Paper 219-30

Programmatically Measure SAS® Application Performance On Any Computer Platform With the New LOGPARSE SAS Macro

Michael A. Raithel, Westat, Rockville MD

Abstract

For years, SAS programmers have manually combed through their SAS logs to gather the performance metrics of their SAS applications. They take pen in hand and dutifully add up the resource usage of the various DATA steps and PROC steps of their SAS programs. Sometimes they enter the performance data into a SAS data set or into a spreadsheet so that it can be further analyzed. This process is cumbersome and time consuming. The new, experimental LOGPARSE SAS Macro eliminates the drudgery of obtaining SAS program performance metrics by hand.

The LOGPARSE Macro parses SAS log files and stores SAS performance metrics in a SAS data set. That SAS data set can be used to programmatically report the performance of individual steps within a SAS program, report the performance of the entire SAS program, or report the performance of individual programs within a SAS application. The LOGPARSE Macro can be used to record SAS performance information on any platform that SAS runs on.

This paper introduces the LOGPARSE SAS Macro and shows how to use it. It provides examples of how to use LOGPARSE "out of the box" and how to configure it to fit your needs. This paper presents samples of SAS resource usage reports based on data obtained via the LOGPARSE Macro. After reading this paper, you will be able to download the LOGPARSE SAS Macro from the SAS Institute web site, configure it to your needs, and begin automatically recording and reporting on your SAS application's computer resource usage.

Introduction

The LOGPARSE SAS Macro is experimental software that recently became available. Though someday it may be added to the SAS Sample Library, today you can only obtain a copy of it by downloading it from the SAS Institute's Support web site. You can do that by performing the following steps:

1. Access the following URL: http://support.sas.com/rnd/scalability/tools/fullstim/fullstim.html .
2. In the first paragraph of text (in the middle column), click on *download this ZIP file* to download the logparse.zip ZIP file to your desktop.
3. Expand the *logparse.zip* ZIP file to extract its contents. That will create a folder containing three SAS Macro programs and a *readme.txt* file.

Where you go from here depends upon the operating system you are going to use the LOGPARSE SAS Macro on. If you are in a Windows environment, then you simply need to move the programs to a Windows directory. However, if you are going to use LOGPARSE SAS Macro on UNIX, Linux, VMS, z/OS or other platforms, then you will have to FTP or otherwise transfer the three SAS Macro programs to that other platform. After that, you can tailor your applications to make use of the LOGPARSE programs and start recording and reporting on computer resource usage.

The three SAS Macro programs that are used to realize the benefits of the LOGPARSE SAS Macro are:

- passinfo.sas
- mvsname.sas
- logparse.sas

Each of these SAS Macro programs is discussed in the following sections of this paper. After that, this paper provides an execution strategy for using the LOGPARSE SAS Macro, discusses the LOGPARSE SAS data set, and provides some examples of simple performance reports.

The examples in this paper illustrate the LOGPARSE SAS Macro implemented on a Linux server. However, it is a relatively simple task to configure it to run on other operating systems.

The PASSINFO SAS Macro

The PASSINFO SAS Macro does exactly what its name implies; it passes information on to the LOGPARSE SAS Macro. It does this by writing very special, stylized SAS session information notes in the SAS log of the programs in which it is invoked. When the LOGPARSE SAS Macro processes a SAS log, it looks for PASSINFO notes in the log. If it finds them, that information is included in the observations in the output SAS data set; if not, the LOGPARSE SAS Macro looks for other performance information.

Here is an example of the type of information that the PASSINFO SAS Macro writes to a SAS log:

```
PASS HEADER BEGIN
PASS HEADER os=LINUX
PASS HEADER os2=Linux
PASS HEADER host=sas2
PASS HEADER ver=9.01.01M3P072804
PASS HEADER date=24jan05:07:53:59
PASS HEADER parm=
MEMSIZE=536870912 Specifies the limit on the total amount of memory to be used by the SAS System
SUMSIZE=0         Upper limit for data-dependent memory usage during summarization
SORTSIZE=134217728 Size parameter for sort
PASS HEADER END
```

You can see that the information has more to do with the operating system, SAS version, and execution date, than it has to do with performance metrics. This macro is the conduit through which this type of information is passed to the LOGPARSE SAS Macro. Here is what the various PASS HEADER notes mean:

- **OS** – This is the abbreviated name of your operating system taken from the value of the *&sysscp* automatic SAS Macro variable.

- **OS2** – This is the long name of your operating system taken from the value of the *&sysscpl* automatic SAS Macro variable. Not all operating systems have a long name, so this may be blank.

- **HOST** – This is the name of your host system. The value is created in the PASSINFO SAS Macro and is derived in different ways for different operating systems.

- **VER** – This is the release number and maintenance level of your SAS software. The value is taken from the value of the &sysvlong SAS automatic variable.

- **DATE** – This is the date and time that your program's SAS session began.

- **PARM** – If it exists, this is the character string passed to SAS from your operating system via the &parm SAS Macro variable.

- **MEMSIZE**, **SUMSIZE**, and **SORTSIZE** are all derived by the LOGPARSE SAS Macro running the OPTIONS Procedure for those particular SAS system options. They are there for informational purposes only; the LOGPARSE SAS Macro does not collect and use them.

The PASSINFO SAS Macro should be executed at the beginning of each SAS program that you intend to collect performance information from. This allows the LOGPARSE SAS Macro to have early access to the information listed above so that it can be written to each observation that the LOGPARSE SAS Macro creates.

The MVSNAME SAS Macro

The MVSNAME SAS Macro is used to determine the name of the OS/390 or z/OS host operating system. This macro is invoked by the LOGPARSE SAS Macro ***only*** when it determines that the host system is an MVS platform. The MVSNAME SAS Macro does not have any parameters for you to specify, nor does it write anything new to the SAS log. It is simply a subroutine completely employed by PASSINFO under certain circumstances to derive the value of HOST, discussed above.

MVSNAME is an ingenious little macro that builds a temporary flat file:

```
filename inpds disk ".t&_tmpnum..clist" disp=(new,catlg)
               space=(trk,(1,1,1))
               lrecl=80 recfm=fb blksize=8000;
```

…and populates it with the REXX statement:

```
SAY 'SYSNAME=' MVSVAR('SYSNAME')
```

Then, it closes the temporary flat file and executes the flat file via a TSO command:

```
tso exec '%bquote(t&_tmpnum..clist(mvsnm))' nolist
```

The result of this command, the host system's name, is piped back into the MVSNAME SAS Macro via a FILENAME statement and stored in the SYSHOST Macro variable. The temporary flat file is deleted and MVSNAME is finished executing. PASSINFO then sets HOST (noted in the PASSINFO section, above) equal to the value of the SYSHOST Macro variable if it is executing on an MVS operating system.

The LOGPARSE SAS Macro

The LOGPARSE SAS Macro parses a SAS log and collects information placed there by PASSINFO and collects performance metrics information written to the log via the FULLSTIMER system option. (More information about the FULLSTIMER system option can be found in the next section, *The Execution Strategy*. It writes this information to observations and stores them in a temporary SAS data set or in a permanent SAS data set, depending upon your specification. The LOGPARSE SAS Macro creates one observation for each DATA step and PROC step that it finds in a SAS log.

This is the syntax for the LOGPARSE SAS Macro:

```
%logparse(saslog, outds, system, pdsloc, append=no)
```

The LOGPARSE parameters are:

SASLOG – This is the full-path name of the SAS log file that is to be processed. For MVS, see *PDSLOC*, below.

OUTDS - This is the name of the output SAS data set where the resulting observations will be stored. This parameter is optional and the following rules hold true:

- If it is not specified, WORK.DATA*n* is created, where *n* is the smallest integer that makes the name unique. This is the same behavior as in the simple SAS program: "data; x = 1; run;".
- If it is specified, OUTDS names the data set used to store the observations created by this invocation of LOGPARSE.
- If the output data set is specified and *append=yes* is also specified, observations from this invocation of LOGPARSE are appended to the already-existing SAS data set named by OUTDS. (Refer to the documentation of the *append=* parameter, below).

SYSTEM – This is the 3-character operating system code. This is another optional parameter. The default value is the sysem on which %logparse() is run. Valid codes are:

- **MVS** – z/OS, OS/390, or MVS. This causes the MVSNAME SAS Macro to be executed.
- **ALP** – OpenVMS Alpha
- **VMS** – OpenVMS VAX
- **OTH** – All other operating systems

PDSLOC – When LOGPARSE is executed on an MVS system and the SAS log file to be analyzed is stored in a Partitioned Data Set (PDS), PDSLOC partially names the PDS and SASLOG names the member. The PDS name is generated with a leading period (the system uses your *userid* for the first level of the name) and the last level is assumed to be *LOGS*.

For example consider a the following PDS member that contains a SAS log:

JPAGE.LEDZEP.LOGS(PROD456)

The following invocation of LOGPARSE would process the SAS log stored in the PROD456 member of the JPAGE.LEDZEP.LOGS PDS:

```
%logparse(prod456, , , ledzep.logs);
```

APPEND – This parameter is used to either create a new SAS data set to store the observations created by this execution of LOGPARSE, or to specify that they are to be appended to an existing SAS data set. This parameter acts in conjunction with the OUTDS parameter, discussed above. APPEND is an optional parameter, with a default value is NO. Valid values are:

- NO = LOGPARSE creates a new SAS file according to the rules for the OUTDS parameter, described above.

- YES = LOGPARSE appends its output to the file named by the OUTDS parameter. If the file does not already exist, then it is created.

Here is an example of specifying the LOGPARSE Macro for a SAS log created in the Linux environment:

```
%logparse(/home/production/Daily.log,logparse.perfprod,OTH,,append=YES);
```

In the example, the *Daily.log* SAS log file in the */home/production* directory is being processed. Observations created by LOGPARSE will be appended to the *logparse.perfprod* SAS data set, since *append=YES* has been specified. *Oth* has been specified for the system (in accordance with the rules for this parameter discussed above) because the SAS program is executed on a Linux server. Since this is not executing on an MVS system, the *pdsloc* parameter was not specified.

The Execution Strategy

Once you have successfully downloaded the logparse macros, you can put them to work collecting SAS application performance information. There are several steps to doing this. The first step is that you must decide how the logparse macros will end up in your SAS programs. There are several different ways that you can get them there:

- %INCLUDE – You can use the %INCLUDE statement to include each of the macros at specific points in your SAS programs.

- AUTOCALL Library – You can place the LOGPARSE, PASSINFO, and MVSNAME SAS Macros in an autocall macro library.

Either of the methods, above, will allow your application programs to have access to the logparse SAS Macros.

Here is the strategy for exploiting the logparse SAS Macros:

1. **Specify the FULLSTIMER option in an OPTIONS statement on the first line of your SAS application programs**. This will direct SAS to print full performance statistics for each DATA step and PROC step to the SAS log. Once these detailed performance metrics are in the SAS log, LOGPARSE can be used to collect them. Here is an example of specifying that option in a SAS program running in the Linux environment:

   ```
   options fullstimer;
   ```

   Exercise caution with this option by checking the *SAS Companion* for your environment to verify its proper use. The FULLSTIMER option cannot be used in the z/OS environment. Instead, you must specify the STIMER option in the system or user config file and then specify FULLSTATS on an OPTIONS statement. See the **References** section at the end of this paper for more information on STIMER, FULLSTATS, and other SAS options that surface processing statistics for SAS running in the z/OS environment.

2. **Invoke the PASSINFO SAS macro on the second line of your SAS application programs**. This may be done in one of two ways, depending upon whether or not you put the macro in an AUTOCALL Library.

   If you are using %INCLUDE to make this SAS Macro available:

```
%INCLUDE "/home/logparse/passinfo.sas";
%passinfo;
```

If you have the PASSINFO SAS Macro in an AUTOCALL library:

```
%passinfo;
```

3. **Run your SAS application programs and be sure to save the SAS log in a file**. The programs will write additional information to the SAS log from invocation of the PASSINFO SAS Macro and from invocation of the FULLSTIMER SAS System option.

4. **Collect application performance information by running the LOGPARSE SAS Macro**. This may be done in one of two ways, depending upon whether or not you put the macro in an AUTOCALL Library.

   If you are using %INCLUDE to make the LOGPARSE SAS Macro available:

```
%INCLUDE "/home/logparse/logparse.sas";
%logparse(/home/production/Daily.log,logparse.perfprod,OTH,,append=YES);
```

   If you have the LOGPARSE SAS Macros in an AUTOCALL library:

```
%logparse(/home/production/Daily.log,logparse.perfprod,OTH,,append=YES);
```

   More than likely, you will create a separate SAS program that collects the performance information from the various SAS programs that make up your application. You will schedule that program to run last in order to collect performance information from the logs of the previously run programs. For example, you might have a program named *collect_performance_data.sas* that has the following SAS code in it:

```
libname logparse "/home/logparsedata";

%INCLUDE "/home/logparse/logparse.sas";

%logparse(/home/production/Extract.log,logparse.perfprod,OTH,,append=YES);
%logparse(/home/production/Process.log,logparse.perfprod,OTH,,append=YES);
%logparse(/home/production/Create.log,logparse.perfprod,OTH,,append=YES);
```

   The program, above, begins with a LIBNAME statement that allocates a permanent SAS data library. That SAS data library contains the PERFPROD SAS data set wherein logparse information is stored for all of your SAS applications. The program next %INCLUDE's the LOGPARSE SAS Macro so that it may be invoked. Then, three separate SAS logs (Extract.log, Process.log, and Create.log) from the same directory are parsed. Observations created during the parsing of the SAS logs are appended to the LOGPARSE.PERFPROD SAS data set. Since this is not running in an MVS environment, the PDSLOC parameter was not specified.

Improving the Placement of the LOGPARSE Macro Invocation

The strategy outlined in the previous section is somewhat intrusive because it calls for a change to your application programs. You must ensure that the FULLSTIMER (or STIMER and FULLSTATS for z/OS) option is in effect at the beginning of each program, and also include and invoke the PASSINFO SAS Macro. So, at worst case, you would be adding the following statements to each of your SAS programs:

```
options fullstimer;
%INCLUDE ""/home/logparse/passinfo.sas";
%passinfo;
```

You may mitigate the intrusion somewhat by making a simple change to the PASSINFO SAS Macro. Simply edit the macro and add the following as the first line of the macro:

```
options fullstimer;
```

This will enable you to simply code the include and invocation of the PASSINFO SAS Macro at the beginning of each application program. Since FULLSTIMER would now be coded inside of PASSINFO, and PASSINFO is at the beginning of the SAS program, it will be in effect throughout the execution of the program. Now, you only have to put this at the top of each program:

```
%INCLUDE ""/home/logparse/passinfo.sas";
%passinfo;
```

If you make the FULLSTIMER change to PASSINFO, and include the PASSINFO SAS Macro in an AUTOCALL Macro library, you can reduce the lines that you have to code at the beginning of each SAS application program to this:

```
%passinfo;
```

There is a way to make specifying the FULLSTIMER option and PASSINFO SAS Macro more automatic and non-intrusive in your SAS application programs. Simply specify both of them in your SAS Application's AUTOEXEC.SAS file. This allows you to skip making changes to individual SAS programs, and guarantees that the FULLSTIMER option and the PASSINFO SAS Macro will always be in effect when the application programs are executed. When you add new programs to your application, they will automatically have performance information written to their logs as long as you use the same AUTOEXEC.SAS file. Consequently, all you have to do is to run the LOGPARSE SAS Macro against the SAS logs to collect application performance information.

The LOGPARSE SAS Data Set

A CONTENTS Procedure listing of the LOGPARSE SAS data set can be found in **Appendix A**. If you take a look at it, you will notice that there are performance metrics from a variety of host operating systems. There are performance metrics from VMS, UNIX, LINUX, Windows, OS/2, and z/OS. If you are running the logparse macros on any one of those operating systems, the variables that hold information for the other operating system metrics will contain missing values. So, there will be a good amount of wasted space in your LOGPARSE SAS data set.

This is not necessarily a bad thing. If you have a multi-platform SAS environment, then you can collect performance information on the various operating systems via the logparse macros and save them in a single LOGPARSE SAS data set on one of the systems. This would give you a distinct, integrated source of performance information for all of your SAS applications running on your disparate SAS servers. You could then create management or user reports of SAS application performance across your organization's entire computer complex from that single LOGPARSE SAS data set.

The reason that the LOGPARSE SAS data set contains metrics from all host systems that SAS can be run on is simple. The SAS Institute had to make a choice between offering a separate LOGPARSE SAS data set for each platform that it supports or offering a single data set that could be used among all of the platforms. It chose to do the latter for simplicity of upkeep and maintenance. This was an intelligent decision. However it may be a

bit wasteful on disk space and a bit confusing for programmers who are not fully cognizant of the performance metrics of their operating system.

You can address the issue of having "too many" variables in your LOGPARSE SAS data set in a variety of ways:

1. Simply ignore it.

2. Compress the LOGPARSE SAS data set and ignore the other host system variables. This would reduce the size of the LOGPARSE SAS data set by compressing out the space occupied by missing values.

3. Create a data view that includes only variables containing metrics for your operating system. You would use the data view when creating reports or accessing the data via the SAS display manager. If users wanted to create performance reports, you could give them the data view, so that they would not puzzle over foreign metrics with missing values.

4. Keep only the performance metrics for your distinct operating system. You could do this by employing the following method:

   - Run the LOGPARSE SAS Macro and store the data in a work SAS data set:

     ```
     %logparse(/home/production/Extract.log,perftemp,OTH,,append=NO);
     ```

   - Append the temporary LOGPARSE SAS data set to the permanent LOGPARSE SAS data set using the KEEP option on the DATA statement. Here is what it would look like for LINUX:

     ```
     proc append base=logparse.perfprod
                 data=perftemp(keep=logfile stepname portdate platform scp
                                    realtime usertime systime cputime pageflt
                                    pagercl pageswp osvconsw osiconsw
                                    blkinput bkoutput obsin obsout varsout
                                    datetime host memused stepcnt);
     run;
     ```

The method you choose to use will depend upon your unique situation and your tolerance for the additional variables in the LOGPARSE SAS data set. So, whatever you choose to do will be ultimately be the right choice.

Here are several SAS data views that keep only the performance metric variables found on some of the more popular operating systems. Note that only operating systems the author is familiar with are represented below. You will have to research the performance metrics on systems that are not included to create views for them.

```
data windows / view=windows;  /*Logparse view for Windows systems */
set  logparse.logparse(keep=logfile stepname portdate platform realtime
                           usertime systime cputime obsin obsout varsout
                           datetime host memused stepcnt);
run;


data zos / view=zos;  /*Logparse view for the z/OS and OS/390 systems */
set logparse.logparse(keep=logfile stepname portdate platform realtime
                             usertime systime cputime scp excpcnt totmemd
                             totmemp tskmemd tskmemp vatime vutime rsmtime
                             abovemem belowmem obsin obsout varsout datetime
```

```
                                host memused stepcnt);
run;


data linux / view =linux;  /*Logparse view for Linux systems */
set logparse.logparse(keep=logfile stepname portdate platform scp realtime
                           usertime systime cputime pageflt pagercl pageswp
                           osvconsw osiconsw blkinput bkoutput obsin obsout
                           varsout datetime host memused stepcnt);
run;


data unix / view =unix;  /*Logparse view for UNIX systems */
set logparse.logparse(keep=logfile stepname portdate platform scp realtime
                           usertime systime cputime pageflt pagercl pageswp
                           osvconsw osiconsw blkinput bkoutput obsin obsout
                           varsout datetime host memused stepcnt);
run;


data vms / view=vms;  /*Logparse view for the VMS system */
set logparse.logparse(keep=logfile stepname portdate platform scp
                           realtime usertime pageflt buffio dirio
                           obsin obsout varsout datetime host stepcnt);
run;
```

The LOGPARSE SAS data set contains forty-three variables. A number of them are not performance metric variables, and can be used to understand the nature of where the data came from and when it was collected. You will undoubtedly use some of these variables in your reporting. Here is a brief description of the non-performance metric variables:

- **DATETIME** – This is the date/time that the SAS session started for the program that executed. You can decompose this variable to create reports of the metrics of SAS application programs run by Date or by Date and Time.

- **HOST** – This is the name of the host server on which the SAS application program ran. For example, if you ran the logparse macros on the SAS2 Linux server, the value of HOST would be "SAS2". If you collect logparse data from several servers and store it in the same LOGPARSE SAS data set, you can use this variable to distinguish between them.

- **LOGFILE** – Contains the full path name of the log file that was parsed to collect the performance information stored in the observation. If the name of the log file contains the program name, then you can use SAS functions such as REVERSE, INDEX and SUBSTR to eke out the name of the SAS program that was executed. An example value is:

  ```
  /home/marsyst/benchmarks/programs/benchmr2.log
  ```

  In this example, the name of the program that created the log was benchmr2.sas. Knowing this, we can simply use SAS functions to grab "benchmr2.log" and change it to "benchmr2.sas" in our reports. See the example in *Creating Performance Reports* for one approach to doing this.

- **OBSIN** – This variable records the number of observations read into the DATA step or PROC step. This value can be checked to determine if the expected number of observations were really input to this particular SAS step. It can also be matched against OBSOUT to determine whether

the number of observations input and output match your expectations. It is a good measure of the volume of data that this SAS step had to process.

- **OBSOUT** – This provides the number of observations written to an output SAS data set during the execution of this PROC step or DATA step—if applicable. It is another measure that you can inspect to determine if the number of output observations match your expectations for your SAS application.

- **PLATFORM** – The PLATFORM variable reports the exact version of the operating platform that the DATA or PROC step executed on. For example, the author executed logparse macros on the following Linux platform: "Linux 2.4.21-27.0.2".

- **PORTDATE** – This metric details the exact version of SAS that was used in this SAS step. You can use this to determine when the version of SAS might have been changed, and how that may have affected the performance of this particular SAS step. An example of PORTDATE for SAS on a Linux server is: "9.01.01M3P072804".

- **SCP** – The SCP variable provides a generic name of the operating system that you are using. It is similar to the PLATFORM variable, but does not have the same long operating system version numbers. For example, it could have a value of "LINUX" or "UNIX".

- **STEPCNT –** This is a subtle, yet very important metric. STEPCNT specifies the order in which this particular DATA step or PROC step executed within the SAS program. You can use it to identify the performance of individual SAS steps by lining it up with the original SAS program (as long as the program has not changed). You can also use it to determine which DATA and PROC steps are consuming the *most* computer resources within a given SAS program. STEPCNT contains an integer value. So, STEPNCT is important because it allows you to zero in on the specific DATA step or PROC step that consumed the performance information stored in the observation.

- **STEPNAME –** This is the name of the SAS step that executed. Some values that you might find in your LOGPARSE SAS data set are:

    - **initialized** – This is the SAS initialization step that occurs at the beginning of each SAS program.
    - **DATA** – This is the value for each SAS DATA step.
    - **SORT** – This is the value for each execution of PROC SORT.
    - **DATASETS** – This value specifies that PROC DATASETS was executed.
    - **SUMMARY** – This is the value for the SUMMARY Procedure.

  Other values could be GRAPH, UNIVARIATE, and MEANS. This variable simply records the name of the SAS DATA or PROC step that was executed. So, STEPNAME helps you to determine exactly which type of SAS step consumed the computer resources reported in the observation. It is an invaluable variable in helping you to understand exactly what is going on—performance wise—within a SAS program.

Creating Performance Reports

Once you have collected logparse data from your SAS applications, you can create performance reports that characterize the performance of individual SAS programs and the performance of SAS applications. You can

do this for a given execution of the program or application, or look at the performance over time. Because the information is automatically collected, and because it is stored in SAS data sets, you can bring all of your SAS programming skills to bear on creating reports.

Here is an example of a SAS program that creates a detailed report of a particular SAS program and a summary report of all SAS programs that are run on a Linux server.

```
data linux / view =linux;  /*Logparse view for Linux systems */
set proddata.logparse(keep=logfile stepname portdate platform scp realtime
                           usertime systime cputime pageflt pagercl pageswp
                           osvconsw osiconsw
                           blkinput bkoutput obsin obsout varsout
                           datetime host memused stepcnt);

length SASprogram $40;
drop logtemp r;

  /*********************************/
  /* Determine the SAS program name */
  /*********************************/
logtemp = reverse(trim(left(logfile)));
r = index(logtemp,'/');
substr(logtemp,r,length(logtemp) - r + 1) = ' ';
SASprogram = translate(trim(left(reverse(logtemp))),"sas","log");

  /*********************************/
  /* Determine the system and OS names*/
  /*********************************/
if host ne " " then call symput('HOST',trim(left(host)));
if scp  ne " " then call symput('SCP',trim(left(scp)));

run;

  /************************************************/
  /* Create a report for an individual SAS program.*/
  /************************************************/
proc print noobs data=linux(where=(SASprogram="benchmr2.sas"));
      var SASprogram stepcnt stepname obsin obsout;
title1 "Audit Report for the benchmr2.sas SAS Program";
run;

  /************************************************/
  /* Create a report of all SAS programs.          */
  /************************************************/
proc summary nway data=linux;
      class SASprogram;
      var cputime usertime blkinput bkoutput obsin obsout;
      output out=summ1(drop=_type_) sum=;
run;

proc print noobs data=summ1;
      by SASprogram;
      id SASprogram;
title1 "Cumulative Performance Statistics";
title2 "For All SAS Programs run on the &HOST &SCP Server";
run;
```

Most of the example above is self explanatory. However, you can see that you must put some effort into manipulating the LOGFILE variable (in the data view step) if you want to get a concise, meaningful SAS

program name from it. Doing so makes your reports easier to read and more professional looking. Sample outputs from executing this program can be found in **Appendix B – Report Samples**.

Conclusions

The experimental LOGPARSE SAS Macro provides an automatic way to store SAS program performance information in SAS data sets. You must download the three logparse macros, install them on your operating system, and configure your SAS applications to be able to invoke them. This requires a minimal amount of planning and effort. The result is that you will have a rich collection of SAS performance data that you can use to track and report the performance of SAS programs and SAS applications. It is well worth the time and effort to take advantage of this great new SAS performance tool!

Disclaimer

The contents of this paper are the work of the author and do not necessarily represent the opinions, recommendations, or practices of Westat.

References

Readme.txt file found in the logparse.zip file on the SAS Institute Support web site:
http://support.sas.com/rnd/scalability/tools/fullstim/fullstim.html

Raithel, Michael A. 2003. *Tuning SAS® Applications in the OS/390 and z/OS Environments, Second Edition.* Cary, NC: SAS Institute Inc.

Acknowledgements

I would like to thank Bill Brideson, from the SAS Institute, for his help in understanding the LOGPARSE SAS Macro. Bill was kind enough to offer me insights into the use and utility of the macro, and to consider some of my suggestions for improving its usability.

Contact Information

Please feel free to contact me if you have any questions or comments about this paper. You can reach me at:

Michael A. Raithel
Westat
1650 Research Boulevard
Room RW4521
Rockville, Maryland 20850

michaelraithel@westat.com

Appendix A – PROC CONTENTS of the LOGPARSE SAS Data Set

The CONTENTS Procedure

Data Set Name	PARSLIB.LOGPARSE	Observations	16
Member Type	DATA	Variables	43
Engine	V9	Indexes	0
Created	Monday, January 24, 2005 08:00:18 AM	Observation Length	848
Last Modified	Monday, January 24, 2005 08:00:18 AM	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	LINUX_32, INTEL_ABI		
Encoding	latin1 Western (ISO)		

Engine/Host Dependent Information

Data Set Page Size	65536
Number of Data Set Pages	1
First Data Page	1
Max Obs per Page	77
Obs in First Data Page	16
Number of Data Set Repairs	0
File Name	/home/marsyst/logparse/logparse.sas7bdat
Release Created	9.0101M3
Host Created	Linux
Inode Number	795488
Access Permission	rw-rw-r--
Owner Name	marsyst
File Size (bytes)	73728

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Label
31	abovemem	Num	8		Total Memory - above line
30	belowmem	Num	8		Total Memory - below line
19	bkoutput	Num	8		Block Output Operations
18	blkinput	Num	8		Block Input Operations
20	buffio	Num	8		Buffered IO
17	consemap	Num	8		Contended Semaphores
9	cputime	Num	8	TIME12.3	CPU Time
40	datetime	Num	8	DATETIME.	Run Date/Time
21	dirio	Num	8		Direct IO
36	events	Num	8		Events
25	excpcnt	Num	8		I/O count
15	excsemap	Num	8		Exclusive Semaphores
41	host	Char	200		System Name
37	locks	Num	8		Locks
1	logfile	Char	200		Log file name
42	memused	Num	8		Memory Used
32	obsin	Num	8		Observations Read
33	obsout	Num	8		Observations Written
14	osiconsw	Num	8		Involuntary Context Switches
13	osvconsw	Num	8		Voluntary Context Switches
10	pageflt	Num	8		Page Faults
11	pagercl	Num	8		Page Reclaims
12	pageswp	Num	8		Page Swaps
4	platform	Char	50		Platform
38	pools	Num	8		Memory Pools Created
39	poolsx	Num	8		Memory Pools Destroyed
3	portdate	Char	25		SAS Port Date
6	realtime	Num	8	TIME12.3	Elapsed Time
24	rsmtime	Num	8		RSM Hiperspace
5	scp	Char	50		Operating System
16	shrsemap	Num	8		Shared Semaphores
43	stepcnt	Num	8		Step Number
2	stepname	Char	20		Step Name
8	systime	Num	8	TIME12.3	System Time
35	threads	Num	8		Threads
28	totmemd	Num	8		Total Memory - data
29	totmemp	Num	8		Total Memory - program
26	tskmemd	Num	8		Task Memory - data
27	tskmemp	Num	8		Task Memory - program
7	usertime	Num	8	TIME12.3	User Time

34	varsout	Num	8	Variables Written
22	vatime	Num	8	Vector Affinity
23	vutime	Num	8	Vector Usage

Appendix B – Sample Reports

Audit Report for the benchmr2.sas SAS Program

SASprogram	stepcnt	stepname	obsin	obsout
benchmr2.sas	0	initialization	.	.
benchmr2.sas	1	DATA	984590	5977270
benchmr2.sas	2	OPTIONS	.	.
benchmr2.sas	3	OPTIONS	.	.
benchmr2.sas	4	OPTIONS	.	.
benchmr2.sas	5	DATA	5977270	5977270
benchmr2.sas	6	SORT	5977270	5977270
benchmr2.sas	7	DATA	435389	435389
benchmr2.sas	8	SORT	435389	435389
benchmr2.sas	9	DATA	6412659	3048774
benchmr2.sas	10	SORT	3048774	3048774
benchmr2.sas	11	DATA	3048774	3048774
benchmr2.sas	12	SORT	1261174	1261174
benchmr2.sas	13	DATA	1261174	483833
benchmr2.sas	14	DATASETS	.	.

Cumulative Performance Statistics
For All SAS Programs run on the SAS2 LINUX Server

SASprogram	_FREQ_	cputime	usertime	blkinput	bkoutput	obsin	obsout
My_Production_Program.sas	49	0:00:04.210	0:00:00.170	1923	5438	123049	239876
benchmr2.sas	32	0:01:23.040	0:52:21.390	29876	86768	54844968	46562516