PhUSE 2017

Poster PP06

SAS Performance Qualification for Clinical Programmers

Tatiana Volodina, Data MATRIX, Saint Petersburg, Russia
Andrey Myslivets, Data MATRIX, Saint Petersburg, Russia

ABSTRACT

The paper presents the program code along with the description of tools and methods that help SAS users to assess the system performance on their work stations and produce the required documentation on the results. Common tasks the performance of which is crucial in the clinical programmer's routine work are presented.

INTRODUCTION

When starting to use a new system, tool or software, certain qualification checks ought to be performed to ensure its correct work, such as:

- Design qualification (DQ);
- Installation qualification (IQ);
- Operational qualification (OQ);
- Process qualification (PQ);
- Performance qualification (P1Q).

SAS provides sufficient tools for performing most of these checks, such as SAS Installation Qualification Tool (SAS IQ) and the SAS Operational Qualification Tool (SAS OQ). These tools support the qualification aspect of the essential migration, integration, and verification processes users need to move from previous versions of SAS to later releases. However, performance qualification cannot be unified in the same way. It implies field-specified tests that check the system performance in real-world scenarios.

This poster presents a program code designed to run performance qualification tests that have been selected taking into account the specifics of clinical programming. Methods used to perform these tests are described, as well as the produced results.

The program code performs qualification tests that check calculations, procedures, methods, etc. that are used the most by a programmer in order to create statistical outputs for Clinical Study Reports (CSRs). The produced results show how much time the system uses for performing each task, so that the user can identify the system capability and its weak points in order to take corrective actions to improve it.

The presented program code can not only be used for performance qualification checks, but it presents an opportunity to compare the efficiency of different programming approaches and SAS procedures, which can be useful in everyday work.

TOOLS

SAS provides predefined macros that allow users to estimate the system performance, for example, number of input and output operations, memory usage, number of threads. In order to execute these macros the Application Response Measurement (ARM) interface should be used.

APPLICATION RESPONSE MEASUREMENT

Application Response Measurement (ARM) is an application programming interface which is used to monitor the availability and performance of software applications. The SAS ARM interface implements a number of features, which are compliant with the ARM 4.0 standards. SAS cooperates with open source ARM agents or vendor products that implement the ARM open API standard.

There are many techniques for measuring response times, but only ARM measures them accurately. Other techniques, although useful in other ways, might measure business service levels by assuming or guessing what a business transaction is, and when it begins and ends. Also, other techniques cannot provide the important information that ARM can, such as whether a transaction completed successfully.

ARM allows to:

- determine the application response times;
- determine the workload and throughput of your applications;
- verify that service-level objectives are being met;
- determine why the application is not available ;
- verify who is using an application;
- determine why a user is experiencing poor response time;

- determine what queries are being issued by an application;
- determine the subcomponents of an application's response time;
- determine which servers are being used;
- calculate the load time for data warehouses.

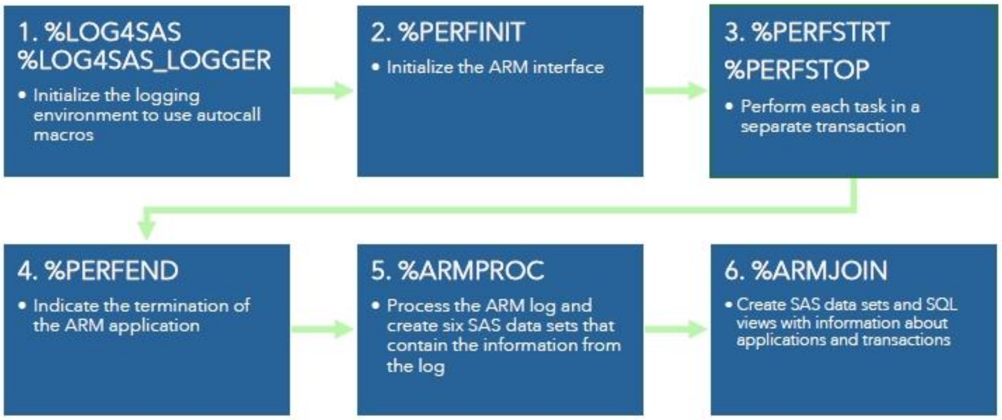
ARM is designed to be a high-speed interface that has minimal impact on applications. An ARM agent is designed to quickly extract the information that is needed and to return control to the application immediately. Processing of the information is done in a different process that can run when the application is otherwise idle.

The SAS ARM interface uses the following features to measure and log application availability, performance, usage, and transaction response time:

- ARM agent, which is an executable program that contains an implementation of the ARM API
- ARM appender, which processes ARM transaction events and sends the events to a specified output destination
- *ARM performance macros, which you strategically place in your SAS programs to define, start, and stop ARM data collection and contain default user metrics*
- ARM system options:
- ARMAGENT=, which specifies an executable module or keyword
- ARMLOC=, which specifies the location of an ARM log
- ARMSUBSYS=, which specifies whether to initialize the ARM subsystems
- default correlators, which are used to track parent and child transactions
- default user metrics, which are used to measure start, update, and stop times
- performance macros, which contain default user metrics
- SAS logging facility, which enables more flexibility and control of the ARM log destinations and message formats

PERFORMANCE QUALIFICATION STEPS

The picture below represents the steps that should be completed in order to run the performance qualification checks in SAS. Each step is described below in more detail.


1. %LOG4SAS
%LOG4SAS_LOGGER
• Initialize the logging environment to use autocall macros
2. %PERFINIT
• Initialize the ARM interface
3. %PERFSTRT
%PERFSTOP
• Perform each task in a separate transaction
4. %PERFEND
• Indicate the termination of the ARM application
5. %ARMPROC
• Process the ARM log and create six SAS data sets that contain the information from the log
6. %ARMJOIN
• Create SAS data sets and SQL views with information about applications and transactions

STEP 1. INITIALIZING THE LOGGING ENVIRONMENT

Initializing the logging facility for SAS programs is necessary if you use the logging facility autocall macros. This task is performed by invoking the %LOG4SAS macro as follows:

- in your autoexec file
- as a statement that is specified in the INITSTMT system option, which can be placed in the SAS configuration file or on the SAS command line

You can also invoke the %LOG4SAS autocall macro by placing it at the beginning of your SAS program as follows:

```
  %log4sas ();
```

The next step is to identify a logger. The macro %LOG4SAS_LOGGER is used for that purpose. In our code it is invoked with the following parameters:

```
  %log4sas_logger (PERF.ARM, 'level=info');
```

Here the name of the logger is specified as “PERF.ARM”. Therefore, ARM is the parent logger, PERF is the ancestor of the ARM logger.

The option “level=info” is used to specify the level “info” at which log events of the specified level and higher are processed by the logger. There are five different levels of the logging events, which, from the lowest to the highest, are as follows:

- TRACE: produces the most detailed information about your application. This level is primarily used by SAS Technical Support or development.
- DEBUG: produces detailed information that you use to debug your application. This level is primarily used by SAS Technical Support or development.
- INFO: provides information that highlights the progress of an application.
- WARN: provides messages that identify potentially harmful situations.
- ERROR: provides messages that indicate that errors have occurred. The application might continue to run.
- FATAL: provides messages that indicate that severe errors have occurred. These errors will probably cause the application to end.

The “info” level has been chosen because it provides the necessary information and does not provide too much unimportant details.

STEP 2. INITIALIZING THE ARM INTERFACE

After the logging environment has been initialized, the ARM interface can be accessed. The %PERFINIT macro names the application instance and initializes the ARM interface:

```
  %perfinit (appname = "Perf_App");
```

Here the name of the application is set as “PERF_APP”.

STEP 3. PERFORM THE TRANSACTION

After the first two steps have been completed, the ARM performance macros can be used.

Our goal is to evaluate the performance of different tasks separately. Therefore, the code should be separated into sections and for each section the performance qualifiers should be calculated. Such sections are called transactions.

In order to signal the start of a transaction, the %PERFSTRT macro is used:

```
  %perfstrt (txnname = "Tab_01_Power_SampleSize");
```

“TXTNAME=” sets the transaction name. The %PERFSTRT macro contains default used metrics which are described below in the corresponding section.

To signal the end of a transaction, the %PERFSTOP macro is invoked:

```
  %perfstop;
```

There can be any number of transactions and each of them can contain any number of datasets and procedures. Each transaction starts with %PERFSTRT and ends with %PERFSTOP.

The %PERFSTRT and %PERFSTOP macros can be nested in other %PERFSTRT and %PERFSTOP macros. When nested, each %PERFSTOP macro that is initiated is paired with the currently active %PERFSTRT macro.

STEP 4. TERMINATE THE APPLICATION

After all the required transactions have been run, the application should be terminated which is indicated by the %PERFEND macro.

```
  %PERFEND;
```

The %PERFEND means that the application does not issue any more ARM calls.

STEP 5. PROCESS THE ARM LOG

As all the performance calculations have been done, the next step is to obtain the results for each transaction. For this purpose the %ARMPROC macro is used:

```
  %armproc ();
```

The %ARMPROC macro creates six SAS data sets. These SAS data sets contain information from calls to the ARM:

- API function calls. The following lists the six SAS data sets:
- INIT—contains information from all ARM_INIT calls;
- GETID—contains information from all ARM_GETID calls;
- START—contains information from all ARM_START calls;
- UPDATE—contains information from all ARM_UPDATE calls;
- STOP—contains information from all ARM_STOP calls;
- END—contains information from all ARM_END calls.

STEP 6. INTERPRET THE RESULTS

The six SAS datasets that have been generated by the %ARMPROC macro are not easily understandable. In order to present the information in a better way and help to ease the results interpretation, the %ARMJOIN macro is invoked:

```
  %armjoin ();
```

It reads the six SAS data sets that are created by the %ARMPROC macro and merges the information from those data sets to create data sets and SAS views for easier reporting of ARM data.

The output is a single SAS library that contains the following:

- information about applications (App);
- a DATA step view that contains information about all start handles, including parent correlator class and parent start handles (TXNView);

- a SAS view that contains information about all update transactions (UPDTVIEW);
- one transaction data set for each application;
- one update data set for each application.

COMMON TASKS OF THE CLINICAL PROGRAMMER

SAS is widely used all over the world in many different industries, such as banking, big data, clinical research. It is important to assess the performance of those tasks that a programmer faces on day-to-day basis and that are a part of his/her routine work. Hence, the performance qualification is entirely field-specific and should be built solely on algorithms which are widely used in the specific area.

We have identified some of the most common tasks of the clinical programmer based on our experience in SAS and in clinical research as an example for the article. Each task is described below in detail.

TASK 1. SAMPLE SIZE ANALYSIS

The sample size analysis can be performed using different procedures, POWER, GLMPOWER and SEQDESIGN will be enough for most sample size estimation tasks. It can be time-consuming to execute these procedures in some cases, for example:

- the number of patients and/or groups is significant;
- the “contrast” statement is used in calculation;
- complicated family-wise error correction methods for adaptive design sample size estimation;
- big number of planned interim analyses

In our code we have indicated 2000 subjects per group and 2 groups, as follows:

```
proc power;
  twosamplemeans test = diff
   groupmeans = .85 | .90
   stddev = 0.5
   sides = 2
   npergroup = 2000
   power = .;
run;
```

TASK 2. CREATING A RANDOMIZATION LIST FOR 4000 SUBJECTS

Randomization list creation is a more complex task for which we have developed a macro that runs the PLAN procedure for each site separately and combines the data after. The part of the code that launches the PLAN procedure is presented below.

```
proc plan seed = &&seed&j;
  factors block = &bl. ordered treat = &tr. random / noprint;
  output out = data&j
  treat nvals = (
  …
  ) random;
run;
```

TASK 3. DELETING 10 DATASETS

After performing the previous task we have got 10 datasetes in the Work library named “dataX” where X is a number of the corresponding site. Now it is possible to assess the performance metrics for deleting these datasets:

```
proc datasets library = work nolist;
  delete Data:;
run;
```

TASK 4. CREATING A DATASET WITH SEVERAL VARIABLES FOR 4000 SUBJECTS

The next task is to create a dataset with one character variable (subjID) and two numeric variables containing random values:

```
data PQ_table (drop = i);
  call streaminit (2218); /* set random number seed */
  do i = 1 to 4000;
    subjID='R'||put(i, z5.);
    prm_1 = rand ("Uniform"); /* u ~ U[0,1] */
    prm_2 = rand ("Uniform") + rand("Uniform"); /* u ~ U[0,1] */
    output;
  end;
run;
```

TASK 5. MERGING DATASETS

Merging several datasets is one of the most common tasks while working with all kinds of data. So the next transaction in our code performs sorting and merging the datasets that we have previously created:

```
data PQ_analysis;
  merge PQ_table Rnd;
  by subjID;
run;
```

TASK 6. CALCULATING DESCRIPTIVE STATISTICS USING PROC MEANS

Being a statistical programmer, it is impossible to be unfamiliar with the descriptive statistics calculation. One of the procedures used for this purpose is MEANS, which we can launch on the dataset created before within out code:

```
proc means data = PQ_analysis;
  by treat site;
  output out = means;
run;
```

TASK 7. CALCULATING FREQUENCY STATISTICS USING PROC FREQ

Variables to be analyzed can be not only numeric, but categorical as well. To calculate frequencies and percentage on the specified variables the FREQ procedure is used:

```
proc freq data = PQ_analysis;
  tables treat * site;
run;
```

TASK 8. PERFORMING T TESTS USING PROC TTEST

As a part of the statistical analysis, different treatment groups ought to be compared to each other by various parameters, and there are numerous methods to accomplish that. One of the most basic methods is the t-test. In SAS the TTEST procedure performs t-tests for one sample, two samples and paired observations. We have chosen to run a single sample t-test within the P1Q program code:

```
proc ttest data = PQ_analysis;
  class treat;
  var prm_1;
run;
```

TASK 9. PERFORMING NONPARAMETRIC TESTS USING PROC NPAR1WAY

When the dependent variable is not considered as a normally distributed interval variable, non-parametric analogs to the independent samples t-test can be used. There are several procedures that allow to perform non-parametric tests in SAS, one of them is using the NPAR1WAY procedure:

```
proc npar1way data = PQ_analysis;
  class treat;
  var prm_1;
run;
```

TASK 10. PERFORMING ANCOVA WITH RANDOM FACTOR USING PROC MIXED

The next common task is to perform ANCOVA with a random factor. MIXED is a very powerful procedure for a wide variety of statistical analyses, therefore, it was our choice for this task:

```
proc mixed data = PQ_analysis;
  class treat;
  model prm_2 = treat prm_1;
  random site;
run;
```

Apart from MIXED, the GLIMMIX, GENMOD procedures can be tested using performance qualification tests. Other large areas include a wide range of regression analysis methods performed by different procedures, ROC curve diagnostics, different cluster and other classification analyses etc.

TASK 11. CREATING A TABLE USING PROC REPORT

Analyses results have to be output and reported properly in order to make them understandable and usable for clinical reports. The most functional procedure for presenting tables and listings is REPORT:

```
proc report data = means nowd;
  column treat _STAT_ prm_1 prm_2 site;
run;
```

Some companies use various macros of different complexity degrees to create statistical output. Sometimes they invoke external scripts (e.g. MS© Visual Basic Scripts) that can significantly affect the system performance.

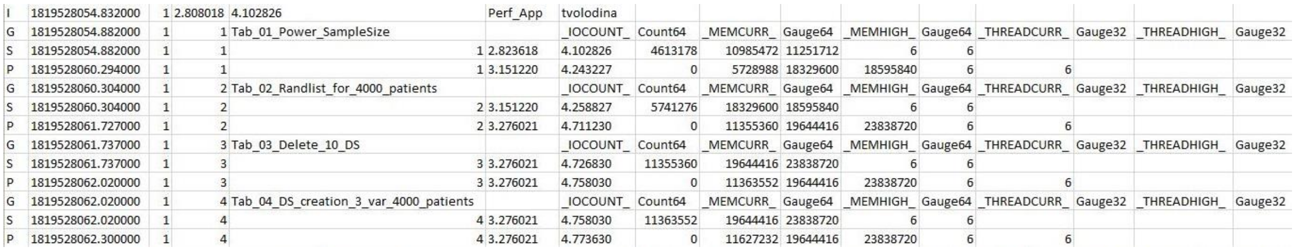
TASK 12. CREATING PLOTS USING PROC GPLOT

The data are presented not only in tables and listings in clinical reports, but it is also common to produce various plots. One way to achieve that is by using the SGPLOT procedure. We launch it from a macro that calculates the required parameters and creates several plots, the piece of code where SGPOT is used is presented below:

```
proc sgplot data = graph;
  xaxis grid values = &x_grid offsetmin = 0.05 offsetmax = 0.05 label = &x_label;
  yaxis grid label = &y_label;
  scatter x = jday y = mean / group = treat
                          yerrorlower = lower
                          yerrorupper = upper
                          markerattrs = (symbol = CircleFilled size = 5);
  series x = jday y = mean / group = treat
                         lineattrs = (pattern = 1);
run;
```

ARM LOG

While each performance check is carried out, the result is written into the log. The file with the ARM log is presented as a table on the picture below. Metrics for each test are presented.



I	1819528054.832000	1	2.808018	4.102826	Perf_App	tvolodina									
G	1819528054.882000	1	1	Tab_01_Power_SampleSize		_IOCOUNT_	Count64	_MEMCURR_	Gauge64	_MEMHIGH_	Gauge64	_THREADCURR_	Gauge32	_THREADHIGH_	Gauge32
S	1819528054.882000	1	1	1	2.823618	4.102826	4613178	10985472	11251712	6	6				
P	1819528060.294000	1	1	1	3.151220	4.243227	0	5728988	18329600	18595840	6	6			
G	1819528060.304000	1	2	Tab_02_Randlist_for_4000_patients		_IOCOUNT_	Count64	_MEMCURR_	Gauge64	_MEMHIGH_	Gauge64	_THREADCURR_	Gauge32	_THREADHIGH_	Gauge32
S	1819528060.304000	1	2	2	3.151220	4.258827	5741276	18329600	18595840	6	6				
P	1819528061.727000	1	2	2	3.276021	4.711230	0	11355360	19644416	23838720	6	6			
G	1819528061.737000	1	3	Tab_03_Delete_10_DS		_IOCOUNT_	Count64	_MEMCURR_	Gauge64	_MEMHIGH_	Gauge64	_THREADCURR_	Gauge32	_THREADHIGH_	Gauge32
S	1819528061.737000	1	3	3	3.276021	4.726830	11355360	19644416	23838720	6	6				
P	1819528062.020000	1	3	3	3.276021	4.758030	0	11363552	19644416	23838720	6	6			
G	1819528062.020000	1	4	Tab_04_DS_creation_3_var_4000_patients		_IOCOUNT_	Count64	_MEMCURR_	Gauge64	_MEMHIGH_	Gauge64	_THREADCURR_	Gauge32	_THREADHIGH_	Gauge32
S	1819528062.020000	1	4	4	3.276021	4.758030	11363552	19644416	23838720	6	6				
P	1819528062.300000	1	4	4	3.276021	4.773630	0	11627232	19644416	23838720	6	6			

However, even using the description of each metric, the file is not easy to understand and it is not a good practice to include it into documentation for regulatory, sponsor and other kinds of audits. For this reason, the %ARMPROC and %ARMJOIN macros are used. They create a more understandable output combining six SAS data sets that are created by the %ARMPROC macro to create data sets and SAS views for easier reporting of ARM data.

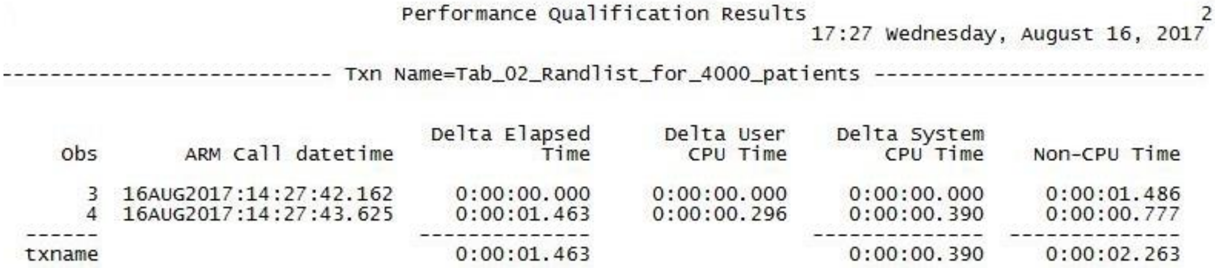
The output is a single SAS library that contains the following:

- The application data set (App) contains one observation for every application that is found in the input data. Each observation contains information such as application name, user ID, transaction counts, average application response time, and so on.
- A DATA step view that contains information about all start handles, including parent correlator class and parent start handles (TXNView). The transaction data sets are named TXN1, TXN2, TXN3, and so on. Each data set corresponds to a single application, and each observation represents a single ARM transaction containing start and stop times, elapsed times, and CPU time. The TXNView view joins all transaction data sets into a single data set.
- A SAS view that contains information about all update transactions (UPDTVIEW).
- One update data set for each application. The update data sets are named UPDT1, UPDT2, UPDT3, and so on. Each data set contains multiple observations for each ARM transaction. Each observation contains the ARM call datetime, an ARM call sequence ID, and, if applicable, elapsed time, CPU time, and update data. The UPDTView view joins all update data sets into a single data set.
- One transaction data set for each application which are easier to use for analyzing individual ARM transactions because all information about a transaction is represented in one observation. However, the transaction data sets do not contain any information from %ARMUPDT macros

The output of the %ARMPROC macro is presented on real life examples (tasks described above) in the Results section.

RESULTS

The %ARMPROC and %ARMJOIN macros produce the results that are usually used for writing P1Q documentation. Results are attached as appendixes to the P1Q documentation.

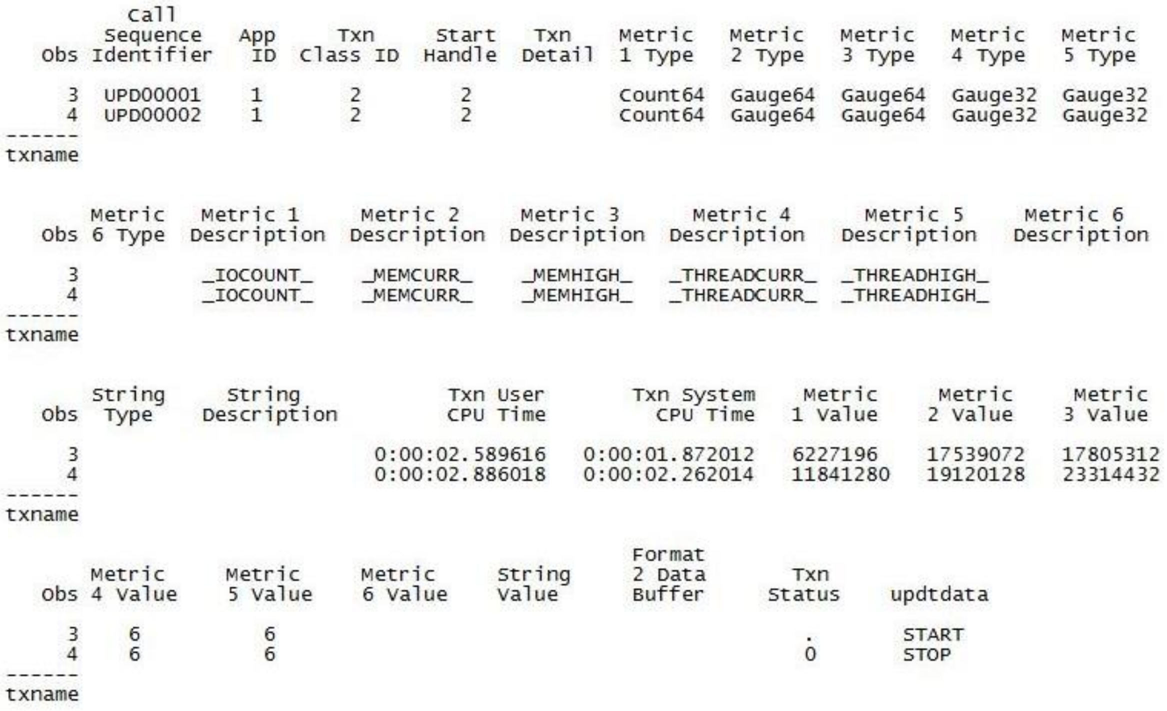


```
                                 Performance Qualification Results                            2
                                                          17:27 Wednesday, August 16, 2017

--------------------------- Txn Name=Tab_02_Randlist_for_4000_patients ---------------------------


                                     Delta Elapsed       Delta User     Delta System
   Obs        ARM Call datetime               Time         CPU Time         CPU Time     Non-CPU Time

     3  16AUG2017:14:27:42.162     0:00:00.000      0:00:00.000      0:00:00.000      0:00:01.486
     4  16AUG2017:14:27:43.625     0:00:01.463      0:00:00.296      0:00:00.390      0:00:00.777
------                          --------------                   --------------   --------------
txname                             0:00:01.463                      0:00:00.390      0:00:02.263
```



```
         Call
       Sequence    App     Txn     Start    Txn    Metric   Metric   Metric   Metric   Metric
   Obs Identifier   ID   Class ID  Handle  Detail  1 Type   2 Type   3 Type   4 Type   5 Type

     3  UPD00001    1        2        2            Count64  Gauge64  Gauge64  Gauge32  Gauge32
     4  UPD00002    1        2        2            Count64  Gauge64  Gauge64  Gauge32  Gauge32
------
txname


       Metric   Metric 1     Metric 2     Metric 3      Metric 4       Metric 5       Metric 6
   Obs 6 Type  Description  Description  Description  Description    Description    Description

     3         _IOCOUNT_    _MEMCURR_    _MEMHIGH_    _THREADCURR_   _THREADHIGH_
     4         _IOCOUNT_    _MEMCURR_    _MEMHIGH_    _THREADCURR_   _THREADHIGH_
------
txname


       String    String           Txn User       Txn System    Metric     Metric     Metric
   Obs  Type   Description        CPU Time         CPU Time    1 Value    2 Value    3 Value

     3                        0:00:02.589616  0:00:01.872012  6227196    17539072   17805312
     4                        0:00:02.886018  0:00:02.262014  11841280   19120128   23314432
------
txname

                                                 Format
       Metric     Metric     Metric     String   2 Data       Txn
   Obs 4 Value    5 Value    6 Value    Value    Buffer      Status    updtdata

     3   6          6                                         .         START
     4   6          6                                         0         STOP
------
txname
```

METRICS

Metric Name	Metric Type	Description
IOCOUNT	COUNT64	The metric value is the total number of disk, tape, or related input and output operations at each %PERFSTRT and %PERFSTOP event. The metric value is obtained from the host operating system and is associated with the input and output operations for that process. The value is a running count at the time of the event.
MEMCURR	GAUGE64	The metric value is the current value for memory used in the process at each %PERFSTRT and %PERFSTOP event. The metric value is obtained from the host operating system.
MEMHIGH	GAUGE64	The metric value is the highest amount of memory used for the life cycle of the current process at each ARM event. The metric value is obtained from the host operating system.
_THREADCURR	GAUGE32	The metric value is the current thread count of the process at each ARM event. The metric value is obtained from internal SAS counters.
_THREADHIGH	GAUGE32	The metric value is the highest number of active threads for the life cycle of the current process at each ARM event. The metric value is obtained from internal SAS counters.

POSSIBLE USES

Apart from checking SAS performance on different devices, the P1Q program code can be used to compare the performance of different SAS algorithms, especially in Big Data. Since SAS allows to perform the same tasks using different procedures and functions, it is important to know which methods and approaches to problem solving are more effective.

These checks can be performed in the P1Q program code by defining each approach or algorithm as one separate transaction. After running all transactions the P1Q results of different approaches should be compared between each other to identify the best one.

CONCLUSION

Companies where SAS is used should perform qualification checks after installing the system and maintain the proper documentation on them. Now these checks are conducted using SAS Installation Qualification Tool (SAS IQ) and the SAS Operational Qualification Tool (SAS OQ). Methods and tools described in this paper allow to conduct SAS performance qualification in addition to IQ and OQ and produce the appropriate documentation.

Moreover, these methods can be used to compare different techniques and procedures by various parameters, such as used memory, number of input and output operations, number of threads. This will help programmers to choose the most efficient and less time-consuming algorithms for performing routine tasks.

CONTACT INFORMATION

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

Tatiana Volodina
Data MATRIX
Work Phone: +7 812 449 86 33
Fax: +7 812 449 86 35
Email: volodina.tatyana@gmail.com

Andrey Myslivets
Data MATRIX
Work Phone: +7 812 449 86 33 (ext. 4011)
Fax: +7 812 449 86 35
Email: andrewmyslivets@gmail.com

Data MATRIX