SAS5060-2016

Exploring SAS® Embedded Process Technologies on Hadoop®

David Ghazaleh, SAS Institute Inc., Cary, NC

ABSTRACT

SAS® Embedded Process offers a flexible, efficient way to leverage increasing amounts of data by injecting the processing power of SAS® where ever the data lives. SAS Embedded Process can tap into the massively parallel processing (MPP) architecture of Hadoop for scalable performance. Using SAS in-database technologies for Hadoop, you can run scoring models generated by SAS® Enterprise Miner™ or, with SAS® In-Database Code Accelerator for Hadoop, user-written DS2 programs in parallel. With SAS Embedded Process on Hadoop you can also perform data quality operations, and extract and transform data using SAS® Data Loader. This paper explores key SAS technologies that run inside the Hadoop parallel processing framework and prepares you to get started with SAS In-Database Code Accelerator and Scoring Accelerator for Hadoop.

INTRODUCTION

Companies are storing all the data they can possibly collect. The volume of data is so large that it becomes impossible to store in one single hard disk or perhaps even a computer. The question becomes: how are you going to process all that data in an effective and efficient manner? SAS Embedded Process technologies on Hadoop is the perfect platform to process your big data. It allows you to combine SAS intelligence and Hadoop processing power.

Apache™ Hadoop allows massive amounts of data to be stored on a cluster of commodity hardware providing an open-source distributed file system and parallel processing framework. In addition to being fault tolerant, Hadoop is developed for low cost, fast, and efficient massively parallelized data manipulation. Multiple copies of the data are automatically stored across all nodes of the cluster. Data processing is also protected against hardware failures. If a processing node goes down, tasks are automatically restarted on another node of the cluster. Hadoop is also scalable. You can virtually have limitless processing power by simply adding more computing nodes to the cluster.

SAS inside Hadoop provides a platform that supports the following functionalities at massive scale on distributed computing environment:

- SAS In-Database Code Accelerator
- SAS Data Quality Accelerator
- SAS Scoring Accelerator
- SAS® Data Loader
- SAS® High-Performance Analytics

Applying the process to the data eliminates data movement and decreases overall processing time. Information becomes more secure because the data never leaves the cluster.

SAS Embedded Process is the core of SAS in-database products. It allows the parallel execution of SAS processes inside Hadoop and many other databases. SAS Embedded Process technology is a portable, lightweight, execution container for SAS DS2 code. SAS Embedded Process is orchestrated by Hadoop MapReduce framework. Load balancing and resources allocation are managed by YARN.

DS2 is a procedural programming language influenced by the SAS DATA step. The DS2 language is mainly focused in parallel execution. Most DATA step functions can be called from a DS2 program. DS2 programs can run in the Base SAS language interface using PROC DS2, SAS High-Performance Analytics, SAS In-Database Scoring Accelerator, SAS In-Database Code Accelerator, and SAS® In-Memory Analytics. The parallel syntax of the DS2 language coupled with SAS Embedded Process allows traditional SAS developers to create portable algorithms that are implicitly executed inside Hadoop.

There are many benefits in bringing together big data, Hadoop processing power, and the intelligence offered by SAS, including:

- Greater storage capabilities. Store all the data you can collect. Experience the maximum accuracy of larger data.
- Greater parallel processing capabilities. Write more complex algorithms to obtain more precise results.
- Faster data growth and processing time. Maximizing and expanding the value of Hadoop across the enterprise is essential and desired. SAS Embedded Process is as scalable as your Hadoop cluster.
- Data management and integration in order to promote broad reuse while being compliant with Information Technology policies and procedures.
- Boost the value of analytics infrastructure while reducing the cost to maintain it.

PREPARING BASE SAS TO ACCESS HADOOP

Before you can use any SAS accelerator, you need to prepare the machine from where you run Base SAS. Access to Hadoop is done through the Java client API provided by the Hadoop distribution you are using.

When interacting with Hadoop, Base SAS starts an auxiliary process called Java Proxy (JProxy). The JProxy process boots a Java Virtual Machine inside it. Base SAS and the JProxy process communicate with each other using inter-process communication resources such as semaphores and shared memory segments. Base SAS can also access Hadoop Distributed File System (HDFS) services through RESTful web services offered by WebHDFS. In this case JProxy is not used. JProxy is used when access to Hadoop requires a Java client API, such as MapReduce/YARN and Hive JDBC Driver.

Figure 1 illustrates the way Base SAS communicates with a Hadoop cluster through JProxy process.

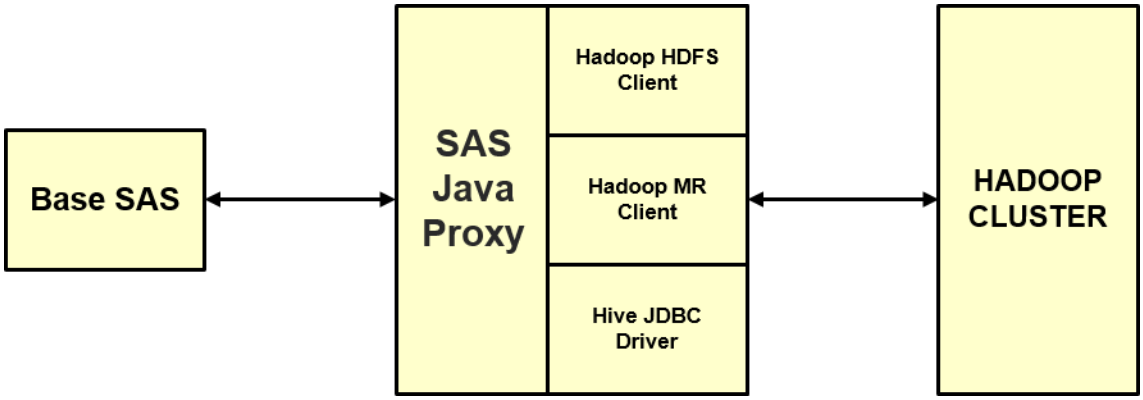

Figure 1. SAS Java Proxy to Hadoop Service Components

SETTING SAS ENVIRONMENT VARIABLES

The Hadoop Java client API used to access Hadoop services is provided by your Hadoop vendor in the form of JAR files. A JAR file is a compressed collections of Java classes. The Hadoop JAR files contain the Java application code deployed to the user's workstation to enable SAS to connect to Hadoop. The JAR files are specific to the version of the Hadoop distribution and the Hadoop components that you are using. Each vendor might provide a different set of Java client API JAR files. SAS does not provide any vendor's specific set of JAR files. Some SAS technologies require that you perform steps to make the Hadoop distribution JAR files accessible to the SAS client machine.

Hadoop services configuration files also need to be available on the client machine. The configuration files must be extracted from the cluster. The method of extraction will depend on the Hadoop vendor you are using. Some vendors provide an administrative console to extract configuration files. You can also

collect the configuration files directly from the local file system in one of the Hadoop nodes. The configuration files are usually stored under */etc/hadoop/conf*. Hive configuration files can be found under */etc/hive/conf*. The basic configuration files you need to collect are: *core-site.xml*, *hdfs-site.xml*, *mapred-site.xml*, *hive-site.xml* and, if applicable, *yarn-site.xml*. The configuration files are used by the Hadoop Java client API to find the Hadoop cluster and related services. The configuration files must be available on the user's workstation under a folder that is accessible by Base SAS.

Once you have collected the Hadoop Java client API JAR files and Hadoop services' configuration files, you must set two environment variable:

1) SAS_HADOOP_JAR_PATH: specifies the location that contains all the Hadoop Java client API JAR files. The environment variable can also be set by using an OPTION SET statement. Here is an example of how to set this environment variable:

```
options set=SAS_HADOOP_JAR_PATH="C:\Hadoop\jars";
```

2) SAS_HADOOP_CONFIG_PATH: specifies the location of the Hadoop services configuration files. The environment variable can also be set by using an OPTION SET statement. Here is an example of how to set this environment variable:

```
options set=SAS_HADOOP_CONFIG_PATH="C:\Hadoop\conf";
```

USING SAS/ACCESS INTERFACE TO CONNECT TO HADOOP

SAS in-database technologies work in conjunction with the SAS/ACCESS Interface to Hadoop. SAS/ACCESS Interface to Hadoop provides enterprise data access and integration between SAS and Hadoop. With SAS/ACCESS Interface to Hadoop you can connect to a Hadoop cluster to read and write data to and from Hadoop. You can analyze Hadoop data with any SAS procedures and the DATA step. SAS/ACCESS Interface to Hadoop works like other SAS engines. You execute a LIBNAME statement to assign a library reference and specify the engine. The LIBNAME statement associates a SAS library reference with Hadoop HDFS or Hive. To define a library reference to Hadoop you need to specify *HADOOP* as the engine name. The following is an example of how to assign a Hive-type LIBNAME. The library reference 'hive' will be associated with a Hive server.

```
libname hive hadoop server="hivenode.sgf2016.com"
                    user=sasdemo
                    subprotocol=Hive2;
```

There is another way to assign a LIBNAME to Hadoop and access data independently from Hive. When you specify the HDFS_METADIR connection option, SAS/ACCESS Interface to Hadoop does not connect directly to a Hive server. It instead creates a library reference that is directly associated with HDFS. The following is an example of how to assign a LIBNAME to HDFS.

```
libname hdfs hadoop server="hadoopnn.sgf2016.com"
                    user=sasdemo
                    HDFS_DATADIR="/user/sasdemo/data"
                    HDFS_METADIR="/user/sasdemo/meta";
```

When using an HDFS-type LIBNAME, SAS/ACCESS Interface to Hadoop creates the file metadata under the folder that is specified in the HDFS_METADIR option. The input file metadata is also known as SAS Hadoop Metadata (SASHDMD) or HDMD. The input file is stored under what is specified in HDFS_DATADIR option. For example, when using *hdfs.mytable* in a DS2 program, SAS procedures or SAS DATA step, the LIBNAME looks for the metadata file */user/sasdemo/meta/mytable.sashdmd.* The input file name is retrieved from an element in the metadata file.

CREATING SAS HADOOP METADATA DESCRIPTOR

SAS can create and use XML-based metadata descriptions of HDFS files and Hive tables with the HDMD procedure. The file type for an XML-based metadata description that PROC HDMD produces is a

SASHDMD descriptor (for example, *mytable.sashdmd*). The SAS Embedded Process requires a SASHDMD descriptor for each input file. When using a Hive-type LIBNAME in the DS2 program, SAS procedure or SAS DATA step, the SASHDMD descriptor is automatically created by the SAS/ACCESS Interface to Hadoop.

Similar to HiveQL data definition language (DDL), SASHDMD descriptors describe the columns in a file with metadata such as column data type, column separator, file location and many other attributes. The file location attribute can point to either a file path or a directory. When pointing to a directory, all files under it are considered parts of a same file. The following example uses PROC HDMD to create a SASHDMD descriptor for the INCOME file stored on HDFS. The INCOME file has four comma-separated columns: a product code, the quantity purchased, the price, and the total purchase amount including tax.

```
proc hdmd
     name=hdfs.income
     file_format=delimited
     sep=','
     data_file='file01.txt';
     column product  int;
     column quantity int;
     column price    real;
     column total    real;
run;
```

Before running PROC HDMD, a LIBNAME must be assigned. PROC HDMD uses the LIBNAME to create the metadata file. The NAME option specifies the name of the metadata file to create. The metadata file is created in the location that is specified in HDFS_METADIR option in the LIBNAME statement. The FORMAT option specifies the format of the input data that is passed to the SAS Embedded Process. The SEP option specifies the character that separates the columns for the records in the delimited input file. The DATA option specifies the path to the input data file relative to the HDFS_DATADIR option in the LIBNAME statement. The COLUMN statement provide specifications for one or more columns. Each column has a name, data type, and options.

At this point, you have prepared the machine from where you run Base SAS to access Hadoop before using any SAS accelerators.

THE ANATOMY OF THE SAS EMBEDDED PROCESS

SAS Embedded Process is a subset of Base SAS software that is sufficient to support the multithreaded SAS DS2 language. SAS Embedded Process runs inside a MapReduce task address space in Hadoop. SAS Embedded Process is a lightweight execution container for specialized DS2 code that makes SAS portable and deployable on a variety of platforms. Running DS2 code directly inside Hadoop effectively leverages the massive parallel processing and native resources. Strategic deployment such as scoring code, data transformation code or data quality code can be applied in this manner.

In order to run DS2 code in parallel inside Hadoop, SAS Embedded Process needs to be installed on every node of the Hadoop cluster that is capable of running a MapReduce task. SAS Embedded Process runs as a MapReduce application. Therefore all the computing resources used by SAS Embedded Process are completely manageable by YARN.

Depending on the type of acceleration you are using, SAS Embedded Process runs on a mapper or a reducer task. MapReduce tasks are started and orchestrated by the MapReduce application master. The SAS Embedded Process is engaged once the mapper or reducer task is started.

SAS EMBEDDED PROCESS INTERNAL COMPONENTS

SAS Embedded Process is written mainly in C language where all the interactions with DS2 container happen. It is also written in Java where all the interactions with the Hadoop environment happen. The Java code is responsible for extracting data from HDFS and passing it to the DS2 container.

Both the C and Java code run on the same Java Virtual Machine (JVM) process that is allocated by the MapReduce application master. The Java and C code have access to shared memory buffers allocated by native code. This eliminates multiple copies of the input data. The shared native buffers are allocated outside of the JVM heap space. It is worth mentioning that native memory allocations are still seen by YARN resource management. Journaling messages generated by the C and DS2 code are written to the MapReduce standard output (*stdout*) or standard error (*stderr*) logs. Messages generated by the Java code are written to the standard job log (*syslog*).

Figure 2 depicts SAS Embedded Process internal architectural components. On the left-hand side are the components provided by the Java code. SAS In-Database Code Accelerator (a) is a full-fledge MapReduce application that engages SAS Embedded Process through a specialized interface called EP Proxy (b). The EP Proxy is responsible for pushing and pulling data to and from SAS Embedded Process Native Interface (c). The HCatalog Reader (d) is capable of reading files that have a SerDe registered in Hive. HCatalog Reader enables you to read data from special format files, such as Parquet, ORC, and RCFile.

The Input Formats and Record Readers (e) retrieve data directly from HDFS. They are controlled by a multi-threaded and multi-split reader framework called Super Reader (f). The Output Formats and Record Writers (g) are responsible for writing DS2 output data directly to HDFS. They are controlled by a multi-threaded writer framework called Super Writer.

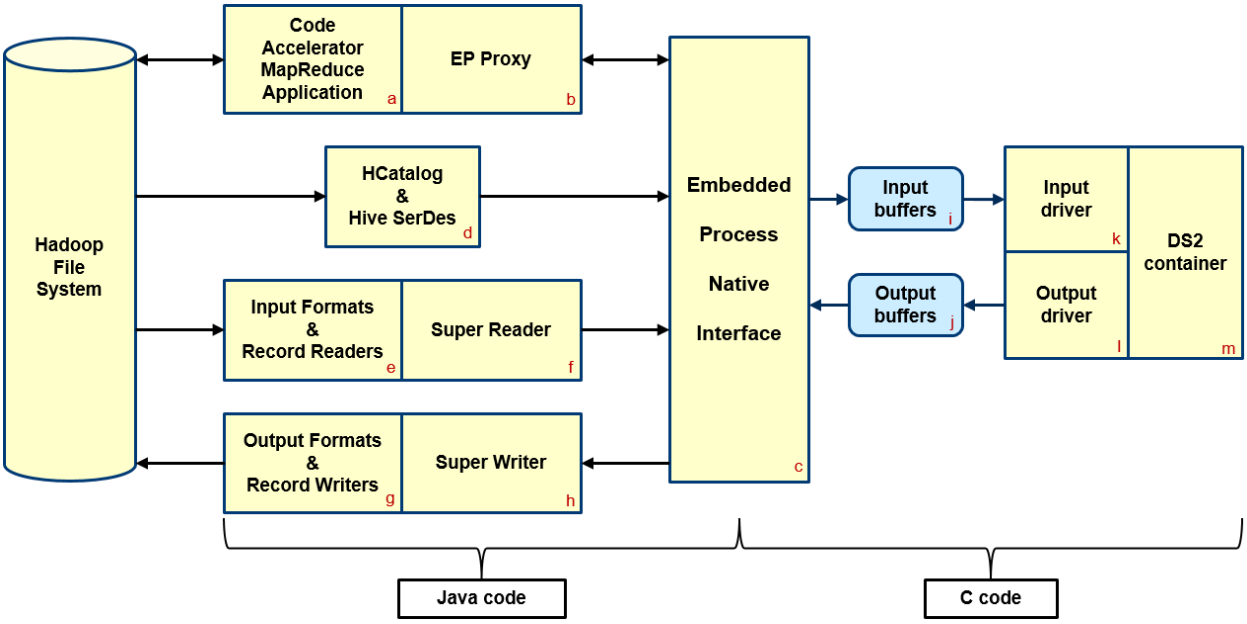

Figure 2. Embedded Process Components

On the right-hand side are the components provided by the C code. SAS Embedded Process Native Interface (c) is the communication channel between Java and C code. Data read from input files are stored directly into the native INPUT buffers (i). The DS2 program runs in the DS2 container (m) and retrieves input data through the input driver (k). The DS2 program operates like a User Defined Table Function (UDTF). It takes a block of records at a time and processes them. At some point in time DS2 creates an output result set and makes it available for consumption. Output data generated by the DS2 program is stored in the native OUTPUT buffers (j) through the output driver (l).

There are three independent sets of threads controlling SAS Embedded Process execution:

- Reader threads: Java threads responsible for reading input splits and filling up input buffers.
- Compute threads: C threads responsible for the execution of the DS2 program.
- Writer threads: Java threads responsible for emptying output buffers and writing data to HDFS.

You now have an understanding of SAS Embedded Process internal components and their responsibilities.

SAS IN-DATABASE CODE ACCELERATOR

The SAS In-Database Code Accelerator for Hadoop publishes a DS2 thread or data program to Hadoop and executes it in parallel inside a MapReduce task. Examples of thread programs include large transpositions, computationally complex programs, scoring models, and BY-group processing. To use the SAS In-Database Code Accelerator, the DS2ACCEL option in the PROC DS2 statement must be set to YES. The Hive table or HDFS file used for input must reside in HDFS and the SAS Embedded Process must be installed on the Hadoop cluster.

Figure 3 depicts the flow of execution of a typical code accelerator MapReduce job.

1) The record reader breaks data into elements appropriate for a single map method [MAP(K, V)] invocation.
2) The mapper performs one-to-one operations. Data arrives as a single key and value pair. The mapper might transform the key and value. The key that is output from the mapper tasks determines which reducers will process the values.
3) The reducer tasks perform many-to-one operations. Data arrives as a single key and a set of values associated with that key.

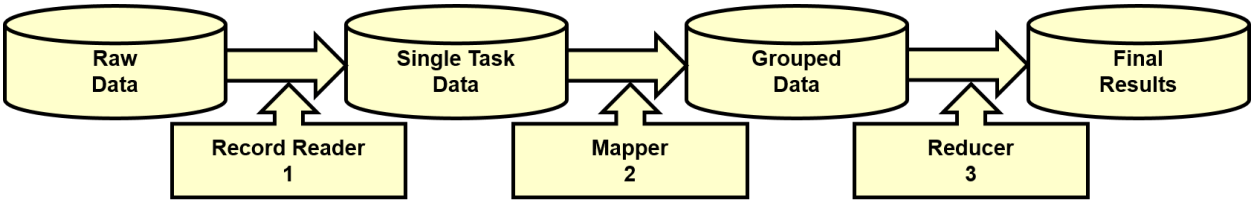

Figure 3. Typical Code Accelerator MapReduce Job

Data is distributed on different nodes of the Hadoop cluster. Each DS2 program running inside a MapReduce task reads a portion of the data. The data partition is also known as a file block or a file split.

When there is no BY statement in the thread program, the number of reducers is set to 0 and the whole program is executed in the mapper tasks. The degree of parallelism will depend on how the data is split. The number of splits controls the number of tasks. The number of parallel tasks depends on the maximum number of concurrent MapReduce slots in the cluster. The mapper task uses its records reader to retrieve data from a given input file split. The records are pushed into the DS2 container through the EP Proxy and SAS Embedded Process input driver. Output records generated by the DS2 program are pulled by the mapper task through the EP Proxy and SAS Embedded Process output driver. Output data is then collected and sent to the next phase.

When there is a BY statement, the degree of parallelism of a thread program is limited to the number of BY groups. Conversely, the DATA program runs singly but can be run in parallel if there is independent BY processing identified by code inspection, which is done by the SAS In-Database Code Accelerator. Either stage could be executed in parallel or singly.

With a BY statement in the thread program and with the BYPARTITION=YES option in the PROC DS2 statement, the mapper tasks partition the data, and the reducer tasks execute the DS2 thread program.

When there is a BY statement in the data program, the number of reducers is set to 1. SAS In-Database Code Accelerator for Hadoop might not produce sorted BY groups when re-partitioning is invoked.

The following is an example of a DS2 program that runs inside Hadoop. It processes a table that contains profiles for around 150,000 real users. The table is referenced in the SET statement inside the RUN method in the thread program. The table lists the musicians each user listens to and a counter indicates how many times each user played each artist. A library reference must be assigned using the LIBNAME statement before PROC DS2 is executed. In this example a LIBNAME to HDFS is used.

```
libname hdfs hadoop server="hadoopnn.sgf2016.com"
                    user=sasdemo
                    HDFS_DATADIR="/user/sasdemo/data"
                    HDFS_METADIR="/user/sasdemo/meta";
```

The table has three columns (USERID, ARTISTID and PLAYCOUNT) and contains 24,296,858 rows. The DS2 program needs to know the input file metadata. The following PROC HDMD creates the input file SASHDMD descriptor.

```
proc hdmd name=hdfs.user_artist_data
          format=delimited
          sep=^A
          data_file='user_artist_data';

     column userid           char(8);
     column artistid         char(8);
     column playcount        double;
run;
```

Here is the SASHDMD descriptor file created by PROC HDMD.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<hdmd>
  <recfm>DELIMITED</recfm>
  <managedData>FALSE</managedData>
  <encoding cei="20">UTF8</encoding><!-- utf-8 -->
  <fieldsep ctrl="1" />
  <fileType>DELIMITED</fileType>
  <inputDir>/user/sasdemo/data/user_artist_data</inputDir>
  <outputDir>/user/sasdemo/data/user_artist_data</outputDir>
  <metaDir>/user/sasdemo/meta</metaDir>
  <expectFile />
  <columns>
    <column>
      <name>userid</name>
      <type>CHAR</type>
      <c_type>char</c_type>
      <charlength>8</charlength>
      <bytelength>8</bytelength>
    </column>
    <column>
      <name>artistid</name>
      <type>CHAR</type>
      <c_type>char</c_type>
      <charlength>8</charlength>
      <bytelength>8</bytelength>
    </column>
    <column>
      <name>playcount</name>
      <type>DOUBLE</type>
      <c_type>double</c_type>
      <bytelength>8</bytelength>
    </column>
  </columns>
</hdmd>
```

PROC DS2 is executed in Base SAS on the user's workstation. Base SAS uses the JProxy process to submit the SAS In-Database Code Accelerator MapReduce job for execution. Depending how it was written, the DS2 program is executed inside a MapReduce mapper and/or reducer task.

```
proc ds2 ds2accel=yes;
thread ComputePlays/overwrite=yes;
  dcl double pCount;
  keep artistid pCount;
  method run();
    set hdfs.user_artist_data;
    by artistid;
    if first.artistid then
      pCount=0;
    pcount+playcount;
    if last.artistid and pcount > 0 then
      output;
  end;
endthread;

data hdfs.results (overwrite=yes);
  dcl thread ComputePlays computeThread;
  method run();
    set from computeThread;
    output;
  end;
enddata;
run; quit;
```

The option DS2ACCEL=YES tells the SAS In-Database Code Accelerator to execute the thread program it inside Hadoop. The thread program groups rows by artist ID and counts the number of times the artist was played. If the total count is greater than zero, the thread program outputs the artist's ID and the play count. The output of the thread program is the input of the data program. The data program outputs the results to the table (*hdfs.results*) that is referenced in the DATA statement. Table *hdfs.results* is stored on HDFS under the folder that is specified in HDFS_DATADIR option in the LBNAME.

The DS2 program has a BY statement in the thread program. When running in the MapReduce job, the mapper tasks partition the data by artist's ID and send them to the reducers. The reducers run the thread program. The number of reducers depends on what is specified in property *mapreduce.job.reduces* in the *mapred-site.xml* configuration file. You should not specify more reducers than the number of available reducer slots in the cluster.

Figure 4 illustrates the SAS In-Database Code Accelerator job submission and execution process.

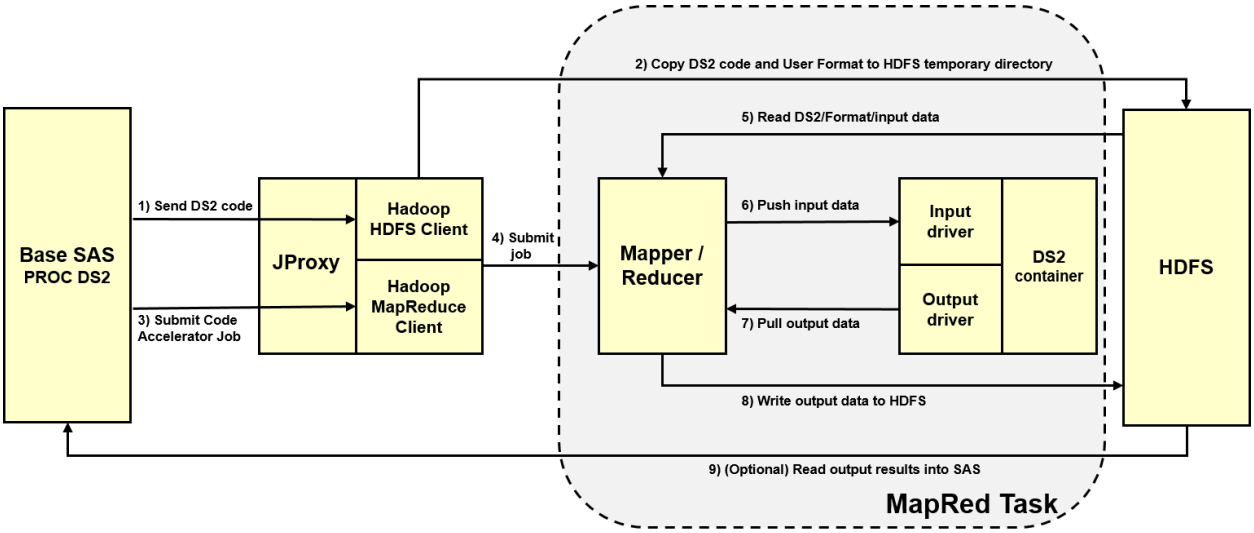

Figure 4. Code Accelerator MapReduce Job Submission and Execution

1) The DS2 compiler prepares the source code to be executed. It creates the SASHDMD descriptor containing the input metadata, loads any additional user-defined formats, and invokes Hadoop client API to store the artifacts on HDFS.
2) The artifacts are stored on a temporary folder on HDFS.
3) DS2 compiler calls into the Input Format class associated with the job to calculate the number of file splits. The number of splits controls the number of mapper tasks allocated to the job.
4) The MapReduce job client API is called to submit the SAS In-Database Code Accelerator job for execution. A MapReduce job is started on the cluster.
5) Depending on how the DS2 source was written, this step might be executed inside a mapper or a reducer task. The task retrieves the DS2 artifacts previously stored on a temporary folder. SAS Embedded Process native interface is instantiated. The DS2 code is prepared and put into execution. If running on a mapper task, the record reader class opens the input file split and starts reading the data. If running on a reducer task, the input data are the records that were collected on the map phase.
6) Input records are pushed into SAS Embedded Process through the EP Proxy and input driver.
7) DS2 processes the input records. Output result sets are given to the mapper/reducer tasks so they can be collected.
8) Output records are collected and sent to the next MapReduce phase or written to an output file on HDFS.
9) You can read the output results back into Base SAS when the job is finished.

The MapReduce job execution can be monitored through a web user interface provided by YARN Resource manager on address http://<yarn-resource-manager-host-name>:8088.

Figure 5 illustrates the YARN resource manager applications interface from where you can see the job in execution.

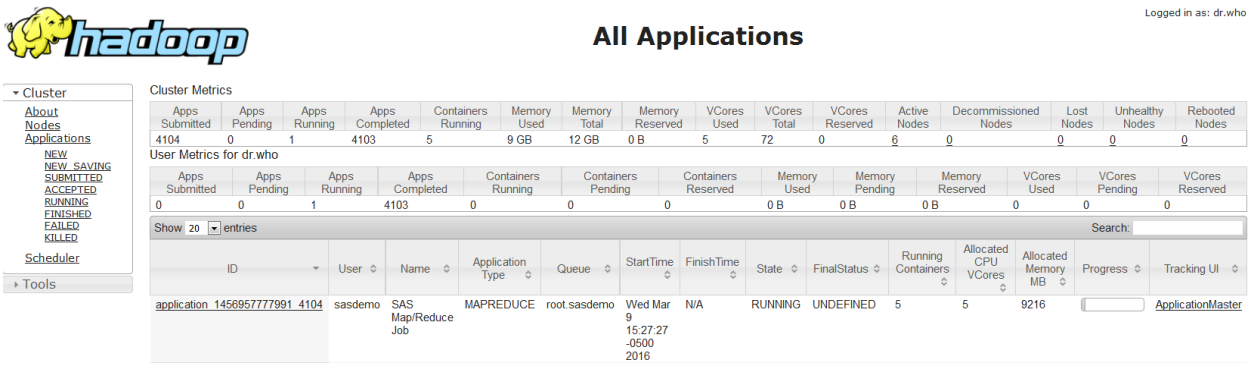

Figure 5. SAS In-Database Code Accelerator MapReduce Job in Execution

The '*ApplicationMaster*' link under '*Tracking UI*' field provides another interface where you can monitor the tasks in execution. After job completion, '*Tracking UI*' field provides a link to the job summary page as illustrated on Figure 6.

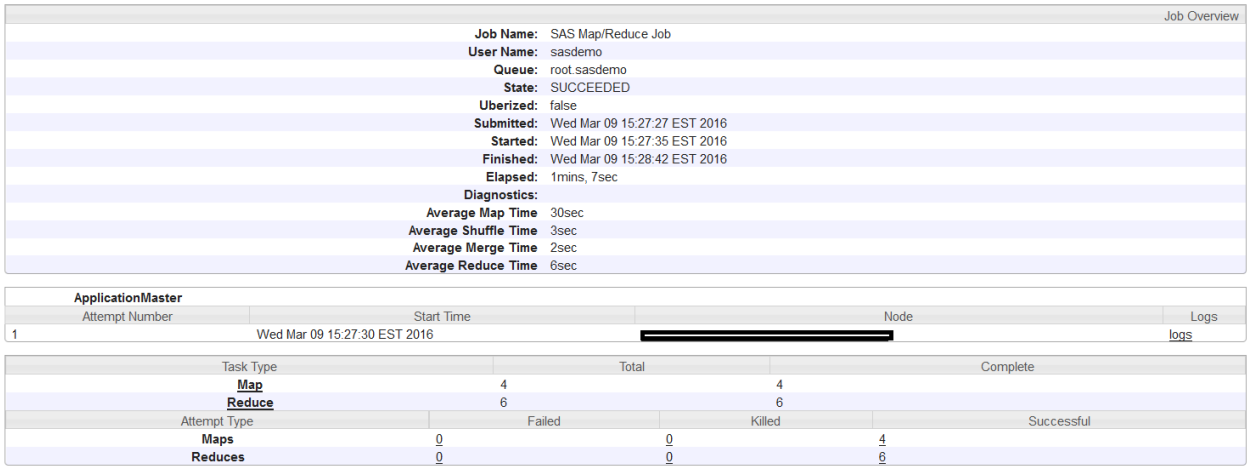

Figure 6. SAS In-Database Code Accelerator MapReduce Job Summary Page

From the job summary page you can navigate to *Maps* or *Reduces* attempts pages, where you can find a link to the attempt log, as illustrated in Figure 7 and Figure 8.

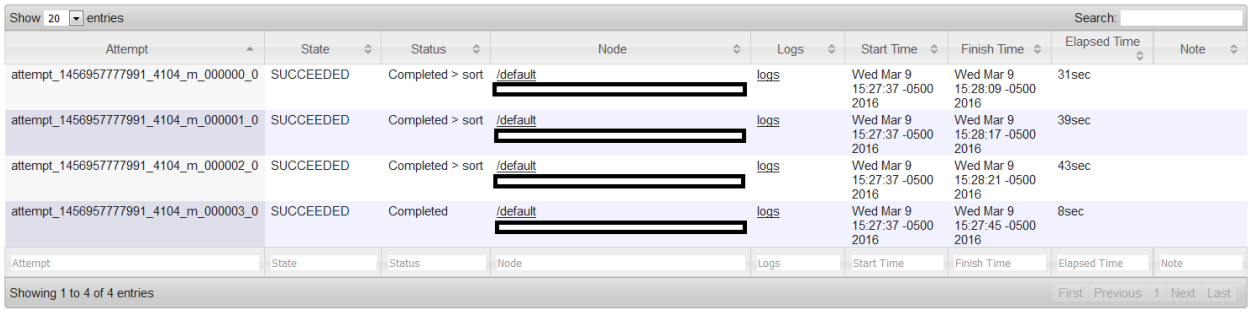

Figure 7. Code Accelerator Maps Attempts Page

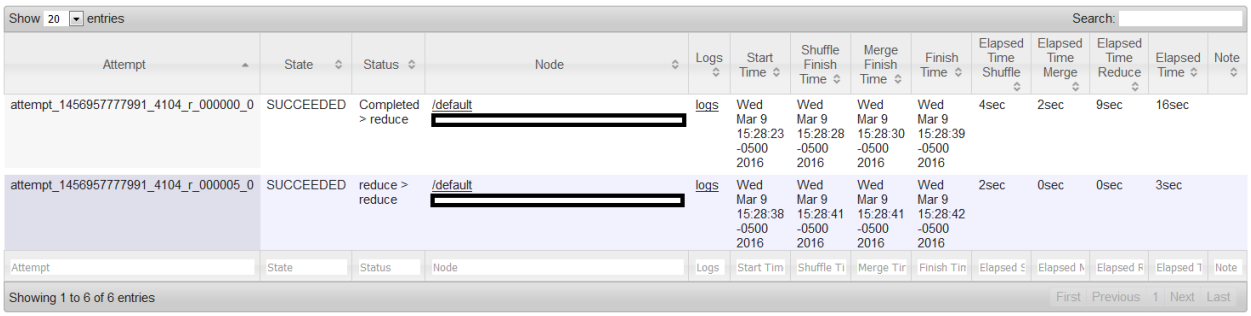

Figure 8. Code Accelerator Reduces attempts page

The link '*logs*' provided under '*Logs*' field takes you to the attempt job log summary. There you can see parts of *stderr*, *stdout* and *syslog* logs. SAS Embedded Process writes its log messages to *stdout* and *stderr* logs. The Java code of the MapReduce tasks writes to the job *syslog*.

Output 1 shows a summary of the log messages generated during execution.

```
Log Type: stderr
Log Length: 0

Log Type: stdout
Log Length: 279

00000013: WARNING: [01S02]Current catalog set to SASEP (0x80fff8bd)
00000018: NOTE: All Embedded Process DS2 execution instances completed with SUCCESS.

Log Type: syslog
Log Length: 14880
Showing 4096 bytes of 14880 total. Click here for the full log.

INFO [main] EmbeddedProcess: JVM file encoding: UTF-8
INFO [main] EmbeddedProcess: JVM process name : 27092@datanode2.sgf2016.com
INFO [main] EmbeddedProcess: JVM process ID   : 27092
INFO [main] EmbeddedProcess: Task ID          : 0
INFO [main] EmbeddedProcess: Attempt ID       : 0
INFO [main] EmbeddedProcess: Job ID           : job_1456957777991_4104
INFO [main] EmbeddedProcess: SAS Embedded Process task environment has been created.
INFO [DONALVARO] Alvaro: Alvaro has been invoked to guide current task execution.
INFO [main] EmbeddedProcess: Fetch size set to 1,985 records. Number of input buffers is 4.
Input buffers will consume 305,688 bytes.
INFO [main] EmbeddedProcess: Number of output buffers is 3. Output buffers will consume
176,862 bytes.
INFO [main] EmbeddedProcess: Task INPUT Information:
INFO [main] EmbeddedProcess:   Task status ..................: SUCCEEDED
INFO [main] EmbeddedProcess:   Input records ................: 4817165
INFO [main] EmbeddedProcess:   Input bytes ..................: 84891993
INFO [main] EmbeddedProcess:   Number of compute threads ....: 1
INFO [main] EmbeddedProcess:   Transcode errors .............: 0
INFO [main] EmbeddedProcess:   Truncation errors ............: 0
INFO [main] EmbeddedProcess:   Input buffer reallocations ...: 0
INFO [main] EmbeddedProcess: Task OUTPUT Summary:
INFO [main] EmbeddedProcess:   Writer status ................: SUCCEEDED
INFO [main] EmbeddedProcess:   Number of output threads .....: 1
INFO [main] EmbeddedProcess:   Output records ...............: 326793
INFO [main] EmbeddedProcess:   Output bytes .................: 3530534
INFO [main] EmbeddedProcess:   Output buffer reallocations ..: 3
INFO [main] org.apache.hadoop.mapred.Task: Task 'attempt_1456957777991_4104_r_000000_0' done.
```

Output 1. Output from Reducer Task Job Log Summary Page

The DS2 sample program provided above writes the output to *hdfs.results*. The output metadata can be displayed by running PROC CONTENTS procedure on the output table. The following source code displays the output metadata.

```
proc contents data=hdfs.results; run;
```

Output 2 displays the output table contents.

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Informat	Label
2	artistid	Char	8	$8.	$8.	artistid
1	pCount	Num	8			pCount

Output 2. SAS In-Database Code Accelerator Output Metadata

The output data can be displayed by running PROC PRINT procedure on the output table. The following is an example of how to print the first 10 observations of a table.

```
proc print data=hdfs.results(obs=10); run;
```

Output 3 displays the first 10 observations of the output file.

Obs	pCount	artistid
1	1401	1000
2	2	10000005
3	21	10000009
4	4	10000010
5	20	10000013
6	1	10000015
7	6	10000022
8	4	10000059
9	1	10000061
10	1	10000068

Output 3. SAS In-Database Code Accelerator Output Results

The SAS In-Database Code Accelerator enables you to run your DS2 code inside the MapReduce framework. Parallel processing and data proximity are fundamental factors to achieve faster results.

SAS IN-DATABASE SCORING ACCELERATOR

Scoring models are created in SAS Enterprise Miner or SAS® Factory Miner. When the scoring function is exported from SAS Enterprise Miner, it creates the scoring model code (*score.sas*), the associated property file that contains model inputs and outputs (*score.xml*) or analytic store file, and a catalog of user-defined formats. These files, called scoring files, are used by a publishing mechanism that will make them available in the Hadoop cluster and ready to be executed.

The generated score model code is a SAS DATA step that contains the scoring functions. In conventional processing, without acceleration, SAS Enterprise Miner uses the SAS/ACCESS Interface to Hadoop to read the input file into SAS allowing the scoring model code to operate on the records. The SAS Scoring Accelerator for Hadoop brings together the robustness of SAS Enterprise Miner and the parallel processing power of Hadoop with a common goal: achieve high performance when executing the scoring functions.

The SAS Scoring Accelerator for Hadoop model publishing macro translates scoring models code to DS2 code. The DS2 code, along with the other scoring files, are stored on HDFS. A model running macro uses the DS2 code and the scoring files to invoke SAS Embedded Process to execute the scoring functions in parallel directly inside Hadoop. Using the parallel processing capabilities of SAS Embedded Process inside Hadoop avoids data movements, yields higher model-scoring performance and faster time to results.

Before running a model inside Hadoop, the scoring files must be published, the input file must be stored on HDFS or Hive, and the input file SASHDMD (SAS Hadoop Metadata) descriptor must be created. When scoring against a table that is stored in Hive, the SASHDMD descriptor is created automatically.

The %INDHD_PUBLISH_MODEL macro publishes the model to Hadoop. The model does not need to be published every time you run it. However, if you make changes to the model, you must republish it. The macro provides an option, ACTION=, that allows you to create, replace or delete a model.

The %INDHD_RUN_MODEL macro uses the files published by the publishing macro and submits the model for execution in the Hadoop cluster. The model, which is the translated DS2 program, runs inside a MapReduce job. The output of the DS2 model program is stored in the HDFS location that is specified by the OUTPUTDATADIR= option.

The publishing and running model macros are executed from Base SAS. Base SAS uses the JProxy process to copy the scoring files to HDFS and to submit the SAS Embedded Process job. SAS Scoring Accelerator for Hadoop is executed inside a MapReduce job. The job runs only the mapper tasks, no reducers. SAS Embedded Process executes the DS2 program that contains the model at the record reader level. When the control returns from the record reader to the MapReduce framework, all records are already processed, the Mapper class is never called, and the task finishes.

Figure 9 illustrates the steps necessary to publish and run a scoring model using SAS Embedded Process on Hadoop. The shaded area represents a MapReduce mapper task executing inside a Java Virtual Machine process on a particular node of a Hadoop cluster.

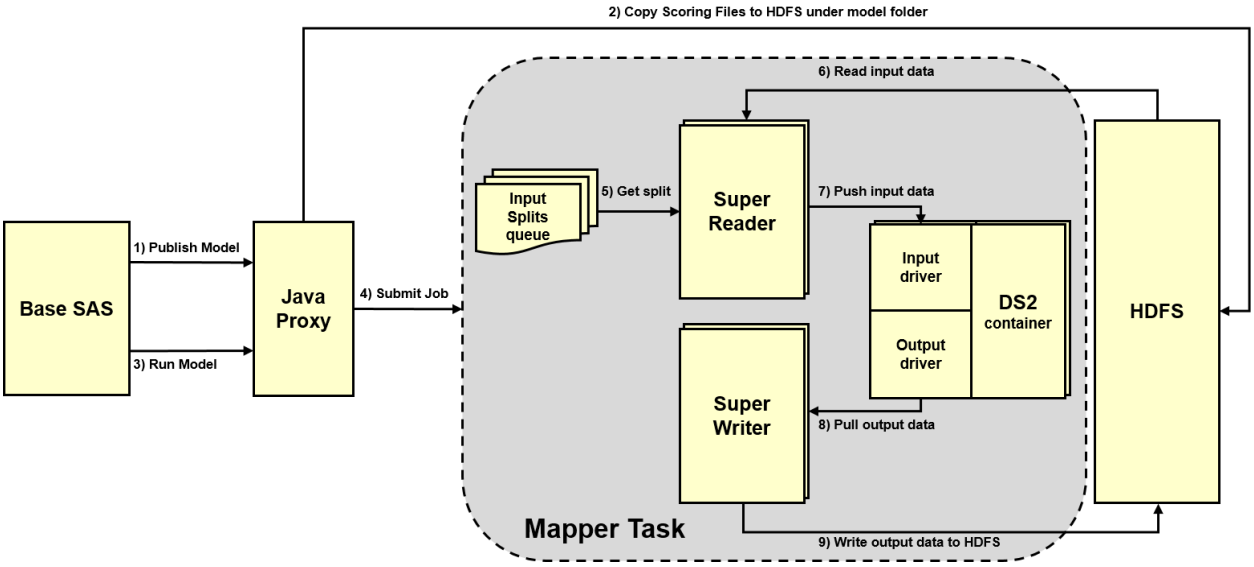

Figure 9. Scoring Accelerator Execution Flow

A SAS Embedded Process mapper task is not a conventional MapReduce task. It is capable of reading multiple file splits in parallel on the same process. It is also capable of writing multiple parts of the output file in parallel on the same process. These technologies are called Super Reader and Super Writer.

In a conventional MapReduce application the input format class is responsible for calculating the number of splits in the input file. The split calculation routine is called before the MapReduce job is submitted for execution. The input format class needs to be on the classpath of the machine that is submitting the MapReduce job. Each file split contains a set of nodes where the file split can be found. One mapper task is assigned to one file split during job submission. For instance: if the input file has 1000 splits, the MapReduce job will create and run 1000 tasks.

The input format class is also responsible for creating a record reader. The record reader class is instantiated by the input format class at run time. The record reader is responsible for opening the split that it was given and reading the records until the end of the split. When the record reader reaches the end of the split, the mapper task ends.

Super Reader does not use the standard MapReduce split calculation. Instead of assigning one split per task, it assigns many. Super Reader calculates the splits, groups them, and distributes the groups to a configurable number of mapper tasks based on data locality. The default maximum number of tasks per

node is 6. This value can be changed by setting the following configuration property in the *mapred-site.xml* configuration file*: sas.ep.superreader.tasks.per.node*.

When it starts, the MapReduce mapper task engages Super Reader input format and Super Writer output format classes. Super Reader creates a configurable number of reader threads. The default number of reader threads is 3. This value can be changed by setting the following configuration property in the *mapred-site.xml* configuration file*: sas.ep.input.threads.*

Super Writer creates a configurable number of writer threads. The default number of writer threads is 2. This value can be changed by setting the following configuration property in the *mapred-site.xml* configuration file*: sas.ep.output.threads.*

Each reader thread takes a file split from the input splits queue, opens the file, positions at the beginning of the split, and starts reading the records. Each record is stored on a native buffer that is shared with the DS2 container. When the native buffer is full, it is pushed to the DS2 container for processing. When a reader thread finishes reading a file split, it takes another file split from the input splits queue.

The DS2 code running inside the DS2 container processes the records it receives. At a given point, DS2 flushes output data to native buffers. Super Writer threads take the output data from DS2 buffers and write them to the Super Writer thread output file part on a designated HDFS location. When all file input splits are processed and all output data is flushed and written to HDFS, the mapper task ends.

Super Reader improves task performance by reading records from many file splits at the same time and by processing the records in parallel. The mapper tasks allocated to run SAS Embedded Process remain active for the entire duration of the MapReduce application. This is to avoid the overhead of having to start one mapper task per file split.

Super Reader supports a variety of file formats, such as delimited text, fixed record length binary, XML, Sequence files, and others. Super Reader also supports a custom reader framework. Users can provide their own input format and reader and plug them into Super Reader framework.

SAS Embedded Process also supports any file type that has a SerDe registered in Hive, such as Parquet, Avro and ORC. SAS Embedded Process uses the HCatalog input format and record reader to read that type of file format. Currently, Super Reader is not engaged by the HCatalog input format and reader.

Super Writer improves performance by writing output data in parallel producing multiple parts of the output file per mapper task. The output of a Scoring Accelerator job is stored on HDFS under the folder that was specified in the run model macro. Super Writer is capable of writing delimited text or fixed length binary records.

SCORING ON HADOOP USING SAS EMBEDDED PROCESS

The following code examples demonstrate how to score on Hadoop using SAS Embedded Process. The model in use is called ALMUSH01, and it is described as follows:

"*A wholesaler has a bushel of mushrooms to look at. He picks up a representative mushroom and checks out the cap color, the gill color, veil color, the odor, presence of spores, and so on. He assigns a qualitative (letter) value for each. The model equates each of these judgements with a numeric value, which can then be used to return a classification of that batch and whether it is a good batch. The wholesaler can use that to determine if he should sell it, and to help set the price.*"

Before using the %INDHD_PUBLISH_MODEL and %INDHD_RUN_MODEL SAS Scoring Accelerator macros, the Base SAS environment needs to be prepared to connect to Hadoop. The Hadoop connection attributes are specified using the INDCONN macro variable. The following is an example of how to set INDCONN macro variable.

```
%let INDCONN=%str(USER=sasdemo);
```

The INDCONN macro variable holds the name of the user that is used to connect to Hadoop. The following SAS statements set the Hadoop client JAR folder and the Hadoop configuration files folder:

```
options set=SAS_HADOOP_JAR_PATH="C:\Hadoop\jars";
```

```
options set=SAS_HADOOP_CONFIG_PATH="C:\Hadoop\conf";
```

The following two library reference definitions are created to give access to HDFS and Hive. The library references are not required to publish a model. They are used farther down to create a SASHDMD descriptor of a HDFS input file or Hive input table. They are also used to demonstrate how to score against a table stored in Hive. The following code assigns the library references to HDFS and Hive:

```
libname hdfs hadoop server="hadoopnn.sgf2016.com" user=sasdemo
                    HDFS_DATADIR="/user/sasdemo/data"
                    HDFS_METADIR="/user/sasdemo/meta"
                    HDFS_TEMPDIR="/user/sasdemo/temp";

libname hive hadoop server="hivenode.sgf2016.com"
                    user=sasdemo
                    subprotocol=Hive2;
```

The first library reference expects the input file to be stored in what is specified in HDFS_DATADIR= option, and the input file metadata (SASHDMD descriptor) to be stored in what is specified in the HDFS_METADIR= option. The folder specified in the HDFS_TEMPDIR= option is used to store temporary files. If omitted, temporary files are stored under */tmp* folder on HDFS. The second library assignment establishes a connection with Hive server on the node specified in SERVER= option.

The following example of %INDHD_PUBLISH_MODEL macro publishes the model to Hadoop.

```
%indhd_publish_model ( dir=u:\sasuser\sgf2016\almush01
                     , modeldir=/user/sasdemo/sgf2016
                     , modelname=almush01
                     , datastep=score.sas
                     , xml=score.xml );
```

The DIR= option specifies the local directory where the scoring model program and its companion files are located. This directory is created by the SAS Enterprise Miner Score Code Export node or SAS Factory Miner. It contains the *score.sas* file and the *score.xml* file. If user-defined formats were used, a format catalog is also published.

The MODELDIR= option specifies the base HDFS path where the scoring model directory is located. The MODELNAME= option specifies the model name. The model name is a directory created under the model directory on HDFS.

The DATASTEP= and XML= options specify the name of the scoring model program and the name of the properties XML file, respectively. Both files are created by using the SAS Enterprise Miner Score Code Export node or SAS Factory Miner. Figure 10 illustrates the content of the model directory on HDFS after it is published.

Contents of directory /user/sasdemo/sgf2016/almush01

Goto : /user/sasdemo/sgf2016 go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
almush01.ds2	file	10.75 KB	3	128 MB	2016-02-18 23:31	rw-r--r--	sasdemo	supergroup
almush01_ufmt.xml	file	6.02 KB	3	128 MB	2016-02-18 23:31	rw-r--r--	sasdemo	supergroup
score.sas	file	15.48 KB	3	128 MB	2016-02-18 23:31	rw-r--r--	sasdemo	supergroup
score.xml	file	21.14 KB	3	128 MB	2016-02-18 23:31	rw-r--r--	sasdemo	supergroup

Figure 10. Contents of the Published Model Directory on HDFS

Once the model is published, you can run it using the %INDHD_RUN_MODEL macro. The metadata file of the input data file used by ALMUSH01 must be present on HDFS. There are three ways that the metadata file can be present depending on where the file exists.

- The file is not a Hive table, it is a file on HDFS: no metadata exists. You must use PROC HDMD to create the metadata file.
- The file is a Hive table. The metadata associated with the table is in Hive, but the metadata is not in SASHDMD format. You either must generate a SASHDMD file for it or use a Hive library reference in %INDHD_RUN_MODEL macro.
- The file is created using the SAS/ACCESS Hadoop engine. When a file is created with a Hadoop LIBNAME statement that contains the HDFS_DATADIR= and HDFS_METADIR options, the HDMD file is automatically generated.

The following code invokes PROC HDMD to create the ALMUSH01 input file metadata.

```
proc hdmd name=hdfs.almush01   format=delimited
          data_file='almush01' sep=^A;

     column capcolor   CHAR(1);   column bruises    CHAR(1);
     column odor       CHAR(1);   column gillsize   CHAR(1);
     column gillcolo   CHAR(1);   column stalksha   CHAR(1);
     column stalkroo   CHAR(1);   column stalksar   CHAR(1);
     column stalkcbr   CHAR(1);   column veilcolo   CHAR(1);
     column ringnumb   CHAR(1);   column ringtype   CHAR(1);
     column sporepc    CHAR(1);   column populat    CHAR(1);
     column habitat    CHAR(1);   column id         DOUBLE;
run;
```

The NAME= option specifies the name of the metadata file to create. The metadata file is created in the location that is specified in HDFS_METADIR= option in the LIBNAME statement. The FORMAT= option specifies the format of the input data that is passed to the SAS Embedded Process. The SEP= option specifies the character to separate the columns in the records. DATA_FILE= option specifies the path to the input data file relative to the HDFS_DATADIR= option in the LIBNAME statement. PROC HDMD will produce file */user/sasdemo/meta/almush01.sashdmd* on HDFS. The input filename specified by SASHDMD descriptor is */user/sasdemo/data/almush01* on HDFS.

Once the input metadata is created, the model is ready to be submitted for execution. The following example invokes %INDHD_RUN_MODEL macro to run the model.

```
%indhd_run_model(  inmetaname=/user/sasdemo/meta/almush01.sashdmd
                 , outdatadir=/user/sasdemo/data/almush01out
                 , outmetadir=/user/sasdemo/meta/almush01out.sashdmd
                 , scorepgm=/user/sasdemo/sgf2016/almush01/almush01.ds2
                 , forceoverwrite=true);
```

The INMETANAME= option specifies the HDFS full path of the input metadata file. The OUTDATADIR= option specifies the directory on HDFS where the output files are stored. The OUTMETADIR= option specifies the directory on HDFS where the output file metadata is stored. The SCOREPGM= option specifies the name of the scoring model program file that is executed by SAS Embedded Process. The FORCEOVERWRITE= option specifies whether the output directory should be deleted before the model is executed.

Assuming that the ALMUSH01 input data is also stored as a table in Hive, you can run %INDHD_RUN_MODEL macro referencing a table name, instead of an input metadata name, using the LIBNAME to Hive. The following example submits the scoring model to be executed against a Hive table.

```
%indhd_run_model(  inputtable=hive.almush01
                 , outdatadir=/user/sasdemo/data/almush01out
```

```
, outmetadir=/user/sasdemo/meta/almush01out.sashdmd
, scorepgm=/user/sasdemo/sgf2016/almush01/almush01.ds2
, forceoverwrite=true);
```

The INPUTTABLE= option automatically creates the SASHDMD metadata descriptor by implicitly calling PROC HDMD.

The execution of the model produces a delimited text or fixed record length binary output file. In addition, the output metadata file is created. To view the output data in Base SAS use PROC PRINT. The following example prints the first ten rows of the output table.

```
proc print data=hdfs.almush01out (obs=10); run;
```

SAS In-Database Scoring Accelerator for Hadoop eliminates the need to move massive data sets between Hadoop and the SAS environment. The parallel processing shortens the time needed to run your models and delivers faster time to results.

CONCLUSION

The challenges imposed by the big data era can be minimized by using SAS Embedded Process. This paper has prepared you to (1) access Hadoop via Base SAS, (2) explain the SAS Embedded Process in Hadoop, and (3) explore the SAS In-Database Code Accelerator for Hadoop and Scoring Accelerator for Hadoop. SAS provides the platform you need to process your data in an effective and efficient manner by using the massively parallel processing power of Hadoop. You are now ready to collect, store and process your data with confidence using the power of SAS.

RECOMMENDED READING

- Secosky, Jason, et al. 2014. “Parallel Data Preparation with the DS2 Programming Language”, *Proceedings of the SAS Global Forum 2014 Conference*. Cary, NC: SAS Institute Inc. Available at http://support.sas.com/resources/papers/proceedings14/SAS329-2014.pdf.
- Ray, Robert. Eason, William. 2016. “Data Analysis with User-Written DS2 Packages”, *Proceedings of the SAS Global Forum 2016 Conference*. Cary, NC: SAS Institute Inc. Available http://support.sas.com/resources/papers/proceedings16/SAS6462-2016.pdf
- Hazejager, W., et al. 2016. “Ten Tips to Unlock the Power of Hadoop with SAS®*”, Proceedings of the SAS Global Forum 2016 Conference*. Cary, NC: SAS Institute Inc. Available at http://support.sas.com/resources/papers/proceedings16/SAS2560-2016.pdf.
- Rausch, N., et al. 2016. “What’s New in SAS Data Management”, *Proceedings of the SAS Global Forum 2016 Conference*. Cary, NC: SAS Institute Inc. Available at http://support.sas.com/resources/papers/proceedings16/SAS2400-2016.pdf.

CONTACT INFORMATION

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

David Ghazaleh
500 SAS Campus Drive
Cary, NC 27513
SAS Institute Inc.
David.Ghazaleh@sas.com
http://www.sas.com