SAS® Grid Manager – Testing and Benchmarking Best Practices for SAS Intelligence Platform

INTRODUCTION

Grid computing offers optimization of applications that analyze enormous amounts of data as well as load balancing and management of multiple applications by providing an environment of shared and dynamically allocated resources. Grid computing delivers value in a highly efficient manner for SAS analytics, data integration (ETL), data mining, and business intelligence. However, the success of a grid depends on the nodes in the grid spending more time computing than they spend communicating. This paper will address some best practices for how to approach testing and benchmarking I/O performance with SAS in a grid.

While this paper focuses on the SAS Intelligence Platform and executing user-written SAS programs in a grid, several other SAS solutions benefit from grid execution such as SAS Data Integration Studio and SAS Enterprise Miner. These solutions offer point-and-click interfaces to enable you to develop complex SAS workflows that can automatically execute in a grid environment without any end-user programming. Performance testing and tuning specific to those products is beyond the scope of this paper. However, the information in this paper is applicable to running other SAS products and solutions in a grid.

SAS AND DATA

When running SAS in a grid environment it is a best practice to have a shared data infrastructure that allows each node in the grid access to storage. SAS applications and solutions typically process enormous amounts of data and the SAS I/O pattern is very sequential. The read and write percentage will vary greatly depending on the SAS application. For example, queries would be mostly reads, Data Integration and ETL tasks would be typically half reads and half writes. It is nearly impossible to predict read write percentages in a typical SAS environment with multiple types of SAS applications. A mixed workload environment would likely be in the range of 60% read and 40% writes.

There are three types of data that are critical to successful execution of SAS in a grid:

- Input data
- Temporary data
- Output data
- Cached data

Input Data Considerations

Each of the fundamental tasks distributed across the grid must have access to all required input data regardless of where the tasks are executed. Sometimes the input data may be small (on the order of 100s of MB's) and other times the data may be large (multiple GB's). In order to achieve the highest efficiency, the compute nodes should spend the majority of their time computing rather than communicating. Compute tasks that require substantial data movement require adequate bandwidth to the data. In this case, the data can either be distributed to the nodes prior to running the application or much more commonly, stored on a file server accessible across the network via a shared file system implementation combined with a typical SAN environment. It is important to consider the amount of data being shared across the network. If the size of the shared files is in the gigabyte range then you need to ensure that you have adequate bandwidth between your grid nodes and file system to maintain the necessary I/O throughput. If the input data is generated by the application and not shared or needed later then there is nothing to worry about. Data can be kept local to the server / grid node where execution occurs.

SASWORK Considerations

SAS does temporary file I/O in the SAS WORK directory, which is a disk directory configurable as a SAS option. SAS I/O activity in SAS WORK can be quite substantial both in terms of intensity and disk space, depending on the SAS job. As you plan your grid you need to evaluate SAS WORK usage by your application and ensure that you have adequate SAS WORK space allocated for each grid node. In some SAS environments SAS WORK may be set to a network location instead of local storage. Cost vs. performance needs to be analyzed in order to determine whether to put SAS WORK on a network device or on local storage.

Output Data Considerations

You also need to consider the amount of data that is output by the application and ensure either that adequate space is allocated in a shared location such that each of the grid nodes can write their output to the shared space or ensure that each of the grid nodes has adequate space to write their respective output to their local file system. It is

recommended to write all output to a shared space if possible so that it is available to all nodes for subsequent execution. You can also write SPDS tables back to a shared storage and then snap the partial tables together in SPDS to create one large final table.

File Caching
File cache can play an important role in SAS application performance. However, because of process and application contention for the limited amount of file cache in a single operating system instance performance can suffer. In a grid environment less contention and more resources (i.e. RAM and CPU) can alleviate this problem and at a potentially reduced hardware cost.

VALIDATION OF GRID INFRASTRUCTURE

When setting up a grid environment for testing and benchmarking it is a best practice to configure a small subset, in the range of 5-10 nodes, of your grid with the same network infrastructure as the planned production environment for the most meaningful results.

It is also a best practice to review your SAS applications and the data that they use and determine what your per process and overall I/O throughput requirements are before you begin testing. This will help you set realistic and beneficial performance goals for your organization.

There are some basic command style tests that can be run to determine if your compute, network and storage infrastructure can provide the I/O throughput sufficient to meet your performance expectations with your SAS applications. SAS typically writes in 8K file chunks to the file system. The most important aspect from an I/O perspective for SAS is MB/s rather than IOPS. You should review the types of SAS applications that you are running and get an idea of what type of throughput you want to get out of your file systems. The following is a good, free tool to sanity check your file system BEFORE installing and running SAS in order to determine if your file system will be able to scale and provide the level of throughput required by your SAS applications.

Non-SAS Tool (UNIX Systems (for Windows Systems refer to the SAS tool in the following section)):
Run the following steps initially on a single grid node that has access to the file system. Once you have completed the testing on a single grid node, determine the scalability by running these steps simultaneously on 2 grid nodes, then 5 grid nodes and then 10 grid nodes. The increments you choose may vary depending on the number of grid nodes in your test environment. The goal would be to see near-linear scalability. For example, 1 server = 100 MB/s sustained, 2 servers to NAS = 199 MB/s, 5 = 490 MB/s, etc. You can use operating system commands/tools or third party tools to monitor and measure the I/O rates.

Do the following to check the IO throughput rates on UNIX systems prior to running any SAS applications. First, create two very simple shell scripts. The first script will test the write IO throughput rate and the second script will test the read IO throughput rates.

iowritetest.sh

```
dd if=/dev/zero of=/filesystem/dd1.data bs=64k count=250000
sync
```

This script creates a file in the /filesystem directory (you will need to replace /filesystem with a valid directory name on your computer). The blocksize for the file is 64k and the total size will be 16GB. You can make the file smaller by changing the count, but please note you need to create a file greater than the amount of physical RAM in your box.

ioreadtest.sh

```
dd if=/filesystem/dd1.data of=/dev/null bs=64k count=250000
```

This script reads the file that was created by the iowritetest.sh script. Again, you will need to replace the /filesystem with the directory name of where you created the file.

Now that the scripts are ready, here is how you execute them on your UNIX system. Please note that on Linux systems you will use the time command instead of the timex command listed below.

```
sync
timex iowritetest.sh
```

sync
timex ioreadtest.sh

The sync command is used to flush all previously unwritten system buffers out to disk. This will insure there are no leftover write activities going on your computer before you start your test. The timex command gives time information for any command that you can run from a UNIX command prompt.

Here are example numbers timex echoed to the xterm screen for the write test:

Real	3:13.85
User	0.66
Sys	2:14.81

Here are example numbers timex echoed to the xterm screen for the read test:

Real	2:11.56
User	0.72
Sys	1:23.76

To determine the Mbytes/second IO throughput rate, you determine the number of MBs written (16 (because we ran a 16GB test) * 1024) and divide it by the number of second of real time it took to do the task.

So, the write IO throughput rate is 16,384 divided by 193.85 for a total of 84.52 MB/sec IO throughput rate.
The read IO throughput rate is 16,384 divided by 131.56 seconds for a total of 124.58 MB/sec IO throughput rate.

SAS Tool (All Systems):
Run the following steps initially on a single grid node that has access to the file system. Once you have completed the testing on a single grid node, determine the scalability by running these steps simultaneously on 2 grid nodes, then 5 grid nodes and then 10 grid nodes. The increments you choose may vary depending on the number of grid nodes in your test environment. The goal would be to see near-linear scalability. For example, 1 server = 100 MB/s sustained, 2 servers to NAS = 199 MB/s, 5 = 490 MB/s, etc. You can use operating system commands/tools or third party tools to monitor and measure the I/O rates.

Do the following to check I/O throughput rates on all systems using two very simple SAS jobs. First, create the two simple SAS jobs. The first job will test the write IO throughput rate and the second job will test the read IO throughput rate.

write.sas

```
OPTIONS FULLSTIMER SOURCE SOURCE2 MSGLEVEL=I MPRINT NOTES;
PROC OPTIONS GROUP=MEMORY;
RUN;

%LET COUNT=250000;
%LET DIGITS=0123456789;
%LET
INITSTR=%DIGITS%DIGITS%DIGITS%DIGITS%DIGITS%DIGITS%DIGITS%DIGITS
%DIGITS%DIGITS%DIGITS;

OPTION BUFSIZE=64K;
LIBNAME PERM "/filesystem";
DATA PERM.DS;
  LENGTH C $255;
  RETAIN C "&INITSTR";
  DROP I;
  DO I=1 to 256*&COUNT;
     OUTPUT;
  END;
```

```
RUN;

X "sync";
RUN;
```

This SAS job creates a file in the /filesystem directory (you will need to replace /filesystem with a valid directory name on your file system). The blocksize for the SAS data file is 64k and the total size will be 16GB. You can make the file smaller by changing the MB= value, but please note you need to create a file greater than the amount of physical RAM in your box

read.sas

```
OPTIONS FULLSTIMER SOURCE SOURCE2 MSGLEVEL=I MPRINT NOTES;
PROC OPTIONS GROUP=MEMORY;
RUN;

LIBNAME PERM "/filesystem";
DATA _NULL_;
   DROP _ALL_;
   SET PERM.DS(DROP=_ALL_);
RUN;
```

This SAS job reads the file that was created by the write.sas SAS job. Again, you will need to replace the /filesystem with the directory name of where you created the SAS data file.

Now that the SAS jobs are ready, here is how you execute them:

```
sync
sas write.sas

sync
sas read.sas
```

To get the numbers for the SAS write test, you need to open the write.log file and scroll to the very bottom. Here are example numbers from the write test:

```
Real time              3:25.70
User CPU time            33.60
System CPU time        1:52.27
```

To get the numbers for the SAS read test, you need to open the read.log file and scroll to the very bottom. Here are example numbers from the read test:

```
Real time              2:38.53
User CPU time            21.97
System CPU time        1:23.89
```

To determine the Mbytes/second IO throughput rate, you determine the number of MBs written (16 (because we ran a 16GB test) * 1024) and divide it by the number of second of real time it took to do the task.

RUNNING SAS ON THE GRID

When moving your SAS workload to the grid it is a best practice to identify a couple of applications that either already make use of MP CONNECT or that lend themselves to parallelization as well as a couple of ad hoc SAS jobs that you may want to execute on your grid environment.

A grid infrastructure provides many benefits to a SAS enterprise deployment including:

- Application acceleration – Significant performance increases are possible for those applications that can be decomposed into independent subtasks of work that can be distributed and executed in parallel on a grid.

- Load balancing and policy enforcement for multiple applications – Grid allows you to create a virtual server environment to execute non-parallel applications from multiple users and to share and provision resources in order to most effectively balance workload and meet service levels across the enterprise.

For application acceleration, identify a couple of SAS jobs that are well suited for decomposition and parallelization. Ideally you will already have SAS programs that make use of the MP CONNECT feature of SAS/CONNECT. If that is the case you can very easily modify these programs to execute with SAS Grid Manager by adding a few statements to the beginning of your program. Refer to "Sample Code to Grid Enable SAS Programs" at http://support.sas.com/rnd/scalability/grid/gridfunc.html for the exact statements needed. No other modifications to your program are necessary.

If you do not have a program that already leverages the MP CONNECT capability, you need to identify an application that has multiple independent units of work. Further, the independent units of work need to consume sufficient compute or I/O resources to warrant distribution and execution on a grid. A good way to determine this is to look at the SAS log file from a recent run and the elapsed time field from the fullstimer output at the end of each job step. The larger the time required to execute the job steps, the more potential there is for performance gains through parallelization. It is also important that this time is being spent in SAS rather than outside of SAS, such as doing an external DBMS query. Once you have identified a program that is a good candidate for parallelization, you will need to modify it to add the MP CONNECT statements as well as the grid enabled statements referenced by the link above. You can find the details of the MP CONNECT statements under "SAS/CONNECT Software" at http://support.sas.com/rnd/scalability/tricks/index.html .

Load balancing and policy enforcement of multiple applications also has the benefit of decoupling the knowledge of the IT infrastructure from the application. Users can submit their SAS applications to a virtual environment and SAS Grid Manager will determine the appropriate resource for executing the application. In addition, policies can be defined to give prioritization to specific users, to enforce time policies such that certain machines are only available during certain times of the day, etc. Also your SAS programs can be submitted to the grid through your favorite interface; whether that is batch submission or interactive submission using SAS Display Manager. This is done without any modifications at all to the SAS programs. You can find the Grid Toolbox at http://support.sas.com/rnd/scalability/grid/download.html which contains batch submission scripts as well as a new key definition for DMS submission.

In many cases submitting whole SAS jobs to the grid will result in faster execution either due to running them with SAS9 which supports multi-threading in many procedures, faster processors on newer grid hardware or because the jobs will be load balanced to the most appropriate machine. Just as important as performance improvements in this case is the ability to define and manage policies from a central location through SAS Grid Manager. This is a very valuable capability that should not be minimized.

CONCLUSION

Grid computing offers tremendous potential to reduce costs and increase performance. The most successful grid environments will be designed to ensure that the nodes in the grid spending more time computing than they spend communicating. When testing and benchmarking a SAS grid environment it is critical that you examine your SAS applications and determine the necessary I/O throughput required for successful execution. Once that has been determined there are several steps outlined in this paper that can be followed to ensure that the infrastructure will meet your performance expectations and to help you move your SAS workload to the grid.

FOR MORE INFORMATION

For more information about SAS and grid computing, visit the following websites:
SAS Scalability and Performance Community
http://support.sas.com/rnd/scalability/grid
SAS grid website
http://www.sas.com/grid