Paper SAS2180-2016

How to Leverage the Hadoop Distributed File System as a Storage Format for the SAS® Scalable Performance Data Server and the SAS® Scalable Performance Engine

Steven Sober, SAS Institute Inc., Cary, NC

ABSTRACT

The SAS® Scalable Performance Data Server and SAS® Scalable Performance Data Engine are data formats from SAS that supports the creation of analytical base tables with tens of thousands of columns. These analytical base tables are used to support daily predictive analytical routines. Traditionally Storage Area Network (SAN) storage has been and continues to be the primary storage platform for the SAS Scalable Performance Data Server and SAS Scalable Performance Data Engine formats. Due to cost constrains associated with SAN storage, companies have added Hadoop to their environments to help minimize storage costs. In this paper we explore how the SAS Scalable Performance Data Server and SAS Scalable Performance Data Engine leverage the Hadoop Distributed File System (HDFS).

INTRODUCTION

In this paper we will use SPD Server to represent the SAS Scalable Performance Data Server format and SPD Engine to represent the SAS Scalable Performance Data Engine format.

The size of our SPD Server and SPD Engine table that is used in all examples is 100 gigabytes. This table has 204 numeric variables. The table is not encrypted, compressed, or indexed. If we wanted to encrypt, compress, or index the table we would use the corresponding data set options when we create the table.

The Hadoop environment I have access to is small and not a candidate to run official benchmarks on. I am testing with Cloudera - CDH 5.4.5 on 5 Linux Servers (RHEL 6) with 4 of the servers acting as data nodes. Each server has 64GB RAM with Intel® Xeon(R) CPU E7-4880 v2 @ 2.50GHz (4 cores) processors. Each server has 251 gigabytes of local storage and for HDFS we have 3.0 terabytes (NFS mount).

Suffice to say that in the real world Hadoop clusters are much bigger. The customers I have worked with tend to have 100 to over 1,000 cores in their Hadoop environments. In these environments due to the size of data, meaning source and target tables are in the hundreds of gigabytes if not terabytes, it is not possible to bring the data back to SAS to run the query efficiently. Leaving the data on Hadoop gives you two advantages:

- Avoid the overhead of having to move the data from the Hadoop cluster to SAS.
- Use the MapReduce framework on the Hadoop cluster.

In these environments everything we do to transform data must run via MapReduce. To accomplish this we use PROC SQL, FedSQL and the SAS In-Database Code Accelerator for Hadoop. In addition to these three technologies, PROC MEANS, FREQ, REPROT, SUMMARY and TABULATE will be converted to SQL, and when possible, run via the MapReduce framework..

To help understand run times (that is, real time), and the location of the processing of WHERE statements, look for the green and yellow highlighted sections in the following SAS logs. In addition, *italics and the underlining of text* will point out additional important information in the SAS logs.

Format of SAS Scalable Performance Data

Both SPD Server and SPD Engine organize data into a file format that has advantages for a distributed file system like HDFS. Advantages of the file format include the following:

- The file format consists of separate components, a data component, a metadata component, and for indexes there are 2 components. See Figure 1.
- The data is partitioned across multiple data components.
- Compression of data is supported.
- Encryption of data, at rest and in transit, is supported.
- Depending on the amount of data and the partition size, the data can consist of one or more physical files, but is referenced as one logical file.

Format of SPD Server

Advantages of the SPD Server format include the following:

- When the data is not encrypted and uncompressed, WHERE clause pushdown to MapReduce is supported.
- SPD Server Access Control Lists to secure the data is supported.
- SPD Server Cluster Tables, a personal favorite format of mine, are supported.
- SPD Server Cluster Tables WHERE clause pushdown is on the roadmap.

Format of SPD Engine

Advantages of the SPD Engine format include the following:

- When the data is not encrypted and uncompressed, WHERE clause processing can be pushed down to MapReduce.
- When the data is not encrypted and uncompressed, SAS® In-Database Code Accelerator for Hadoop is supported.
- When the data is not encrypted and uncompressed, it will parallel lift the data into the memory of the SAS High-Performance Analytics Server.
- When the data is not encrypted and uncompressed, the Hive SerDe for SPD Engine Data allows other clients Read access to the data.

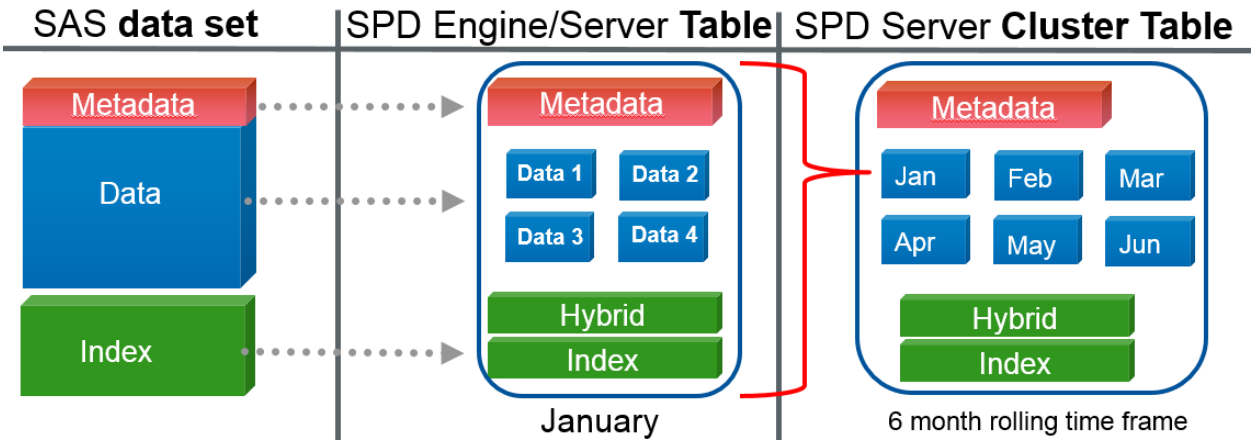

Figure 1. SAS Data Set, SPD Engine Table or SPD Server Table, and SPD Server Cluster Tables

Like SAS data sets, the SPD Server and SPD Engine tables support analytical base tables containing hundreds of thousands of columns. These analytical base tables become source tables to visualization, reporting, and predictive analytical routines.

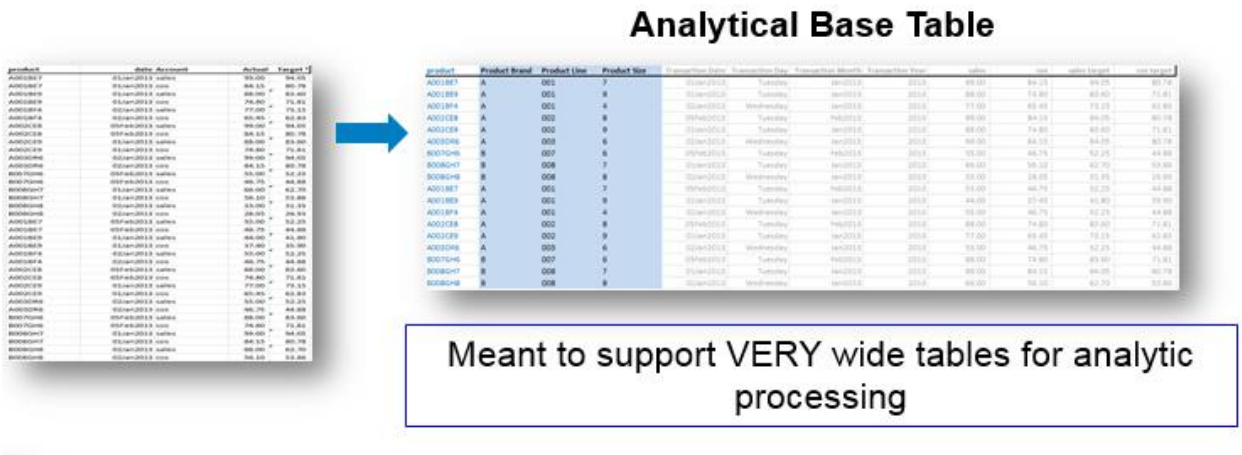

Figure 2. An SPD Engine Table or SPD ServerTable Transformed into an SPD Engine Table or SPD Server Analytical Base Table Using PROC TRANSPOSE

HADOOP DISTIBUTIONS

SPD Server

In SPD Server 5.2, support for the Hadoop Distributed File System (HDFS) was added. Here is the supported Hadoop distribution, with or without Kerberos:

- Cloudera CDH 5.2

SPD Engine

In the third maintenance release for SAS 9.4, SPD Engine, which is delivered with Base SAS, expands the supported Hadoop distributions, with or without Kerberos:

- Cloudera CDH 4.x
- Cloudera CDH 5.x
- Hortonworks HDP 2.x
- IBM InfoSphere - BigInsights 3.x
- MapR 4.x (for Microsoft Windows and Linux operating environments only)
- Pivotal HD 2.x

HOW TO CREATE SAS SCALABLE PERFORMANCE DATA SERVER AND OR ENGINE TABLES ON THE HADOOP DISTRIBUTED FILE SYSTEM

SAS SCALABLE PERFORMANCE DATA ENGINE

SAS Programmers

Let's start by reviewing the options in the following SPD Engine LIBNAME statement:

```
LIBNAME MYSPDE SPDE '/user/sasss1'
      HDFSHOST=DEFAULT
      PARALLELWRITE=YES
      PARALLELREAD=YES
      ACCELWHERE=YES;
```

- *MYSPDE* is the libref we reference in our SAS code to process the SPD Engine data stored on HDFS.
- *SPDE* is the engine SPD Engine uses to process SPD Engine tables.
- '/user/sasss1' is the path on HDFS from which our SPD Engine data is stored.

- *HDFSHOST*=DEFAULT defaults to the Hadoop system that the following SAS/ACCESS Interface to Hadoop OPTIONS point to.
    - options set=SAS_HADOOP_CONFIG_PATH=
    - options set=SAS_HADOOP_JAR_PATH=
- *PARALLELWRITE*=YES tells SPD Engine to use parallel processing to write data to HDFS.
- *PARALLELREAD*=YES tells SPD Engine to use parallel processing to read data stored in HDFS.
- *ACCELWHERE*=YES tells SPD Engine, when possible, to push all WHERE clauses down to Hadoop.

In the third maintenance release for SAS 9.4, you can now request parallel processing for all Write operations in HDFS. A thread is used for each CPU on the SAS client machine. For example, if eight CPUs exist on the SAS client machine, then eight threads are used to write data. To request parallel processing for Write operations, use the PARALLELWRITE= LIBNAME statement option, or the PARALLELWRITE= data set option. Note, the data set option will override the LIBNAME statement option. Also note, there are restrictions when writing data in parallel. For one, we cannot use parallel processing for a Write operation and also request to create an index. Nor can we use parallel processing for a Write operation and also request BY-group processing or sorting.

In the following example PARTSIZE= is the size of the data partition used when creating the table. It is specified in megabytes, gigabytes, or terabytes. If a value is specified without M, G, or T, the default is megabytes. That is, PARTSIZE=64 is the same as PARTSIZE=64M. The default partition size is 128 megabytes and each partition is stored as a separate data component file. In this example we are using a partition size of 10 gigbytes.

Creating an Uncompressed SPD Engine Table:

```
69      +DATA MYSPDE.&source (COMPRESS=NO
70      +                              PARTSIZE=10G);
71      +       ARRAY x{100};
72      +       ARRAY c{100};
73      +       DO i=1 TO 625000 * &Num_GB;
74      +         DO j=1 TO 100;
75      +             x{j}=RANUNI(2);
76      +             c{j}=INT(RANUNI (1)*4);
77      +         END;
78      +         y=INT(RANUNI (1)*2);
79      +         joinvalue=INT(RANUNI (1)*20);
80      +         OUTPUT;
81      +       END;
82      +RUN;
```

INFO: Parallel write processing is being performed using 4 threads.
NOTE: The data set MYSPDE.TESTDATA has 62500000 observations and 204 variables.
NOTE: DATA statement used (Total process time):
real time 6:59.35
cpu time 7:47.34

When we look on HDFS (Figure 3), we see that we have created two items: an SPD Engine metadata file (that is, testdata.mdf.0.0.0.spds9) and a directory that has a naming convention of "_spde" appended to the table name (that is, testdata_spde).

```
[sasss1@eeclxvm109 ~]$ hadoop fs -ls '/user/sasss1/spde'
Found 4 items
drwxr-xr-x   - sasss1 supergroup          0 2016-01-07 13:53 /user/sasss1/spde/sasds2_spd_wye0ho93s44stfz9j
-rw-r--r--   3 sasss1 supergroup      58304 2016-02-03 19:52 /user/sasss1/spde/testdata.mdf.0.0.0.spds9
drwxr-xr-x   - sasss1 supergroup          0 2016-01-07 14:03 /user/sasss1/spde/testdata2_ds2spde
drwxr-xr-x   - sasss1 supergroup          0 2016-02-03 19:52 /user/sasss1/spde/testdata_spde
```

Figure 3. Results of the Hadoop command: hadoop fs –ls /user/sasss1

When we look in the temporary directory used in the creation of the table, /user/sasss1/$000017d003476b777d9f2b3_spde (Figure 4), we see the data partitions (components) for our table.

```
[sasss1@eeclxvm109 ~]$ hadoop fs -ls '/user/sasss1/spde/$0000117d003476b777d9f2b3_spde'
Found 8 items
-rw-r--r--   3 sasss1 supergroup 10737411072 2016-02-03 19:48 /user/sasss1/spde/$0000117d003476b777d9f2b3_spde
/$0000117d003476b777d9f2b3.dpf.00003ccd.0.1.spds9
-rw-r--r--   3 sasss1 supergroup 10737411072 2016-02-03 19:48 /user/sasss1/spde/$0000117d003476b777d9f2b3_spde
/$0000117d003476b777d9f2b3.dpf.00003ccd.1.1.spds9
-rw-r--r--   3 sasss1 supergroup 10737411072 2016-02-03 19:48 /user/sasss1/spde/$0000117d003476b777d9f2b3_spde
/$0000117d003476b777d9f2b3.dpf.00003ccd.2.1.spds9
-rw-r--r--   3 sasss1 supergroup 10737411072 2016-02-03 19:48 /user/sasss1/spde/$0000117d003476b777d9f2b3_spde
/$0000117d003476b777d9f2b3.dpf.00003ccd.3.1.spds9
-rw-r--r--   3 sasss1 supergroup  2607022080 2016-02-03 19:48 /user/sasss1/spde/$0000117d003476b777d9f2b3_spde
/$0000117d003476b777d9f2b3.dpf.00003ccd.4.1.spds9
-rw-r--r--   3 sasss1 supergroup  2607022080 2016-02-03 19:48 /user/sasss1/spde/$0000117d003476b777d9f2b3_spde
/$0000117d003476b777d9f2b3.dpf.00003ccd.5.1.spds9
-rw-r--r--   3 sasss1 supergroup  2607022080 2016-02-03 19:48 /user/sasss1/spde/$0000117d003476b777d9f2b3_spde
/$0000117d003476b777d9f2b3.dpf.00003ccd.6.1.spds9
-rw-r--r--   3 sasss1 supergroup  2607022080 2016-02-03 19:48 /user/sasss1/spde/$0000117d003476b777d9f2b3_spde
/$0000117d003476b777d9f2b3.dpf.00003ccd.7.1.spds9
```

Figure 4. Results of the Hadoop Command: hadoop fs –ls /user/sasss1/$000017d003476b777d9f2b3_spde

You will notice in Figure 4 that there are four partition files that are 10 gigabytes while four other partition files are less than 10 gigabytes. This is because we told the SPD Engine to write the data to HDFS in parallel. The SAS server on which I ran this code has 4 cores, so 4 threads are writing to HDFS in parallel. Eventually we will have 10 partition files that are 10 gigabytes in size.

To improve I/O operation performance, consider setting a different SPD Engine I/O block size. Keep in mind that the larger the block size, the fewer I/O operations when reading the data. For example, when reading a data set, the block size can significantly affect performance. When retrieving a large percentage of the data, a larger block size improves performance. However, when retrieving a subset of the data such as with WHERE processing, a smaller block size performs better. You can specify a different block size with the IOBLOCKSIZE= LIBNAME statement option or the IOBLOCKSIZE= data set option when you create the SPD Engine table.

A note about compression and encryption: both are supported using a data set option. Encryption is for data in transit as well as data at rest. Note that you cannot compress and encrypt a table; you can only compress or encrypt. If you compress or encrypt a table, the SPD Engine table WHERE clause processing cannot be pushed down to MapReduce.

For indexing, we will always benchmark to validate that performance benefits outweigh the cost of (run time (that is, real time) of the following:

- ceating our tables using the LIBNAME and or data set option PARALLELWRITE=YES
- WHERE statements that return a subset of rows

Creating a Compressed SPD Engine Table:

When creating compressed SPD Engine tables, set PARALLELLWRITE=NO. In our example we will simply let it default to NO by leaving it off the LIBNAME statement. Compressed SPD Engine tables can use the PARALLELREAD=YES option as shown in this example:

```
63      +LIBNAME source SPDE '/user/sasss1/spde' HDFSHOST=default ACCELWHERE=YES
PARALLELREAD=YES;
    Engine:      SPDE
    Physical Name: /user/sasss1/spde/
67      +DATA MYSPDE.&source (COMPRESS=BINARY
68      +                              PARTSIZE=10G);
69      +       ARRAY x{100};
```

```
70      +       ARRAY c{100};
71      +       DO i=1 TO 625000 * &Num_GB;
72      +         DO j=1 TO 100;
73      +              x{j}= RANUNI (2);
74      +              c{j}= INT(RANUNI (1)*4);
75      +         END;
76      +         y= INT(RANUNI (1)*2);
77      +         joinvalue=INT(RANUNI(1)*20);
78      +         OUTPUT;
79      +       END;
80      +RUN;
```

NOTE: The data set SOURCE.TESTDATA has 62500000 observations and 204 variables.

NOTE: Compressing data set MYSPDE.TESTDATA decreased size by 33.35 percent.

NOTE: DATA statement used (Total process time):

real time 24:32.81

cpu time 21:59.71

SPD SERVER

SAS Administrator

In order for us to store SPD Server data on HDFS, the SAS administrator needs to add a domain to our LIBNAMES.PARM configuration file as well as a few parameters to the SPDSSERV.PARM configuration file.

Let's review Figure 5. The first three domains point to a cluster file system in a SAS Grid Manager environment. The fourth domain, *LIBNAME=spdshdfs*, is pointing to the HDFS path that we use to store our data. Like a clustered file system or SAN storage, the only Hadoop user ID needing Read, Write, Alter, Control access to the HDFS path is the user ID that started SPD Server.

```
[sas@eeclxvm107 site]$ cat libnames.parm

libname=speedy pathname=/opt/LSF_Share/sas/spds/metadata/speedy
    owner=sasss1 DYNLOCK=yes
    roptions="datapath=('/opt/LSF_Share/sas/spds/data1/speedy'
                        '/opt/LSF_Share/sas/spds/data2/speedy')
              indexpath=('/opt/LSF_Share/sas/spds/indexes/speedy')
              ";

libname=dynamic pathname=/opt/LSF_Share/sas/spds/metadata/dynamic
    owner=sasss1 DYNLOCK=yes
    roptions="datapath=('/opt/LSF_Share/sas/spds/data1/dynamic'
                        '/opt/LSF_Share/sas/spds/data2/dynamic')
              indexpath=('/opt/LSF_Share/sas/spds/indexes/dynamic')
              ";

libname=user pathname=/opt/LSF_Share/sas/spds/metadata/user
    owner=sasss1
    roptions="datapath=('/opt/LSF_Share/sas/spds/data1/user'
                        '/opt/LSF_Share/sas/spds/data2/user')
              indexpath=('/opt/LSF_Share/sas/spds/indexes/user')
              ";

libname=spdshdfs pathname=/user/sas/spds/speedy HADOOP=YES;
```

Figure 5. LIBNAMES.PARM

Let's review the parameters of the SPDSERV.PARM file in Figure 6.

- HADOOPXML=- and HADOOPJAR= point to the XML and JAR files that were created when we ran the SAS Software Deployment wizard for SAS/ACCESS Interface to Hadoop.
- HADOOPWORKPATH= specifies the path to the directory in the Hadoop Distributed File System that stores the temporary results of the MapReduce output.
- HADOOPACCELJVER= is the version of Java on the Hadoop cluster. Valid values are 1.6, 1.7, and 1.8.
- HADOOPACCELWH will attempt to push all SPD Server WHERE clauses to MapReduce.

```
#
# pointers to hdfs
#
HADOOPCFG=/opt/sasinside/XMLS;
HADOOPJAR=/opt/sasinside/JARS;
HADOOPWORKPATH=/user/sas/spds/hadoopworkpath;
#
# requried for WHERE push down to MapReduce
#
HADOOPACCELJVER=1.7;
HADOOPACCELWH;
```

Figure 6. SPDSERV.PARM

SAS Programmers

We will start by reviewing our SPD Server LIBNAME statement:

```
83      +LIBNAME MYSPDS  SASSPDS IP=YES DISCONNECT=YES LIBGEN=YES
```

```
84      +  HOST='localhost'
85      +  SERV='5400'
86      +  SCHEMA='spdshdfs'
87      +  USER='sasss1'
88      +  PASSWORD=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
89      +  ;
```

NOTE: User sasss1(ACL Group ADMGRP) connected to SPD(LAX) 5.2 HF(1) server at 10.36.150.120.

NOTE: Libref MYSPDS was successfully assigned as follows:

Engine: SASSPDS

Physical Name: :29121/user/sas/spds/speedy/

- *MYSPDS* is the libref we will reference in our SAS code to process the SPD Server data stored on HDFS.
- *SASSPDS* is the engine SPD Server uses to process data stored on HDFS.
- HOST='localhost' is the name of the computer where SPD Server is running.
- *SCHEMA*='spdshdfs' references the SPD Server domain defined to LIBNAMES.PARM (See Figure 5). The path defined to this domain is the location on HDFS where we will store our SPD Server data.
- *USER*= is our SPD Server user ID.
- *PASSWORD*= is our SPD Server password.
- *IP*=YES will attempt to push all PROC SQL down to SPD Server for processing.
- *DISCONNECT*=YES specifies when to close network connections between the SAS client and the SPD Server. Closure can occur either after all librefs are cleared or at the end of a SAS session.
- *LIBGEN*=YES option is used in explicit SQL connection statements. When you set LIBGEN= yes, SPD Server is configured to generate additional domain connections that enable you to perform SQL joins across different SPD Server domains.

Creating a Compressed SPD Server Table:

```
91      +DATA MYSPDS.&source (COMPRESS=BINARY);
92      +       ARRAY x{100};
93      +       ARRAY c{100};
94      +       DO i=1 TO 625000 * &Num_GB;
95      +         DO j=1 TO 100;
96      +              x{j}=RANUNI(2);
97      +              c{j}= INT(RANUNI (1)*4);
98      +         END;
99      +         y= INT(RANUNI (1)*2);
100     +          joinvalue=INT(RANUNI (1)*20);
101     +          OUTPUT;
102     +       END;
103     +RUN;
```

NOTE: The data set MYSPDS.TESTDATA has 62500000 observations and 204 variables.

NOTE: Compressing data set MYSPDS.TESTDATA decreased size by 33.35 percent.

NOTE: DATA statement used (Total process time):

real time 24:20.03

cpu time 10:26.28

Creating an Uncompressed SPD Server Table Using PROC APPEND:

```
92          +PROC APPEND BASE=MYSPDS.TESTDATA data=BDAT.TESTDATA;
93          +RUN;

NOTE: Appending BDAT.TESTDATA to MYSPDS.TESTDATA.
NOTE: BASE data set does not exist. DATA file is being copied to BASE file.
INFO: Engine's block-read  method cannot be used because:
INFO:   -  The file is compressed
NOTE: There were 62500000 observations read from the data set BDAT.TESTDATA.
NOTE: The data set MYSPDS.TESTDATA has 62500000 observations and 204
variables.
NOTE: PROCEDURE APPEND used (Total process time):

      real time             11:34.68
      cpu time              9:29.99
```

HOW TO READ SAS SCALABLE PERFORMANCE DATA SERVER AND OR ENGINE TABLES ON THE HADOOP DISTRIBUTED FILE SYSTEM

I'll take you through an example whereby the size of our SPD Server and SPD Engine table is 100GB and all WHERE statements return 14% of the data. To validate the processing location of WHERE statements we will set the SPD Server and SPD Engine macro SPDSWDEB to YES.

SAS SCALABLE PERFORMANCE DATA ENGINE

Uncompressed data

When SPD Engine data is not compressed, we can push our WHERE statements down to MapReduce by ensuring that the SPD Engine LIBNAME option ACCCELWHERE= is set to YES. To validate the location of the processing of WHERE statements, set the SPD Engine macro SPDSWDEB=YES. If you set SPDSWDEB=YES, the SAS log provides information about the processing location of WHERE statements:

```
12      +%LET SPDSWDEB=YES; *Validates the processing location of WHERE statements;

80      +DATA MYSPDE.&source.3;
81      +   SET MYSPDE.&source;
82      +   WHERE joinvalue < 3;

whinit: WHERE (joinvalue<3)
whinit returns: ALL EVAL2
83      +RUN;

WHERE processing is optimized on the Hadoop cluster
Hadoop Job ID: job_1450299322397_0061
INFO: Parallel write processing is being performed using 4 threads.
```

```
NOTE: There were 9375485 observations read from the data set MYSPDE.TESTDATA.
      WHERE joinvalue<3;
NOTE: The data set MYSPDE.TESTDATA3 has 9375485 observations and 204 variables.
NOTE: DATA statement used (Total process time):
      real time           8:55.79
      cpu time            26.11 seconds


86         +PROC SQL;
87         +   CREATE TABLE MYSPDE.&source._sql AS
88         +   SELECT x1, x3, x5, joinvalue FROM MYSPDE.&source
89         +   WHERE joinvalue < 3;

whinit: WHERE (joinvalue<3)
whinit returns: ALL EVAL2
WHERE processing is optimized on the Hadoop cluster
Hadoop Job ID: job_1450299322397_0062
INFO: Parallel write processing is being performed using 4 threads.
NOTE: Table MYSPDE.TESTDATA_SQL created, with 9375485 rows and 4 columns.

90         +QUIT;
NOTE: PROCEDURE SQL used (Total process time):
      real time           7:45.81
      cpu time            11.79 seconds
```

When SPD Engine data is not compressed or encrypted, we can also process that data using SAS In-Database Code Accelerator for Hadoop. SAS In-Database Code Accelerator for Hadoop is part of the bundle of software you get when you license SAS® Data Loader for Hadoop. The SAS In-Database Code Accelerator for Hadoop uses the SAS DS2 language to run DS2 code in a MapReduce framework. To accomplish this, we create a DS2 THREAD program that contains our source table(s) and the business rules we need to apply to that data. To execute that THREAD program in parallel on every data node in your Hadoop cluster, we declare it to the DS2 DATA program and then execute it using the SET statement of that DATA program. To validate the PROC DS2 code ran in a MapReduce framework we will set the SAS option MSGLEVEL to I:

```
34         +OPTIONS MSGLEVEL=I;

104        +PROC DS2;
105        +THREAD work.thread / OVERWRITE=YES;
106        +DCL DOUBLE count;
107        +METHOD RUN ();
108        +   SET myspde.&source;
109        +   BY joinvalue;
110        +   IF FIRST.joinvalue THEN count = 0;
111        +   count + 1;
112        +   IF LAST.joinvalue;
113        +END;
114        +ENDTHREAD;
115        +DATA MYSPDE.&source.2 (OVERWRITE=YES);
116        +DECLARE THREAD work.thread thrd;
117        +METHOD RUN ();
118        +   SET FROM thrd;
119        +END;
120        +ENDDATA;
121        +RUN;
```

```
NOTE: Created thread thread in data set work.thread.
NOTE: Running THREAD program in-database
NOTE: Running DATA program in-database
NOTE: Execution succeeded. No rows affected.
122      +QUIT;

NOTE: PROCEDURE DS2 used (Total process time):
      real time           20:39.18
      cpu time            2.59 seconds
```

When SPD Engine data is not compressed, all SAS High-Performance procedures will lift that data, in parallel, into the memory of the SAS High-Performance Analytics Server:

```
128      +PROC HPLOGISTIC DATA = MYSPDE.&source;
129      +PERFORMACE NODES=ALL DETAILS;
130      +CLASS j / PARAM= GLM;
131      +          WEIGHT c98 ;
132      +MODEL y (EVENT = "0" ) = /  LINK= LOGIT;
133      +RUN;

NOTE: No explanatory variables have been specified.
INFO: Read the content of utility file /opt/sasinside/XMLS/core-site.xml.
INFO: Read the content of utility file /opt/sasinside/XMLS/hdfs-site.xml.
INFO: Read the content of utility file /opt/sasinside/XMLS/mapred-site.xml.
INFO: Read the content of utility file /opt/sasinside/XMLS/yarn-site.xml.
NOTE: The HPLOGISTIC procedure is executing in the distributed computing environment with 4 worker
nodes.
NOTE: You are modeling the probability that y='0'.
NOTE: Convergence criterion (GCONV=1E-8) satisfied.
NOTE: The PROCEDURE HPLOGISTIC printed pages 7-8.
NOTE: PROCEDURE HPLOGISTIC used (Total process time):
      real time           1:59.46
      cpu time            3.29 seconds
```

Compressed data

When the SPD Engine data is compressed or encrypted, we cannot push our WHERE statements down to MapReduce. In my testing, the real time is faster than pushing that WHERE statement down to MapReduce. This is more of a reflection on the small Hadoop cluster and data size I am using in our testing, and this might not be the case with a large Hadoop cluster. For the DATA step test, the compressed real time was 3:38.48 quicker. For the PROC SQL test, the compressed real time was 3:18.39 quicker:

```
87       +DATA MYSPDE.&source.3;
88       +   SET MYSPDE.&source;
89       +   WHERE joinvalue < 3;

whinit: WHERE (joinvalue<3)
The data set is compressed
WHERE processing cannot be optimized on the Hadoop cluster
whinit returns: ALL EVAL2
90       +RUN;

NOTE: There were 9375485 observations read from the data set MYSPDE.TESTDATA.
      WHERE joinvalue<3;
NOTE: The data set MYSPDE.TESTDATA3 has 9375485 observations and 204 variables.
```

```
NOTE: DATA statement used (Total process time):
      real time           5:17.31
      cpu time            5:06.40

93         +PROC SQL;
94         +   CREATE TABLE MYSPDE.&source._sql AS
95         +   SELECT x1, x3, x5, joinvalue FROM MYSPDE.&source
96         +   WHERE joinvalue < 3;

whinit: WHERE (joinvalue<3)
The data set is compressed
WHERE processing cannot be optimized on the Hadoop cluster
whinit returns: ALL EVAL2
NOTE: Table MYSPDE.TESTDATA_SQL created, with 9375485 rows and 4 columns.

97         +QUIT;
NOTE: PROCEDURE SQL used (Total process time):
      real time           4:27.42
      cpu time            4:45.59
```

When the SPD Engine data is compressed, we cannot process that data using the MapReduce framework using DS2 code and the SAS In-Database Code Accelerator for Hadoop:

```
111        +PROC DS2;
112        +THREAD work.thread / OVERWRITE=YES;
113        +DCL DOUBLE count;
114        +METHOD RUN ();
115        +   SET MYSPDE.&source;
116        +   BY joinvalue;
117        +   IF FIRST.joinvalue THEN count = 0;
118        +   count + 1;
119        +   IF LAST.joinvalue;
120        +END;
121        +ENDTHREAD;
122        +DATA MYSPDE.&source.2 (overwrite=yes);
123        +DCL THREAD work.thread thrd;
124        +METHOD RUN ();
125        +   SET FROM thrd;
126        +END;
127        +ENDDATA;
128        +RUN;

NOTE: Created thread thread in data set work.thread.
NOTE: Running THREAD program in-database
NOTE: Running DATA program in-database
ERROR: Failed to run DS2INDB.
ERROR: The path was not found: /user/sasss1/spde//sasds2_spd_k2s8ho_es44t8bpnr.
ERROR: Error returned from tkedsPubINDBDS2.
NOTE: Execution succeeded. No rows affected.
129        +QUIT;

NOTE: PROCEDURE DS2 used (Total process time):
      real time           7.56 seconds
      cpu time            1.48 seconds
```

When the SPD Engine data is compressed, SAS High-Performance procedures will “front load” the data into the memory of the SAS High-Performance Analytics Server. By “front load” we mean the data must leave the Hadoop cluster and flow through the SAS Server into the memory of the SAS High-Performance Analytics Server. In this case, the compressed real time was 8:56.43 slower.

```
135     +PROC HPLOGISTIC DATA = MYSPDE.&source;
136     +PERFORMACE NODES = ALL DETAILS;
137     +CLASS j / PARAM= GLM;
138     +       WEIGHT c98 ;
139     +MODEL y (EVENT = "0" ) = /  LINK= LOGIT;
140     +RUN;

NOTE: No explanatory variables have been specified.
NOTE: Data set MYSPDE.TESTDATA cannot be processed in the Embedded Process Environment
because the data set is either encrypted or compressed.
NOTE: The data MYSPDE.TESTDATA are being routed through the client.
NOTE: The HPLOGISTIC procedure is executing in the distributed computing environment with 4 worker
nodes.
NOTE: You are modeling the probability that y='0'.
NOTE: Convergence criterion (GCONV=1E-8) satisfied.
NOTE: There were 62500000 observations read from the data set MYSPDE.TESTDATA.
NOTE: The PROCEDURE HPLOGISTIC printed pages 3-4.
NOTE: PROCEDURE HPLOGISTIC used (Total process time):
      real time           10:15.89
      cpu time            4:23.25
```

Hive SerDe for SPD Engine Data

In the third maintenance release for SAS 9.4, SAS provides a custom Hive SerDe for SPD Engine Data that is stored in HDFS. The SerDe makes the data available for applications outside of SAS to query.

The following are required to access SPD Engine tables using the SPD Engine SerDe:

- You must deploy SAS Foundation using the SAS Deployment Wizard. Select SAS Hive SerDe for SPDE Data.
- You must be running a supported Hadoop distribution that includes Hive 0.13:
    - Cloudera CDH 5.2
    - Hortonworks HDP 2.1 or later
    - MapR 4.0.2 or later

The following table features are not supported:

- compressed or encrypted tables
- tables with SAS informats
- tables that have user-defined formats
- password-protected tables
- tables owned by the SAS Scalable Performance Data Server

In addition, the following processing functionality is not supported by the SerDe and requires processing by the SPD Engine:

- Write, Update, and Append operations
- If preserving observation order is required.

SAS Administrators

Log on to the Hadoop cluster head node. To register the SPD Engine tables to the Hive metastore we need to append to the HADOOP_CLASSPATH the SAS_HADOOP_CONFIG_PATH as well as the

directory paths for the Hive JAR files and XMLs files:

export HADOOP_CLASSPATH=$HADOOP_CLASSPATH:$SAS_HADOOP_CONFIG_PATH:opt/cloudera/parcels/CDH/lib/hive/lib/*:/opt/cloudera/paels/CDH/lib/hadoop-mapreduce/lib/*

In the following two examples we are registering the SPD Engine tables "players and stats" to the "sasss1" database. The following Hadoop commands executes the SerDe JAR files and registers the SPD Engine tables to the Hive metastore:

hadoop jar /opt/cloudera/parcels/CDH/lib/hive/lib/sas.HiveSerdeSPDE.jar com.sas.hadoop.serde.sp.hive.MetastoreRegistration -table players -mdflocation /user/sasss1/spde -database sasss1

hadoop jar /opt/cloudera/parcels/CDH/lib/hive/lib/sas.HiveSerdeSPDE.jar com.sas.hadoop.serde.sp.hive.MetastoreRegistration -table stats -mdflocation /user/sasss1/spde -database sasss1

The following options are also supported:

-database database_name to specifies a Hive metastore database if you are not using the default database.

-mdflocation is the path on HDFS that we used on our SPD Engine LIBNAME.

-renametable table-name to assigns a different name to the table stored in HDFS.

-owner owner-name that assigns an owner name. The default owner name is Anonymous.

SAS Programmers

In this example, we create a new Hive table by joining the two SPD Engine tables that are registered to the Hive metastore:

```
58      +/* HIVE LIBREF */
59      +LIBNAME HIVE HADOOP PORT=10000  TRANSCODE_FAILl=warning
HOST="eeclxvm109.unx.sas.com"   DATABASE=sasss1  USER=sasss1;
```

NOTE: Libref HIVE was successfully assigned as follows:

Engine: HADOOP

Physical Name: jdbc:hive2://eeclxvm109.unx.sas.com:10000/sasss1

```
102     +PROC SQL;
103     +   CREATE TABLE HIVE.PLAYER_STATS AS
104     +   SELECT t1.Name,
105     +          t1.Team,
106     +          t2.nHits,
107     +          t2.nRuns
108     +      FROM HIVE.PLAYERS t1
109     +           INNER JOIN HIVE.STATS t2 ON (t1.Name = t2.Name);
```

NOTE: SAS variable labels, formats, and lengths are not written to DBMS tables.

NOTE: Table HIVE.PLAYER_STATS created, with 322 rows and 4 columns.

HADOOP_23: Executed: on connection 4

DROP TABLE sasdata_17_43_43_325_00001

Summary Statistics for HADOOP are:

Total SQL execution seconds were: 0.137603

Total SQL prepare seconds were: 16.867544

Total SQL describe seconds were: 0.002308

Total seconds used by the HADOOP ACCESS engine were 19.764125

```
110      +QUIT;
```

NOTE: PROCEDURE SQL used (Total process time):

real time 20.61 seconds

cpu time 0.17 seconds

SAS SCALABLE PERFORMANCE DATA SERVER

Uncompressed data

When SPD Server data is not compressed or encrypted, we can push our WHERE statements down to MapReduce by ensuring that the SAS administrator has added the parameter HADOOPACCELWH to the SPDSERV.PARM configuration file. Like SPD Engine, we can validate where the WHERE statements are processed by setting the SPD Server macro SPDSWDEB=YES:

```
105      +DATA MYSPDS.&source.3;
106      +   SET MYSPDS.&source;
107      +   WHERE joinvalue < 3;
whinit: WHERE (joinvalue<3)
whinit: wh-tree presented
        /-NAME = [joinvalue]
 --CLT----|
        \-LITN = [3]
```

WHERE processing Hadoop cluster optimization candidate

```
whinit returns: ALL EVAL6
108      +RUN;
```

WHERE processing is optimized on the Hadoop cluster

NOTE: There were 9375485 observations read from the data set SOURCE.TESTDATA.

WHERE joinvalue<3;

NOTE: The data set SOURCE.TESTDATA3 has 9375485 observations and 204 variables.

NOTE: DATA statement used (Total process time):

real time 10:55.34

cpu time 58.80 seconds

```
111      +PROC SQL;
112      +   CREATE TABLE MYSPDS.&source._sql AS
113      +   SELECT x1, x3, x5, joinvalue FROM source.&source
```

```
114      +   WHERE joinvalue < 3;
whinit: WHERE (joinvalue<3)
whinit: wh-tree presented
        /-NAME = [joinvalue]
 --CLT----|
        \-LITN = [3]
```

WHERE processing Hadoop cluster optimization candidate

```
whinit returns: ALL EVAL6
```

WHERE processing is optimized on the Hadoop cluster

```
NOTE: Table SOURCE.TESTDATA_SQL created, with 9375485 rows and 4 columns.
115      +QUIT;
NOTE: PROCEDURE SQL used (Total process time):
    real time          9:36.21
    cpu time           2.72 seconds
```

Because SPD Server is not supported by the SAS In-Database Code Accelerator for Hadoop, we cannot run our PROC DS2 code.

Like SPD Engine compressed or encrypted data, all SAS In-Memory technologies will “front load” the SPD Server data into the SAS High-Performance Analytics Server:

```
153      +PROC HPLOGISTIC DATA=MYSPDS.&source;
154      +PERFORMANCE NODES=ALL DETAILS;
155      +CLASS j / PARAM= GLM;
156      +     WEIGHT c98 ;
157      +MODEL y (EVENT = "0" ) = /  LINK= LOGIT;
158      +RUN;


NOTE: No explanatory variables have been specified.
```

NOTE: The HPLOGISTIC procedure is executing in the distributed computing environment with 4 worker nodes.

```
NOTE: You are modeling the probability that y='0'.
NOTE: Convergence criterion (GCONV=1E-8) satisfied.
NOTE: There were 62500000 observations read from the data set MYSPDS.TESTDATA.
NOTE: The PROCEDURE HPLOGISTIC printed pages 3-4.
NOTE: PROCEDURE HPLOGISTIC used (Total process time):
    real time          12:05.23
    cpu time           10.08 seconds
```

By comparing PROC HPLOGISTIC real time for our compressed SPD Server table to the real time for our uncompressed SPD Engine table, we realize the cost of front loading our compressed SPD Server real time as 10:05.77 seconds. When comparing PROC HPLOGISTIC real time for our compressed and uncompressed SPD Server table we see our uncompressed table delta in real time is 00:00.12.1

seconds.

Compressed data

When SPD Server data is compressed, we cannot push our WHERE statements down to MapReduce. But like we saw with SPD Engine, this is not a concern. For the DATA step test, the compressed real time was 04:34.05 quicker. For the PROC SQL test, the compressed real time was 04:37.70 quicker:

```
108      +DATA MYSPDS.&source.3;
109      +   SET MYSPDS.&source;
110      +   WHERE joinvalue < 3;
whinit: WHERE (joinvalue<3)
whinit: wh-tree presented
         /-NAME = [joinvalue]
 --CLT----|
         \-LITN = [3]
whinitj: cannot do where pushdown; table compressed
whinit returns: ALL EVAL2
111      +RUN;
NOTE: There were 9375485 observations read from the data set MYSPDS.TESTDATA.
     WHERE joinvalue<3;
NOTE: The data set MYSPDS.TESTDATA3 has 9375485 observations and 204 variables.
NOTE: Compressing data set MYSPDS.TESTDATA3 decreased size by 33.37 percent.
NOTE: DATA statement used (Total process time):
     real time          6:21.29
     cpu time           1:00.19
114      +PROC SQL;
115      +   CREATE TABLE MYSPDS.&source._sql AS
116      +   SELECT x1, x3, x5, joinvalue FROM MYSPDS.&source
117      +   WHERE joinvalue < 3;
whinit: WHERE (joinvalue<3)
whinit: wh-tree presented
         /-NAME = [joinvalue]
 --CLT----|
         \-LITN = [3]
whinitj: cannot do where pushdown; table compressed
whinit returns: ALL EVAL2
NOTE: Compressing data set MYSPDS.TESTDATA_SQL decreased size by 11.12 percent.
NOTE: Table MYSPDS.TESTDATA_SQL created, with 9375485 rows and 4 columns.
118      +QUIT;
NOTE: PROCEDURE SQL used (Total process time):
```

real time 4:58.51

cpu time 2.77 seconds

For our PROC HPLOGISTIC test, the compressed SPD Server table real time of 00:10.03.81 was 00:02:01.42 quicker when compared to uncompressed SPD Server real time of 00:12:05.23:

```
156     +PROC HPLOGISTIC DATA=MYSPDS.&source;
157     +PERFORMACE NODES=ALL DETAILS;
158     +CLASS j / PARAM= GLM;
159     +     WEIGHT c98 ;
160     +MODEL y (EVENT = "0" ) = /  LINK= LOGIT;
161     +RUN;
```

NOTE: No explanatory variables have been specified.

NOTE: The HPLOGISTIC procedure is executing in the distributed computing environment with 4 worker nodes.

NOTE: You are modeling the probability that y='0'.

NOTE: Convergence criterion (GCONV=1E-8) satisfied.

NOTE: There were 62500000 observations read from the data set MYSPDS.TESTDATA.

NOTE: The PROCEDURE HPLOGISTIC printed pages 3-4.

NOTE: PROCEDURE HPLOGISTIC used (Total process time):

real time 10:03.81

cpu time 9.76 seconds

CONCLUSION

To compress or not to compress SPD Server and or SPD Engine tables that are stored on HDFS - that is the question.

SPD SERVER

In my testing I have concluded it is best to compress SPD Server tables. The reason for this is that there is a savings in the amount of HDFS storage needed. In our example of a 100-gigabyte table with 204 numeric values, the compressed table is 33% smaller.

SPD ENGINE

The answer depends on how you plan to process that data:

Compressed SPD Engine Table

- There is a savings in the amount of HDFS storage needed.
- PARALLELREAD is supported.

Uncompressed SPD Engine Table

- PARELLELWRITE is supported.
- PARELLELREAD is supported.
- Hive Serde is supported.
- If you have licensed SAS High-Performance procedures, SAS® Visual Analytics, SAS® Visual Statistics, or SAS® IMSTAT, the following items are true:
    - You will have quicker run times (that is, real time) for your SAS High-Performance procedures.
    - For SAS Visual Analytics, SAS Visual Statistics, and SAS IMSTAT, it will lift data in to memory quicker.
- It you have licensed SAS In-Database Code Accelerator for Hadoop, you will be able to run DS2 code in the MapReduce framework when staging your data for visualization, reporting, and predictive analytics.

REFERENCES

Simpson, G. 2014 “Managing Large Data with SAS® Dynamic Cluster Table Transactions.” *Proceedings of the SAS Global Forum 2014 Conference*. Cary, NC: SAS Institute Inc. Available http://support.sas.com/resources/papers/proceedings14/SAS255-2014.pdf

ACKNOWLEDGMENTS

I would like to thank Guy Simpson, Lisa Brown, Brian Kinnebrew, Clark Bradley, and Lisa Dodson for their insights and support.

RECOMMENDED READING

- *SAS 9.4 SPD Engine: Storing Data in the Hadoop Distributed File System, Third Edition*
- SAS(R) 9.4 In-Database Products: User's Guide, Sixth Edition
- *SAS 9.4 DS2 Language Reference, Fifth Edition*

CONTACT INFORMATION

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

Steven Sober
SAS Institute Inc.
919 531 9644
Steven.Sober@sas.com
http://www.sas.com