A Multi-processing Tool to Batch Submit a List of Programs with Real Time Feedback and Dashboard Email Notification

Huashan Huo & Fanyu Li, PPD, Beijing

ABSTRACT

In clinical research, it is very common that a large number of SAS programs are to be repeatedly batch run due to program modifications or new data updates. In the past few years, several papers authored by pharmaceutical industry programmers (Gilbert Chen 2002; Shu 2006; Prescod Cawley 2010; Wong Sun 2010; Conover 2011; Andrew E. Hansen 2013) were published in this area to describe methods and tools for automating this process. The purpose of this paper is to introduce a tool for multi-processing batch submitting a large number of SAS programs concurrently. Overall execution time is significantly reduced by using this tool. Users can also get real-time status feedbacks on SAS execution progress and email notifications with a Dashboard report on the completion status once batch run completed.

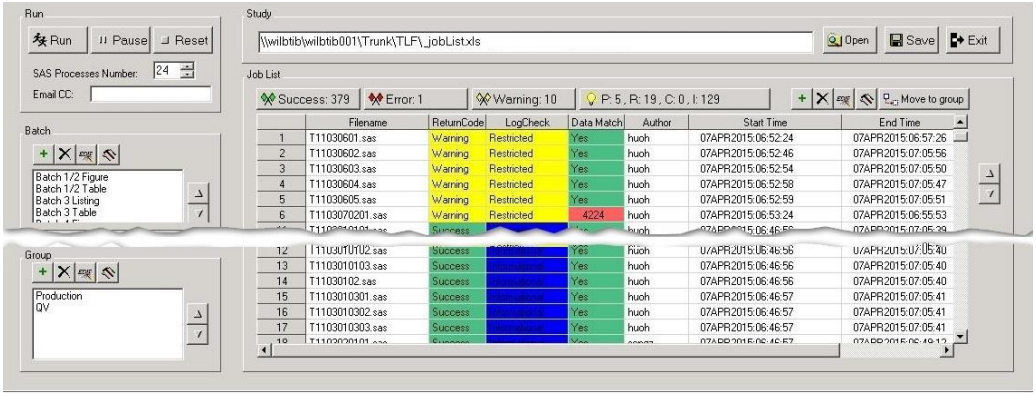
INTRODUCTION

This paper introduces a high efficiency tool. The idea was started when examining a clinical study which has more than 1,000 tables, listings and figures. With each data update, programs were to be rerun and the manual rerun would be very complicated and time consuming. By using existing batch processing tools, the process of rerunning this study still cost more than seven hours rerunning programs one by one. That prompted us to develop a new tool which is capable of running multiple programs concurrently and keeps tracking the status of these program reruns. In addition, this tool automatically emails its users with status updates.

In order to conduct the parallel batch submission, the hierarchy of programs must be defined first. One could use batch and group to define a level. There is no sequence within levels so that programs within each level could be batch submitted in parallel. However, among different levels, there is a sequence. Different levels must be batch submitted in sequence. For example, the production programs must be submitted prior to the validation programs. This could maintain the logic and maximize the utilization of computer resources. SYSTASK asynchronous mode is the cornerstone of our tool. Our analysis shows that this new tool saves more than 2/3 of the time compared with the previous methods. This tool manages all TLF programs through SVN. The benefit is that it automatically accesses author IDs by the SVN and then automatically obtains user email address, thus there is no need to frequently update and manually provide author IDs and email address information. This tool could self-discover the change of author IDs. However, there are still needs to consider how to preserve independent execution environment for the programs and how to concurrently write a same dataset without the SAS/SHARE, these are to be discussed in the following sections.

METHODS

This tool was implemented using SAS/AF and SYSTASK statement. The general idea is to use SAS/AF to implement the GUI, by calling the %MULTI_SESSION_RUN macro in RSUBMIT. This macro could use SYSTASK statement to launch a predefined number of SAS sessions to run in parallel in SAS. The following is the GUI display.



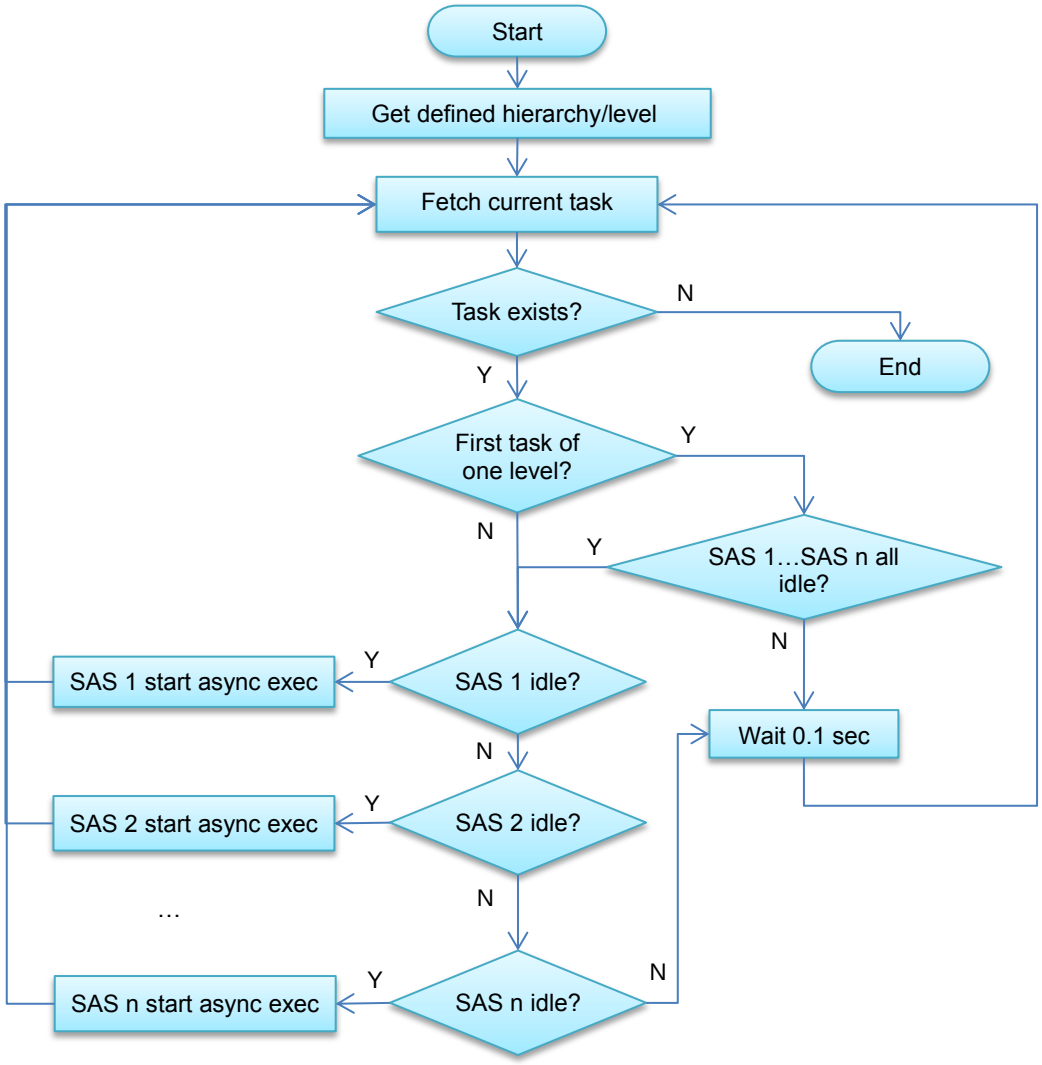
1. Using SAS/AF to develop GUI

If using CONNECTWAIT=NO and RSUBMIT statement, user could get the control of GUI without waiting for the completion of a running program and get the latest status of all programs. SAS/AF is a classic visual programming tool. There is a specific introduction in SAS help. SAS/AF is just one choice to develop GUI. We also could use SAS %WINDOW and %DISPLAY to achieve a real-time interaction. Due to the complexity of the macros and previous papers (Alden 2000, Mace 2002), we will not discuss this part in this paper. We chose SAS/AF since it has a stronger interactions than %WINDOW and %DISPLAY. You can also use color marks to identify the different types (Success, Error, Warning, the Log issue, etc.). By clicking on the program name of GUI, the tool can automatically open the corresponding SAS program and the corresponding Log. That makes debugging and fixing the programs much more easily. The table viewer controls also bring filter selection function and that makes it easy to find what you need from numerous programs.

2. General overview of the %MULTI_SESSION_RUN macro

When running this macro, user should create a list of programs which contains all programs needed in parallel batch submission. There are many methods to assemble this. We recommend using excel file as a best practice. %MULTI_SESSION_RUN calls can independent operation. For example, you can put the macro into a scheduled task to run.

Brief flow chart of the macro is as follows:



%MULTI_SESSION_RUN can start a predefined number of SAS sessions to run SAS programs in parallel in the background using SYSTASK statement. At the same time, it can automatically examine the running status of each SAS session. When an idle SAS session is found, it is closed and a new session is assigned to run. In this case, we do not utilize the existing idle session to start another program run in order to preserve independent execution environment for the programs. This is because during an execution, a program may modify system options, global macro variables, temporary data sets, formats, templates, etc. and may interfere with the execution of next program run. The macro may end the execution of a program that freezes or runs into errors. It automatically looks up the program owner and the executer's information and sends a summary email to them when a problem occurs.

```
%MULTI_SESSION_RUN(
   studyName =,
   programList = %str(&xlsPathName),
   sessionNum = 16,
   timeout = 3600,  /*seconds*/
   onErrorExit = Y,
   genDashboard = Y,
   copyto =
);
```

Parameter explanations：

studyName - Study name

programList - A list file of programs which contains all programs need parallel batch submission

sessionNum - The number of parallel batch submission SAS session

timeout - The maximum number of seconds that WATIFOR should suspend the current SAS session

onErrorExit - Quit when meet an error

genDashboard - Whether to create the Dashboard in email

copyto - The copyto email address (email will be sent to the executer and program owner as well）

3. Program execution level definition

The hierarchy of the programs must be defined in a parallel batch submission. Using Excel to define the batch and group level is easy to read and maintain.

	A	B	C	D	E	F	G	H
1	Batch	Group	CodeName	Path	Skip	Pause	Sysparm	ReturnCode
2	Batch 1 Table	Production	T1.sas		0	0		0
3	Batch 1 Table	Production	T2.sas		0	0		0
4	Batch 1 Table	QV	VT1.sas		0	0		0
5	Batch 1 Table	QV	VT2.sas		0	0		0
6	Batch 2 Table	Production	T3.sas		0	0		0
7	Batch 2 Table	Production	T4.sas		0	0		0
8	Batch 2 Table	QV	VT3.sas		0	0		0
9	Batch 2 Table	QV	VT4.sas		0	0		0

We could use batch and group to define a level. For example, the production of the batch1 Table will be in one level. There is no sequence within one level so that they could be parallel batch submitted. However, among different levels, there is a sequence, and different levels must be batch submitted in the right sequence. For example, the batch 1 table production should be prior to batch 1 table QV (quality validation). Because we need results of batch 1 table production when we run the batch 1 table QV.

4. Submit a program

Each program will be submitted by %runOneJob macro and SYSTASK statement. The structure is as follows:

```
%macro runOneJob;
   %let curNumofLevel=%eval(&curNumofLevel.+1);
   /*Add code here: check file exist, etc. */
   SYSTASK COMMAND "'c:\progra~1\sas9~1.2\sasfou~1\9.2\sas.exe'
      -autoexec ""&G_fullpath.AUTOEXEC.SAS""
      -initstmt '||'""libname here '''
||"d:\&&&&jobNameOfLevel&&L&level..N&curNumofLevel.."||''';""
      -sysin ""&G_fullpath.&&&&jobNameOfLevel&&L&level..N&curNumofLevel...sas""
      -log ""&G_fullpath.&&&&jobNameOfLevel&&L&level..N&curNumofLevel...log""
      -print ""&G_fullpath.&&&&jobNameOfLevel&&L&level..N&curNumofLevel...lst""
      -noterminal -rsasuser -nosplash -nologo
      " nowait status=done&j
      taskname="&&&&jobNameOfLevel&&L&level..N&curNumofLevel";
   %if &SYSRC ne 0 %then %do;
      %put Batch SAS job &G_fullpath.
         &&&&jobNameOfLevel&&L&level..N&curNumofLevel...sas
         start failed: SYSRC=&SYSRC..;
   %end;
   %global running&j;
   %let running&j=&&&&jobNameOfLevel&&L&level..N&curNumofLevel..;
%mend runOneJob;
```

Key points:

- SYSTASK asynchronous mode

SYSTASK allows user to execute operating system-specific commands from within user's SAS session or application. SYSTASK runs these commands as asynchronous tasks, which means that these tasks execute independently of all other tasks that are currently running. Asynchronous tasks run in the background, so user can perform additional tasks while the asynchronous task is still running. SYSTASK asynchronous mode is the cornerstone of our tool.

- SYSTASK NOWAIT option

Determines whether SYSTASK COMMAND suspends execution of the current SAS session until the task has completed. For tasks that are started with the NOWAIT argument, we use the WAITFOR statement to suspend execution of the SAS session until the task has finished.

- RSASUSER ensure the SAS session independent

This option specifies to open the SASUSER library for read access. It is useful since then all SAS sessions will use a single SASUSER library without conflict risk.

- The method of concurrently write dataset

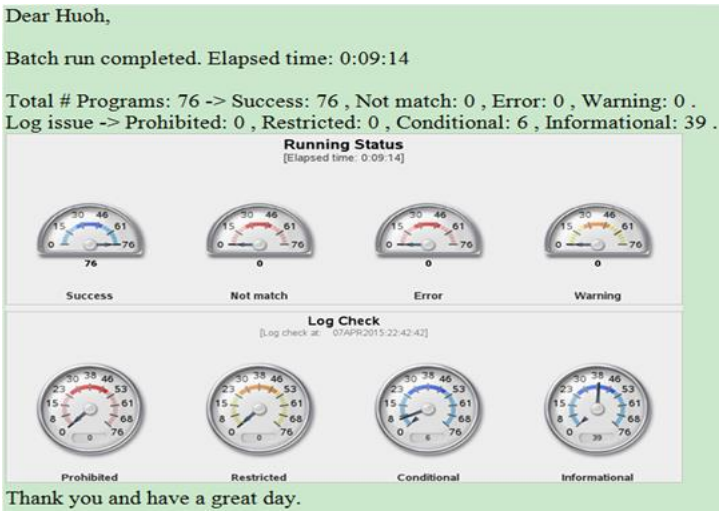
When programs are run in parallel, concurrent conflicts in writing a same dataset can happen if user does not have a permit to use the SAS/SHARE. We could use INITSTMT statement to solve this problem. We use INITSTMT statement to create a library "Here" for each session. This library points to a solely physical path (different programs have different paths). When all program executions complete, one could set all datasets together.

-initstmt '||"""libname here '" ||"d:\&&&&jobNameOfLevel&&L&level..N&curNumofLevel.."||"';""

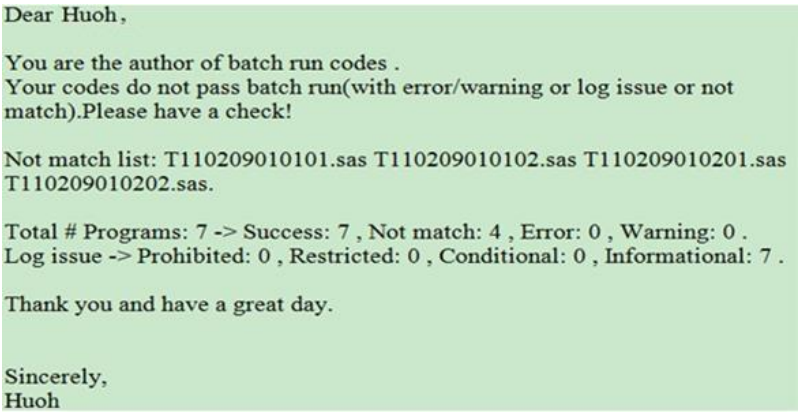
5. Sending emails

When batch submission completes and results are available, this tool sends emails to authors and program executor. The following Dashboard is created by GKPI, GSLIDE and GREPLAY and will be sent as the email body instead of attachment for convenience reasons.

Email example 1: Summary information



Email example 2: Mismatch information



6. Parallel efficiency and resource utilization

Even with a single core CPU, this tool could improve the efficiency significantly. By scheduling multiple sessions in parallel, as opposed to in sequence, the utilization of CPU resource is greatly improved. This tool works the best on multi-CPU or multi-core systems. To maximize the power of such systems, tasks are dispatched to different CPUs/cores, instead of the same CPU in a time-division-multiplex mode. A computer system with a single CPU concurrently running multiple applications would not be ideal for parallel executions. Memory capacity is another important factor since each running session consumes a certain amount of memory. When the memory taken by multiple sessions reaches the limit of physical capacity, virtual memory kicks in and that will negatively affect system performance.

7. Next step

We plan to use SAS/IT to enhance this tool to support Browser/Server mode by using other programming languages. That will help facilitate the development, deployment, maintenance and use of this tool. For now, this current tool is a stand-alone operation and can't be operated among multiple machines for workload balance.

CONCLUSION

Program execution time is significantly reduced by using this tool. Users can also get real-time status feedbacks on SAS execution progress and email notifications with a Dashboard report on the completion status once batch run completes. However, the execution time reduction is also in relation to the CPU cores and memory capacity. We plan to enhance this tool to support Browser/Server mode in the future. In a batch submit program environment, concurrency is key to improve execution efficiency. A GUI system that is capable of real-time feedback allows tool users to continuously track the status and progress of such executions, and therefore it becomes convenient for programmers to debug and modify programs. With the help of completion email notification, users are not required to frequently manually check for completion status. Lastly, error log and exact-program-name notification features greatly facilitate programmers to identify exact errors and to correct programs.

REFERENCES

Alden, Kay. 2000 "SAS' Best Kept Secret: Macro Windows® for Applications Development" Proceedings of the Twenty-Fifth Annual SAS Users Group International Conference, paper 76.

Cawley, Jim, Prescod Jillian 2010 "Need Reporting Deliverables the Painless and Easy Way? A Batch Submit Macro of Course!" Proceedings of the PharmaSUG 2010 Conference, paper CC10.

Chen, Ling Y., Gilibet Steven A. 2002 "Run All Your SAS(R) Programs in One Program Automatically" Proceedings of the Twenty-Seventh Annual SAS Users Group International Conference, paper 105.

Conover, William 2011 "BAT Files: Run all Your Programs with One Click in PC SAS" Proceedings of the PharmaSUG 2011 Conference, paper PO01.

Mace, Michael A. 2002 "%WINDOW: You Can Talk to the Users, and They Can Talk Back" Proceedings of the Twenty-Seventh Annual SAS Users Group International Conference, paper 192.

Murphy, Howard. 2007 "Changing Data Set Variables into Macro Variables" Proceedings of the 2007 SAS Global Forum, paper 050.

SAS Institute Inc. 2013 *SAS® 9.2 Companion for Windows* Cary, NC: SAS Institute Inc.

Shu, Haibin 2006 "Highly Effective Batch Processing" Proceedings of the PharmaSUG 2006 Conference, paper AD09.

Sun, Helen, Wong Cindy 2010 "A Macro to Create a Batch Submit SAS Program" Proceedings of the 2010 SAS Global Forum, paper 092.

RECOMMENDED READING

Cogswell, Denis. 2005 "More Than Batch – A Production SAS® Framework" Proceedings of the Thirtieth Annual SAS Users Group International Conference, paper 21.

Furdal, Stanislaw. 2008 "Quick Windows Batches to Control SAS® Programs Running Under Windows and UNIX" Proceedings of the SAS Global Forum 2008, paper 17.

Andrew E. Hansen. 2013 "A Macro to Batch Submit a List of Programs with Real Time Feedback" PharmaSUG 2013, paper AD15

ACKNOLEDGEMENTS

The author would like to thank Jianrong Li, Ping-Chung Chang and Xianyi Kong for their insightful comments in reviewing an earlier publication of this paper.

DISCLAIMER

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

CONTACT INFORMATION

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

Name: Huashan Huo
Enterprise: PPD Inc.
Address: 25th Floor, Raffles City Beijing Office
Tower No.1 Dongzhimen South Street,
Dongcheng District, Beijing 100007, China
Work Phone: +86 10-61846092
Fax: +86 10-61846099
E-mail: Huashan.Huo@ppdi.com
Web: www.ppdi.com

Name: Fanyu Li
Enterprise: PPD Inc.
Address: 25th Floor, Raffles City Beijing Office
Tower No.1 Dongzhimen South Street,
Dongcheng District, Beijing 100007, China
Work Phone: +86 10-61846118
Fax: +86 10-61846099
E-mail: Fanyu.Li@ppdi.com
Web: www.ppdi.com

Appendix:

Parallel batch submission brief code:

- Define the hierarchy and level, create macro variable to each programs.

```
data _null_;
   set jobs end=last;
   by level;
   retain totalLevels 0 JobTotalOfLevel 0;
   if last.level then totalLevels=totalLevels+1;
   if first.level then JobTotalOfLevel=0;
   JobTotalOfLevel=JobTotalOfLevel+1;
   call symputx('jobNameOfLevel'||strip(put(level,8.))
                ||'N'||strip(put(JobTotalOfLevel,8.))
               ,scan(CodeName,1,'.'));
   if last.level then do;
      call symputx('JobTotalOfLevel'||strip(put(level,8.)),
                    put(JobTotalOfLevel,8.));
      call symputx('L'||strip(put(totalLevels,8.)),strip(put(level,8.)));
   end;
   if last then call symputx('totalLevels',put(totalLevels,8.));
   call symputx('jobNameOfAll'||strip(put(_n_,8.))
               ,scan(CodeName,1,'.'));
   call symputx('totalJobs',put(_n_,8.));
run;
```

- Submit the code according to the hierarchy and level sequence

```
%do level = 1 %to &totalLevels.;    /*all level loop: start*/
   %let curNumofLevel=0;

   data _null_;
     call sleep(10,0.01);
   run;

   %do %while (&curNumofLevel<&&&&JobTotalOfLevel&&L&level);/*1 level loop*/
      %do j = 1 %to &sessionNum; /* loop by session number */
         %if &curNumofLevel<&&&&JobTotalOfLevel&&L&level %then %do;
            /* Check the job running status : start*/
            /* If found any session with error then exit loop */
            %do s = 1 %to &sessionNum;
               %if &&done&s ne 0 and &&done&s ne 1 and &&done&s ne %then %do;
               /* >=2: SAS issued error or ABORT*/
                  /* jump out level loop*/
                  %put Batch SAS job &&running&s issued error or aborted!;
                  /* save which job failed */

                  %if %upcase(&onErrorExit.)=Y or %upcase(&onErrorExit.)=YES
                  %then %do;
                     /* Kill tasks */
                     SYSTASK KILL "&&running&s";
                     /* Jump out of level loop */
                     %goto exitLevelLoop;
                  %end;
               %end;
            %end;
            /* Check the job running status: end */

            /*-------find success session then submit code <<<<<<<<*/
            /* all steps success: 0, with warning: 1, null means new session */
            %if %trim(&&done&j) ne %str() %then %do;
               %if &&done&j=0 and %trim(&&running&j) ne %str()
```

```
                  or &&done&j=1 and %trim(&&running&j) ne %str() %then %do;
                  /*Add program to record running status*/
               %end;

               %runOneJob; /* Submit a new program to run */
            %end;
            /*-------find success session then submit code >>>>>>>>*/
         %end;
      %end; /* loop by session number */
   %end; /* 1 level loop */

   /* wait for all jobs of a level complete or timeout */
   WAITFOR _all_
      %do w = 1 %to &&&&JobTotalOfLevel&&L&level;
         "&&&&jobNameOfLevel&&L&level..N&w"
      %end;
      TIMEOUT=&timeout.
      /* SUPPOSE: one level job will not cost &timeout. seconds */
   ;
   /* if one level failed, jump out of level loop */
   %if &SYSRC ne 0 %then %do;
   /* Here &SYSRC is return value of WAITFOR statement, ne 0 means timeout */
      %put Batch SAS job (Level: &&L&level) timed out.;
      SYSTASK KILL
         %do f = 1 %to &&&&JobTotalOfLevel&&L&level;
            "&&&&jobNameOfLevel&&L&level..N&f"
         %end;
      ;
      %goto exitLevelLoop;
   %end;
%end; /* all level loop: end */
```