

Optimizing SAS® Grid Computing with SAS® Scalable Performance Data Server and Dynamic Data Partitioning

Andy Knight, SAS Institute Inc.

ABSTRACT

Making optimal use of SAS® Grid Computing relies on the ability to spread the workload effectively across all of the available nodes. With SAS® Scalable Performance Data Server (SPD Server), it is possible to partition your data and spread the processing across the SAS® grid computing environment. In an ideal world it would be possible to adjust the size and number of partitions according to the data volumes being processed on any given day. This paper discusses a technique that enables the processing performed in the SAS Grid Computing environment to be dynamically reconfigured automatically at run time in order to optimize the use of SAS Grid Computing and to provide significant performance benefits.

INTRODUCTION

Making optimal use of SAS Grid Computing relies on the ability to spread the workload effectively across all of the available grid nodes. With SAS Scalable Performance Data Server (SPD Server), and the SAS Data Integration Studio® Loop transformation, it is possible to partition your data and spread the processing across the nodes in the SAS Grid Computing environment.

Ideally we would like to be able to adjust the size and number of partitions according to the data volumes being processed on any given day. This would enable the process to adapt to rapidly changing data volumes.

This paper describes a technique that enables the processing performed in the SAS Grid Computing environment to be adapted at run time without manual intervention, in order to optimize the use of SAS Grid Computing and to provide significant performance benefits.

LOADING DATA IN PARALLEL

Traditionally, parallel processing a data load involves splitting the input data set, processing the parts, and then joining the parts back together. (See Figure 1.)

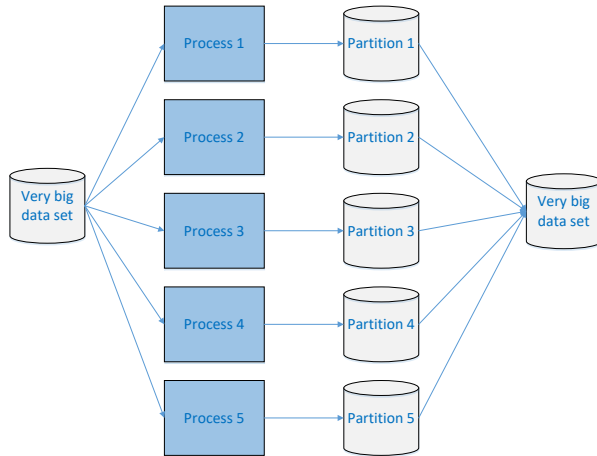


Figure 1 - Parallel Processing Data Flow

With a traditional relational database management system (RDBMS) this means defining the number of partitions in advance, or worse, copying the partitions back to a main data set after the individual processes have completed.

To achieve the best performance it is important that I/O is kept to a minimum. Copying the partitions back to a main data set is likely to cost more time than parallel processing saves.

Defining the number of partitions in advance can save us the additional I/O required to consolidate the data when processing is complete, but it is a solution that does not take into account the volume of data being processed. A fixed number of partitions are loaded by the same number of parallel processes, every time, without any consideration of fluctuating data volumes. Changes in data volumes will mean that either too little parallelization occurs, which extends run times, or too much occurs, which consumes more grid resources than required, to the detriment of other work being processed on the grid.

THE IDEAL SOLUTION

Ideally the solution would be customized to fit the data volume being processed. Customization would involve using more partitions and additional processing streams to cope with high data volumes, and reducing the number of partitions and processing streams for lower data volumes in order to release grid resources that are not required.

A dynamic solution that is driven by data volumes could be configured to fit a fixed processing window, this solution would make the time taken to process the data more predictable, even with fluctuating data volumes. (See Figure 2.)

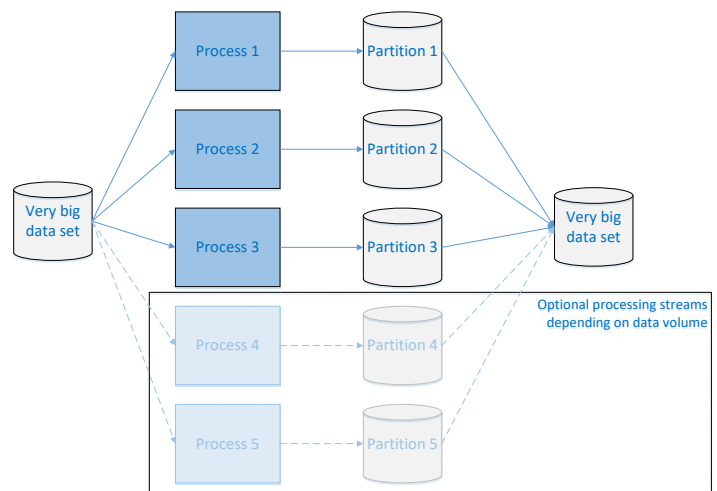


Figure 2 - Dynamic Parallel Processing Data Flow

A DYNAMIC APPROACH WITH SPD SERVER

SAS Scalable Performance Data Server (SPD Server) is a SAS product that has been around for a long time. It provides a lot of parallel processing capabilities through multi-threading, and this helps to ensure that each grid node is fully exploited. It also provides a table “Clustering” capability.

Using SPD Server it is possible to create multiple data sets with identical structure, and to combine these data sets into a single clustered data set.

This SPD Server version of partitioned tables has features not found in tables used by a traditional RDBMS. First, the partitioning feature is completely dynamic. The number of partitions can be defined at run time and they can be varied every time. There is no need to identify the number of partitions required in advance. Second, and most importantly, the process of joining the partitions into a single table does not require any data to be moved or copied. Clustering a collection of SPD Server tables into a single table is an operation performed in the SPD Server metadata, not an operation in physical data. SPD Server metadata (which is different from SAS Metadata) provides details about the SPD Server tables, their indexes, and clustering. Building an SPD Server clustered table only involves updating SPD Server metadata, making it very quick indeed.

DESIGNING THE SOLUTION

A dynamic approach to parallelization requires careful design, in order to perform these functions.

- Identify the number of partitions required

We want the number of partitions and parallel workstreams to be driven by data volumes, so a sensible approach would be to count the number of records on the input data set. For this we need to use a function that provides the record count from the table header, rather than by reading the entire input data table, which would sacrifice most of the benefits we hope to gain. Although this approach sounds obvious, some storage technologies, particularly those provided a traditional RDBMS are remarkably reluctant to give up information of this sort, and if the input data is being provided as a view, it is unlikely to be available.

- Split the data evenly between the partitions

For the dynamic process to succeed it is important that the partitions are sized as evenly as possible. This can be achieved with a surrogate key, or with an existing key if the data contains evenly distributed key values. Using the record number is a good way of guaranteeing that the partitions will be of equal size, but there are other possibilities. For example, if we know that there is an even distribution of account numbers in the input data, then the account number might prove to be a better key. The choice of key might be impacted by the processing that is going to be carried out upon the data, which might improve if the data is processed by specific key ranges.

- Process the data in parallel streams

Using the SAS Data Integration Studio Loop transformation, parallel processing streams can be executed, with the number of streams defined by the loop control data set. Building this control data set to reflect the number of partitions that are identified by the record count allows us to dynamically adjust the number of processing streams.

- Cluster the data sets into a single logical table

When the parallel processing streams have all completed successfully, we can use PROC SPDO to cluster the resulting partition tables into a single logical table. Because this is an SPD Server metadata operation, it is extremely quick, and does not involve moving any of the data.

PUTTING IT ALL TOGETHER

The SAS Data Integration Studio job flow shown in Figure 3 below provides dynamic data partitioning and parallelized processing with SPD Server.

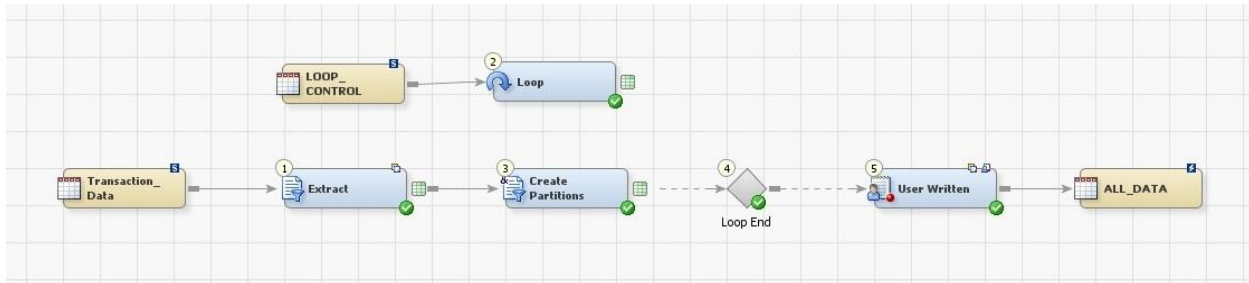


Figure 3 - Dynamic Parallel Processing Job Flow

The number of partitions required is calculated from an estimate of the number of records per partition. A production job might fetch this value from a parameter file, but in Figure 4 the value is coded into the jobs precode for simplicity. In a production environment you might need to tune this value to the most effective partition size for your data. It would be better to set this value in a parameter table so that it can be changed without updating the jobs code.

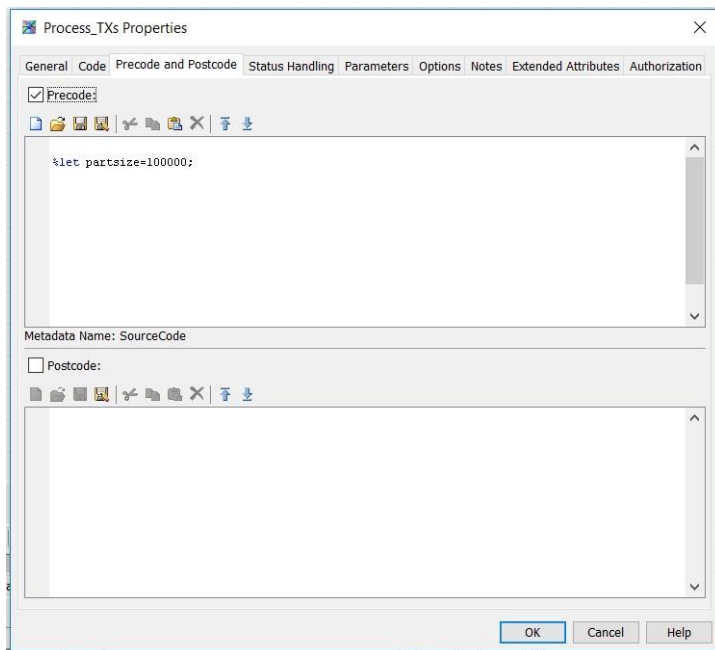


Figure 4 - Setting the Partition Size in Precode

The precode of the Extract transformation fetches the number of records in the input data set, and uses that value together with the partition size to calculate the number of partitions and parallel processes for this run.

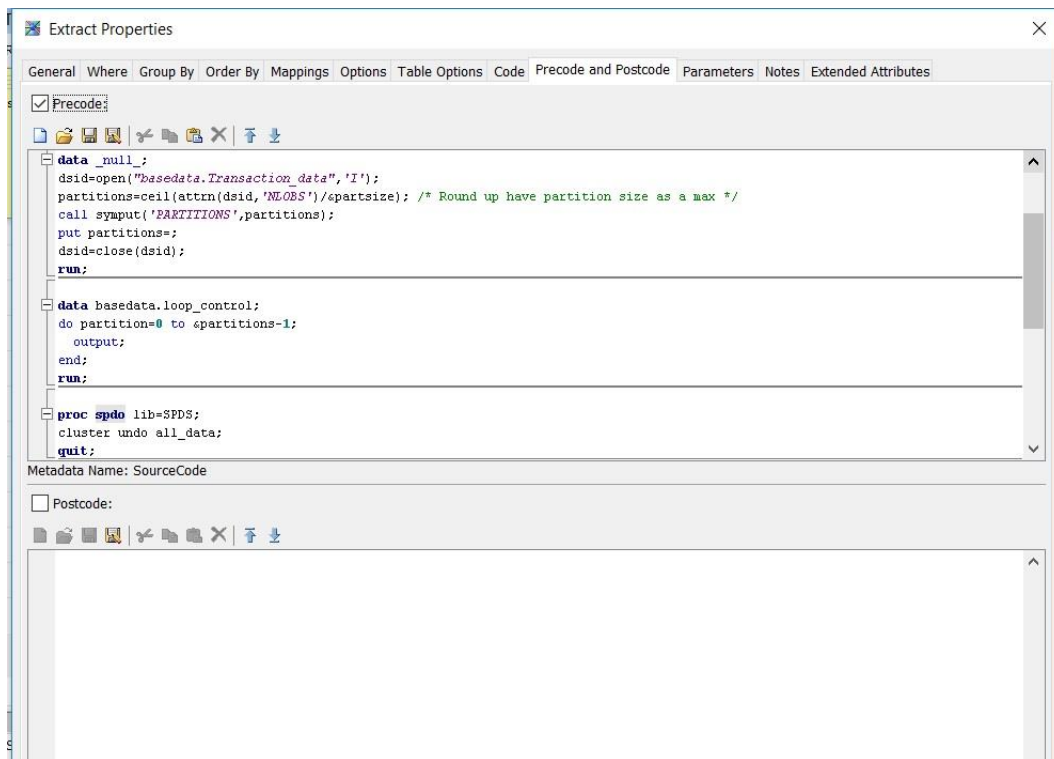


Figure 5 - Calculating the Number of Partitions from the Data Volume

The Extract precode shown in Figure 5 is used to create the Loop Control table. This table contains a single variable “PARTITION,” with values from 0 to 1 fewer than the number of partitions required. These values match the numbers of the partitions that we will create, for reasons that will become apparent.

The last task that the Extract precode performs is to “UNDO” the SPD Server cluster. This SPD Server metadata operation removes the links between the partitions of the existing SPD Server table, which effectively deletes the cluster, without removing the partition tables themselves. In this example, the partition tables will be re-created each time, and the existing tables are left in place. Individual circumstances might specify that it is better to delete the actual partition tables, or simply to update them. This choice depends on the data being processed.

This final step in the Extract transformation precode ensures that the cluster does not exist, which prevents problems when we try to re-create it.

The Extract transformation splits the input data and assigns each record to a partition for processing. The partition is calculated using the MOD function. MOD divides one number by another, and returns the remainder. MOD always returns a value between 0 and 1 fewer than the divisor. This is why we loaded our control table with partition numbers from 0 to 1 fewer than the number of partitions that are required.

| # | Column | Column Descrip... | Type | Le |
|---|------------|-------------------|-----------|----|
| 1 | account... | | Numeric | |
| 2 | tx_type | | Character | |
| 3 | tx_amo... | | Numeric | |

| # | Column | Colum... | Expression | Type | Length | Informat | Format |
|---|------------|----------|-----------------------------|-----------|--------|----------|--------|
| 1 | account_no | | | Numeric | 8 | (None) | z12. |
| 2 | tx_type | | | Character | 1 | (None) | (None) |
| 3 | tx_amount | | | Numeric | 8 | (None) | 10.2 |
| 4 | partition | | mod(account_no,&partitions) | Numeric | 8 | (None) | (None) |

Figure 6 - Assigning the Partition Number with the MOD Function

Creating a new variable in the input data set normally involves passing the entire data set and calculating the new value for each record. This action is obviously something we wish to avoid, because we would lose some of the benefits of parallelization with each pass of the input data set.

One way around this is to delay the actual pass of the data and the calculation of the new column until the point where we actually have to read the data. We can do this by creating the output data set from the Extract transformation as a view. This will retain the code required to create the output data set, but not actually execute it until the data is read by another transformation. In this situation, using a view saves us one pass of the entire data set. (see **Error! Reference source not found.**)

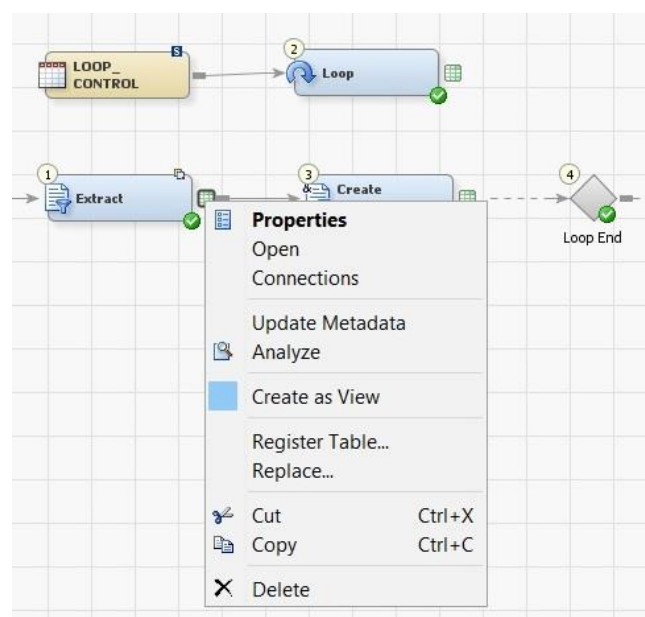


Figure 7 - Extract Output as a View

The SAS Data Integration Studio Loop transformation executes all of the transformations between the Loop and the Loop End transformations repeatedly. It executes these transformations once for each record on the Loop Control table. The variable in the Loop Control table is presented as a global macro variable value to the transformations being processed. This enables each processing iteration to identify itself and the partition table it will use. The options provided by the Loop transformation can be used to control the processing, make it sequential or parallel, and define the number of processes to be submitted concurrently.

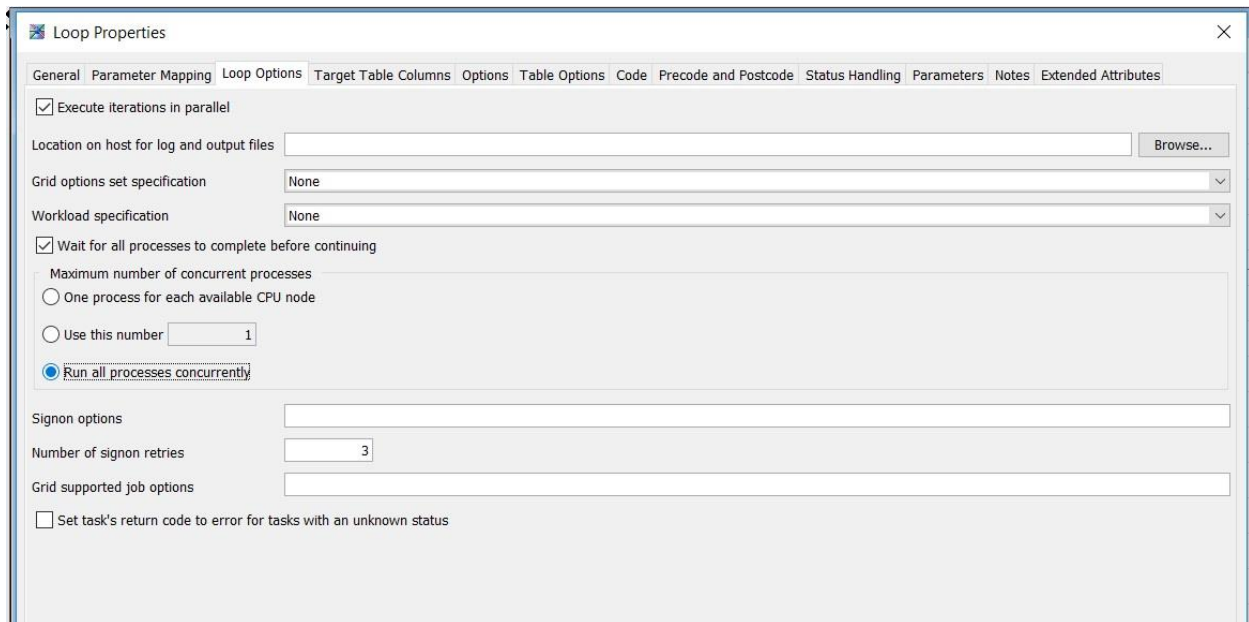


Figure 8 - Loop Transformation Processing Options

The options shown in Figure 8 specify that the loop iterations will run in parallel, and that the loop will wait for all processes to complete before continuing.

These options in a SAS grid processing environment provide the parallelization we are looking for, with the records in the Loop Control table defining the number of parallel processing streams to be created.

The maximum number of concurrent processes option might vary, depending on the resources of the individual grid environment. If the grid has ample resources, the “Run all processes concurrently” option might be appropriate. However, if resources might become an issue, the number of concurrent processes might need to be restricted by using the other Loop transformation processing options available, or by overriding the number of partitions to be used.

The input data is read from the view created by the Extract transformation and the “PARTITION” column is added. This column is used to define the data for each partition and processing stream. The processing transformation applies a WHERE clause to filter the data it will receive. (See Figure 9.)

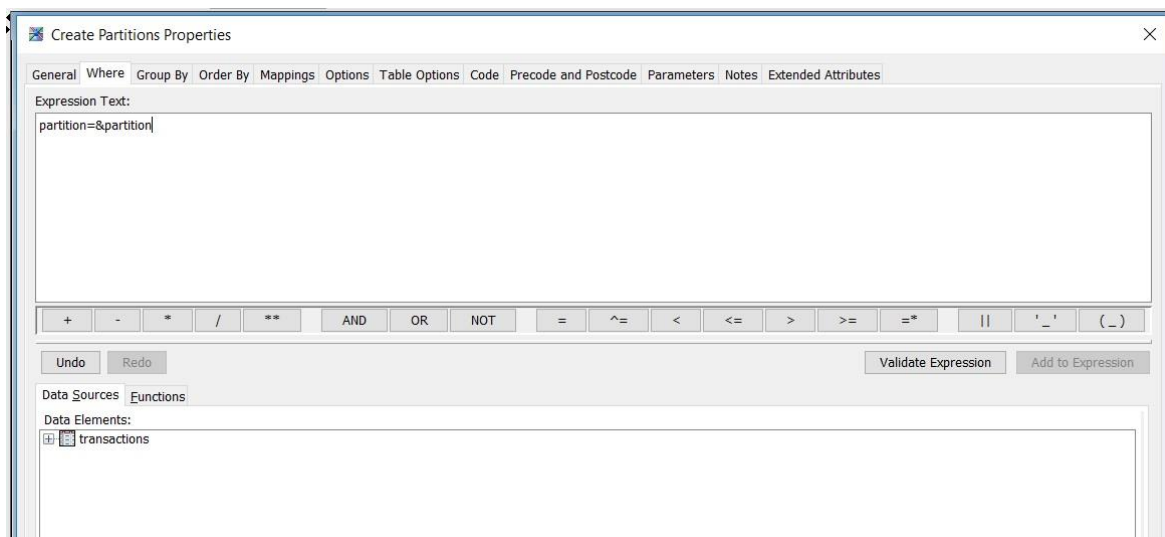


Figure 9 - Filtering the Data for Each Partition

The additional “PARTITION” column created in the view of the input data set is not required for the output data set that created by the processing, so the mappings of the processing transformation do not map this column through to the output data set. (See Figure 10.)

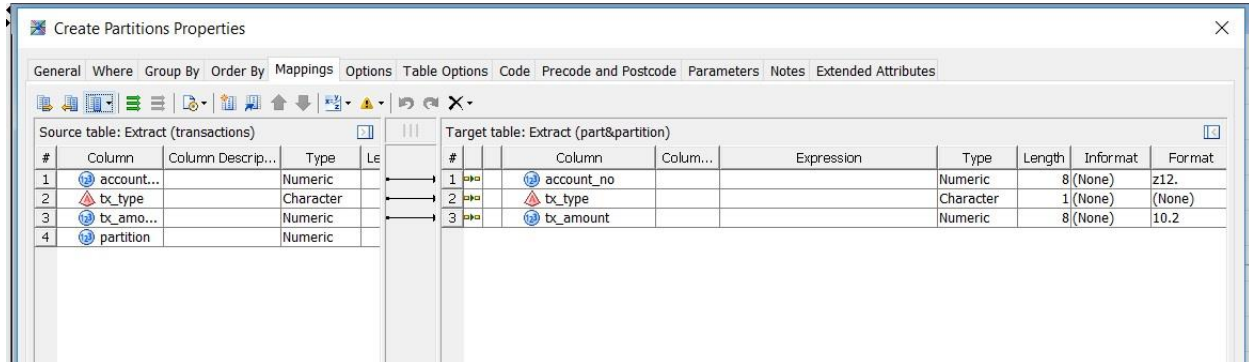


Figure 10 - Processing Mappings to Drop the PARTITION Column

The final configuration setting within the Loop transformation specifies the output data set. As each processing transformation executes, its input from the view is filtered by the WHERE clause, so that only data relevant for the current partition is read.

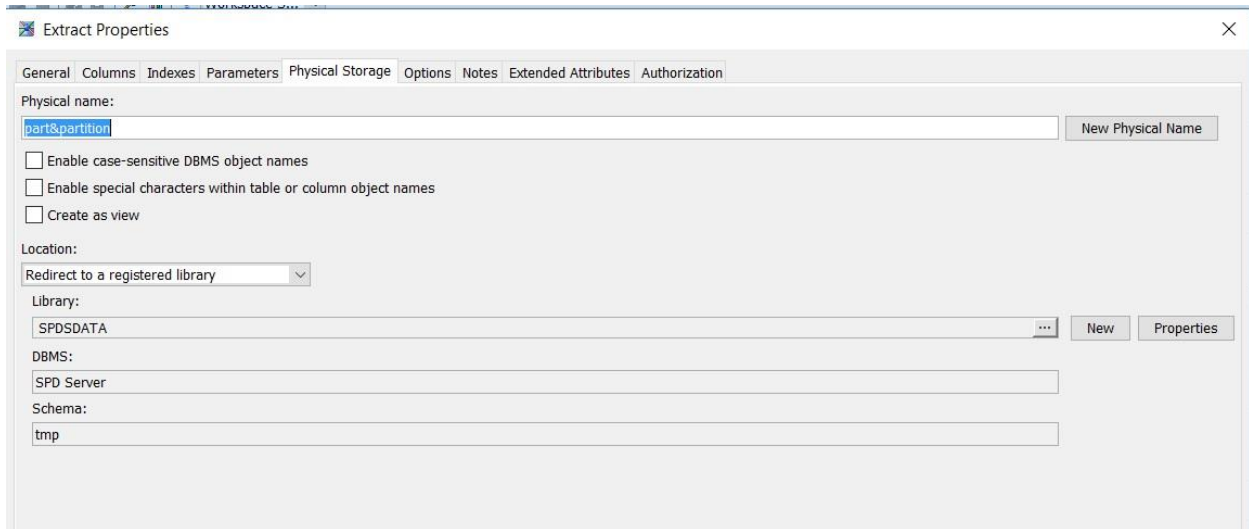


Figure 11 - Specifying the Output Partition Table

The output data set from the processing inside the Loop transformation must be separated by the value of the PARTITION column, and stored in a permanent SPD Server data library. Each SPD Server partition data set has a physical name that includes the partition number. (See Figure 11.) This ensures that all of the partition tables are created with distinct table names, which prevents them from overwriting each other.

After all of the parallel processes have completed, the Loop transformation exits and processing continues. The final step is to build the SPD Server cluster from the partition tables created within the Loop transformation.

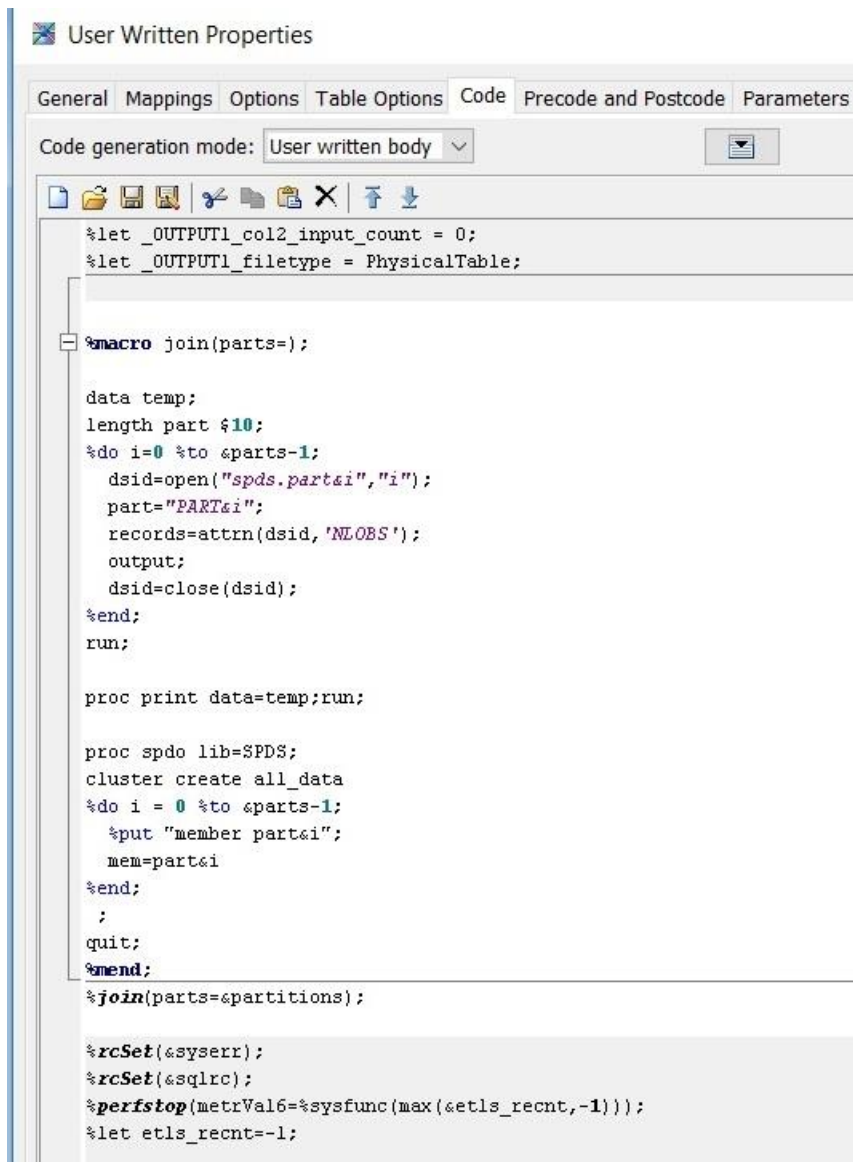


Figure 12 - Building the SPD Server Cluster

The macro in the user-written code shown in Figure 12 provides a basic report, which details the number or records in each partition.

| Obs | part | dsid | records |
|-----|--------|------|---------|
| 1 | PART0 | 1 | 97953 |
| 2 | PART1 | 1 | 96733 |
| 3 | PART2 | 1 | 97168 |
| 4 | PART3 | 1 | 97562 |
| 5 | PART4 | 1 | 97421 |
| 6 | PART5 | 1 | 97697 |
| 7 | PART6 | 1 | 97579 |
| 8 | PART7 | 1 | 97404 |
| 9 | PART8 | 1 | 97653 |
| 10 | PART9 | 1 | 97647 |
| 11 | PART10 | 1 | 97522 |
| 12 | PART11 | 1 | 97702 |
| 13 | PART12 | 1 | 97607 |
| 14 | PART13 | 1 | 97477 |
| 15 | PART14 | 1 | 97455 |
| 16 | PART15 | 1 | 97794 |
| 17 | PART16 | 1 | 97517 |
| 18 | PART17 | 1 | 97609 |

Figure 13 - Partition Table Record Count Report

The report (see Figure 13) is optional, but provides a useful insight into distribution of records between the partitions. Once again the record counts are obtained from the table header, in order to improve the speed of processing.

The SPD Server cluster is built with PROC SPDO, using a macro loop to specify the partition tables. (See Figure 14.)

```

NOTE: Enter ";" to get info on all the SPDO commands.
      Enter "SPDSMAC;" to get list and setting of all SPD macros.
MPRINT(JOIN):  proc spdo lib=SPDS;
"member part0"
"member part1"
"member part2"
"member part3"
"member part4"
"member part5"
"member part6"
"member part7"
"member part8"
"member part9"
"member part10"
716
"member part11"
"member part12"
"member part13"
"member part14"
"member part15"
"member part16"
"member part17"
MPRINT(JOIN):  cluster create all_data mem=part0 mem=part1 mem=part2 mem=part3 mem=part4 mem=part5 mem=part6 mem=part7 mem=part8
mem=part9 mem=part10 mem=part11 mem=part12 mem=part13 mem=part14 mem=part15 mem=part16 mem=part17 ;
NOTE: MAXSLOT= set for Dynamic growth by default.
NOTE: CLUSTER ALL_DATA has been created with -1 maximum slots.
MPRINT(JOIN):  quit;

NOTE: PROCEDURE _DISARM_      STOP! _DISARM_ 2016-11-04T08:31:16.892+00:00! _DISARM_ WorkspaceServer! _DISARM_ SAS! _DISARM_ !
_DISARM_ 21331968! _DISARM_ 16351232! _DISARM_ 10! _DISARM_ 10! _DISARM_ 0! _DISARM_ 5254681821! _DISARM_ 0.000000! _DISARM_
0.042000! _DISARM_ 1793867476.851000! _DISARM_ 1793867476.893000! _DISARM_ 0.000000! _DISARM_ ! _ENDDISARM_
NOTE: PROCEDURE _DISARM_ STOP! _DISARM_ 2016-11-04T08:31:16.892+00:00! _DISARM_ WorkspaceServer! _DISARM_ SAS! _DISARM_ !

```

Figure 14 - Building the SPD Server Cluster

PROC SPDO executes very quickly, because it does not need to move any data in order to combine the partition tables into a single logical entity. The log in Figure 15 shows just how quick the final cluster build process can be.

```

139
"member part11"
"member part12"
"member part13"
"member part14"
"member part15"
"member part16"
"member part17"
MPRINT(JOIN):  cluster create all_data mem=part0 mem=part1
mem=part9 mem=part10 mem=part11 mem=part12 mem=part13 mem=p
NOTE: MAXSLOT= set for Dynamic growth by default.
NOTE: CLUSTER ALL_DATA has been created with -1 maximum slo
MPRINT(JOIN):  quit:

NOTE: PROCEDURE _DISARM|  STOP| _DISARM| 2016-11-07
_DISARM| 23175168| _DISARM| 17561952| _DISARM| 12| _D:
0.180000| _DISARM| 1794146143.393000| _DISARM| 179414
NOTE: PROCEDURE SPDO used (Total process time:
real time      0.18 seconds
cpu time       0.01 seconds

1325
1326  /*---- End of User Written Code ----*/
1327
1328  %rcSet(%syserr);
1329  %rcSet(%sqlrc);
1330  %perfstop(metrVal6=%sysfunc(max(%etls_recnt,-1))
MPRINT(PERFSTOP):  options notes nosource nosource2 nosymb
NOTE: _DISARM|A505MEI9.BN000000E|triton|UserWritten| _DISARM
SAS_Data_Integration_Studio| _DISARM| SAS| _DISARM| -.

```

Diagram Code Log Output

Details

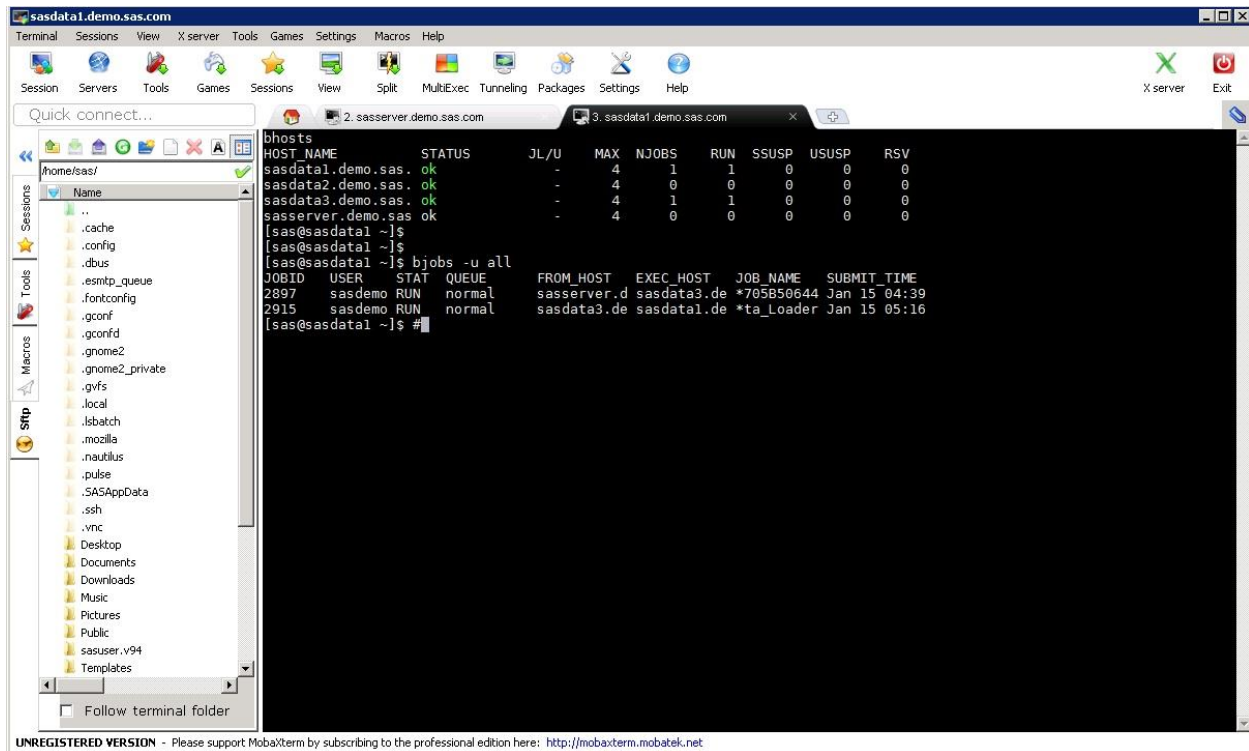
Columns Status Warnings and Errors Statistics Control Flow

Figure 15 - Step End Statistics for Cluster Build

MONITORING THE GRID

The Load Sharing Facility (LSF) commands “bjobs” and “bhosts” allow us to examine the jobs running or queueing to run on the grid, and the current status of the grid nodes.

Initially, no data load jobs are present in the grid.;



```
bhosts
HOST_NAME      STATUS      JL/U      MAX      NJOBS      RUN      SSUSP      USUSP      RSV
sasdata1.demo.sas ok          -         4         1         1         0         0         0
sasdata2.demo.sas ok          -         4         0         0         0         0         0
sasdata3.demo.sas ok          -         4         1         1         0         0         0
sasserver.demo.sas ok          -         4         0         0         0         0         0
[sas@sasdata1 ~]$
[sas@sasdata1 ~]$
[sas@sasdata1 ~]$ bjobs -u all
JOBID  USER  STAT  QUEUE  FROM_HOST  EXEC_HOST  JOB_NAME  SUBMIT_TIME
2897   sasdemo  RUN   normal  sasserver.d sasdata3.de *705B50644 Jan 15 04:39
2915   sasdemo  RUN   normal  sasdata3.de sasdata1.de *ta_loader Jan 15 05:16
[sas@sasdata1 ~]$ #
```

UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: <http://mobaxterm.mobatek.net>

Figure 16 - Activity before the Loop Transformation

Figure 16 illustrates the state just after the job was submitted from SAS Data Integration Studio. The active jobs are SAS Data Integration Studio itself and the main data loader job. At this stage, the main data loader job has not started to spawn additional jobs from the Loop transformation.

As the Loop transformation spawns parallel jobs to load data into the partitions, the jobs begin to use up the available grid node job slots. When the grid nodes reach capacity, SAS Grid Manager flags the nodes as Closed.

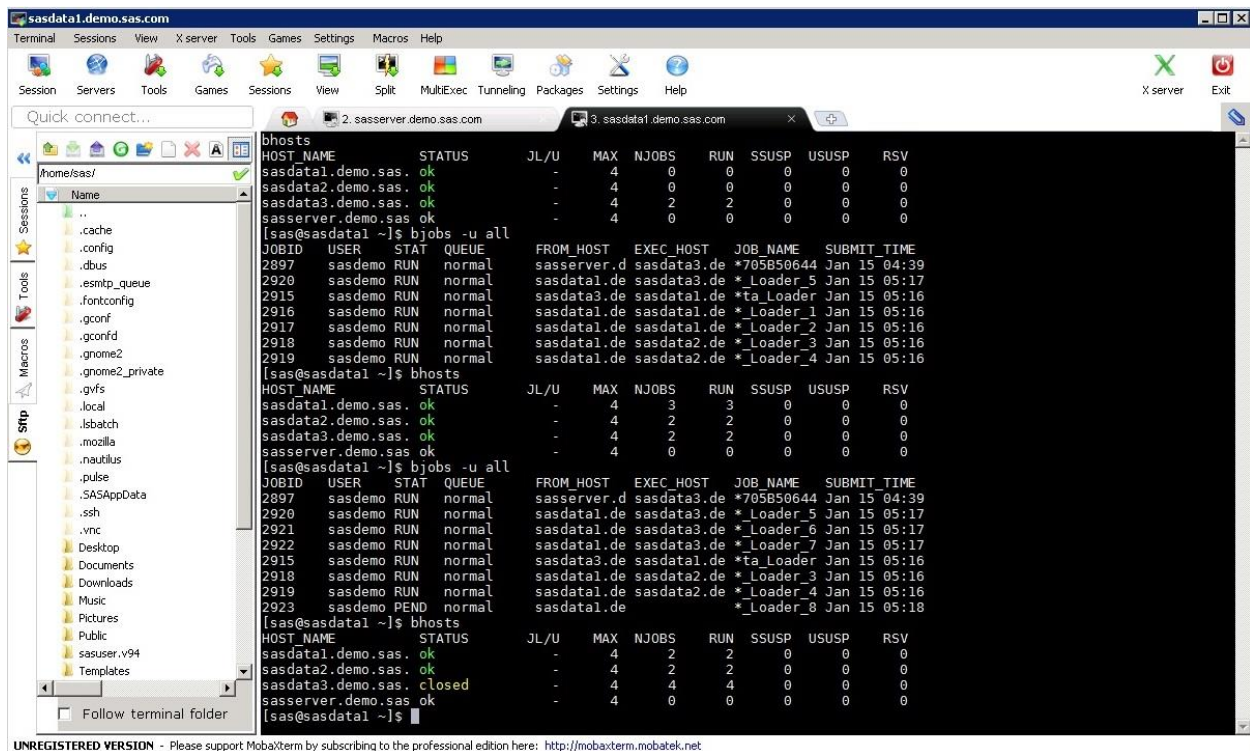


Figure 17 – Activity While Loop Transformation Runs

In Figure 17, we can see that grid node sasdata3 has reached capacity, and has been closed to new work by SAS Grid Manager.

When the active jobs complete and the grid nodes drop back to being within capacity, SAS Grid Manager opens the node to new work by flagging it to OK status.

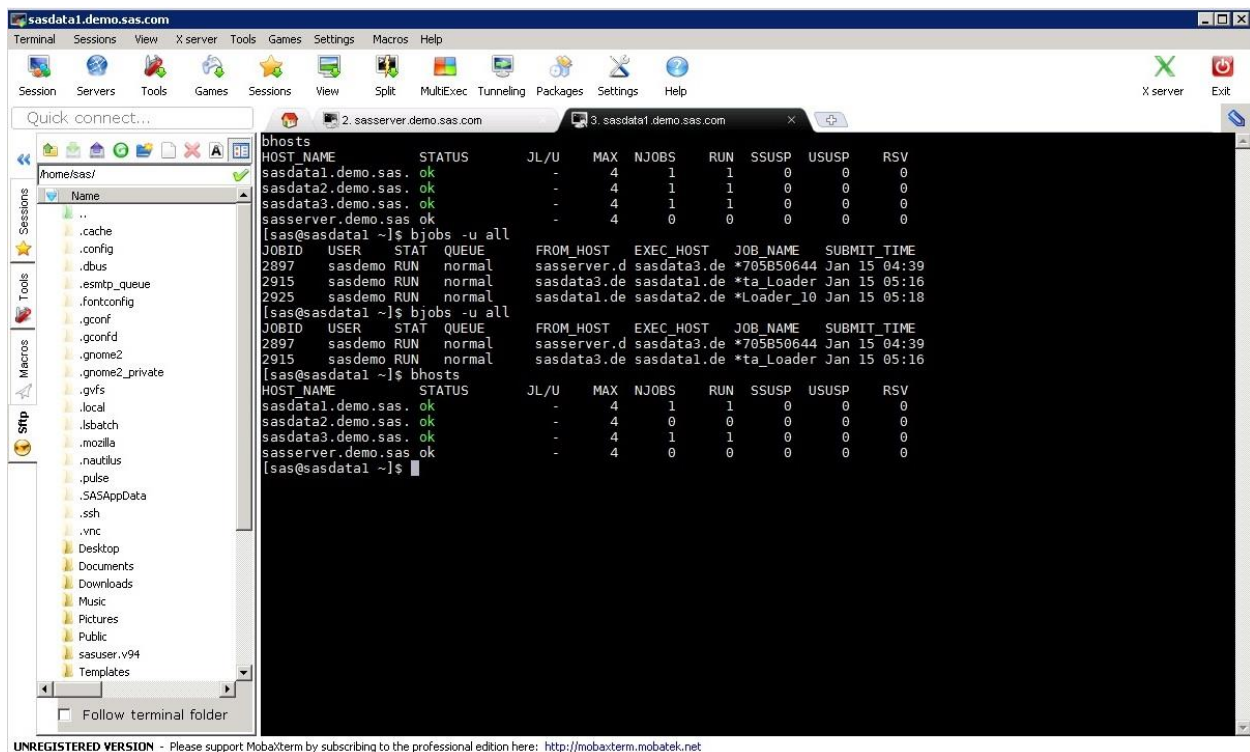


Figure 18 – Activity as Loop Transformation completes

LIMITATIONS AND CONSTRAINTS

Much of the benefit of this approach relies upon there being an even distribution of key values in the selected partitioning key. In the example shown here, we used a simple incremental number generated for each observation. This gives us a reliable key with an even distribution. Using a business key, such as account number or customer ID, will not necessarily result in an even key distribution, and this can lead to partitions of different sizes being created, which can reduce the effectiveness of this technique.

The testing environment used for this example, is similar to many testing and development environments in that it has “over-allocated” CPU resources. This means that although the grid nodes can be defined as having 4 CPU cores, in reality much of the environment’s memory and CPU resource is virtual, so each grid node might have as few as a single CPU core. Over-allocated CPU cores results in resource contention, which has a serious impact on performance. Reconfiguring LSF to reflect the true number of available CPU cores cures the contention problem, but doesn’t provide a realistic view of the impact of varying the partition size.

The best approach is to ignore job elapsed time, and concentrate on maximizing job concurrency. If the level of parallel job execution is high, the savings will be realized in the production environment, because production environments are rarely over-allocated.

CONCLUSION

In conclusion, this technique not only provides significant savings in batch load times, it does so without monopolizing the resources of the grid. It adds a degree of predictability to the length of time required to run the batch process, regardless of changes in the volumes of data being processed.

This list summarizes the features of the technique.

- We set the partition size to the number of records.
- The total number of input records and the partition size is used to set the number of partitions.
- The number of partitions is used to generate the Loop Control table.
- We pass the data as little as possible, using a view to reduce the required number of passes of the data.
- In the view we use an expression to create a partition number.
- We use a SAS Data Integration Studio Loop transformation to iterate through the processing for each partition.
- We create a separate table for each data partition.
- The processing within the loop can include one or more transformations or even one or more SAS Data Integration Studio jobs.
- When the Loop transformation completes, PROC SPDO is used to cluster the partition tables.
- Virtualized environments are fine, provided the resources are not over-allocated.

ADDITIONAL CAPABILITIES AND REFERENCES

SPD SERVER CLUSTERING CAPABILITIES

In the example in the main section of this paper, only the simple clustering capabilities of SPD Server have been demonstrated. You can explore these additional features, which can be used for more advanced requirements.

- Cluster create – create a clustered SPD Server table
- Cluster undo – delete a cluster (but leave the partition tables intact)
- Cluster add – Add a single table to a cluster
- Cluster remove – Remove a single table from a cluster

SPD SERVER WITH HADOOP

For very large data volumes, a clustered file system such as Hadoop can be used to store the data behind SPD Server.

RECOMMENDED READING

- SAS Institute Inc. 2014. *SAS Data Integration Studio 4.9: User's Guide*. Cary, NC: SAS Institute Inc. Available <http://support.sas.com/documentation/cdl/en/etlug/67323/PDF/default/etlug.pdf>
- SAS Institute Inc. 2016. *Grid Computing in SAS 9.4, Fifth Edition*. Cary, NC: SAS Institute Inc. Available <http://support.sas.com/documentation/cdl/en/gridref/69583/PDF/default/gridref.pdf>
- SAS Institute Inc. 2016. *SAS Scalable Performance Data Server 5.3: User's Guide*. Cary, NC: SAS Institute Inc. Available <http://support.sas.com/documentation/cdl/en/spdsug/68963/PDF/default/spdsug.pdf>
- SAS Institute Inc. 2016. *SAS Scalable Performance Data Server 5.3: Administrator's Guide*. Cary, NC: SAS Institute Inc. Available <http://support.sas.com/documentation/cdl/en/spdsag/68967/PDF/default/spdsag.pdf>
- SAS Institute Inc. 2016. *SAS 9.4 SPD Engine: Storing Data in the Hadoop Distributed File System, Fourth Edition*. Cary, NC: SAS Institute Inc. Available <http://support.sas.com/documentation/cdl/en/engspdehdfsug/69725/PDF/default/engspdehdfsug.pdf>

CONTACT INFORMATION

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

Andy Knight
SAS Institute Inc. (UK)
+44 (0)782 353 8809
andy.knight@sas.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.