SCSUG 2016 - Paper 41

Take a SPA Day with the SAS® Performance Assessment (SPA): Baselining Software Performance Across Diverse Environments To Elucidate Performance Placement and Performance Drivers

Troy Martin Hughes

ABSTRACT

Software "performance" is often measured through *execution efficiency*—that is, program execution time, with *higher performing* software executing more rapidly than lower performing software. Intrinsic factors affecting software execution efficiency can include the use of efficient coding techniques and other software development best practices. Factors extrinsic to software that affect performance can include SAS® system options, configuration, add-on modules, third-party software, and hardware and network infrastructure. Variability in the data sets (e.g., data type, number of observations, number of variables) processed by SAS software also heavily drives execution time, and these combined and commingled factors complicate the comparison of one SAS environment or installation to another. Moreover, many SAS practitioners may work in only one or in a few SAS environments, giving them limited insight into how performance of *their* SAS environment compares to *other* SAS environments. The SAS Performance Assessment (SPA) survey, first described at SAS Global Forum 2016, is a theorized survey of the SAS community that could be used to collect performance metrics from diverse organizations having equally diverse SAS environments. By running standardized programs that manipulate standardized data sets, the relative performance of unique environments could be compared. Moreover, as the number and variability of the SAS installations of SPA participants continued to increase, the role that individual factors play in software performance would continue to be detangled and better understood. SPA would ultimately enable SAS users to identify not only how their specific SAS environment compared to other environments, but also what set of specific modifications could be implemented to increase SAS performance.

INTRODUCTION

The SAS Performance Assessment (SPA) is a theorized survey of the SAS community intended to identify, analyze, and categorize performance variability within and among SAS environments. By comparing performance metrics of standardized programs that process standardized data sets in disparate SAS environments, the impact of the environment on SAS performance (measured through software completion time) could be analyzed. Once data from sufficient and sufficiently diverse environments had been collected, the role that specific environmental attributes (e.g., SAS version, SAS add-on modules, third-party software, or other hardware or infrastructure) play in system performance could be more readily understood. Moreover, once a correlation between environmental attributes and performance is understood, users seeking to improve performance would be able to identify and quantify infrastructure improvements that would benefit performance. For example, an organization would be able to evaluate not only that its SAS environment ran at 40 percent of the median speed of other organizations, but also the infrastructure attributes of higher-performing SAS environments, after which it could weigh the return on investment (ROI) of making either hardware or software upgrades.

Any SAS environment is able to demonstrate performance variability *within* its environment through repeated measures analysis of performance metrics captured within SAS logs. However, without comparison of performance *among* disparate SAS environments, each SAS environment effectively operates within a vacuum—aware of its own performance (and variability thereof), yet unaware of how it compares to other environments. For example, consider an extract-transform-load (ETL) process that runs in 20 minutes on average with execution times ranging from 15 to 25 minutes. SAS practitioners might excitedly high-five each other whenever the process completes in 15 minutes—at the apex of the local performance spectrum—touting "lightning performance." However, a second SAS environment might run the same processes on the same data, yet complete in less than a minute on average, blowing the so-called "lightning performance" out of the water. Only by running identical code on identical data in disparate environments can the overall spectrum of performance be understood. SPA aims to facilitate this objective through inter-site comparison.

The author first introduced the SPA concept at SAS Global Forum 2016, and marketed it subsequently at SAS regional user conferences. Although SPA exists only conceptually at this point, future collaboration among the SAS user base—to include completion of a formalized survey by users across diverse SAS environments—would empower SAS users globally to understand, quantify, and maximize the performance of their respective SAS environments. SPA would for the first time in history answer the questions *Where does this SAS rank?* and *How do I make this SAS better?*

ASSESSING SYSTEM OPTIONS AND SYSTEM RESOURCES

SPA must fully describe any SAS environment to ensure as many environmental variables as possible are captured. For example, at the most basic level, SAS version (e.g., 9.4) must be recorded, as earlier versions lacked multithreading capabilities, and later versions understandably implement other processing efficiencies.

System resources themselves can be managed (and limited) through SAS system options. The CPUCOUNT option describes the number of CPUs available to the SAS session, and can be varied to improve performance where additional CPUs are available. CPUCOUNT is demonstrated using the following code:

```
%let cpucount=%sysfunc(getoption(cpucount));
%put &=cpucount;
```

Using the default SAS configuration (that ships with *my* SAS installation), the log demonstrates that only four CPUs are available:

```
CPUCOUNT=4
```

However, by specifying CPUCOUNT=ACTUAL in the OPTIONS statement, the number of CPUs utilized by SAS is set to the maximum number that are available:

```
options cpucount=actual;
%let cpucount=%sysfunc(getoption(cpucount));
%put &=cpucount;
```

The code executes, and both notifies the user of the number of available CPUs (8), and instructs SAS to begin using all available CPU resources:

```
CPUCOUNT=8
```

It is impossible not to wonder how many SAS practitioners, teams, or organizations are unnecessarily limiting performance of their SAS software simply because they have failed to understand or maximize performance settings!

For the purposes of SPA, it is critical that these system option settings be captured (as demonstrated using the GETOPTION function). Other system options such as memory size (XMRLMEM) and sorting memory size (SORTSIZE) would need to be similarly captured:

```
%let memsize=%sysfunc(getoption(xmrlmem));
%put &=memsize;
%let sortsize=%sysfunc(getoption(sortsize));
%put &=sortsize;
```

Finally, for system resource metadata collection, it is imperative that the FULLSTIMER option be selected so that resource utilization can be best captured:

```
options fullstimer;
```

These cursory SAS options comprise some but not all of the environmental variables that SPA would need to collect. Other qualitative metadata would need to be captured, such as the operating system, hardware characteristics, and network characteristics, so that these could be compared among SAS installations to assess their respective impacts on performance.

STANDARDIZING DATA SETS FOR PERFORMANCE TESTING

In addition to executing the same code across different SAS environments, a performance assessment also must process the same data; only by standardizing both code and data can performance be baselined and compared. Many SAS white papers and much technical documentation rely on SAS built-in data sets that ship (or download) with SAS installations, and which are maintained in the timeless SASHELP library. By referencing SASHELP data sets, instructors and authors are assured that any SAS user has access to the referenced data set. However, performance testing, including load testing and stress testing that aim to evaluate how SAS performs when facing increasing larger data sets, requires both larger data sets and data sets that are incrementally (and precisely) varied in size and/or structure.

Thus, as SPA cannot rely on SASHELP data sets for performance testing, and as downloading hundreds or thousands of data sets from some Internet source would be horribly inefficient and ineffective, SPA must instead rely on data sets created by code and maintained as part of the SPA survey itself. In so doing, any SPA participant could run standardized code, create standardized data sets, and be ensured that those data are identical to the data sets being generated by all other SPA participants.

For example, the following one-billion observation (7.9GB) data set can be created programmatically to ensure that different SAS environments each utilize the same data:

```
data one_billion (drop=i);
    length num1 8;
    call streaminit(123);
    do i=1 to 1000000000;
       num1=int(rand('uniform')*100000);
       output;
       end;
run;
```

Where the WORK library is used to maintain data sets utilized by SPA, the only consideration becomes a preliminary assessment that the logical drive maintaining the WORK library has sufficient storage capacity to process SPA programs. Thereafter, all SPA-related data sets would be automatically deleted when the SAS system terminates and dumps WORK contents.

ISOLATING PERFORMANCE METRICS

SPA aims to analyze and compare software *execution efficiency*—not *efficiency*—within and across environments, where *execution efficiency* measures runtime, whereas *efficiency* measures runtime with respect to system resource utilization. Thus, although input/output (I/O)-intensive, CPU-intensive, and memory-intensive operations are distinguished, execution time is the only software performance metric that SPA measures. By making subtle variations to code, the effect of system resource utilization on software performance can be analyzed independently. For example, SPA results might demonstrate that a specific environment performs I/O functions faster than average (when compared to other SAS users that have participated in SPA) but processes data slower. Further analysis could elucidate environments that are CPU-bound or I/O-bound respective to other environments.

The most commonly referenced SAS performance metrics are *real time* (i.e., execution time) and *CPU time*. Real time measures actual time that elapses while software is executing. Because it can be observed by stakeholders or programmatically assessed by recording when a process starts and stops, real time is the construct included in contractual service level agreements (SLAs), and against which performance requirements are measured. Because real time can be affected by other SAS processes or unrelated programs that are executing concurrently on a system, it tends to be slower and more variable than CPU time. CPU time, conversely, measures only the effort that the processor expends to execute SAS processes. It is generally less than the execution time (excepting multithreaded processing), and is less variable than real time because it is less influenced by concurrently executing processes and programs. Thus, CPU time is typically more useful during testing, development, and optimization of software, whereas real time will be found in SLAs and used once software is in production.

The following DATA step ingests the One_billion data set, and copies the data into New_billion, comprising I/O read and write operations:

```
data new_billion;
    set one_billion;
run;
```

The log (with FULLSTIMER activated) demonstrates the performance, with CPU time representing the sum of *user CPU time* and *system CPU time*:

```
NOTE: There were 1000000000 observations read from the data set WORK.ONE_BILLION.
NOTE: The data set WORK.NEW_BILLION has 1000000000 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time           34.95 seconds
      user cpu time       27.32 seconds
      system cpu time     4.43 seconds
      memory              654.56k
      OS Memory           11756.00k
```

Thus, CPU time can be calculated to be 31.75 seconds (27.32 + 4.43). However, to isolate only the I/O *read* functionality (and remove the I/O *write* functionality), the following DATA step utilizes _NULL_ to instruct SAS not to write the ingested data to disk:

```
data _null_;
    set one_billion;
run;
```

The log demonstrates the increased performance when the One_billion data set is read from (but not re-written to) disk:

```
NOTE: There were 1000000000 observations read from the data set WORK.ONE_BILLION.
NOTE: DATA statement used (Total process time):
      real time           17.31 seconds
      user cpu time       15.56 seconds
      system cpu time     1.62 seconds
      memory              538.40k
      OS Memory           11756.00k
```

Because the data are not required to be rewritten in the second DATA step, the performance costs of reading and writing data can be roughly distinguished. For example, the CPU time required to read and write the data (31.75 seconds) was almost twice the CPU time required to only read the data set (17.18 seconds). From these results, you also might infer that approximately 54.11 percent of the CPU time is the cost of *reading* and 45.89 percent of the CPU is the cost of *writing*. *In truth, distinguishing the respective costs of these disparate yet intertwined operations is more complex than this simple example; however, analyses such as these begin the process of detangling operations.*

With the cost of I/O operations now captured, some additional functionality could be conceptualized, and could be utilized to identify and roughly distinguish computational effort from I/O effort. For example, consider the need to compute the average (mean) of the numeric variable (Num1) in the One_billion data set. Of course, this is better accomplished using built-in SAS procedures like MEANS; however, this straightforward computation can be readily understood and coded. The following DATA step now prints the average (mean) of all Num1 values:

```
data _null_;
    set one_billion end=eof;
   retain tot 0;
   tot + num1;
   if eof then do;
      tot = tot / _n_;
      put tot=;
```

```
      end;
run;
```

The log demonstrates not only the average value of Num1 but also DATA step resource utilization:

```
tot=49999.42505
NOTE: There were 1000000000 observations read from the data set WORK.ONE_BILLION.
NOTE: DATA statement used (Total process time):
      real time           26.21 seconds
      user cpu time       21.21 seconds
      system cpu time     1.76 seconds
      memory              541.93k
      OS Memory           16888.00k
```

Thus, as the equivalent read-only DATA step had used 17.18 seconds of CPU time, and the current DATA step used 22.97 seconds, the cost of calculating the average value can be inferred to take approximately 5.79 seconds.

One final DATA step now performs the same calculation, albeit also (unnecessarily) creating the New_billion data set:

```
data new_billion (drop=tot);
    set one_billion end=eof;
   retain tot 0;
   tot + num1;
   if eof then do;
      tot = tot / _n_;
      put tot=;
      end;
run;
```

The log demonstrates the DATA step incurred a CPU time of 102.18 seconds:

```
tot=49999.42505
NOTE: There were 1000000000 observations read from the data set WORK.ONE_BILLION.
NOTE: The data set WORK.NEW_BILLION has 1000000000 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time           2:19.45
      user cpu time       1:11.73
      system cpu time     30.45 seconds
      memory              631.21k
      OS Memory           16888.00k
```

This CPU time of 102.18 seconds is contrasted with the 31.75 seconds incurred when reading One_billion and writing New_billion (without computing the average). These results help demonstrate the complexity in detangling operations and operational performance where both I/O and computational processes are occurring. For example, these results demonstrate that the operational cost of some functionality varies with respect to whether SAS is writing a DATA set. For example, when only reading the One_billion data set, calculating the average of Num1 incurred only 5.79 seconds of CPU time, whereas when both reading and writing a data set, this same operation incurred 70.43 seconds (102.18 – 31.75).

Throughout this section, CPU times are represented with two-decimal precision, and although these may be accurate individual measures, they remain solitary data points that are susceptible to (sometimes significant) operational variability. In the next section, the critical requirement of repeated measures analysis is discussed and demonstrated, which does incur added SPA runtime, but with the benefit of delivering more consistent and stable performance metrics.

REPEATED MEASURES ANALYSIS

Isolated performance metrics can be dangerously misleading due to variability often experienced within an environment. As already mentioned, relying on CPU time (as opposed to real time) for performance evaluation is one best practice that supports more consistent performance measurement. Another best practice requires the evaluation of multiple data points that describe the same operation—that is, *repeated measures*—so that the average value (demonstrated here as the mean, but sometimes captured as the median where data can be skewed) can be calculated to stabilize performance metrics.

The following REPEATED_MEASURES macro now executes the read-only DATA step twenty times to assess what variability may exist when data are ingested:

```
%macro repeated_measures(iter= /* number of iterations */);
%do cnt=1 %to &iter;
data _null_;
    set one_billion;
run;
   %end;
%mend;

%repeated_measures(iter=20);
```

The first two runs in the log demonstrate some of the variability that can exist in both real time and CPU time, whereas memory utilization remains constant:

```
NOTE: There were 1000000000 observations read from the data set WORK.ONE_BILLION.
NOTE: DATA statement used (Total process time):
      real time           17.58 seconds
      user cpu time       15.86 seconds
      system cpu time     1.48 seconds
      memory              539.40k
      OS Memory           16888.00k


NOTE: There were 1000000000 observations read from the data set WORK.ONE_BILLION.
NOTE: DATA statement used (Total process time):
      real time           17.36 seconds
      user cpu time       15.73 seconds
      system cpu time     1.61 seconds
      memory              538.50k
      OS Memory           16888.00k
```

Even if it is established that the DATA step takes 17.30 seconds to run on average, there remain both fixed and variable CPU costs that must be identified and isolated. For example, what is the *fixed cost* of reading the first observation versus the *variable cost* of reading incrementally more observations? Similarly, what is the *fixed cost* of reading the first variable versus the *variable cost* of reading incrementally more variables? Stated another way, what is the fixed cost of reading *any* SAS data set versus the variable costs of reading data sets whose size in bytes grows incrementally larger? Only load testing, in which one or more data set attributes is incrementally modified, can begin to represent the complexity and interaction of fixed and variable performance costs.

LOAD TESTING WITH INCREMENTAL OBSERVATION COUNT

Load testing varies the *load*, which often represents the quantity of data being operationalized; it is disambiguated from *stress testing*, in which the load is increased until functional failure results. Load testing can incorporate numerous data attributes, but for demonstration purposes, only observation count is varied herein. Note that changing the observation count indirectly increases data set size, so both metrics should be captured. It is always advisable to capture all available metrics that describe and quantify the data set at each interval, as these metadata can be painful, if not impossible, to recreate or recapture.

The LOAD_TESTING macro builds and reads incrementally larger data sets, ranging from 100 million to one billion observations; thus, ten separate data sets are evaluated:

```
%macro load_testing(obs_min= /* min obs */,
   obs_max= /* max obs */,
   obs_iter= /* iterate obs by this value */);
%do obs=&obs_min %to &obs_max %by &obs_iter;
* create test data;
data test_data (drop=i);
    length num1 8;
    call streaminit(123);
    do i=1 to &obs;
       num1=int(rand('uniform')*100000);
       output;
       end;
run;
* evaluate file size;
proc sql noprint;
   select filesize format=15.
      into :filesize
         from dictionary.tables
   where libname="WORK" and memname="TEST_DATA";
quit;
* convert bytes to MB;
%let filesize=%sysevalf(&filesize/(1024**2),comma15.);
* read test data (the functionality being evaluated);
%put &=obs &=filesize;
data _null_;
    set test_data;
run;
   %end;
%mend;

%load_testing(obs_min=100000000, obs_max=1000000000, obs_iter=100000000);
```

The first DATA step creates the data set and the SQL procedure evaluates its size—both are necessary precursors, but neither are the focus of SPA. Rather, performance testing is focused on only the final DATA step that reads Test_data. The first iteration (of ten) of LOAD_TESTING is demonstrated in the log:

```
NOTE: The data set WORK.TEST_DATA has 100000000 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time           4.79 seconds
      user cpu time       2.89 seconds
      system cpu time     0.40 seconds
      memory              401.59k
      OS Memory           17660.00k

NOTE: PROCEDURE SQL used (Total process time):
      real time           0.00 seconds
      user cpu time       0.00 seconds
      system cpu time     0.00 seconds
      memory              5187.81k
      OS Memory           22784.00k

OBS=100000000 FILESIZE=775.25
```

```
NOTE: There were 100000000 observations read from the data set WORK.TEST_DATA.
NOTE: DATA statement used (Total process time):
      real time           2.50 seconds
      user cpu time       1.89 seconds
      system cpu time     0.15 seconds
      memory              538.40k
      OS Memory           17660.00k
```

It is through the combination of both repeated measures and load testing that reliable and consistent performance metrics can be generated and captured. Thus, the REPEATED_LOAD_TESTING macro combines this functionality and iterates over increasingly larger loads while pausing at each load to repeatedly sample it:

```
%macro repeated_load_testing(iter= /* number of iterations */,
   obs_min= /* min obs */,
   obs_max= /* max obs */,
   obs_iter= /* iterate obs by this value */);
%do obs=&obs_min %to &obs_max %by &obs_iter;
* create test data;
data test_data (drop=i);
    length num1 8;
    call streaminit(123);
    do i=1 to &obs;
       num1=int(rand('uniform')*100000);
       output;
       end;
run;
* evaluate file size;
proc sql noprint;
   select filesize format=15.
      into :filesize
         from dictionary.tables
   where libname="WORK" and memname="TEST_DATA";
quit;
* convert bytes to MB;
   %let filesize=%sysevalf(&filesize/(1024**2));
   %do cnt=1 %to &iter;
   * read test data (which is the functionality being evaluated);
      %put &=obs &=filesize &=cnt;
   data _null_;
    set test_data;
   run;
      %end;
   %end;
%mend;

%repeated_load_testing(iter=20, obs_min=100000000, obs_max=1000000000,
obs_iter=100000000)
```

Note that the outer loop iterates over the number of observations, whereas the inner loop iterates over the sample size of repeated measures. Repeated measures should always occupy the innermost loop, as this enables a data set to be created only one (and have its file size evaluated only once), after which it can be repeatedly accessed to evaluate the functionality of primary interest. It is for this reason that the first DATA step and SQL statement reside in the *outer* loop, whereas the DATA step that reads One_billion resides in the *inner* loop.

The truncated log represents only the first two (of 200) iterations that execute:

```
NOTE: The data set WORK.TEST_DATA has 100000000 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time           5.10 seconds
      user cpu time       2.81 seconds
      system cpu time     0.31 seconds
      memory              401.59k
      OS Memory           17660.00k

NOTE: PROCEDURE SQL used (Total process time):
      real time           0.00 seconds
      user cpu time       0.00 seconds
      system cpu time     0.00 seconds
      memory              5187.81k
      OS Memory           22784.00k

OBS=100000000 FILESIZE=775.25 CNT=1

NOTE: There were 100000000 observations read from the data set WORK.TEST_DATA.
NOTE: DATA statement used (Total process time):
      real time           2.76 seconds
      user cpu time       1.78 seconds
      system cpu time     0.23 seconds
      memory              538.40k
      OS Memory           17660.00k

OBS=100000000 FILESIZE=775.25 CNT=2

NOTE: There were 100000000 observations read from the data set WORK.TEST_DATA.
NOTE: DATA statement used (Total process time):
      real time           5.46 seconds
      user cpu time       2.79 seconds
      system cpu time     0.23 seconds
      memory              538.40k
      OS Memory           17660.00k
```

The first twenty iterations (only two of which are shown) represent repeated measures run on the 100-million-observation data set, after which the next twenty iterations (not shown) represent repeated measures run on the 200-million-observation data set. Note that as more attributes are incorporated into an analysis (such as number of variables, or variable length), the number of iterations grows exponentially, and can quickly surpass the capabilities of realistic assessment. In these cases, attribute sampling may be warranted in lieu of strict attribute iteration.

For example, if you also wanted to vary One_billion by dynamically creating between one and fifteen numeric variables—in addition to sampling observation count ten times between 100 million and one billion observations, and repeating each measurement 20 times—this would generate 3,000 data points (15 x 10 x 20). Representative sampling can substantially reduce the number of data points, and may be essential when the number and range of data set attributes surpasses the ability to collect all data points. After all, SPA is intended to be run for no more than a few hours on the SAS installation of a gracious host—not to monopolize a SAS instance for a month as it churns through quadrillions of covariate combinations.

PERFORMANCE MEASUREMENTS EVALUATION

By this point, it should be clear that the diversity of operations that SPA would need to run, inasmuch as the need to vary size (and potentially other attributes) of the data sets being manipulated by the operations (not to mention repeating those performance measurements), imbues a complexity prohibitive of manual log review. Performance metrics would

need to be automatically extracted through log parsing, and this could be done either by SPA participants (as part of the survey), or subsequently upon collection of SPA logs. The benefit of log parsing done by SPA participants would be the requirement that only the resultant metrics data set would be returned to SPA organizers, as opposed to raw log files.

Subsequent evaluation of performance metrics would enable individual programming operations to be represented for a single SAS installation. That is, the variability *within* an organization could be shown with respect to factors such as observation count and data set size. For example, Figure 1 demonstrates a box plot generated after the previous execution of the REPEATED_LOAD_TESTING macro, and shows the linear progression of CPU time as the number of observations is incrementally increased, as well as the subtle performance variability evident through repeated measures.

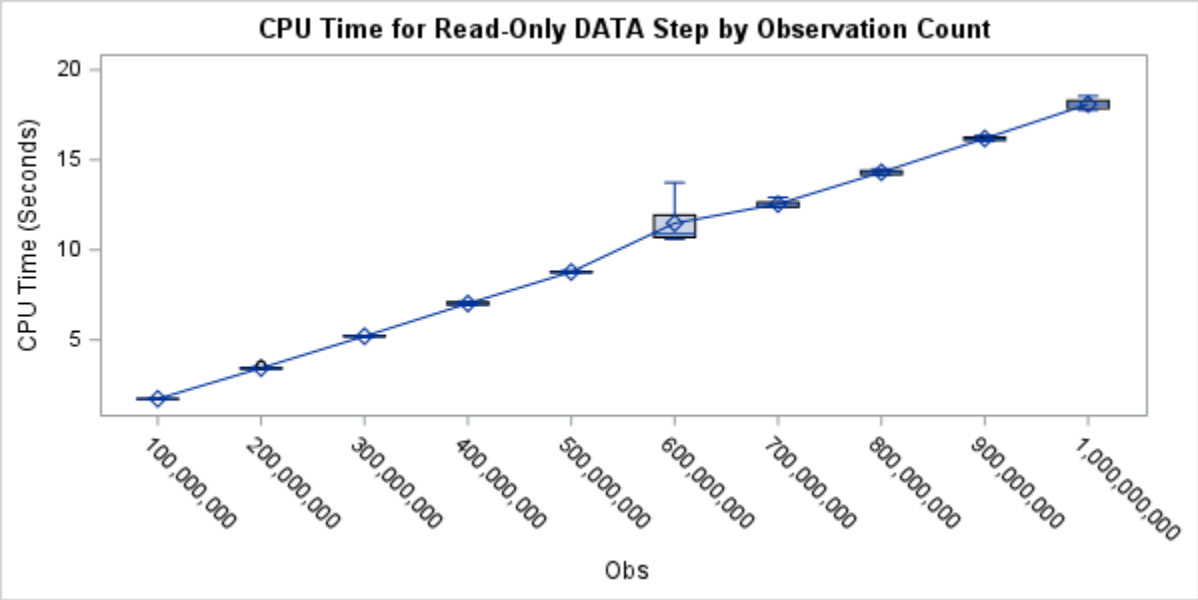

Figure 1. Results of REPEATED_LOAD_TESTING Macro with 200 Data Points

These metrics alone might prove valuable to an organization seeking to understand its SAS environment better—for example, how it handles reading, writing, and transforming multi-gigabyte data sets. However, the data do not yet provide insight into how this single SAS instance compares with the performance of other SAS installations elsewhere. For this insight, SPA would need to be performed across a swath of SAS environments.

PERFORMANCE MEASUREMENTS PLACEMENT

Performance placement describes the assessment of software performance in a specific environment relative to other (or, theoretically, all) environments. Because the SPA survey would require SAS users to run identical programs on identical data, and because repeated measures obtain more reliable metrics, any performance disparity among sites could be credited to their independent configurations and infrastructure rather than other sources of variability such as the uniqueness of data or the software processing them.

For example, Figure 1 demonstrates software performance for a single SAS environment; however, if the same component of SPA were run in a separate environment, the results could be compared. Would the second environment also demonstrate a linear relationship that perfectly correlated CPU time with observation count? Would it be faster or slower? Would either of the environments transition from a linear to curvilinear (or other) function at some minimum or maximum observation count or other threshold? And as the number of participants in the SPA survey increased, and commensurately, too, the infrastructure variability in those environments, attention could turn from blandly identifying whose SAS was slow and whose SAS was fast to more remarkably identifying the specific characteristics correlated with higher performing environments.

CONCLUSION

The SAS Performance Assessment (SPA) conceptualizes an online survey that could be completed by global SAS users to baseline and compare the performance of individual SAS environments the world over. Running for only a few hours, SPA would process the same data using the same programs, thus minimizing external sources of performance variability while collecting performance metrics on a host of operations. By obtaining additional qualitative and quantitative indicators about the nature and structure of the SAS environment, these data could ultimately be correlated with performance metrics to demonstrate the collective and intertwined roles they play in SAS environment performance.

CONTACT INFORMATION

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

Name: Troy Martin Hughes
E-mail: troymartinhughes@gmail.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.