sleep 60;

%end;

%mend;
```

The numeric value `1` inside the parentheses is considered a true condition, so the macro loop will run forever. As a real-time SAS server monitor, there is no reason to reset this condition. If we ever need to terminate the monitor, we can do it manually from the command line.

Also note the sleep time of 60 seconds. This dictates that the monitor will wake up once every minute, perform the monitoring duties, and go back to sleep for another minute, and the cycle continues endlessly. To change the interval, just replace it with another time value, for example, 300 for five minutes.

The `<action statements>` represent all the activities that the monitor conducts when it is awake. The very first activity is checking the target partition's disk space usage, as follows:

```
filename target pipe "df -lk | grep &partition";

DATA _null_;
   infile target pad;
   input line $120.;

   tempspaceused = ceil(scan(line, 3, " ") / 1048576);
   call symput("tempspaceused", tempspaceused);
RUN;
```

With the fileref `target`, the pipe device is used when the DATA step is executed to retrieve the line that contains the partition usage information from the Unix command **`df`**, with the resultant number in unit of kilobytes (KB). For example, 104857600 will mean 104857600KB, or 100GB. The division converts the number to gigabytes from kilobytes. The disk space currently in use (in GB) is then written to a macro variable for further processing. Please note that your particular Unix version may not show the available disk space in the third position of the line even with the same command, so you may need to test and confirm this, and make adjustments if necessary.

Once we know how much disk space is currently being used, we can use a series of `%if – %else` conditions to determine what to do next, based on our predefined thresholds:

```
%if &tempspaceused >= 190 %then %do;
   <identify the process with the largest temporary library>
   <terminate the process and clean up the session>
   <notify the user and administrator about the termination>
%end;
%else %if &tempspaceused >= 180 %then %do;
   <identify the three processes with the largest temporary libraries>
   <notify the users and administrator about the space issue>
%end;
%else %if &tempspaceused >= 150 %then %do;
   <notify the administrator about the potential space issue>
%end;
```

Since the first condition includes the most processing activities, we will examine its implementation in more detail below. Instead of identifying the top process, we will actually identify the top three processes. Obviously, you may want to perform certain common functions prior to entering the conditional checks.

To identify the processes with the biggest temporary libraries, we again use the pipe device with a fileref. We also use the `%sysexec` macro statement to execute operating system commands. They can often be used interchangeably for the same result, although the syntax and operations are different. Please note that you need to substitute the `<machinename>` with your real SAS server machine name (without the angle brackets). As mentioned earlier, you need to test the commands with your particular version of the Unix operating system to make sure that they generate similar results. Usually, some minor adjustments are all you need.

```
filename userwork pipe "ls -l /&partition | grep _<machinename> |
                        awk '{print $9 $3}'";

%sysexec %str(cd /&partition;
              ls | grep _<machinename> | xargs du -ks
                 > /my/monitor/directory/sessionsizes.txt);
```

Because the Unix directory names for a SAS session's work library and utility files contain the SAS process id (in hexadecimal format) as well as the server machine name, as shown above, we can use them to link the work library to the possible utility files directory and add up the space. The total amount is the space used by the SAS session. The utility files directory sometimes does not exist. The following DATA step processes all users' work libraries and utility files directories:

```
DATA usersessions (drop=line);
   length username $ 8 worksession $ 30 pid $ 4 type $ 1;
   infile userwork pad;
   input line $38.;

   username = substr(line, 31);
   worksession = substr(line, 1, 30);
   pid = substr(worksession, 17, 4);

   if substr(worksession, 5, 4) = 'work' then
      type = "w";
   else
      type = "u";
RUN;
```

Please note that the length of `line` is specific to your site and probably needs to be adjusted, along with the defined lengths of the variables and `substr` function starting positions. A type of "w" indicates a work library, while a type of "u" means a utility files directory.

The follow DATA step collects the disk space occupied by each directory (either a work library or a utility files directory) in the temporary partition. Unlike the DATA step above, you do not need to change anything here for your particular Unix operating system, except maybe the `byte(9)` part. In Solaris 9, a horizontal tab separates the directory / file name and its total disk space size when using the **du** utility.

```
DATA sizesessions (drop=line);
   length size 8 worksession $ 30;
   infile "/my/monitor/directory/sessionsizes.txt" pad;
   input line $60.;

   size = scan(line, 1, byte(9));
   worksession = scan(line, 2, byte(9));
RUN;
```

The two data sets are then merged to add the size information to each work library and utility files directory, which also has a process id. Each process's disk space is then summarized and ranked, and the top three processes (with the largest temporary disk space usage) are linked to the users' email addresses.

```
DATA usersizesessions;
   merge usersessions sizesessions;
   by worksession;
RUN;
```

```
PROC SQL noprint;

   create table sessiontotalsize as
   select pid, sum(size) as totalsize
   from usersizesessions
   group by pid
   order by totalsize desc;


   create table usersizesessionsfinal as
   select a.*, b.totalsize
   from usersizesessions a, sessiontotalsize (obs=3) b
   where a.pid = b.pid;


   create table usersizesessionsfinalwithemail as
   select a.*, b.emailaddress
   from usersizesessionsfinal a, monitor.emailaddresses b
   where a.username = b.username
   order by totalsize desc, pid, type;

QUIT;
```

The following DATA step creates a string to hold both the work library and the utility files directory so that a command to delete them can be later constructed for the users. The process ids are also converted into decimal format, again for the purposes of easily identifying and killing them.

```
DATA usersizesessionsfinalwithemail (drop=worksession size type);
   length bothdirectories $ 61;
   retain bothdirectories;
   set usersizesessionsfinalwithemail;
   by descending totalsize pid;

   decimalpid = input(pid, hex4.);

   if first.pid then do;
      if type = 'u' then
         bothdirectories = worksession;
      else
         bothdirectories = " ";
   end;

   if last.pid then do;
      bothdirectories = worksession || " " || compress(bothdirectories);
      output;
   end;
RUN;
```

Finally, the most important information about the three top processes is put into macro variables:

```
PROC SQL noprint;

   select trim(translate(scan(emailaddress, 1, "@"), " ", ".")),
          ceil(totalsize / 1048576),
          bothdirectories,
          emailaddress,
          decimalpid
      into : fullname1 - : fullname3,
           : totalsize1 - : totalsize3,
           : library1 - : library3,
           : emailaddress1 - : emailaddress3,
           : pid1 - : pid3
   from usersizesessionsfinalwithemail;

QUIT;
```

Now that we know what the top three processes are, terminating the top one is easy, so is cleaning up its session. Here are the commands:

```
%sysexec %str(/usr/local/bin/sudo -u &username1 kill -9 &pid1;
              cd /&partition;
              /usr/local/bin/sudo -u &username1 rm -r &library1);
```

We do not have to deal with number two and number three sessions in the same way because they can actually be quite small and harmless. If one of them does grow to more than 100GB in size, or when the total used disk space tops 190GB and one of them happens to be the top session, then the same fate is awaiting it. You will need to determine if it is necessary to automatically remove the top session's temporary space when it reaches 100GB but there is still ample disk space for other users. It is usually a good idea to give the offending user a second chance whenever possible. Killing his or her process is drastic and severe enough.

The last step is sending out notifications by email. Here is an example of notifying the top three users and the administrator:

```
filename notify email to="&emailaddress1 &emailaddress2 &emailaddress3"
                      cc="admin@movies.com"
                      subject="Your SAS session(s) may soon fail due to low disk
                               space on /&partition";

DATA _null_;
   file notify;

   put "Dear SAS users:" //
       "The available space on /&partition is %eval(200 - &tempspaceused)GB." /
       "There is a high probability that the disk will be full very soon, in" /
       "which case all SAS sessions with work libraries on the partition will" /
       "fail, yours included." //
       "You are one of the users with the largest work libraries on /&partition:"//
       "User Name          Size   SAS Work Library and Utility File Directory" //
       "&fullname1" @21 "&totalsize1.GB" @28 "&library1" /
       "&fullname2" @21 "&totalsize2.GB" @28 "&library2" /
       "&fullname3" @21 "&totalsize3.GB" @28 "&library3" //
       "You should either terminate your own SAS session or make sure that the" /
       "other two sessions are terminated by their respective owner(s) – if you" /
       "feel your session must go on." //
       "To kill your SAS session and thoroughly clean up the work library, you" /
       "can open a Telnet session to the SAS server and copy the appropriate" /
       "commands below to the Unix command prompt. You can either copy the" /
       "entire line below and hit enter to execute all four commands together," /
       "or copy and execute the four commands one at a time. In the latter case," /
       "you do not need to copy the semicolon following each command." //
       "For &fullname1:" /
       "kill -9 &pid1; cd /&partition; rm -r &library1; cd" //
       "For &fullname2:" /
       "kill -9 &pid2; cd /&partition; rm -r &library2; cd" //
       "For &fullname3:" /
       "kill -9 &pid3; cd /&partition; rm -r &library3; cd" //
       "If no corrective action is taken by the top space user (&fullname1)" /
       "within five minutes, his or her SAS session will be terminated, and" /
       "the space will be reclaimed." //
       "Thank you for your prompt attention to this urgent matter!" //
       "The SAS Server Monitor";
RUN;
```

Again, you may need to modify the positions of certain variables when creating the contents of the notifications.

DISCUSSIONS

While all the necessary elements are in place, putting them together can be sometimes challenging. For those users who have not used the SAS capability to execute operating system commands from within the SAS session, or who are not particularly familiar with the listed Unix commands and utilities, building an automated real-time SAS server monitor can be a little (or a lot) intimidating. With patience and perseverance and, above all, diligent Googling and Unix manning, you will eventually get the job done. The Big Brother must prevail!

Once you have a prototype monitor in place, the possibilities suddenly become limitless. The above examples covered only the basic activities, with an emphasis on demonstrating the necessary components. You should fine-tune them to suit your special needs. You can monitor the CPUs, the memory usage, and the I/O system in addition to the disk space, either separately or within one monitoring program. In fact, you can watch over all your important disk partitions and file systems, not just the temporary partition for work libraries. In addition to email notifications, you may also explore pager and cell phone notifications where applicable. If you want a daily or weekly report on resources statistics, you can collect them at the regular intervals, store them in a permanent location, and summarize them daily or weekly before sending them out. For the artistic types, you can even add the SAS ODS into the mix and generate truly impressive graphics for those reports, or alternately send them to a Web server for easier and broader access.

Some people may be concerned about the possibility of an unexpected server shutdown. In its current implementation, this real-time monitor needs to be manually started or restarted. However, there is a very simple solution. You can create a small SAS program (let's call it The Little Brother) whose sole function is checking to see if The Big Brother is running, and launching the real-time monitor if necessary. By scheduling The Little Brother to run once every hour or more frequently via the Unix cron facility, the real-time monitor is guaranteed to be running within an hour of the SAS server coming back to life. The Big Brother likes to watch over the SAS server and will be watching constantly.

For all the round-the-clock monitoring and reporting activities, the system load from the monitoring program is minimal. Our regular monitor currently wakes up once every minute to watch over the four CPUs, the memory, and disk space usage on six important disk partitions. It also generates two daily statistics and status reports besides checking for any new users and dead SAS sessions. The total CPU time as consumed by the monitor is about two minutes on a single CPU per day, which is truly negligible. The rewards, however, are tremendous. Ever since the monitor was put in place more than one and a half years ago, we have never had a CPU overload, our memory has been sufficient, and we have had no disk space full incidents as long as the monitor is running. Together with The Little Brother, we have been able to pinpoint when our SAS server was brought down unexpectedly and when it was restarted, without having to contact the Unix administrators. There are no longer dead SAS sessions lying around on our SAS server, even though we have about three dozen users of various proficiencies accessing the SAS server via different channels. Above all, the hard-working, mostly-sleeping, and never-complaining server monitor has freed us from the tedious daily management chores of the Unix machine. Our managers are happy too because the accumulated daily reports clearly show the trend of resources utilization and point to potential new bottlenecks. It even shows how much CPU time each user has incurred during the past 24 hours. OK, maybe the last part is not such a good thing!

Since the SAS software works on most operating platforms, it is equally possible to build a SAS server monitor for your particular platform, such as Windows or the mainframe, if you so desire. Whenever possible, you should try to incorporate the operating system utilities that perform relevant functions into your monitoring and reporting system.

CONCLUSION

Every company needs to have some kind of monitoring mechanism to prevent system overload as much as possible, or the results can be disastrous. Rather than spending a lot of money on professional administration and commercial tools, you can create your own customized solution by utilizing Base SAS DATA step, SAS macro, and operating system commands. You will benefit from zero cost, effective resources management, extreme flexibility, and a better understanding of the server system. More importantly, you get to play The Big Brother like in George Orwell's novel, except everyone truly loves you. Really!

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Houliang Li
2556 Carrington Way
Frederick, MD 21702
(301) 682-6832
houliang_li@yahoo.com