Pinching Off Your SAS Log: Shifting the Perception of "SAS Logs" from a Post Hoc Validation Used Haplessly and Often Too Late…to a Dynamic Quality Control That Drives Exception Handling and Program Flow

Troy Martin Hughes

ABSTRACT

Too often in SAS® literature, the role of the program log is narrowly conceptualized as a static, post hoc review that is utilized only to validate program success and to detect program failure. Especially when software development occurs outside of a formalized software development life cycle (SDLC), as is often the case in end user development environments, software integrity and reliability are often gained not through software quality suffused throughout code but rather through manual examination of the log to demonstrate both that the software did what it was supposed to do and that it did not do what it was not supposed to do. While SAS practitioners commonly read program logs and although an opus of SAS of literature even demonstrates parsing algorithms that automate the log validation process, in nearly all log literature the logs are analyzed only after software termination—not during execution. This text introduces dynamic log parsing during software execution which facilitates flexible response to system resource utilization. The FULLSTIMER macro enables software to detect anomalous or undesirable CPU, input/output (I/O), or memory resource consumption and to respond dynamically, thus facilitating finer-tuned software that can analyze and react to poor performance or inefficiency.

INTRODUCTION

The normal uses of the SAS log—analysis after software has terminated, either after normal or abnormal execution…this is commonly achieved through manual reading of the SAS logs…

More advanced parsing of the SAS log can be achieved through automated solutions that parse SAS logs after program termination. These are widely demonstrated throughout SAS literature to validate program success through failure detection…

In the most advanced demonstrations in literature, SAS logs are parsed automatically and, based upon the results, failures may be communicated through emails that are automatically sent to stakeholders, dashboards that are updated, or other modalities…

This text builds upon the opus of SAS log parsing and advances this by introducing the FULLSTIMER macro that reads system resource information printed only to SAS logs. Rather than producing a single, monolithic SAS log for the program, logs are "pinched off" at defined intervals for analysis, saved to discrete text files, and analyzed in real time to drive further dynamic processing…

Examples demonstrate how this method can be utilized to analyze the relative efficiency of SAS processes to alert inefficient or poor performing software. For example, as the complexity of some procedures increase, or the size of data sets increase, or the number of SAS jobs running on a server or system increases, performance can decrease. Through implementation of the FULLSTIMER macro, sometimes subtle fluctuations in system performance can be measured and analyzed to predict subsequent performance or functional failure. Another example demonstrates how the FULLSTIMER macro can be utilized to automatically determine the optimal number of parallel processes to run in distributed, divide-and-conquer design.

CONCLUSION

While normal usage of the SAS log—whether through manual or automated parsing—can often detect and demonstrate program failure, implementation of the FULLSTIMER macro can actually detect subtle variations in system resource utilization to predict failure and prevent it through dynamic program flow.

REFERENCES

APPENDIX A. FULLSTIMER MACRO

```
%macro readfullstimer(textfile= /* path, name, and extension of text file */);
* converts all times to seconds SS.xx format from HH:MM:SS.ss format;
%let syscc=0;
%global fullstimerRC;
%global realtime;
%global usercputime;
%global systemcputime;
%global memory;
%global osmemory;
%global stepcount;
%global switchcount;
%global pagefaults;
%global pagereclaims;
%global volcontextswitches;
%global involcontextswitches;
%global blockinops;
%global blockoutops;
%global errtextlong;
%let fullstimerRC=;
%let realtime=;
%let usercputime=;
%let systemcputime=;
%let memory=;
%let osmemory=;
%let stepcount=;
%let switchcount=;
%let pagefaults=;
%let pagereclaims=;
%let volcontextswitches=;
%let involcontextswitches=;
%let blockinops=;
%let blockoutops=;
%let errtextlong=;
data _null_;
   length tab $100 errtextlong $2000 erryes 8;
   infile "&textfile" truncover;
   input tab $100.;
   if _n_=1 then errtextlong='';
   if _n_>=5 then do; *** needs to be made dynamic based on which version of
FULLSTIMER is used;
      if find(tab,"ERROR")>0 or find(tab,"WARNING")>0 then erryes=1;
      else if find(tab,"real time")>0 then erryes=0;
      if erryes=1 then errtextlong=strip(errtextlong) || ' *** ' || strip(tab);
      call symput('errtextlong',strip(errtextlong));
      if lowcase(substr(tab,1,9))='real time' then do;
         if count(scan(substr(tab,10),1,' '),':')=0 then call
symput('realtime',scan(substr(tab,10),1,' '));
         else if count(scan(substr(tab,10),1,' '),':')=1 then call
symput('realtime',(input(strip(scan(substr(tab,10),1,':')),8.0) * 60)
               + input(strip(scan(substr(tab,10),2,':')),8.2));
         else if count(scan(substr(tab,10),1,' '),':')=2 then call
symput('realtime',(input(strip(scan(substr(tab,10),1,':')),8.0) * 3600)
```

```
               + (input(strip(scan(substr(tab,10),2,':')),8.0) * 60) +
input(strip(scan(substr(tab,10),3,':')),8.2));
         end;
      else if lowcase(substr(tab,1,13))='user cpu time' then do;
         if count(scan(substr(tab,14),1,' '),':')=0 then call
symput('usercputime',scan(substr(tab,14),1,' '));
         else if count(scan(substr(tab,14),1,' '),':')=1 then call
symput('usercputime',(input(strip(scan(substr(tab,14),1,':')),8.0) * 60)
               + input(strip(scan(substr(tab,14),2,':')),8.2));
         else if count(scan(substr(tab,14),1,' '),':')=2 then call
symput('usercputime',(input(strip(scan(substr(tab,14),1,':')),8.0) * 3600)
               + (input(strip(scan(substr(tab,14),2,':')),8.0) * 60) +
input(strip(scan(substr(tab,14),3,':')),8.2));
         end;
      else if lowcase(substr(tab,1,15))='system cpu time' then do;
         if count(scan(substr(tab,16),1,' '),':')=0 then call
symput('systemcputime',scan(substr(tab,16),1,' '));
         else if count(scan(substr(tab,16),1,' '),':')=1 then call
symput('systemcputime',(input(strip(scan(substr(tab,16),1,':')),8.0) * 60)
               + input(strip(scan(substr(tab,16),2,':')),8.2));
         else if count(scan(substr(tab,16),1,' '),':')=2 then call
symput('systemcputime',(input(strip(scan(substr(tab,16),1,':')),8.0) * 3600)
               + (input(strip(scan(substr(tab,16),2,':')),8.0) * 60) +
input(strip(scan(substr(tab,16),3,':')),8.2));
         end;
      else if lowcase(substr(tab,1,6))='memory' then call
symput('memory',scan(substr(tab,7),1,' k')/1024); *convert from KB to MB;
      else if lowcase(substr(tab,1,9))='os memory' then call
symput('osmemory',scan(substr(tab,10),1,' k')/1024); *convert from KB to MB;
      else if lowcase(substr(tab,1,10))='step count' then do;
         call symput('stepcount',scan(substr(tab,11),1,' '));
         call symput('switchcount',scan(substr(tab,11),4,' '));
         end;
      else if lowcase(substr(tab,1,11))='page faults' then call
symput('pagefaults',scan(substr(tab,12),1,' '));
      else if lowcase(substr(tab,1,13))='page reclaims' then call
symput('pagereclaims',scan(substr(tab,14),1,' '));
      else if lowcase(substr(tab,1,26))='voluntary context switches' then call
symput('volcontextswitches',scan(substr(tab,27),1,' '));
      else if lowcase(substr(tab,1,28))='involuntary context switches' then call
symput('involcontextswitches',scan(substr(tab,29),1,' '));
      else if lowcase(substr(tab,1,24))='block input operations' then call
symput('blockinops',scan(substr(tab,25),1,' '));
      else if lowcase(substr(tab,1,25))='block output operations' then call
symput('blockoutops',scan(substr(tab,26),1,' '));
      end;
   retain erryes errtextlong;
run;
* set all missing values to -9;
%let macrolist=realtime usercputime systemcputime memory osmemory stepcount
switchcount pagefaults
      pagereclaims volcontextswitches involcontextswitches blockinops blockoutops;
%let i=1;
%do %while(%length(%scan(&macrolist,&i,,S))>1);
   %let mac=%scan(&macrolist,&i,,S);
```

```
   %put MAC: &mac &&&mac;
   %if %length(&&&mac)=0 %then %let &mac=-9;
   %let i=%eval(&i+1);
   %end;
%let errtextlong=%sysfunc(compress("&errtextlong",,kns));
%if &syscc>0 %then %do;
   %let fullstimerRC=FAILURE;
   %return;
   %end;
%mend;
```

For example, to implement the %READFULLSTIMER macro on the DATA step that creates PERM.Subset, the following code is executed.

```
libname perm 'c:\perm';
%let path=%sysfunc(pathname(perm));
options fullstimer;
proc printto log="&path/out.txt" new;
run;
data perm.subset;
   set perm.sortme (where=(num1<.1));
run;
proc printto;
run;
%readfullstimer(textfile=&path/out.txt);
```