Paper RIV-02

Using Linux Shell Commands, vi Editor, and Base SAS® to Parse through Log Files and Gather Log-information

Fuad J. Foty, U.S. Census Bureau, Suitland, MD

ABSTRACT

Reading log files can be fun but not when one has to examine several dozens or sometimes several thousands of those files and calculate real time and cpu time to see if the process is running according to plan. There are many ways to do that, the manual way or some smart automated way. By using the Linux shell commands such as tail, grep and cut, the vi editor, and Base SAS®, one can parse through log files and gather log-information such as cpu time, real time, and other important information that one needs to make decisions.

INTRODUCTION

The purpose of this paper is to produce a useful report generated quickly by executing a korn shell script supplied by several positional parameters on the Linux command line where the script is calling a SAS program which gathers information from SAS log files that are successfully completed.

Unlike other processers that gather performance metrics during the execution of a SAS program, this processor does not require the user to supply any special options or parameters during the execution of a SAS run, rather, it parses through previously executed SAS run logs and generates a quick report on the performance of the SAS run in comparison to other SAS runs in a similar project or program.

By examining the SAS logs, one can learn a lot about the SAS jobs. From CPU and REAL time measures, to what procedures were used and at what frequency, a programmer/supervisor/administrator can use the information to plan and predict what will happen with similar runs on different input SAS data sets with various observation values.

The processor works best when log files are located conveniently in a separate subdirectory and the log filenames have a standard **prefix**. The standard prefix is an important aspect of the system because one can use the processor to generate a report of a subset of those SAS log files in a previous report. There is no limitation on the number of SAS log files the processor can process. I've tested the system on **6443 log files** and it produced a nice report in less that 3 minutes. The report can easily be cut and pasted into an email message to share with co-workers and/or supervisors/managers. Throughout this paper, I will be using a korn shell script example running on a Red Hat Enterprise Linux (RHEL) operating system.

ASSUMPTIONS AND PREREQUISITES

To understand the SAS code and the Korn shell script, one needs to have a basic knowledge of the Korn Shell scripting and the Linux/Unix operation system commands and procedures. Knowing the **vi editor** and its functionality and how it can be used in an automated fashion would also be helpful. In addition to that, one needs to know the functionality and meaning of the SAS "**x command**" and the pipe channel of communication and how to use it in a SAS **"_null_"** procedure.

To produce clean output reports, the user needs to have all SAS **".log"** files located in one subdirectory. All of those files must be SAS log files and contain no errors in them. Those SAS jobs are basically done and ready to be processed through the parser program which expects a standard filename prefix for each log file. Let us start by looking at the korn shell script first.

THE KORN SHELL SCRIPT

To run the parsing operation on a Linux machine, I recommend using a **korn shell script**. Lets take a look at the following script:

Korn Shell Script: parse_logs.ksh

```
#!/bin/ksh
export SASEXE=/usr/local/bin/sas9
export SYSTIME=`date +%Y%m%d%H%M%S`
export opt=$1
export Log_Dir=$2
export pf=$3

# ------------------------------------------------------#
#  START PROGRAM EXECUTION
# ------------------------------------------------------#
echo " "
echo "**************************************************"
echo "Running parse_logs.ksh Processor...   "
echo "**************************************************"
echo " "

ProgDir=/sampwgt/yyyy/dev/foty0001/
ProgName=parse_logs
NewLogName="${ProgName}_$3_$SYSTIME"

cd $Log_Dir
echo "==>Parsing through $Log_Dir/$3*.log Please wait..."

$SASEXE $ProgDir/$ProgName.sas \
        -work /temp1/sastmp \
        -print ~/$NewLogName.lst

echo "==>Parsing Finished..."
echo " "
more ~/$NewLogName.lst
```

The above korn shell script does the invocation of the SAS program which performs the parsing operation on the SAS log files. It expects **three** positional parameters on the Linux/Unix command line. The first positional parameter (**$1**) is the script's options. There are currently three options to choose from. The following table shows a list of those options:

Option	Description
-t	Prints a list of CPU Versus REAL Time Comparison and their associated cumulative totals of all SAS log files with certain filename prefix
-p	Prints a tally list of SAS procedures such as the sum of all proc SORT, proc FORMAT, proc SUMMARY, proc TRANSPOSE, and proc DELETE within all SAS log files with certain filename prefix
-o	Prints a tally list of total observations read or processed of all SAS log files with certain filename prefix

The above options may be strung together as desired. For Example, "**-tpo**" produces all three reports and the "**-t**" option only produces a report of CPU time versus REAL time comparison alongside their associated cumulative totals.

The second positional parameter (**$2**) is an actual physical subdirectory where the log files are located. The third positional parameter (**$3**) is the SAS log filename prefix. The report(s) will be generated on those SAS log files with the specified filename prefix. Let's take a closer look at the following invocation of the SAS logs parser supplied at the Linux dollar (**$**) prompt:

```
Invoking ``parse__logs.ksh'' script COMMAND:

$ parse_logs.ksh -tpo /sales/logs/ sales_
```

The "**parse_logs.ksh**" is the korn shell script which calls the SAS program that does the parsing operation on the SAS log files in the "**/sales/logs/**"subdirectory with the "sales_" filename prefix. For example, if there are "**sales_01.log**" through "**sales_15.log**" log filenames located physically in the "/sales/logs" subdirectory, then the processor generates **fifteen rows** of information in each of the **three** reports specified by options "**-tpo**". If the second positional parameter ($2) has "**sales_1**" instead of "sales_", then it will generate reports on just "**sales_10.log**" through "**sales_15.log**" instead of the fifteen log files specified earlier.

RUNNING THE KORN SHELL SCRIPT

By executing the above korn shell script command, several echoed messages appear on the standard output screen. The message below gets displayed first followed by the reports specified by the options list.

Here is the screenshot of the first message that users will see:

```
************************************************
Running parse_logs.ksh Processor...
************************************************

==>Parsing through /sales/logs/sales_*.log Please wait...
==>Parsing Finished...
```

The "**Please wait…**" message line will be displayed first until the message line "**Parsing Finished…**" gets printed followed by the actual reports specified in the options list. The reports get displayed on the standard output screen and an actual physical output listing filename gets created and placed in the user's home directory with a nice time stamp (Example: **parse_logs_sales_20100608104907.lst**). If the user issues the command twice, then there will be two listing output filenames with different timestamps. This is done intentionally and conveniently to avoid overwriting similar reports with the same filename prefix. The processor uses the "**more**" command from Linux/Unix to display the content of the **".lst**" listing filename for easy scrolling with the spacebar.

THE OUTPUT REPORTS

After executing the **``parse_logs.ksh -tpo /sales/logs/ sales_''** shell script at the Linux/Unix command prompt, and assuming that there are fifteen SAS log filenames "**sales_01.log**" through "**sales_15.log**" in the **"/sales/logs/"** SAS log subdirectory with options "-**tpo**" and "**sales**_" filename prefix, then the processor will generate three reports. The **"-t"** option outputs a display of CPU and REAL time comparisons, while options **"p"** and **"o"** output displays of reports #2 and #3 with various SAS procedures tallies and observation counts within each SAS run.

Report #2, display a tally of certain SAS procedures counts used in the SAS log, while Report #3 displays the total observation counts. Let us take a closer look at those reports. First, we will look at Report #1.

REPORT #1:

***** SAS Logs Directory **/sales/logs/** *****

***** **CPU** Versus **REAL** Time Comparison With Log Files Group **sales_*.log** *****

Obs	Log_Name	real_cum	real_time_n	cpu_cum	cpu_time_n
1	sales_01	00:00:05:04	00:00:05:04	00:00:03:44	00:00:03:44
2	sales_02	00:00:08:38	00:00:03:34	00:00:05:49	00:00:02:06
3	sales_03	00:00:11:25	00:00:02:48	00:00:08:07	00:00:02:17
4	sales_04	00:00:11:41	00:00:00:16	00:00:08:11	00:00:00:05
5	sales_05	00:00:14:55	00:00:03:14	00:00:10:30	00:00:02:19
6	sales_06	00:00:18:53	00:00:03:57	00:00:11:12	00:00:00:41
7	sales_07	00:00:36:33	00:00:17:41	00:00:14:05	00:00:02:54
8	sales_08	00:00:42:31	00:00:05:58	00:00:17:06	00:00:03:01
9	sales_09	00:00:46:50	00:00:04:19	00:00:20:51	00:00:03:45
10	sales_10	00:00:49:02	00:00:02:12	00:00:22:47	00:00:01:57
11	sales_11	00:01:17:56	00:00:28:54	00:00:37:20	00:00:14:33
12	sales_12	00:01:56:28	00:00:38:32	00:00:45:05	00:00:07:45
13	sales_13	00:02:08:60	00:00:12:32	00:00:51:50	00:00:06:45
14	sales_14	00:02:18:32	00:00:09:32	00:00:58:32	00:00:06:42
15	sales_15	00:02:31:43	00:00:13:11	00:01:05:22	00:00:06:50

As you can see, **REPORT #1** displayed all **fifteen** SAS log filenames with their corresponding real time (labeled real_time_n) and cpu time (labeled cpu_time_n). In addition to those two columns, the processor also calculated and displayed a running cumulative total of the REAL time (labeled real_cum) and CPU time (labeled cpu_cum) as well.

REPORT #2 on the other hand, displays a tally list of SAS procedures used in the SAS logs. It tallies the following "PROC" procedures:

- "PROC SORT"
- "PROC FORMAT"
- "PROC SUMMARY"
- "PROC TRANSPOSE"
- "PROC DELETE"

This list can be expanded to include other "PROCS". On the next page, you will see a display of REPORT #2 which shows the "PROC SORT" as the most utilized "PROC" being used in the fifteen SAS logs, while "PROC SUMMARY", "PROC TRANSPOSE" and "PROC DELETE" were not used at all:

REPORT #2: ***** Log Files Group **sales_*.log** Procedures Tally *****

Obs	Log_Name	SORT_CNT	FORMAT_ CNT	SUMMARY_ CNT	TRANSPOSE_ CNT	DELETE_ CNT
1	sales_01	12	0	0	0	0
2	sales_02	8	1	1	1	0
3	sales_03	19	0	0	0	0
4	sales_04	18	0	0	0	0
5	sales_05	8	0	0	0	0
6	sales_06	5	0	0	0	0
7	sales_07	6	0	0	0	0
8	sales_08	3	0	0	0	0
9	sales_09	5	0	0	0	0
10	sales_10	4	0	0	0	0
11	sales_11	16	1	0	0	0
12	sales_12	5	0	0	0	0
13	sales_13	3	0	0	0	0
14	sales_14	3	0	0	0	0
15	sales_14	3	0	0	0	0

REPORT #3 displays a list of the total observations read (labeled **obs_sum**) either on input or output during the execution of each SAS program which generated the SAS log files. The obs_sum is a running total of ALL observations used on input and output SAS data sets. This number is especially useful when investigating the total CPU and/or REAL time it took to execute certain SAS jobs. The following table shows what the report looks like:

REPORT #3: ***** Log Files Group **sales_*.log** Summary of Obs Read Tally *****

Obs	Log_Name	obs_sum
1	sales_01	221589320
2	sales_02	41110515
3	sales_03	147492490
4	sales_04	3635145
5	sales_05	89526342
6	sales_06	29314992
7	sales_07	88954716
8	sales_08	73918136
9	sales_09	44401532
10	sales_10	160384599
11	sales_11	308848987
12	sales_12	135468793
13	sales_13	91064221
14	sales_14	90399198
15	sales_15	90434902

The above three reports may be customized to include other features that are important to the user's organization. This can be done by taking the attached SAS code located in Appendix-A and "playing" around with it to generate those special features. The current features in the three reports above are sufficient enough to my current work here at the U.S. Census Bureau. Users are more than welcome to do that as long as they give reference/credit to this paper.

THE SAS PROGRAM THAT GENERATED THOSE REPORTS

I will be dissecting the SAS program that generated the above three reports into the following **five** parts. Here is a list of those parts:

PART-1 - Determine the options from the command line
PART-2 - The find-xargs-tail-sed-tr combination in a data _null_ step
PART-3 - Applying "%s" command in the vi editor
PART-4 - Read, Tally and filter the parse files
PART-5 - Print the reports

The next five sections examine each part separately.

PART-1: DETERMINE THE OPTIONS FROM THE COMMAND LINE

Before parsing through any SAS logs, one must read the positional parameters supplied by the user on Linux/Unix command line. An easy way to remember those three options is as follows:

"t" is for time
"o" is for observations
"p" is for procedure

The above options are preceded by the minus sign (**"-"**) read from the command's first ($1) positional parameter. In the next table, it shows the SAS-code which reads in the minus symbol first and determines which of the three distinct options are present following the minus sign. The code then sets their corresponding flags (flg_t for "t" option, flg_p for the "p" option, and flg_o for the "o" option) to the value of one (1) when the option appears in the option list. The options may be in any order that the user prefers. Once the flags are set, then the parsing begins to output the specified report(s).

Reading the parse options:

```
%let opt = %sysget(opt);
%put opt= &opt;
%let minus = %qsubstr(&opt,1,1);
%let flg_t = 0; %let flg_p = 0; %let flg_o = 0;
%if "&minus" = "-" %then %do;
    %do i = 2 %to %length(&opt);
        %if %qsubstr(&opt,&i,1) = t %then %let flg_t=1;
        %if %qsubstr(&opt,&i,1) = p %then %let flg_p=1;
        %if %qsubstr(&opt,&i,1) = o %then %let flg_o=1;
    %end;
%end;
```

The code for PART-2 through PART-5 depends on whether the flags are set to "1" or not. It checks the flags before proceeding through the parsing operation. Let's examine now PART-2 which does the actual parsing.

PART-2: THE FIND-XARGS-TAIL-SED-TR COMBINATION IN A DATA "_NULL_" STEP
To understand PART-2, one must first understand the following sub-topics:

- The data "**_null_"** and the use of the "**x**" command.
- The Korn shell commands such as: **tail**, **sed**, **echo**, **tr –d**, **cat**, **rm**, **grep**, and how to further direct the output of those command to a file.
- **Vi editor** and its text substitution "%s" feature.

FIND-XARGS-TAIL-SED-TR COMBINATION:

```
x "find . -type f -name '&pf*.log' -print | xargs tail -5 |
         sed 's/\(NOTE]*\).*/\1/' |
         sed 's/[NOTE]*//g' |
         sed '/^$/ d' |
         tr -d '\f' > ~/_cpu.txt";
```

Let's take a look at the x command operation specified in the table above. The x command tells the SAS session that it wants to spawn out to the operating system and return later to the SAS session after it finishes the execution of the commands specified.

The command-set begins with the "**find**" command that tells the operating system to look for files that start with a prefix **(&pf)** and ends with a ".**log**" in the filename. The result of the find command is fed through the **xargs** to regulate the batches of how many files get processed at-a-time so that the error message "**Argument list too long**" does not appear when dealing with thousands of SAS log files.

The result of the **find-xargs** command combination is then handed over to the "**tail**" command. The "**tail -5**" takes the last five lines of the SAS log file. Those last five lines have the cpu and real time results of each SAS log.

The result of the "**tail -5**" command is piped through several "**sed**" commands. The "sed" is referred to as the **stream editor** which performs basic text transformations on an input stream (a file or input from a pipeline). While in some ways similar to an editor which permits scripted edits (such as "ed"), sed works by making only one pass over the input(s), and is consequently more efficient. But it is sed's ability to filter text in a pipeline which particularly distinguishes it from other types of editors. So it removes text that is not needed for the generation of the reports.

The "sed" command's output is then piped through the "**tr**" command. The "**tr**" command Translates, squeezes, and/or deletes characters from standard input or pipes, writing it to standard output. The "**tr –d**" in particular deletes characters that are not needed.

The entire piping operation gets directed to an output temporary file called **"_cpu.txt**". This is the file that the processor will be read to extract CPU and REAL time information.

We move next to PART-3 of the SAS code.

PART-3: APPLYING "%S" COMMAND IN THE VI EDITOR

The **VI editor** is a powerful tool to manipulate certain characters in the order that the user likes. The **"%s"** refers to substituting old text with new text throughout the file. Let's take a closer look at the following table:

VI Editor and the ``%s'' command:

```
x 'echo ":%s/.log <==/#/g" > ~/temp_cmd~';
x 'echo ":%s/real time//g" >> ~/temp_cmd~';
x 'echo ":%s/cpu time//g" >> ~/temp_cmd~';
x 'echo ":%s/ seconds//g" >> ~/temp_cmd~';
x 'echo ":%s/ //g" >> ~/temp_cmd~';
x 'echo ":%s/\n//g" >> ~/temp_cmd~';
x 'echo ":%s/^[0-9][0-9][.]/0:&/g" >> ~/temp_cmd~';
x 'echo ":%s/^[0-9][.]/0:0&/g" >> ~/temp_cmd~';
x 'echo ":%s/^[0-9][:][0-9][0-9][.]/0:0&/g" >> ~/temp_cmd~';
x 'echo ":%s/^[0-9][0-9][:][0-9][0-9][.]/0:&#/g" >> ~/temp_cmd~';
x 'echo ":%s/^$\n//g" >> ~/temp_cmd~';
x 'echo ":%s/\(133he]*\).*/\1/g" >> ~/temp_cmd~';
x 'echo ":%s/133he\n//g" >> ~/temp_cmd~';
x 'echo ":%s/[.0-9][0-9]$/&#/g" >> ~/temp_cmd~';
x 'echo ":wq!" >> ~/temp_cmd~';
x "vi ~/_cpu.txt < ~/temp_cmd~ > /dev/null 2> /dev/null";
```

The first line in the table has the following command:

(":%s/.log <==/#/g")

The "%s" tells the vi editor to substitute all occurrences of the string **".log <= ="** with the pound sign (#) throughout the entire file ("/g" for global). This substitution is important to extract a unique SAS log filename that the processor can use to match and sort on. The instructions that are built in the above code is directed to a command file (`~/temp_cmd~`) that we later use in the last line of the above table where we invoke the vi editor and supply it with that command file. The other substitutions (%s commands) also format the text output of the CPU and REAL time that are delimited by the pound sign (#) as indicated in the next table:

Content of _cpu.txt:

```
sales_01#0:05:03.61#0:03:43.59#
sales_02#0:03:33.93#0:02:05.97#
sales_03#0:02:47.68#0:02:17.16#
sales_04#0:00:15.94#0:00:04.74#
sales_05#0:03:14.47#0:02:19.07#
sales_06#0:03:57.18#0:00:41.31#
sales_07#0:17:40.80#0:02:53.50#

etc...
```

PART-4: READ, TALLY AND FILTER THE PARSED FILES

This step is done entirely in a couple of data steps. As you can see from the previous table, the filename and its corresponding CPU time and REAL time is conveniently delimited by the pound sign ("#"). This will help us read the table into a SAS data set where more parsing and tallying operations get done.

Let's take a look at the following code:

The data set of _cpu:

```
 * Read the cpu time file;
data _cpu (keep=Log_Name cpu_time_n cpu_cum real_time_n real_cum);
   retain d_cpu 0 h_cpu 0 m_cpu 0 s_cpu 0;
   retain d_real 0 h_real 0 m_real 0 s_real 0;
   infile _cpu truncover dlm='#';
   length Log_Name $30.;
   length real_time $10. real_time1 $10. real_time2 $10. real_cum $11. real_time_n $11;
   length cpu_time $10. cpu_cum $11. cpu_time_n $11;

   input Log_Name $ real_time1 $ real_time2 $ cpu_time;

   Log_Name = substr(Log_Name,3,28);

   if '0' <= substr(real_time1,1,1) <= '9' and
      substr(real_time1,2,1) = ':' and '0' <= substr(real_time1,3,1) <= '9' and  '0' <= substr(real_time1,4,1)   <= '9' and
      substr(real_time1,5,1) = ':' and '0' <= substr(real_time1,6,1) <= '9' and  '0' <= substr(real_time1,7,1)   <= '9' and
      substr(real_time1,8,1) = '.' and '0' <= substr(real_time1,9,1) <= '9' and  '0' <= substr(real_time1,10,1) <= '9' and
      '0' <= substr(real_time2,1,1) <= '9' and substr(real_time2,2,1) = ':' and  '0' <= substr(real_time2,3,1) <= '9' and
      '0' <= substr(real_time2,4,1) <= '9' and substr(real_time2,5,1) = ':' and  '0' <= substr(real_time2,6,1) <= '9' and
      '0' <= substr(real_time2,7,1) <= '9' and substr(real_time2,8,1) = '.' and  '0' <= substr(real_time2,9,1) <= '9' and
      '0' <= substr(real_time2,10,1) <= '9' then do;
      real_time = real_time1;
      cpu_time  = real_time2;
   end;
   else if
        <<<  more else if statements  see Appendix-A  >>>
   end;

   <<<  more if statements see Appendix-A    >>>

   id_num+1;

   h_real = h_real + h_r;   m_real = m_real + m_r;  s_real = s_real + s_r;
   h_cpu = h_cpu + h_c;  m_cpu = m_cpu + m_c;  s_cpu = s_cpu + s_c;
   if s_real  > 60 then do;  s_real =  s_real  - 60; m_real = m_real + 1; end;
   if m_real > 60 then do; m_real = m_real - 60; h_real  = h_real  + 1; end;
   if h_real > 24 then do;   h_real = h_real - 24;  d_real = d_real + 1;  end;
   if s_cpu > 60 then do;  s_cpu = s_cpu - 60; m_cpu = m_cpu + 1;  end;
   if m_cpu > 60 then do; m_cpu = m_cpu - 60; h_cpu = h_cpu + 1; end;
   if h_cpu > 24 then do; h_cpu = h_cpu - 24; d_cpu = d_cpu + 1; end;

   cpu_time_n  = "00:" || put (h_c,z2.) || ":" || put (m_c,z2.) || ":" || put(s_c,z2.);
   real_time_n = "00:" || put (h_r,z2.) || ":" || put (m_r,z2.) || ":" || put(s_r,z2.);

   cpu_cum  = put(d_cpu,z2.)  || ":" || put (h_cpu,z2.)  || ":" || put (m_cpu,z2.) || ":" || put (s_cpu,z2.);
   real_cum = put(d_real,z2.) || ":" || put (h_real,z2.) || ":" ||   put(m_real,z2.) || ":" || put (s_real,z2.);
run;
```

The above code parses through each line of the extracted text and figures out the REAL time and CPU time for each SAS log file. It also calculates the cumulative sum of the REAL and CPU times. The sum is useful to see if the system contains a log file that has unusually high CPU number. The cumulative CPU and REAL time measures have an upper limit of **ninety-nine days**. That is more than enough cumulative CPU and/or REAL time that any system will "ever" need.

PART-5: PRINT THE REPORTS

This is the easiest step to do, a "**PROC PRINT**". After doing the hard work above, the data is conveniently sorted by SAS log filenames ascending order with their corresponding information. All one has to do now is issue a "PROC PRINT" step specified in the following table:

Print the Result:

```
proc print data =filter_cpu(drop=real_time cpu_time);
   title1 "***** SAS Logs Directory: &Log_Dir *****";
   title2 "***** CPU Versus REAL Time Comparison With Log Files Group
&pf*.log *****";
run ;
```

With the above "PROC PRINT", the report gets generated.

Similar steps are taken when options **"p"** and **"o"** are specified. Option **"p"** produces a parsed text file that looks as follows:

Content of _proc.txt:

```
sales_01:SORT:
sales_01:SORT:
sales_01:SORT:
sales_01:SORT:
sales_01:SORT:
sales_01:SORT:
sales_01:SORT:
sales_01:SORT:
sales_01:SORT:
sales_01:SORT:
sales_01:SORT:
sales_01:SORT:
sales_02:FORMAT:
sales_02:SUMMARY:
sales_02:SORT:
sales_02:SORT:
sales_02:TRANSPOSE:
sales_02:SORT:
sales_02:SORT:

*** etc
```

This shows a list of all procedures that were found in the SAS log with their corresponding SAS log filenames. It is now easy to tally the procedures by filename.

The final parsing operation is triggered by option **"o"** which tallies the total number of observations read and/or used in each SAS job. The next page shows the content of the **"_obs.txt"** parsed text file.

```
Content of _obs.txt:

sales_01:31430:WGHT.ACS_CTY_HU09.
sales_01:3143:WORK.HU05.
sales_01:31430:WGHT.ACS_CTY_HU09.
sales_01:3143:WORK.HU06.
sales_01:31430:WGHT.ACS_CTY_HU09.
sales_01:3143:WORK.HU07.
sales_01:31430:WGHT.ACS_CTY_HU09.
sales_01:3143:WORK.HU08.
sales_01:31430:WGHT.ACS_CTY_HU09.
sales_01:3143:WORK.HU09.
sales_01:3143:WORK.HU05.
sales_01:3143:WORK.HU06.
*** etc
```

The above parsed text file contains observations read from all SAS data sets. Each log file has various observations coming from different SAS data sets. It does not distinguish between input or output SAS data sets. It simply tallies the total number of observations that got processed during the SAS log session.

With the above detailed information, REPORTS #1, #2 and #3 get generated depending on their corresponding options supplied at the Linux/Unix command line. The content of this parsing program may be modified to include other features that users around the world desire. Again, the entire code is located in the Appendix-A section of this paper. Feel free to copy the code and modify it as needed to satisfy your organization's requirements. But make sure to give reference/credit to this paper when presenting it publicly.

CONCLUSION

The "**parse_logs**" is a useful processor that will generate several reports about the content of the SAS logs present in a particular subdirectory with a standard filename prefix. The paper showed the korn shell script and the SAS program that worked together to produce reports regarding CPU time, REAL time, and various procedures and observation number counts and tallies. The processor assumes that the SAS log files are error free and located in one physical subdirectory. There is no limit on the number of SAS log files that it can process. I've processed 6443 SAS logs and produced the reports in less than 2 CPU seconds or 44 real time minutes with options "-top". But when using only the "-t" option, it produced the report under three minutes. The processor is a very powerful tool to use and I encourage all readers to copy the code to their machines and demonstrate it to your organization.

REFERENCES

- *Programmatically Measure SAS® Application Performance On Any Computer Platform With the New LOGPARSE SAS Macro, Michael A. Raithel, Westat, Rockville MD,* SUGI 30, System Architecture, Paper 219-30, http://www2.sas.com/proceedings/sugi30/219-30.pdf, 4/2005.
- *Clearing the Pipe: Overcoming System Limitations of the SAS Pipe Channel of Communication, Fuad J. Foty, U.S. Census Bureau, Suitland MD,* NESUG 2009, Applications Big & Small, http://www.nesug.org/Proceedings/nesug09/ap/ap13.pdf, 9/2009.
- Korn Shell Documentation, http://kornshell.com/doc/
- Red Hat Linux Operating System Documentation, http://www.redhat.com/docs/
- Vi Helpfile, http://www.vmunix.com/~gabor/vi.html
- Unix in a Nutshell, A Desktop Quick Reference for System V & Solaris 2.0, Daniel gilly and the staff of O'Reilly & Associates, Inc.

ACKNOWLEDGMENTS

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.
Other brand and product names are trademarks of their respective companies.

CONTACT INFORMATION

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

Fuad J Foty
U.S. Census Bureau
4600 Silver Hill Road
Room 3K460F
Suitland, MD 20746
301-763-5476
fuad.j.foty@census.gov

* *

Appendix A- The Entire “parse_logs.sas” SAS Program

```
options mprint symbolgen;

%macro main;

libname MyLib "" ;
filename _cpu "~/_cpu.txt";
filename _proc "~/_proc.txt";
filename _obs "~/_obs.txt";

%let opt = %sysget(opt);
%let pf  = %sysget(pf);
%let Log_Dir = %sysget(Log_Dir);
%put pf= &pf;
%put opt= &opt;
%put Log_Dir= &Log_Dir;

/* Determine the options */
%let minus = %qsubstr(&opt,1,1);
%let flg_t = 0;
%let flg_p = 0;
%let flg_o = 0;
%if "&minus" = "-" %then %do;
   %do i = 2 %to %length(&opt);
      %if %qsubstr(&opt,&i,1) = t  %then %let flg_t=1;
      %if %qsubstr(&opt,&i,1) = p %then %let flg_p=1;
      %if %qsubstr(&opt,&i,1) = o %then %let flg_o=1;
   %end;
%end;

/* Process cpu and real time tally option*/
%if &flg_t = 1 %then %do;
  data _null_;
    x "find . -type f -name '&pf*.log' -print | xargs tail -5 |
       sed 's/\(NOTE]*\).*/\1/' |
       sed 's/[NOTE]*//g' |
       sed '/^$/ d' |
       tr -d '\f' > ~/_cpu.txt";
    x 'echo ":%s/.log <==/#/g" > ~/temp_cmd~';
    x 'echo ":%s/real time//g" >> ~/temp_cmd~';
    x 'echo ":%s/cpu time//g" >> ~/temp_cmd~';
    x 'echo ":%s/ seconds//g" >> ~/temp_cmd~';
    x 'echo ":%s/ //g" >> ~/temp_cmd~';
    x 'echo ":%s/\n//g" >> ~/temp_cmd~';
    x 'echo ":%s/^[0-9][0-9][.]/0:&/g" >> ~/temp_cmd~';
    x 'echo ":%s/^[0-9][.]/0:0&/g" >> ~/temp_cmd~';
    x 'echo ":%s/^[0-9][:][0-9][0-9][.]/0:0&/g" >> ~/temp_cmd~';
    x 'echo ":%s/^[0-9][0-9][:][0-9][0-9][.]/0:&#/g" >> ~/temp_cmd~';
    x 'echo ":%s/^$\n//g" >> ~/temp_cmd~';
    x 'echo ":%s/\(133he]*\).*/\1/g" >> ~/temp_cmd~';
    x 'echo ":%s/133he\n//g" >> ~/temp_cmd~';
    x 'echo ":%s/[.0-9][0-9]$/&#/g" >> ~/temp_cmd~';
    x 'echo ":wq!" >> ~/temp_cmd~';
    x "vi ~/_cpu.txt < ~/temp_cmd~ > /dev/null 2> /dev/null";

    x "cat ~/_cpu.txt | tr -d '\n' | tr '==>' '\012' > ~/_cpu.txt2";
    x 'echo ":%s/[#][a-z]$/\1/g" > ~/temp_cmd~';
    x 'echo ":wq!" >> ~/temp_cmd~';
    x "vi ~/_cpu.txt2 < ~/temp_cmd~ > /dev/null 2> /dev/null";

    x "cat ~/_cpu.txt2 | grep '#' | sed 's/[.][#]/./g' | sort > ~/_cpu.txt";
    x "\rm ~/temp_cmd~";
    x "\rm ~/_cpu.txt2";
  run;
```

```
* Read the cpu time file;
data _cpu (keep=Log_Name cpu_time_n cpu_cum real_time_n real_cum);
   retain d_cpu 0 h_cpu 0 m_cpu 0 s_cpu 0;
   retain d_real 0 h_real 0 m_real 0 s_real 0;
   infile _cpu truncover dlm='#';
   length Log_Name $30.;
   length real_time $10. real_time1 $10. real_time2 $10. real_cum $11. real_time_n $11;
   length cpu_time $10. cpu_cum $11. cpu_time_n $11;

   input Log_Name $ real_time1 $ real_time2 $ cpu_time;

   Log_Name = substr(Log_Name,3,28);

   if '0' <= substr(real_time1,1,1) <= '9' and
      substr(real_time1,2,1) = ':' and '0' <= substr(real_time1,3,1) <= '9' and  '0' <= substr(real_time1,4,1)   <= '9' and
      substr(real_time1,5,1) = ':' and '0' <= substr(real_time1,6,1) <= '9' and  '0' <= substr(real_time1,7,1)   <= '9' and
      substr(real_time1,8,1) = '.' and '0' <= substr(real_time1,9,1) <= '9' and  '0' <= substr(real_time1,10,1) <= '9' and
      '0' <= substr(real_time2,1,1) <= '9' and substr(real_time2,2,1) = ':' and  '0' <= substr(real_time2,3,1) <= '9' and
      '0' <= substr(real_time2,4,1) <= '9' and substr(real_time2,5,1) = ':' and  '0' <= substr(real_time2,6,1) <= '9' and
      '0' <= substr(real_time2,7,1) <= '9' and substr(real_time2,8,1) = '.' and  '0' <= substr(real_time2,9,1) <= '9' and
      '0' <= substr(real_time2,10,1) <= '9' then do;
      real_time = real_time1;
      cpu_time  = real_time2;
   end;
   else if '0' <= substr(real_time2,1,1) <= '9' and
      substr(real_time2,2,1) = ':' and  '0' <= substr(real_time2,3,1) <= '9' and  '0' <= substr(real_time2,4,1)   <= '9' and
      substr(real_time2,5,1) = ':' and  '0' <= substr(real_time2,6,1) <= '9' and  '0' <= substr(real_time2,7,1)   <= '9' and
      substr(real_time2,8,1) = '.' and  '0' <= substr(real_time2,9,1) <= '9' and  '0' <= substr(real_time2,10,1) <= '9' and
      '0' <= substr(cpu_time,1,1) <= '9' and substr(cpu_time,2,1) = ':' and '0' <= substr(cpu_time,3,1) <= '9' and
      '0' <= substr(cpu_time,4,1) <= '9' and substr(cpu_time,5,1) = ':' and '0' <= substr(cpu_time,6,1) <= '9' and
      '0' <= substr(cpu_time,7,1) <= '9' and substr(cpu_time,8,1) = '.' and '0' <= substr(cpu_time,9,1) <= '9' and
      '0' <= substr(cpu_time,10,1) <= '9' then do;
      real_time = real_time2;
   end;

   if '0' <= substr(real_time,1,1) <= '9' and
      substr(real_time,2,1) = ':' and '0' <= substr(real_time,3,1) <= '9' and '0' <= substr(real_time,4,1) <= '9' and
      substr(real_time,5,1) = ':' and '0' <= substr(real_time,6,1) <= '9' and '0' <= substr(real_time,7,1) <= '9' and
      substr(real_time,8,1) = '.' and '0' <= substr(real_time,9,1) <= '9' and '0' <= substr(real_time,10,1) <= '9' then do;
       h_r =input(substr(real_time,1,1),1.);
       m_r =input(substr(real_time,3,2),2.);
       s_r =input(substr(real_time,6,5),4.2);
   end;
   else do;
       h_r = 0; m_r = 0; s_r = 0; real_time='0:00:00.00';
   end;

   if '0' <= substr(cpu_time,1,1) <= '9' and
      substr(cpu_time,2,1) = ':' and '0' <= substr(cpu_time,3,1) <= '9' and '0' <= substr(cpu_time,4,1) <= '9' and
      substr(cpu_time,5,1) = ':' and '0' <= substr(cpu_time,6,1) <= '9' and '0' <= substr(cpu_time,7,1) <= '9' and
      substr(cpu_time,8,1) = '.' and '0' <= substr(cpu_time,9,1) <= '9' and '0' <= substr(cpu_time,10,1) <= '9' then do;
       h_c =input(substr(cpu_time,1,1),1.);
       m_c =input(substr(cpu_time,3,2),2.);
       s_c =input(substr(cpu_time,6,5),4.2);
   end;
   else do;
       h_c = 0; m_c = 0; s_c = 0; cpu_time='0:00:00.00';
   end;

   id_num+1;
```

```
    h_real = h_real + h_r;   m_real = m_real + m_r;  s_real = s_real + s_r;
    h_cpu = h_cpu + h_c;  m_cpu = m_cpu + m_c;  s_cpu = s_cpu + s_c;
    if s_real  > 60 then do;  s_real =  s_real  - 60; m_real = m_real + 1; end;
    if m_real > 60 then do; m_real = m_real - 60; h_real  = h_real  + 1; end;
    if h_real > 24 then do;   h_real = h_real - 24;  d_real = d_real + 1;  end;
    if s_cpu > 60 then do;  s_cpu = s_cpu - 60; m_cpu = m_cpu + 1;  end;
    if m_cpu > 60 then do; m_cpu = m_cpu - 60; h_cpu = h_cpu + 1; end;
    if h_cpu > 24 then do; h_cpu = h_cpu - 24; d_cpu = d_cpu + 1; end;

    cpu_time_n  = "00:" || put (h_c,z2.) || ":" || put (m_c,z2.) || ":" || put(s_c,z2.);
    real_time_n = "00:" || put (h_r,z2.) || ":" || put (m_r,z2.) || ":" || put(s_r,z2.);

    cpu_cum  = put(d_cpu,z2.)  || ":" || put (h_cpu,z2.)  || ":" || put (m_cpu,z2.) || ":" || put (s_cpu,z2.);
    real_cum = put(d_real,z2.) || ":" || put (h_real,z2.) || ":" ||   put(m_real,z2.) || ":" || put (s_real,z2.);
  run;

  proc sort data=_cpu; by Log_Name; run;

  data filter_cpu(drop=i flg real_time_save real_cum_save
                       cpu_time_save cpu_cum_save);
   set _cpu;
   length real_time_save $10. real_cum_save $11.;
   length cpu_time_save $10. cpu_cum_save $11.;
   retain real_time_save real_cum_save cpu_time_save cpu_cum_save;
   by Log_Name;
   flg = 0;
   do i = 1 to 30;
     if substr(Log_Name,i,1) = "'" or
        substr(Log_Name,i,1) = ";" or
        substr(Log_Name,i,1) = "/" or
        substr(Log_Name,i,1) = ":" then flg = 1;
   end;

   if substr(Log_Name,1,9) = "parse_log" then flg = 1;

   if flg = 0 then do;
     if real_time ^= ' ' then real_time_save = real_time;
     if real_cum  ^= ' ' then real_cum_save  = real_cum;
     if cpu_time  ^= ' ' then cpu_time_save  = cpu_time;
     if cpu_cum   ^= ' ' then cpu_cum_save  = cpu_cum;

     if last.Log_Name then do;
       real_time  = real_time_save;
       real_cum  = real_cum_save;
       cpu_time  = cpu_time_save;
       cpu_cum   = cpu_cum_save;
       output;
     end;
   end;
  run;

  proc print data =filter_cpu(drop=real_time cpu_time);
    title1 "***** SAS Logs Directory: &Log_Dir *****";
    title2 "***** CPU Versus REAL Time Comparison With Log Files Group &pf*.log *****";
  run ;

%end;
```

```
/* Process procedures tally option*/
%if &flg_p = 1 %then %do;
  data _null_;
    x "find . -type f -name '&pf*.log' -print | xargs grep -i procedure > ~/_proc.txt";
    x 'echo ":%s/.log//g" > ~/temp_cmd~';
    x 'echo ":%s/NOTE: PROCEDURE //g" >> ~/temp_cmd~';
    x 'echo ":%s/ used (Total process time):/:/g" >> ~/temp_cmd~';
    x 'echo ":wq!" >> ~/temp_cmd~';
    x "vi ~/_proc.txt < ~/temp_cmd~ > /dev/null 2> /dev/null";
    x "\rm ~/temp_cmd~";
  run;

  data _proc (keep=Log_Name Proc_Type);
    infile _proc truncover dlm=':';
    length Log_Name $30. Proc_Type $15.;
    input Log_Name $ Proc_Type $;

    Log_Name = substr(Log_Name,3,28);

    flg = 0;
    if substr(Log_Name,1,6) = "parse_" then flg = 1;

    if flg = 0 then output;

    proc sort; by Log_Name;

  run;

  data _proc (drop=Proc_Type);
    set _proc;
    retain SORT_CNT 0
           FORMAT_CNT 0
           SUMMARY_CNT 0
           TRANSPOSE_CNT 0
           DELETE_CNT 0 ;
    by Log_Name;

    if Proc_Type = "SORT" then SORT_CNT+1;
    if Proc_Type = "FORMAT" then FORMAT_CNT+1;
    if Proc_Type = "SUMMARY" then SUMMARY_CNT+1;
    if Proc_Type = "TRANSPOSE" then TRANSPOSE_CNT+1;
    if Proc_Type = "DELETE" then DELETE_CNT+1;

    if last.Log_Name then do;
      output;

      SORT_CNT=0;
      FORMAT_CNT=0;
      SUMMARY_CNT=0;
      TRANSPOSE_CNT=0;
      DELETE_CNT=0;
    end;
  run;

  data _proc2;
    set _proc;
    by Log_name;
    if sort_cnt = . then SORT_CNT=0;
    if format_cnt = . then FORMAT_CNT=0;
    if summary_cnt = . then SUMMARY_CNT=0;
    if transpose_cnt = . then TRANSPOSE_CNT=0;
    if delete_cnt = . then DELETE_CNT=0;
  run;

  proc print data =_proc2;
    title1 "***** SAS Logs Directory: &Log_Dir *****";
    title2 "***** Log Files Group &pf*.log Procedures Tally *****";
  run ;

%end;
```

```
/* Process observations tally option*/
%if &flg_o = 1 %then %do;
  data _null_;
    x "find . -type f -name '&pf*.log' -print | xargs grep -i 'observations read' > ~/_obs.txt";
    x 'echo ":%s/.log//g" > ~/temp_cmd~';
    x 'echo ":%s/NOTE: There were //g" >> ~/temp_cmd~';
    x 'echo ":%s/ observations read from the data set /:/g" >> ~/temp_cmd~';
    x 'echo ":wq!" >> ~/temp_cmd~';
    x "vi ~/_obs.txt < ~/temp_cmd~ > /dev/null 2> /dev/null";
    x "\rm ~/temp_cmd~";
  run;

  *Read the Log Name files;
  data _obs (keep=Log_Name obs_num);
    infile _obs truncover dlm=':';
    length Log_Name $30. obs_num $15.;
    input Log_Name $ obs_num $;

    Log_Name = substr(Log_Name,3,28);

    flg = 0;
    if substr(Log_Name,1,6) = "parse_" then flg = 1;

    if flg = 0 then output;

    proc sort; by Log_Name;

  run;

  data _obs (drop=obs_num);
    set _obs;
    retain obs_sum 0;

    by Log_Name;

    obs_sum = obs_sum + input(obs_num,9.);

    if last.Log_Name then do;
      output;
      obs_sum=0;
    end;
  run;

  data _obs2;
    set _obs;
    by Log_name;
    if obs_sum = . then obs_sum=0;
  run;

  proc print data =_obs2;
    title1 "***** SAS Logs Directory: &Log_Dir *****";
    title2 "***** Log Files Group &pf*.log Summary of Obs Read Tally *****";
  run ;

%end;

data _null_;
  x "\rm ~/_cpu.txt";
  x "\rm ~/_proc.txt";
  x "\rm ~/_obs.txt";
run;

%mend main;
%main;
```