Paper SAS1956-2015

The SAS® Scalable Performance Data Engine: Moving Your Data to Hadoop without Giving Up the SAS Features You Depend On

Lisa Brown, SAS Institute Inc.

ABSTRACT

If you are one of the many customers who want to move your SAS® data to Hadoop, one decision you will encounter is what data storage format to use. There are many choices, and all have their pros and cons. One factor to consider is how you currently store your data. If you currently use the Base SAS® engine or the SAS® Scalable Performance Data (SPD) Engine, then using the SPD Engine with Hadoop will enable you to continue accessing your data with as little change to your existing SAS programs as possible. This paper discusses the enhancements, usage, and benefits of the SPD Engine with Hadoop.

INTRODUCTION

Hadoop is an open-source, software framework under the Apache Software Foundation. Hadoop is used for storing and processing big data in a distributed fashion on large clusters of commodity hardware. Hadoop is an attractive technology for a number of reasons:

- cost-effective data storage
- scalable architecture
- failure resiliency

If you decide to move your data to Hadoop for processing with SAS, one of the first decisions is what file format to use for data storage. In making this decision, there are a number of things to consider, such as what storage format are you currently using and what type of processing is most important to you. If you currently store your data in a relational database, then moving your data to a format supported by the SAS/ACCESS® Interface to Hadoop engine might be a logical choice. If you currently store your data in the Base SAS engine file format or the SPD Engine file format, then moving your data to the SPD Engine file format with Hadoop will enable you to make the fewest changes in your SAS programs. On the other hand, if loading data into the SAS® High-Performance Analytics framework or the SAS® LASR™ Analytic Server as efficiently as possible is most important to you, then a good choice would be the SAS Data in HDFS (SASHDAT) engine.

Each SAS LIBNAME engine for accessing data in Hadoop has pros and cons depending on what features are most important to you. This paper gives a basic overview of features supported by each engine, but focuses on using the SPD Engine with Hadoop to maintain SAS features.

CHOOSING THE SPD ENGINE FOR DATA ACCESS WITH HADOOP

THE SPD ENGINE VERSUS. OTHER HADOOP LIBNAME ENGINES

SAS provides a number of LIBNAME engines that allow access to data stored in Hadoop. Which engine works best for you will depend on the features that are most important to you.

- SPD Engine with Hadoop
  - Provides the same SAS language support that exists when the data is located on a traditional file system. If you currently use the Base SAS engine or the SPD Engine, in most cases you will need only minor changes to your existing program to use your SPD Engine data with Hadoop.
  - Supports parallel data access with SAS High-Performance Analytics products using the SAS® Embedded Process for Hadoop. This data access allows data to be moved in parallel between the Hadoop data nodes and the compute nodes without being pulled back to the SAS client. Specifically this parallel data access process is used for the following:

        - Loading and unloading data into the SAS LASR Analytic Server through the SAS® LASR™ procedure.
        - Data access for all high-performance analytical procedures.
    - Most data access pulls data back to the SAS client for processing. One exception is the parallel data access using the SAS Embedded Process for Hadoop. Another exception is WHERE filtering using the ACCELWHERE= LIBNAME or data set option. Within certain constraints, specifying ACCELWHERE=YES allows the SPD Engine to process the WHERE filter on the Hadoop cluster, returning only a subset of data to the SAS client.
- SAS/ACCESS Interface to Hadoop engine
    - Supports SQL processing on the cluster when your data is stored in Hive.
    - Supports access to data stored in HDFS, but not registered in Hive, by using metadata created through PROC HDMD.
    - Supports parallel data access with SAS High-Performance Analytics products using the SAS Embedded Process for Hadoop. This data access allows data to be moved in parallel between the Hadoop data nodes and the compute nodes without being pulled back to the SAS client. Specifically this parallel data access process is used for the following:
        - Loading and unloading data into the SAS LASR Analytic Server through the SAS LASR procedure.
        - Data access for all high-performance analytical procedures.
    - Does not support certain SAS features such as SAS special missing values and the ability to perform random updates on data sets.
- SASHDAT engine
    - Used to distribute data in the Hadoop Distributed File System (HDFS) that is provided by SAS High-Performance Deployment of Hadoop.
    - File format designed for high-performance analytics. The SASHDAT file format allows the most efficient data access with the high-performance analytical procedures and when loading and unloading data into the SAS LASR Analytic Server.
    - Does not support read access other than when used with the high-performance analytical procedures and the LASR procedure. Because the data volumes in HDFS are typically very large, the engine is not designed to read from HDFS and transfer data back to the SAS client.

Table 1 summarizes the support provided by each engine.

	SPD Engine	**Hadoop Engine**	**SASHDAT Engine**
Read access	yes	yes	no *only metadata read access
Write access	yes	yes	yes
Update access	yes	no *insert supported but no record update	no
Parallel read/write with high-performance analytics	yes *some restrictions	yes	yes *best performance for parallel data access

	SPD Engine	Hadoop Engine	SASHDAT Engine
SQL processing in the Hadoop cluster	no *limited where filter push down	yes *with Hive data only	no
DS2 processing in the Hadoop cluster (Code Accelerator for Hadoop)	no	yes	no

Table 1 Features Supported by LIBNAME Engines Accessing Hadoop

SAS FEATURES YOU CAN DEPEND ON

Not all SAS features are supported by all LIBNAME engines that work with Hadoop. Table 1 identifies some SAS features and which LIBNAME engine supports them. These differences in behavior between engines are important to consider as you choose a file format in Hadoop. The following is a more complete list of features supported by the SPD Engine:

- Ability to replace a file
- SAS missing values, including special missing values
- Encryption
- Compression
- Indexes
- Passwords
- SAS formats
- User-defined formats
- Physical ordering of returned observations
- Update operations
- Random access

These features are supported by the SPD Engine whether the data is stored in a traditional file system or HDFS.

THE SPD ENGINE VERSUS THE BASE SAS ENGINE

If you currently have your data in Base SAS engine data sets, you might be wondering why SAS doesn't support that file format with Hadoop. The simple answer is that the file format does not lend itself to the type of processing done with Hadoop. The Base SAS engine file format contains both data and metadata intermixed throughout a single file. This format is problematic for data used with Hadoop for two reasons:

- The Java code used to read the data within the MapReduce framework assumes that the data is composed of data records only. To handle the mixed data and metadata records, a large amount of intelligence would need to be moved from the engine to the Java code. The resulting data access would not be very efficient.
- Updating data records in HDFS requires rewriting the data file. Having one large file versus partitioned data files, such as those that the SPD Engine supports, would require rewriting larger amounts of data for every update process.

The Base SAS engine file format and the fact that it is not a multi-threaded engine prevent it from being as scalable as needed given the size of data typically stored and processed with Hadoop.

In contrast, the SPD Engine is a multi-threaded engine and separates metadata and data into different files. The SPD Engine file format consists of metadata, data, and index files, as shown in Figure 1. The SPD Engine creates multiple threads, each accessing different data partitions. The partitioned data format is central to building a scalable engine and works well when used with Hadoop.

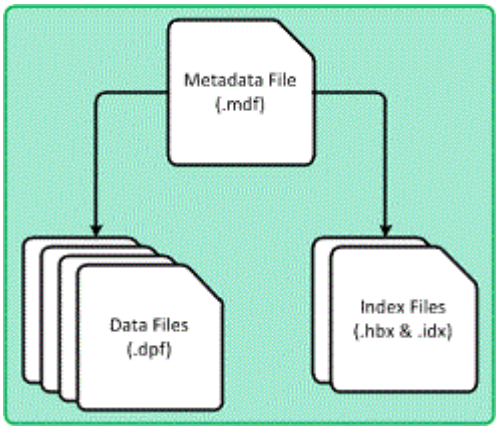

Figure 1. SPD Engine Data Model

If you are currently storing your data in the Base SAS engine file format, you need to be aware that there are some differences in the features supported by each engine. Table 2 summarizes the primary differences between features supported by the SPD Engine and the Base SAS engine:

	Base SAS Engine	SPD Engine
Audit trails	X	
Cross-Environment Data Access (CEDA)	X	
Extended attributes	X	
Generation data sets	X	
Integrity constraints	X	
SAS catalog and SAS views	X	
SAS/SHARE®	X	
SAS/CONNECT®	X	
Compression	X	X
Encryption	X	X
Partitioned data		X
Parallel WHERE		X
Separate metadata file		X
Uses multiple indexes		X
Multi-threaded		X
Parallel WHERE optimizations		X

Table 2. Comparison of SPD Engine and Base SAS Engine Features

If you already use the SPD Engine on a traditional file system, or if you use the Base SAS engine using only the features that are also supported by the SPD Engine shown in Table 2, your existing SAS programs should run with minimal changes to the LIBNAME statement.

GETTING SPD ENGINE DATA INTO HADOOP

After you have chosen the SPD Engine for data access with Hadoop, you likely want to move some existing data from another file system to HDFS. Moving your data to the Hadoop cluster using the SPD Engine file format is as simple as using PROC COPY. This example shows how to use PROC COPY to

move data stored in a local Base SAS engine data set to the Hadoop cluster using the SPD Engine file format:

```
libname baselib 'c:\temp';
libname spdelib '/data/sample' hdfshost=default;

proc copy in=baselib out=spdelib;
run;
```

If your existing data is already in the SPD Engine file format on another file system, you must use the SPD Engine to move the data to the Hadoop cluster. File system tools cannot be used to correctly copy SPD Engine tables to the Hadoop cluster for the following reasons:

- Any SPD Engine data processing that occurs on the Hadoop cluster requires complete SPD Engine records within each HDFS block. To guarantee this record configuration, the SPD Engine calculates the appropriate block size to use when creating data sets in HDFS.
- The SPD Engine has a requirement with respect to the directory structure in HDFS, which differs slightly from using the SPD Engine on other file systems. When working with Hadoop, the SPD Engine creates a subdirectory under the data path directory to store the data partition files. By default, the data path directory is the physical path specified in the LIBNAME statement. But the DATAPATH= LIBNAME option can be used to specify an alternate location. A data path subdirectory is created for each SPD Engine data set. The subdirectory is named with the specified data set name and the suffix "_spde". For example, if you create a data set named CLASS in the directory /user/mydata/test, the metadata and index files will be in the directory /user/mydata/test, and the data partition files will be in /user/mydata/test/class_spde.

SPD ENGINE FEATURES YOU NEED TO UNDERSTAND WHEN USING HADOOP

Some SPD Engine features behave differently or have special considerations when running with Hadoop.

WHERE PROCESSING OPTIMIZATION

WHERE processing with the SPD Engine is already partially optimized by taking advantage of multiple threads to process the data. When you are using the SPD Engine with Hadoop, the ACCELWHERE= option can be used for further optimization. This option indicates that the WHERE processing should be done on the Hadoop cluster versus pulling the data back to the SAS client for processing. In many cases, pushing the processing to the cluster can enhance performance. The SPD Engine provides this functionality by dynamically generating a MapReduce job specific for the WHERE processing being requested, and it runs it on the Hadoop cluster, returning only the result set from the job.

As of the second maintenance release of SAS 9.4, the types of WHERE expressions that can be processed with this technique are limited to only one condition with these comparison operators: EQ, NE, GT, LT, GE, or LE. Subsequent releases of the SPD Engine will reduce these restrictions.

Because the setup and use of the MapReduce framework has its own overhead, performance improvements are more likely when the data set is large and the WHERE expression qualifies only a relatively small subset of data. You will need to experiment with your own data to determine at what point using the ACCELWHERE=YES option provides performance gains.

Using the MSGLEVEL=I system option, messages written to the log will indicate if the data sub-setting is occurring on the Hadoop cluster.

PARALLEL DATA ACCESS

Traditional SPD Engine read operations are performed within a single thread. Reading in a single thread allows the engine to preserve row order, which is important in many SAS operations. The general exception to single-threaded input is WHERE processing. SPD Engine WHERE processing is multi-threaded and does not claim to preserve the row order when returning observations.

When using the SPD Engine with Hadoop you can also get multi-threaded or parallel reads by using the SPDEPARALLELREAD= system option, the PARALLELREAD= LIBNAME option, or the PARALLELREAD= data set option. Using one of these options will send the SPD Engine down the same multi-threaded code path as if a WHERE expression were specified. Within each thread, data is transferred concurrently from each data node on the cluster to the SAS client. Due to this implementation, you might see messages that refer to a generated WHERE expression.

There are cases where the SPD Engine can determine that sort order is important. In these cases, the SPD Engine disables the parallel read processing and writes a note to the log indicating that it has been disabled.

There are also cases where the SPD Engine cannot identify that observation order is required. Requesting parallel read processing in these cases results in an error. An example of this case is the use of the POINT= option in the DATA step as shown in Output 1.

```
15  +data tempdata.sample;
16  +   do num=1 to 500 by 25;
17  +     set tempdata.students(parallelread=yes) point=num;
18  +     if _error_ then abort;
19  +     output;
20  +   end ;
21  +   stop;
22  +run;

ERROR: Internal error in the SPD Engine on file STUDENTS_.DATA.  To
circumvent the problem, sort the data and store it in a temporary file.
Then execute the procedure on that temporary file.
```

Output 1. Output from a CREATE TABLE Statement

Additional information about parallel processing can be obtained by setting the MSGLEVEL= I system option.

UPDATE PROCESSING

The SPD Engine does support updating records when the file is stored in HDFS. However, updating is an expensive process and should be done sparingly. HDFS does not natively support updating records. To provide this update functionality, the SPD Engine has to re-create one or more files: any updated data partition files, the metadata file, and any updated index partition files that might exist. This process is repeated per update on a record-by-record basis. Unlike the SPD Engine used with a traditional file system, when used with Hadoop, the SPD Engine supports the use of index partition files to mitigate the cost of file re-creation necessary for updates. Just as with data partition files, only the index partition file affected by the update needs to be re-created.

Another thing to note is that using the SAS DATA step UPDATE or MODIFY statement provides better performance than using PROC SQL. PROC SQL will open, update, and close the file for each updated record, which in turn re-creates the data partition file, the metadata file, and in some cases the index partition files.

CONCURRENT ACCESS TO DATA

HDFS will not allow more than one writer to concurrently access a file. However, it does not have any restrictions on one writer and multiple readers accessing a file concurrently. Allowing a writer and one or more readers to access a file concurrently can cause errors and possibly corrupt files with SPD Engine data. For this reason, the SPD Engine provides the following default file-locking behavior:

- Whenever write access to an SPD Engine data set is granted, a write-lock file is created.
- Whenever the SPD Engine receives a request for read or write access to an SPD Engine data set, the engine first checks to see if a write-lock file exists for that data set. If it does, the data set is already open for write access. Therefore, the request will be denied because a writer gets exclusive access to the data set.
- If the SPD Engine receives a request for read access to a data set and no write lock file exists, then read access is granted.

The problem with this locking strategy comes when a write request is received after one or more read requests have been granted. Because by default the SPD Engine does not create read-lock files, the engine does not know that the data set is already open for read access. Once the writer has concurrent access with one or more readers, the data set can be updated or deleted while the readers are accessing the data set.

To prevent a writer from accessing the data set concurrently with one or more readers, you can specify the environment variable SPDEREADLOCK and set the value to YES. The setting of this environment variable to YES indicates to the SPD Engine that a read-lock file should be created for every read request granted for an SPD Engine data set. By creating a read-lock file, the SPD Engine can prevent a data set from write access when one or more readers have access to the data set.

The reason read lock files are not created by default is two-fold:

- Creation of lock files results in the SPD Engine creating a lock directory under the directory specified by the libname path. Supporting read-locks requires write access to the library, even if you only need read access to the data.
- The likelihood of orphaned lock files increases.
  - Orphaned lock files are SPD Engine lock files that are unintentionally left in existence after a data set has been closed. The files are referred to as orphaned because there is no corresponding open SPD Engine data set. Unfortunately, the SPD Engine cannot tell if a lock file is orphaned or not, and, if one exists, it can block access to that data set for subsequent open requests.
  - Orphaned lock files are usually caused by SAS being abruptly terminated in a way that does not allow the SPD Engine the opportunity to go through the normal data set close sequence. One example of this type of abrupt termination is when the machine that SAS is running on is turned off without first shutting down the SAS process.
  - Because, in many scenarios, read access to a data set is more prevalent than write access, only creating write lock files by default reduces the risk of orphaned lock files. However, if you anticipate multiple users requesting concurrent read and write access to your SPD Engine data sets, you should use the SPDEREADLOCK environment variable to increase the level of protection.

The SPD Engine stores lock files in the following way:

1. The SPD Engine creates a lock directory in the Hadoop cluster. The lock directory has the name of the data set followed by the suffix '_spdslock9', such as BigFile_spdslock9.
2. The lock directory includes the lock files. A lock filename starts with the data set name, followed by .data, either .read or .write, a 24-character hexadecimal identifier, and the suffix .spds_lock9, such as BigFile.data.read013701690063e1151c774922.spds_lock9.

In most situations, you do not see the lock directory because lock files are deleted when the SAS process completes. Orphaned lock files are the exception to this statement. If you end up with an orphaned lock file, you need to delete the file in order to gain read or write access to the SPD Engine data set. In order to safely delete an orphaned lock file, you need to make sure that it is really orphaned and not actually in use by another user accessing the SPD Engine data set. There is no easy way to determine if a lock file has been orphaned. If you are the only one accessing the data set referenced by the lock file, and you do

not have any SAS programs running that would be using the data set, then you can assume it is an orphaned lock file and safely delete it. On the other hand, if there could be other users accessing the data set, then you need to be more diligent in determining if any of those users have SAS programs running that are accessing the data set.

SAS HIGH-PERFORMANCE ANALYTICS

The SPD Engine can work with SAS high-performance analytical procedures whether the data is stored in HDFS or a traditional file system. However, when using SPD Engine with Hadoop, the engine can provide parallel data access to high-performance analytical procedures by working with the SAS High-Performance Analytics infrastructure and the SAS Embedded Process for Hadoop. This data access support allows data to move in parallel between the data nodes of the Hadoop cluster and the compute nodes of the computing appliance, without data being pulled through the SAS client.

There are restrictions that prevent parallel data access when using the SPD Engine with high-performance analytical procedures. When these restrictions are not met, data access using the SPD Engine is still available but will not be parallel access and will be pulled through the SAS client before being distributed to the compute nodes for processing.

SPD Engine restrictions for parallel read access with high-performance analytical procedures are as follows:

- The data set cannot be compressed.
- The data set cannot be encrypted.
- The data set cannot contain deleted records.
- The STARTOBS= and ENDOBS= data set options cannot be specified.

SPD Engine restrictions for parallel write access with high-performance analytical procedures are as follows:

- The SAS client machine must be Linux x64 or Solaris x64.
- The ALIGN=, COMPRESS=, ENCRYPT=, and PADCOMPRESS= data set options cannot be specified.

If parallel data access using the SAS High-Performance Analytics infrastructure and the SAS Embedded Process for Hadoop cannot be supported for any reason, a note will be written to the log, and data access will be provided through the SAS client using the SPD Engine.

When providing parallel data access with high-performance analytical procedures, the SPD Engine uses the temporary directory /tmp as a default location for writing the work files that are needed for the parallel data access process. You can define the environment variable SPDE_HADOOP_WORK_PATH and set its value to a different directory. For the SPD Engine to successfully use this location for its work files, the directory must exist, and you must have write access.

CONCLUSION

The SPD Engine is just one way that SAS provides access to data in Hadoop. One of the biggest benefits to using the SPD Engine is that it provides broad support of SAS features without requiring a lot of changes to existing SAS programs. Adding Hadoop support to the SPD Engine is an important milestone in the SAS strategy to have as many touch-points with the Hadoop ecosystem and at as many levels as possible.

More details on using the SPD Engine with Hadoop, including best practice recommendations, can be found in the SAS Institute technical paper “SAS® SPD Engine and Hadoop Working Together.”

REFERENCES

SAS Institute Inc. 2014. SAS Institute technical paper. "SAS® SPD Engine and Hadoop Working Together." http://support.sas.com/resources/papers/SPDE_Hadoop.pdf.

ACKNOWLEDGMENTS

I would like to extend my gratitude to the folks that helped with this paper: Scott Vance, Jim Craig, Deanna Warner, Miguel Bamberger, Diane Olson, and Jochen Kirsten.

RECOMMENDED READING

- SAS Institute Inc. 2014. *SAS® 9.4 SPD Engine: Storing Data in the Hadoop Distributed File System, Second Edition.* http://support.sas.com/documentation/cdl/en/engspdehdfsug/67403/PDF/default/engspdehdfsug.pdf.
- SAS Institute Inc. 2014. *SAS® SPD Engine and Hadoop Working Together* http://support.sas.com/resources/papers/SPDE_Hadoop.pdf.

CONTACT INFORMATION

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

Lisa Brown
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
Lisa.Brown@sas.com