Paper 059-2011

Run Your SAS® Job Faster: Parallel Processing with Desktops on LAN

Sijian Zhang, Robert Brown, Yu Zhang
University of Alabama at Birmingham, Birmingham, AL

ABSTRACT

We all want to run SAS jobs faster, especially the large ones. The common solution is to use more powerful machines, which are not cheap. However, in some circumstances, the similar effect can be achieved without adding any hardware. If a SAS job is dividable, and each section can be run independently; and on the local area network (LAN), other computers with SAS installed can be accessed with right user permission, then this parallel method can be applied to speed up the processing. This paper discusses some features of command psexec.exe, SAS job division, and load balancing. Through the experimental tests and a real case application, the key programming skills are explained, and the significant speed increase effect is illustrated.

KEYWORDS

Parallel processing, psexec.exe, SAS job division, LAN.

INTRODUCTION

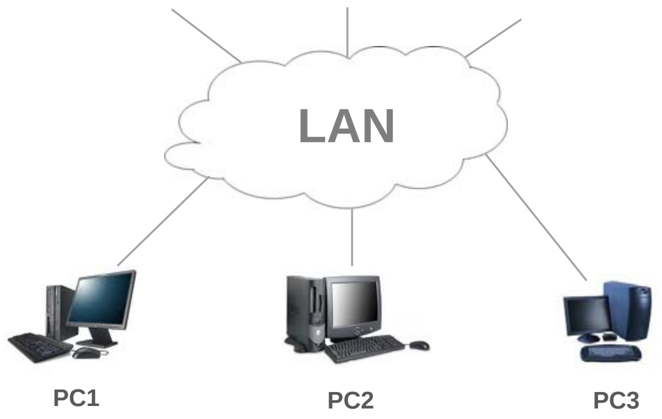
The essence of parallel processing is "divide and conquer". Generally speaking, more people will finish a job sooner than less people do. However, there are two assumptions for this statement to be true: 1) the job can be divided and done independently; and 2) the work is well organized. The same assumptions apply to the approach in this paper.

A popular setting in many institutes and companies is that the desktops or notebooks with SAS installed are connected to a LAN. This paper will show you a way that uses the command psexec.exe to communicate among the network computers, and to facilitate processing SAS jobs in parallel. After reading through this paper, you will find that the approach applied here is simple, effective and low cost. Given the computers with SAS installed and connected to a LAN, no new hardware and software need to be purchased. What 'low cost' means is that you need to download a free program (psexec.exe) from Microsoft website, setup right user permissions on the computers involved, and write the code.

Requirements:

Hardware:

The hardware requirement is simple: computers are connected to a LAN. It is a common setting in most of workplaces. In this paper, three desktops (PC1, PC2 and PC3) are used. Sijian Zhang's desktop PC1 is the master that initiates the execution and sends the requests to PC2 and PC3, which are located in different offices. The three desktops can access to the same network drive on our server.



	PC1	PC2	PC3
Model:	Dell OPTIPLEX 755	Dell OPTIPLEX GX620	Dell OPTIPLEX GX280
CPU:	Core™2 Duo, 3.0 GHz	Pentium® 4, 3.4 GHz	Pentium® 4, 3.2 GHz
RAM:	3.25 GB	3.0 GB	2.5 GB
OS:	Win XP Pro	Win XP Pro	Win XP Pro

Software:
1. SAS for Windows installed on each desktop.
2. psexec.exe program copied to the master computer only, which is PC1 in this paper.
   - Source: technet.microsoft.com/en-us/sysinternals/bb897553.aspx
   - Steps: download free PsTools; unzip it; and copy psexec.exe to a folder of a local drive. In this paper, it is copied to C:\PsTools on PC1.

Permission:
The programmer's user name needs to be listed on all desktops involved as the administrator. In this paper, Sijian Zhang is an administrator on PC1, PC2 and PC3.

PROGRAMMING AND TESTING

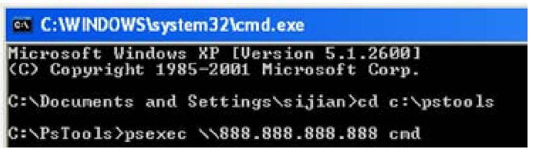
All coding work is done on PC1. Nothing needs done on PC2 and PC3 except setting up the administrator permission at the beginning. The wonderful thing of this programming is that during most of our testing runs, the users of PC2 and PC3 can still work on their daily tasks as usual. They do not know if a SAS process is running on the background, unless they are informed.

psexec.exe

This command plays a key role in our programming. Basically, it allows you to execute a program on a remote computer. It is one of the tools in PsTools that you can download from the Microsoft website. These tools, like other DOS commands, are run in DOS window, and usually used by network administrators or IT people. With the tools, they can operate at their workstations and do a lot of services to any other computers connected to the network on the background. As a terminal user, you have no idea what is going on, even you are using the computer, unless they tell you.

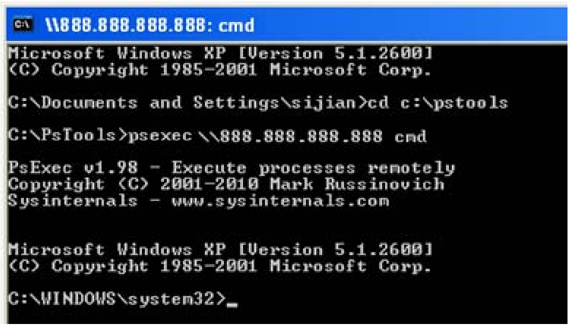
You can find the syntaxes and some brief application examples of psexec.exe in its download webpage. You will be amazed by its powerful functionalities. Here are the three ways how we use it during our programming and testing.

First way: Inexplicit logon
Syntax: psexec \\[target computer name or IP address] [command to execute on target computer]



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\sijian>cd c:\pstools

C:\PsTools>psexec \\888.888.888.888 cmd
```

(A)



```
\\888.888.888.888: cmd
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\sijian>cd c:\pstools

C:\PsTools>psexec \\888.888.888.888 cmd

PsExec v1.98 - Execute processes remotely
Copyright (C) 2001-2010 Mark Russinovich
Sysinternals - www.sysinternals.com


Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>_
```

(B)

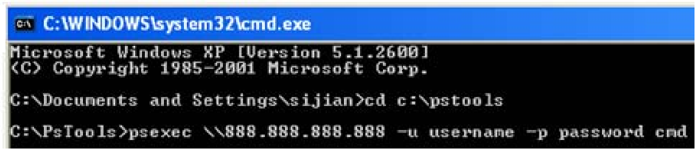
Fig 1

On PC1, when I click on "Start" on the lower left corner of the system window, select "Run" from the pop-up list, then type "cmd" in Run window and click "OK", a DOS window will be open as Fig 1 (A). If I want to do some work on PC2 via its DOS window, I can use PC2's name or IP address (888.888.888.888, a fake one for demonstration in this paper) as the target machine and "cmd" as the program to run on PC2, see the last command in Fig 1 (A). After hitting Enter, the DOS window will become Fig 1 (B). Now, I can operate from PC1 on PC2 as if I manually log on PC2 in front of it and then enter its DOS window. You can see in Fig 1 (B) that the window title has been changed to the target machine's IP address, and the version of psexec.exe is v1.98. In this process, PC2 user does not need to log off; actually, PC2 user is still doing some work on PC2 and not aware of it.

In the first and second way, the program to run should already exist on the remote machine. In the above case, cmd.exe is in the system path of PC2. Otherwise, the copy option (-c) should be used, like in the third way on next page.

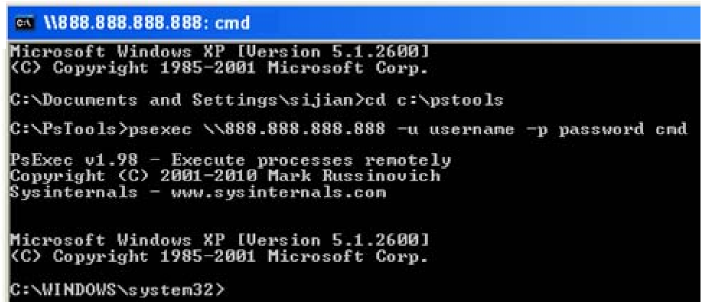
Second way: Explicit logon
Syntax: psexec \\[target computer name or IP address] –u [username] –p [password] [command to execute on target computer]

(A)



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\sijian>cd c:\pstools

C:\PsTools>psexec \\888.888.888.888 -u username -p password cmd
```

(B)

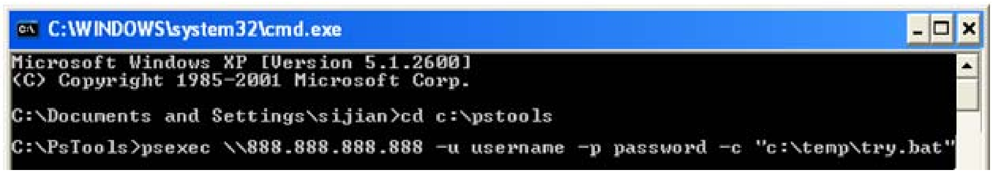


```
\\888.888.888.888: cmd
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\sijian>cd c:\pstools

C:\PsTools>psexec \\888.888.888.888 -u username -p password cmd

PsExec v1.98 - Execute processes remotely
Copyright (C) 2001-2010 Mark Russinovich
Sysinternals - www.sysinternals.com


Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>
```

Fig 2

The only difference from the first way is that the second way uses explicitly the user name and password to logon to PC2. The advantage of the second way is that it allows you to access to network resources from PC2, such as network drives.

Third way: Explicit logon, copy a program from PC1 to PC2, and run it on PC2.
Syntax: psexec \\[target computer name or IP address] –u [username] –p [password] –c [command to execute on target computer]



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\sijian>cd c:\pstools

C:\PsTools>psexec \\888.888.888.888 -u username -p password -c "c:\temp\try.bat"
```

Fig 3

In this way, PC1 user can copy a program, a batch file try.bat here for example, from PC1 c:\temp to the system path of PC2, run it on PC2, and remove it after execution. In other words, before and after the execution on PC2, the batch file try.bat does not exist on PC2. If some operations in try.bat need to access to a network drive, it can be done by using NET command in try.bat to do the mapping.

SAS Testing Code

1. Testing Core Unit
   In order to make the testing simple and easy to control, we set the following code as the core computation unit, and let the machines run it repeatedly by a macro loop. We can use the number of the macro loops to adjust the total computation load.

```
data Test;
    do i=1 to 300000;
        x=log(i);
    end;
run;
```

   This unit takes 0.01 seconds CPU time to run on PC1. We can control the length of time of this unit by changing the value of index variable "i" in the do-loop.

2. Load Balancing
   Since the three desktops are not the same, the computation load needs to be balanced among them to achieve the effect that they start and end at the same time. We did some tests and found out that it was close enough to get the ideal effect when the load ratio was PC1 : PC2 : PC3 = 49.6 : 24.6 : 25.8, which was then used to allocate the job load among the three desktops in all afterward tests. For example, if the total number of the loops is 100,000, by this ration, PC1, PC2 and PC3 will take 49,600, 24,600 and 25,800 loops to run respectively.

3. Working Code
   After load balancing, three working SAS programs are generated and saved on a network drive as Test_PC1.sas, Test_PC2.sas, and Test_PC3.sas, which will be run on PC1, PC2 and PC3 respectively. They look the same except the numbers of the loops that control the computation load.

   For example, if the total number of loops is 100,000, the working code for the three desktops:

Table 1

Desktop	Program to Run	Working Code
PC1	Test_PC1.sas	`%inc "M:\ ···\ParaTest.sas"; %paraTest(1,49600);`
PC2	Test_PC2.sas	`%inc "M:\ ···\ParaTest.sas"; %paraTest(49601,74200);`
PC3	Test_PC3.sas	`%inc "M:\ ···\ParaTest.sas"; %paraTest(74201,100000);`

   In “ParaTest.sas” (see Appendix 1), there is a macro %paraTest(m,n), which loops the testing core unit as many times as the parameters m (start number) and n (end number) indicate.

Dispatcher

Once the desktops and working code are ready, the next step is to signal the machines to run together. Since psexec.exe can execute only one program on the remote computer each time, the following two short programs are used for this purpose.

For example, we want to run the allocated job on PC2. The first one (Connect and Run PC2.bat) is to logon the remote desktop PC2 and start the second one (Run SAS from PC2.bat), which maps a server drive and starts running SAS program. There is only one statement in the first one:

```
c:\pstools\psexec \\888.888.888.888  -u username -p password -c "M:\··· \Run SAS
from PC2.bat"
```

You can convert “Connect and Run PC2.bat” to an .exe file using Windows Iexpress or other programs if there is a security concern about the username and password. As explained in the psexec.exe section, this code let PC1 logon to PC2, copy “Run SAS from PC2.bat” from the server drive M to PC2 and run it PC2.

The second program is to map to a network drive and start the SAS job. Two statements are in “Run SAS from PC2.bat”:

```
net use M: \\ ServerName\DirName
"C:\Program Files\SAS\SASFoundation\9.2\sas" -sysin "M:\ ··· \Test_PC2.sas" -log
"M:\ ··· \"
```

On PC2, the first statement maps M: to a network drive, then the second one starts SAS program to run “Test_PC2.sas” located on M: drive and assign a location to save the log file.

Testing

Now, it is the show time. The following code is launched on PC1.

```
options noxsync noxwait;
x call "C:\Program Files\SAS\SASFoundation\9.2\sas"
       -sysin "&path\Test_PC1.sas" -log "&path";
x call "&path\Connect and Run PC2.bat";
x call "&path\Connect and Run PC3.bat";
```

Option NOXSYNC tells SAS to process as soon as the command is issued. With NOXSYNC in effect, SAS executes an X statement and returns to current session to execute the next statement without waiting. This allows the three desktops to start the parallel processing in batch mode at the “same” time. Well, it feels about the same time while the time for SAS to issue the three X statements one and the time for network drive mapping are not noticeable.

Option NOXWAIT tells SAS that DOS window goes off automatically as soon as the process is finished.

In the testing phase, we started the total number of the macro loops from 10 to 600,000, 42 different testing loads were run on single desktop and on three ones in parallel. After each testing run, we checked the log files and recorded the start and end times. For each testing load, we have one pair of times (start and end, see the macro in Appendix 1) for PC1 (regular processing), and three pairs of times from the three desktops (parallel processing). Since all the three desktops' system times are synchronized with the server, we can use the difference between the latest end time and earliest start time as the parallel processing time.

Testing Results

Here are the results when we ran the tests from 10 to 600,000 total loops.

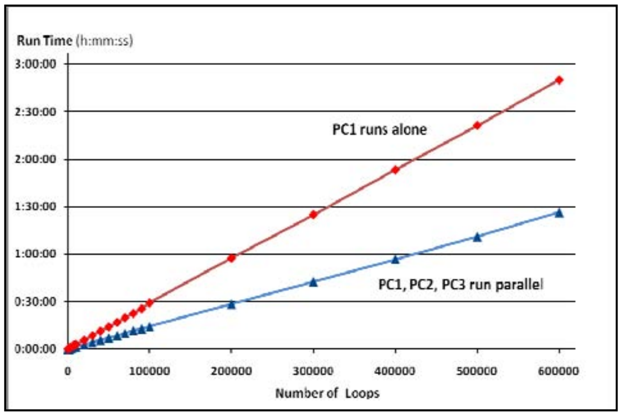


Fig 4

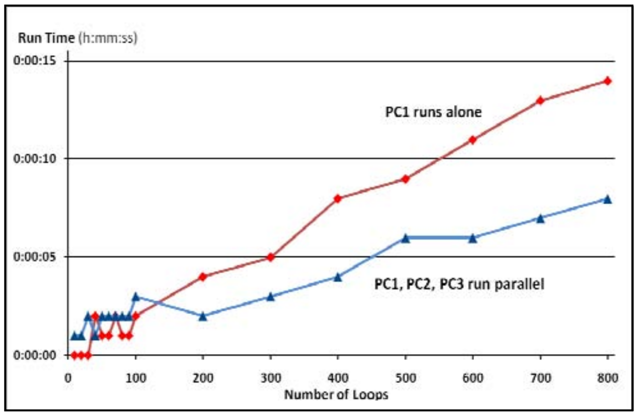


Fig 5

The result showed in Fig 4 is what we expected: the speed is about doubled by using parallel processing. Since the load ratio is PC1 : PC2 : PC3 = 49.6 : 24.6 : 25.8, the run time of three desktops should be about half time of PC1 running alone. If we launch the tests from either PC2 or PC3, we will see about four times speed increase by using parallel processing. This result is pretty stable as the number of loops is large. However, when the computation load is small, the overhead (computer status, network status, communication between computers, SAS program initiation, etc.) takes a significant portion of the total time. In our tests, when the total number of loops is less than 200, and the run time less than 4 seconds, one machine (PC1) runs the same or faster than three together, and the run time is not stable, see the lower left corner of Fig 5. In reality, if a program runs less than several minutes, no one will bother to use parallel processing.

A REAL CASE

We applied the above approach to the data preparation part of our routine medical event review report generation for INTERMACS® (Interagency Registry for Mechanically Assisted Circulatory Support). At the end of this part, the large datasets are subset into small datasets by Event ID, which will make the next report generation part more efficient, and data verification easier. In parallel processing, the same three desktops work together to slice two large datasets (patient_overview_para.sas7bdat and ae_all_para.sas7bdat) to many small data set.

To help visualize how this application is run, you can imagine that there is a super long hotdog (a dataset with a large number of observations) marked by the six-inch unit (Event ID), and three chefs (PC1, PC2, and PC3) are going to work together and cut it into six inch long pieces (small datasets by Event ID). First, the chefs' cutting speeds are tested, then they are given the workloads according to their cutting speeds. If the start and end units (Event IDs) are assigned properly, they can do their jobs independently, in the way that starting and finishing are at the same times. This is how this application is implemented.

To make the testing results comparable, we run the same code (see Appendix 2) in batch mode first by a single machine (PC1), and then by the three (PC1, PC2 and PC3) in parallel. After several tests, the load balancing ratio are set as PC1 : PC2 : PC3 = 35.2 : 32.5 : 32.3. There are 2,014 Event IDs in the current data. In Table 2 is how the job divided. The execution process is the same as the process in the testing section above.

Table 2

Desktop	Program to Run	Working Code
PC1	PC1.sas	`%inc "M:\ ···\ data_slicer_para.sas";` `%data_slicer(1,708);`
PC2	PC2.sas	`%inc "M:\ ···\ data_slicer_para.sas";` `%data_slicer(709,1363);`
PC3	PC3.sas	`%inc "M:\ ···\ data_slicer_para.sas";` `%data_slicer(1364,2014);`

The run time by PC1 only is 31 minutes and 40 seconds; by the three desktops in parallel 15 minutes 33 seconds. The parallel processing is more than double the speed in this application. As the number in the registry accumulated, the advantage of the parallel approach will be more obvious in the future.

DISCUSSION

What is the situation that can motivate us to try parallel processing? In my previous job, on one side, there was a large SAS job that automatically ran overnight to merge data and generate many reports; on the other side, about sixty desktops with SAS installed in the labs could be used when right user permissions were assigned. Sometimes, there were mistakes in the resulting datasets and reports due to the machine or network failure, or wrong parameters. The rerun may take at least half of a working day. If a parallel processing could be used, it would save a lot of waiting time for the new datasets and reports. Occasionally, I saw a note on a faculty member's computer screen for several days - "Don't disturb! Simulation is running.", or something like that. If you have the similar experiences, you may want to have a try.

Our LAN consists of a server (SunOS) and some desktops with Windows, which is a pretty popular setting. We did not test this approach in other settings. If your setting likes ours, and you want to try this approach, you should be prepared for the following issues.

a) Permission. To set a computer user as the administrator on all the computers involved sounds easy, but it took us many days to get it right. If you are not familiar with this, the help from your IT people will save you a lot of time.

b) Job Division. This is the primary condition for parallel processing. The largest undividable unit of the job determines the parallel processing run time. The real SAS jobs could be much more complex than the testing case in this paper. The programming of process control and coordination could be a challenge. How you make the division depends on both your understanding of the job and your computation resources available.

c) Computation Load Balancing. If all the computers in parallel processing are the same, the load balancing will be easy. But in reality, they are often not. First, you should make sure that all the machines are capable to handle their assigned workloads. Second, since the longest run time of the slowest machine plus overhead is the total parallel processing time, the load balancing is very important. Before the real run, you should test a small workload, and take some time to adjust its allocation among computers so that all the computers can start and finish processing approximately at the same time. Different kinds of jobs need to set different balance ratios even in the same system setting. In this paper, you can see that the ratio for testing is quite different from the ratio for the real case. The ratio of computation intensive job can be quite different from I/O intensive one. Our tests show that CPU speed is the determining factor on the computation intensive job, while BUS speed, RAM size and speed could have a significant impact on the job that has a lot of I/O operations.

d) Scalability. Running three desktops, we can easily check their running status and results by monitoring the individual machines and viewing the log files. However, if you are going to run three hundred, or three thousand computers in the same time, you will need an automated method to first detect the availability of the computers and check the status of each machine before starting the parallel job; then to balance the work load among the ones ready to run; and last to check if all the machines have done what they are assigned to do, and fix the problems if one or two machines fail to finish their assigned loads.

CONCLUSION

The hardware and software setting for the parallel processing approach described above is commonly available. To implement the approach, two things need to happen: to identify the right SAS jobs; and to write the code in the similar way of this paper. The obvious advantages of this approach are good feasibility and low cost. You can make use of your current system to significantly increase your computation power. Actually, this is a generic programming approach that can be modified and expanded to other programs rather than SAS. Next time, if someone asks how fast you can run a huge SAS job, given the assumptions and requirements met, you may say, "It will depend on the number of computers that I can use."

REFERENCES

1. http://technet.microsoft.com/en-us/sysinternals/bb897553.aspx (psexec.exe download, syntax instructions, and examples)

ACKNOWLEDGMENTS

We would like to thank M. Michelle Buchecker, SAS Regional Education Director, for her review of this paper and educational advices.

CONTACT INFORMATION

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

Sijian Zhang, MS, MBA
Database Analyst II
INTERMACS
Division of Cardiothoracic Research
Department of Surgery
University of Alabama at Birmingham
790 Lyons-Harrison Research Building
703 19th Street South
Birmingham, AL 35294
sijian@uab.edu

Appendix 1

```
%macro paraTest(m,n);
%let timeStart=%sysfunc(datetime(), datetime21.2);

options nosource nonotes;
%do m=&m %to &n;
data Test_Data_&n;
    do i=1 to 300000;
        x=log(i);
    end;
run;
%end;
options source notes;

%let timeEnd =%sysfunc(datetime(), datetime21.2);
%put Start time: &timeStart;
%put End time:   &timeEnd;
%mend paraTest;
```

Appendix 2

```
%macro data_slicer(m,n);
%let timeStart=%sysfunc(datetime(), datetime21.2);

libname ae "..."; libname mem_ae "...";
libname pt "..."; libname mem_pt "...";

data ae_all; set ae.ae_all_para; run;
data patient_overview; set pt.patient_overview_para; run;
data event_id_n; set mem_ae.event_id_n; run;

%macro AE(event_id);
data mem_ae.AE_&event_id;
      set ae_all(where=(event_id=&event_id));
run;
%mend AE;

%macro Pt(event_id);
data mem_pt.Pt_&event_id;
      set patient_overview(where=(event_id=&event_id));
run;
%mend Pt;

%macro run_AE;
data _null_;
      set event_id_n(firstobs=&m obs=&n);
      call execute ("%AE("||event_id||")");
run;
%mend run_AE;

%macro run_Pt;
data _null_;
      set event_id_n(firstobs=&m obs=&n);
      call execute ("%Pt("||event_id||")");
run;
%mend run_Pt;

options nosource nonotes;
%run_Pt;
%run_ae;
options source notes;

%let timeEnd =%sysfunc(datetime(), datetime21.2);
%put Start time: &timeStart;
%put End time:   &timeEnd;
%mend data_slicer;
```