SESUG Paper 251-2018

Life in the Fast Lane: SAS® Macro Language with Parallel Processing

Merry G. Rabb, RTI International

ABSTRACT

The SAS® macro language is widely used for implementing programs that are reusable, flexible and easily repeatable. When moving to a SAS Grid you gain the ability to run independent tasks within your SAS program in parallel in order to shorten run time. If those programs are built around SAS macros used as code modules and SAS macro variables used as parameters to control execution, you may need to change or restructure your code to duplicate the program logic in a distributed environment. When using macro programming in conjunction with RSUBMIT blocks for parallel processing, the code inside a RSUBMIT block is executed in a remote session, but macro code can be compiled in the main SAS program task. If a block of code that you plan to submit to a remote session contains macro calls, macro programming logic or even just macro variable references, some thought needs to be given to how and when the macro processing should occur. This paper looks at methods for incorporating SAS macro programming logic when using parallel processing.

INTRODUCTION

In 2016 RTI International migrated over 400 SAS users to a grid, which provided an opportunity to improve the performance of some of our long-running jobs with parallel processing. SAS jobs often contain independent tasks that can be split up and distributed across multiple nodes of a SAS grid to run in parallel, which can often result in a reduction in total elapsed time for the job. When converting an existing program to take advantage of parallel processing there are a number of issues to deal with, including how to divide up the processing, how many parallel sessions to have, how to pass information between parallel sessions, and how to make data available to all parallel tasks. Many of our programs contain SAS macro language elements including macro variables, and macro programs that use conditional or looping logic to generate SAS code. This paper will focus on the challenges that can arise when introducing remote processing into macro-driven programs.

MACRO VARIABLES

When moving a program into a distributed environment, some changes may be needed even if the code only contains macro variable references, not macro definitions or macro calls. Macro variables created in your main program (the client session) will not automatically be available in a remote session. Rather than recreate the macro variables in each remote session, you can use the %SYSLPUT statement to assign a value to a macro variable in a server session. This statement is executed in your main program and makes a macro variable or group of macro variables available in the remote session.

```
%let rc=%sysfunc(grdsvc_enable(_all_, resource=SASApp));
options autosignon;
%SYSLPUT state=MD/REMOTE=task1;
rsubmit task1 wait=no connectpersist=no inheritlib=(aq);
    proc freq data=aq.aqtest1(where=(DAYS_OZONE > 0 and state="&state"));
      tables DAYS_OZONE/missing;
      title "&state";
    run;
endrsubmit;
```

In the above example, the %SYSLPUT statement creates the macro variable STATE if it does not already exist, sets the value to MD, and makes it available in the server session. Note that the %SYSRPUT statement can be used to move macro variable values from the server session back to the client session, if needed.

RSUBMIT BLOCKS WITHIN A MACRO

Sometimes you want to use a macro to generate one or more RSUBMIT blocks, for example you may be generating code for one or more remote sessions with a %DO loop or using %IF logic. The RSUBMIT block might contain SAS statements, macro statements, or new macro definitions. SAS statements and macro definition statements will always be executed in the server session. However macro statements such as %DO, %IF,%LET and %PUT can be resolved and executed in the client session. Perhaps you intended the statements to execute in a server session rather than the client session. We will take a look at some examples of adding RSUBMIT blocks to a sample macro and techniques that can be used to ensure that the macro statements are processed where and when you intended.

SIMPLE LOOP EXAMPLE

Suppose you have a macro that loops through lists of information to generate several reports. This sample program using air quality at the Core Based Statistical Area (CBSA) level, and has several measures with variable names starting with the prefix DAYS_ such as DAYS_OZONE or DAYS_SO2. The macro runs the Frequency procedure on one variable for one state using two %DO loops and scanning parameter lists to select each state or variable requested. The original example program and output, before adding parallel processing, are shown below:

```
%macro aqrpt1(dataset=,state=,source=,limit=0);
  %do i=1 %to 2;
    %do j=1 %to 2;
      %let st=%scan(&state, &i);
      %let src=%scan(&source, &j);
      %let var=DAYS_&src;
      proc freq data=&dataset(where=(&var > &limit and state="&st"));
        tables &var/missing;
        title "&st &var";
      run;
    %end; /* do j */
  %end; /* do i */
%mend aqrpt1;
```

The following macro call will produce four tables of output: two tables per state, one each for the variable DAYS_OZONE and DAYS_PM2_5 (see Figure 1)

```
%aqrpt1(dataset=aq.aqtest1, state=%str(MD PA), source=%str(OZONE PM2_5),
limit=0)
```

MD DAYS_OZONE

Days_Ozone	Frequency
26	1
30	1
74	1
77	1

MD DAYS_PM2_5

Days_PM2_5	Frequency
13	1
16	1
50	1
63	1

PA DAYS_OZONE

Days_Ozone	Frequency
27	1
40	1
53	1
54	1
57	1
59	2

PA DAYS_PM2_5

Days_PM2_5	Frequency
36	1
48	1
49	1

Figure 1

Figure 1 Frequency procedure output from aqrtp1 macro.

If we want to introduce parallel processing and submit each frequency procedure in a separate server session, we might modify the program as follows, with the RSUBMIT block surrounding the SAS code inside the inner (%DO J=) loop:

```
libname aq "/data/mrabb";
%let rc=%sysfunc(grdsvc_enable(_all_, resource=SASApp));
options autosignon;

%macro aqrpt1g(dataset=,state=,source=,limit=0);
   %do i=1 %to 2;
     %do j=1 %to 2;
       %SYSLPUT _LOCAL_/REMOTE=task&j&i;
       rsubmit task&j&i wait=no connectpersist=no inheritlib=(aq);
       %let st=%scan(&state, &i);
       %let src=%scan(&source, &j);
       %let var=DAYS_&src;
      %PUT ******* state=&st var=&var **************;
        proc freq data=&dataset(where=(&var > &limit and state="&st"));
          tables &var/missing;
          title "&st &var";
        run;
       endrsubmit;
     %end;
   %end;
   waitfor _all_;
   signoff _all_ ;
 %mend;
```

Calling this macro does produce output so at first you might think it worked. However it only produces the first three tables. If you look at the log you'll see that in the first iteration of the loop, the macro variables ST, VAR and J are not found. Each subsequent time through the loop the values of those macro variables lag behind one iteration from what you might expect. In the partial log shown below, the %PUT statement output displays the expected values for the macro variables for the loop where J=2 (VAR=DAYS_PM_2_5) and I=1 (state=MD) but the submitted code is using the previous value of VAR (DAYS_OZONE) from the loop where J was equal to 1.

```
NOTE: Remote submit to TASK21 commencing.
******* state=MD var=DAYS_PM2_5 **************
NOTE: There were 4 observations read from the data set AQ.AQTEST1.
      WHERE (DAYS_OZONE>0) and (state='MD');
NOTE: The PROCEDURE FREQ printed page 1.
```

Why does this happen? When you run a program that uses an RSUBMIT statement inside a macro, the local macro processor on the client where you submitted the program processes the code within your RSUBMIT block before sending the processed block to the remote server to run. Because of the timing it appears that the macro processing is ahead of the SAS code processing.

We want the %LET statements to execute in the remote session. There are a couple of ways to approach this problem. One way would be to delay their execution with a macro quoting function. You can use the %NRSTR function to "hide" the %LET statements from the client session macro processor and allow them to be submitted remotely. Change the % at the beginning of the macro programming statement found inside the RSUBMIT block to %NRSTR(%%) as shown below:

```
rsubmit task&j&i wait=no connectpersist=no inheritlib=(aq);
%nrstr(%%)let st=%scan(&state, &i);
%nrstr(%%)let src=%scan(&source, &j);
%nrstr(%%)let var=DAYS_&src
%nrstr(%%)PUT ******* state=&st var=&var **************;
```

Another approach would be to replace the macro statements with data step logic. In this case we could replace the %LET statements in the server session with a DATA _NULL_ step and create the macro variables using CALL SYMPUT. For example replace the %LET statements in the sample program with:

```
data _null_;
    length state $2 var $12;
    state=scan("&state",&i);
    var=cats('DAYS_', scan("&source",&j) );
    call symput('var',var);
    call symput('st',state);
run;
```

COMPLEX DOUBLE LOOP EXAMPLE

It may actually be less efficient to run each procedure in its own remote session than it would be to group the PROCs into fewer sessions. You may decide to only generate two remote sessions, each with two PROC FREQs, by submitting one remote session for each outer loop (%DO I) and then the inner loop has the "%DO J" SAS code inside of it. If you try to accomplish this by moving the location of the RSUBMIT block outside of the %DO J loop, as shown below, the %DO J loop generates code, but the code doesn't work in the remote session.

By using the macro options MLOGIC and MPRINT we can tell that the %DO loop executes and sends code to the remote session, but by the time that code executes the macro variable J created by the %DO loop is no longer available. This causes the %LET for the macro variable SRC to fail because the %SCAN function doesn't have a valid second argument. Here is example code and partial log to illustrate this:

```
%macro aqrpt3(dataset=,state=,source=,limit=0);
   %do i=1 %to 2;
      %SYSLPUT _LOCAL_/REMOTE=task&i;
      rsubmit task&i wait=no connectpersist=no inheritlib=(aq);
      %do j=1 %to 2;
         %nrstr(%%)let st=%scan(&state, &i);
         %nrstr(%%)let src=%scan(&source, &j);
         %nrstr(%%)let var=DAYS_&src;
          %nrstr(%%)PUT ******* state=&st var=&var **************;
         proc freq data=&dataset(where=(&var > &limit and state="&st"));
           tables &var/missing;
           title "&st &var";
         run;
      %end; /* do j */
      endrsubmit;
   %end;  /* do i */
   waitfor _all_;
   signoff _all_ ;
 %mend;

%aqrpt3(dataset=aq.aqtest1, state=%str(MD PA), source=%str(OZONE PM2_5),
limit=0)
```

```
MLOGIC(AQRPT3):  %DO loop index variable J is now 3; loop will not iterate again.
MPRINT(AQRPT3):   endrsubmit;
1     %let st=%scan(&state, &i);
2     %let src=%scan(&source, &j);
WARNING: Apparent symbolic reference J not resolved.
ERROR: A character operand was found in the %EVAL function or %IF condition where a numeric
operand is required. The condition was: &j
ERROR: Argument 2 to macro function %SCAN is not a number.
3     %let var=DAYS_&src;
4     %PUT ******* state=&st var=&var **************;
******* state=MD var=DAYS_ **************
```

We know we can use %NRSTR to delay a macro statement inside of RSUBMIT block such that it executes on the server. However if we try that technique here, by adding %NRSTR to the %DO J= and %END statements, then as soon as the %DO J statement inside the RSUBMIT block is encountered we get the error message that “The %DO statement is not valid in open code.”

To solve this, make the code for the “DO J” loop into a separate macro. You will need to make the new macro available within the RSUBMIT block so that the macro can be defined & called remotely. There are at least three ways to accomplish this. First, you can embed the macro definition inside the RBUSMIT block. Second, you can compile and store the compiled macro, then reference it in the program as an AUTOCALL macro. Third, you can %INCLUDE the macro definition in the remote session. The example below depicts a solution using this last approach :

```
 /* new %DO J macro – saved as aqrpt3j.inc */
%macro aqrpt3j(dataset=,state=,source=,limit=0);
   %do j=1 %to 2;
     %let st=%scan(&state, &i);
     %let src=%scan(&source, &j);
     %let var=DAYS_&src;
     %PUT ******* state=&st var=&var **************;
      proc freq data=&dataset(where=(&var > &limit and state="&st"));
        tables &var/missing;
      title "&st &var";
      run;
   %end;
 %mend;

 /* Calling program, defines and calls outer */
 /* macro which calls the %agrpt3j macro     */

libname aq "/data/mrabb";
 %let rc=%sysfunc(grdsvc_enable(_all_, resource=SASApp));
 options autosignon;

 %macro aqrpt3;
   %do i=1 %to 2;
     %SYSLPUT _LOCAL_/REMOTE=task&i;
     rsubmit task&i wait=no connectpersist=no inheritlib=(aq);
     %include "/programs/aqrpt3j.inc;
     %aqrpt3j(dataset=aq.aqtest1, state=%str(MD PA),
              source=%str(OZONE PM2_5), limit=0) ;
    endrsubmit;
   %end;
   waitfor _all_;
   signoff _all_ ;
 %mend;

 %aqrpt3
```

CONCLUSION

When you introduce remote processing into a program that uses macro logic to generate SAS code, macro statements that you intend to resolve and execute in a server session might execute in the client sessions instead. The timing of SAS word scanner processing in a distributed environment is complex. Incorporate remote processing with care, use options such as MPRINT and MLOGIC to track what is actually happening, and be familiar with the different macro techniques you can use to control when execution occurs.

REFERENCES

SAS Customer Support "Interaction between Compute Services and Macro Processing"
http://support.sas.com/documentation/cdl/en/connref/61908/HTML/default/viewer.htm#a001584568.htm

Haigh, Doug 2016, "Divide and Conquer—Writing Parallel SAS® Code to Speed Up Your SAS Program"

Proceedings of the SESUG 2016 Conference, Bethesda, Maryland. Available at:
http://analytics.ncsu.edu/sesug/2016/PA-265_Final_PDF.pdf

Rabb, Merry and Brown, Keith. 2017. "Parallel Processing in a SAS® Grid."

Proceedings of the SESUG 2017 Conference, Cary NC. Available at:

https://analytics.ncsu.edu/sesug/2017/SESUG2017_Paper-85_Final_PDF.pdf

CONTACT INFORMATION <HEADING 1>

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

Merry G. Rabb
RTI International
919-990-8343
mrabb@rti.org