

Sparking Your Data Innovation: SAS® Integration with Apache Spark

Kumar Thangamuthu, SAS Institute Inc.

ABSTRACT

Apache Hadoop is a fascinating landscape of distributed storage and processing. However, the environment can be a challenge for managing data. With so many robust applications available, users are treated to a virtual buffet of procedural and SQL-like languages to work with their data. Whether the data is schema-on-read or schema-on-write, Hadoop is purpose-built to handle the task. In this introductory session, learn best practices for accessing data and deploying analytics to Apache Spark from SAS®, as well as for integrating Spark and SAS® Cloud Analytic Services for powerful, distributed, in-memory optimization.

INTRODUCTION

Apache Hive on Apache Hadoop has been the de facto standard for interacting with Hadoop data for batch processing. Batch processing focuses on data management, ETL types of processing, and huge volumes of data. Hive uses the MapReduce framework to process data, a batch engine. As you probably know already, performance can be a problem. High-latency is among the most notable issues with MapReduce. And unfortunately, business-style queries were also an afterthought. MapReduce is a disk-based batch engine, and it takes time to set up multiple tasks in a job for execution.

Apache Spark offers another option to execute jobs in Hadoop. The goal of Spark is to keep the benefits of MapReduce's scalable, distributed, fault-tolerant processing framework, while making it more efficient and easier to use.

This paper contains code examples to integrate Hadoop and SAS using Spark as the data access service. Examples used are from SAS/ACCESS Interface to Hadoop with the option to execute in Spark. However, the same examples can be executed in Hive with just a change to a parameter option.

WHAT IS SPARK?

Apache Spark is a distributed general-purpose cluster-computing framework. Spark's architectural foundation is the resilient distributed dataset (RDD), a read-only multiset of data items distributed over a cluster of machines and maintained to enable fault tolerance. Spark and its RDDs were developed in response to limitations of the MapReduce cluster computing paradigm, which enforces a particular linear data flow structure for distributed programs. MapReduce programs read input data from disk, map a function across the data, reduce the results of the map, and store reduction results on disk. Spark's RDDs function as a working set for distributed programs that offers distributed shared memory.

Spark is platform-independent, but SAS products require Spark to be running on a Hadoop cluster.

INTRODUCTION TO SAS/ACCESS INTERFACE TO HADOOP

SAS/ACCESS Interface to Hadoop enables you to work with data from three supported modes of operation:

- Hive/MapReduce
- Spark

- HDMD

With SAS/ACCESS Interface to Hadoop, SAS can read and write data to and from Hadoop as if it were any other relational data source to which SAS can connect. This interface provides fast, efficient access to data stored in Hadoop.

In SAS Viya, SAS/ACCESS Interface to Hadoop includes SAS Data Connector to Hadoop. All users with SAS/ACCESS Interface to Hadoop can use the serial SAS Data Connector to Hadoop. If you have licensed SAS In-Database Technologies for Hadoop, you will also have access to the SAS Data Connect Accelerator to Hadoop. SAS Data Connect Accelerator to Hadoop can load or save data in parallel between Hadoop and SAS using SAS Embedded Process, as a Hive/MapReduce or Spark job. To access and process Hadoop data in Spark, SAS/ACCESS Interface to Hadoop uses a PLATFORM= parameter option.

The SAS® Viya Data Connector or SAS® Viya Data Connect Accelerator enables you to load large amounts of data into the CAS server for parallel processing. SAS® Cloud Analytic Services (CAS) is the cloud-based run-time environment for data management, distributed computing, and high-performance analytics with SAS Viya. A platform for distributed computing, CAS can run in the cloud while providing the best-in-class analytics that SAS is known for.

When possible, SAS/ACCESS Interface to Hadoop also does streaming reads and streaming writes directly from the Hadoop Distributed File System (HDFS) to improve performance. This differs from the traditional SAS/ACCESS engine behavior, which exclusively uses database SQL to read and write data.

STORING SAS DATA ON HADOOP CLUSTER

SAS/ACCESS Interface to Hadoop uses an HDMD (Hadoop Metadata) mode of operation. When you specify the `HDFS_METADIR=connection` option, SAS data sets are persisted on HDFS in a format that can be read directly by SAS. This is a useful way to store large amounts of SAS data on a low-cost Hadoop cluster. Metadata about the SAS data set is persisted as a file with the SASHDMD file type. SAS/ACCESS creates SASHDMD metadata when it writes output from SAS. As an alternative, the HDMD procedure can create these metadata files.

SAS/ACCESS INTERFACE TO HADOOP ON SPARK CONFIGURATIONS

We will look at examples that use the MVA™ SAS LIBNAME statement and CAS CASLIB statement to connect to Hadoop and process data. The SAS connection to the Hadoop cluster requires two paths on the SAS client to locations containing Hadoop JAR files and Hadoop configuration files. Contents for these two paths are gathered using the SAS HadoopTracer script.

ENVIRONMENT VARIABLES

The following two environment variables are required when connecting to Hadoop using the LIBNAME statement.

1. SAS_HADOOP_JAR_PATH

Specifies the directory path for the Hadoop and Spark JAR files. If the pathname contains spaces, enclose the pathname value in double quotation marks. To specify multiple pathnames, concatenate pathnames by separating them with a colon (:) in a UNIX environment.

For example, if the Hadoop JAR files are copied to the location **/third_party/Hadoop/jars/lib** and Spark JAR files are copied to the

location **/third_party/Hadoop/jars/lib/spark**, then the following OPTIONS statement syntax sets the environment variable appropriately:

```
options  
set=SAS_HADOOP_JAR_PATH="/third_party/Hadoop/jars/lib:/third_party/Hadoop/jars/lib/spark";
```

2. SAS_HADOOP_CONFIG_PATH

Specifies the directory path for the Hadoop cluster configuration files. If the pathname contains spaces, enclose the pathname value in double quotation marks.

For example, if the cluster configuration files are copied from the Hadoop cluster to the location **/third_party/Hadoop/conf**, then the following OPTIONS statement syntax sets the environment variable appropriately.

```
options set=SAS_HADOOP_CONFIG_PATH "/third_party/Hadoop/conf";
```

These environment variables are not used by CASLIB statements. Hadoop JAR and config paths are specified as parameters in the CASLIB statement, which we will discuss shortly:

- hadoopjarpath = "Hadoop and Spark JAR files path"
- hadoopconfigdir = "Hadoop Configuration files path"

CONNECTING TO A HADOOP CLUSTER

There are two ways to connect to a Hadoop cluster using SAS/ACCESS Interface to Hadoop, based on the SAS platform:

- LIBNAME statement to connect from MVA SAS
- CASLIB statement to connect from CAS

LIBNAME STATEMENT

The SAS/ACCESS LIBNAME statement enables you to assign a traditional SAS libref connection to a data source. After you assign the libref, you can reference database objects (tables and views) as if they were SAS data sets. The database tables can be used in DATA steps and SAS procedures.

Here is a LIBNAME statement that connects to a Hadoop cluster:

```
libname hdplib hadoop server="hadoop.server.com"  
      port=10000  
      user="hive"  
      schema='default'  
      properties="hive.execution.engine=SPARK";
```

Here are some important items to note in this LIBNAME statement:

- Libref – This LIBNAME statement creates a libref named hdplib. The hdplib libref is used to specify the location where SAS will find the data.
- SAS/ACCESS Engine Name – In this case, we are connecting to Hadoop, so we specify the HADOOP option in the LIBNAME statement.
- The SERVER= option tells SAS which Hadoop Hive server to connect to. In this case, we are connecting to the Hive server. This value will generally be supplied by your system administrator.
- The PORT= option specifies the port where the Hive server is listening. 10000 is the default, so it is not required. It is included just in case.

- USER= and PASSWORD= are not always required.
- The SCHEMA= option is used to specify the Hive schema to which you want to connect. It is optional; by default, it connects to the "default" schema.
- The PROPERTIES= option specifies Hadoop properties. Choosing *SPARK* for the property `hive.execution.engine` enables SAS Viya to use Spark as the execution platform.

```

76 libname hdplib hadoop server="hadoop.server.com"
77     port=10000
78     user="hive"
79     schema='default'
80     properties="hive.execution.engine=SPARK";
NOTE: HiveServer2 High Availability via ZooKeeper will not be used for this
connection. Specifying the SERVER= or PORT= libname
option overrides configuration properties.
NOTE: Libref HDPLIB was successfully assigned as follows:
Engine:          HADOOP
Physical Name:
jdbc:hive2://hadoop.server.com:10000/default?hive.execution.engine=SPARK

```

Output 1. SAS Log Output from a LIBNAME Statement

Once the libref has been created, any data processed, or jobs executed using the libref will use Spark as the execution platform.

CASLIB STATEMENT

A caslib is an in-memory space in SAS® Viya to hold tables, access control lists, and data source information. All data is available to CAS through caslibs, and all operations in CAS that use data are performed with a caslib in place.

Here is the CASLIB statement to the Hadoop data source with Spark as the execution platform:

```

caslib splib sessref=mysession datasource=(srctype="hadoop",
    dataTransferMode="auto",
    username="hive",
    server="hadoop.server.com",
    hadoopjarpath="/opt/sas/viya/config/data/hadoop/lib:/opt/sas/viya/conf
ig/data/hadoop/lib/spark",
    hadoopconfigdir="/opt/sas/viya/config/data/hadoop/conf",
    schema="default"
    platform="spark"
    dfdebug="EPALL"

    properties="hive.execution.engine=SPARK");

```

Here is an explanation of the parameters that are used to create a caslib:

- CASLIB – A library reference. The caslib is the space holder for the specified data access. The splib cas library is used to specify the Hadoop data source.
- sessref – Holds the CAS library in a specific CAS session. *Mysession* is the current active CAS session.
- DATASOURCE= Holds Hadoop connection options. A few options are common across all data sources, such as SRCTYPE=, SERVER=, and SCHEMA=. There are also

Hadoop-specific parameters, such as PLATFORM=, HADOOPJARPATH=, HADOOPCONFIGDIR=.

- SRCTYPE= As you have probably guessed from the name, this option is used to specify the type of data source that the connection is intended to.
- DATATRANSFERMODE= Specifies the type of data movement between CAS and Hadoop. This option accepts one of three values – serial, parallel, auto. When AUTO is specified, CAS choose the type of data transfer based on available license in the system. If Data Connect Accelerator to Hadoop has been licensed, parallel data transfer will be used, otherwise serial mode of transfer is used.
- USERNAME= and PASSWORD= are not always required.
- HADOOPJARPATH= Specifies Hadoop and Spark JAR files location path on the CAS cluster.
- HADOOPCONFIGDIR= Specifies Hadoop configuration files location path on the CAS cluster. These config files are used to connect to Hadoop from CAS.
- SCHEMA= An option that is used to specify the Hive schema to which you want to connect. It is optional, but by default it connects to the “default” schema.
- PLATFORM= An option that is used to specify the type of Hadoop platform to execute the job or transfer data using SAS Embedded Process. Default value is “mapred” for Hive MapReduce. When “Spark” is used, data transfer and job executes as a Spark job.
- DFDEBUG= An option that is used to get additional information back from SAS Embedded Process that is used to transfer data in the SAS log.
- The PROPERTIES= Specifies Hadoop properties. Choosing “SPARK” for the property hive.execution.engine enables SAS Viya to use Spark as the execution platform.

```
76 caslib splib datasource=(srctype="hadoop",
77 dataTransferMode="auto",
78 server="hadoop.server.com",
79
hadoopjarpath="/opt/sas/viya/config/data/hadoop/lib:/opt/sas/viya/config/data/hadoop/lib/spark",
80 hadoopconfigdir="/opt/sas/viya/config/data/hadoop/conf",
81 username="hive"
82 schema="default"
83 platform="spark"
84 dfdebug="EPALL"
85 properties="hive.execution.engine=SPARK");
NOTE: 'SPLIB' is now the active caslib.
NOTE: Cloud Analytic Services added the caslib 'SPLIB'.
NOTE: Action to ADD caslib SPLIB completed for session MYSESSION.
```

Output 2. SAS Log Output from a CASLIB Statement

CAS libraries can be part of a session, where users have access to the data source tables for the lifetime of the temporary session. But if you need to store a caslib permanently, a caslib can be promoted to a global space where all users can access its tables or data. In fact, by default a global library called “public” is available in CAS clusters.

The PLATFORM option is used by the SAS Embedded Process to process and execute data in Spark.

DATA ACCESS USING SPARK

Spark provides the ability to read HDFS files and query structured data from within a Spark application. With Spark SQL, data can be retrieved from a table stored in Hive using a SQL statement and the Spark Dataset API. Spark SQL provides ways to retrieve information about columns and their data type and supports the HiveQL syntax.

SAS Data Connect Accelerator for Hadoop with the Spark platform option uses Hive as the query engine that will be used to access Spark data.

Using SAS Data Connect Accelerator for Hadoop, data can be loaded to CAS or saved to Hadoop from CAS in parallel using the SAS Embedded Process, which is installed on all Hadoop cluster nodes. Data movement happens between Spark and CAS through SAS generated Scala code. This approach is useful when data already exists in Spark and either needs to be used for SAS analytics processing or moved to CAS for massively parallel data and analytics processing.

LOADING DATA FROM HADOOP TO CAS USING SPARK

There are many important reasons to load data from Hadoop to CAS. Processing data in CAS offers advanced data preparation, visualization, modeling and model pipelines, and finally model deployment. Model deployment can be performed using available CAS modules or pushed back to Spark if the data is already in Hadoop, an example of which we will see soon.

Here is an example of the code to load data from Hadoop to CAS using Spark:

```
proc casutil
```

```
    incaslib=splib  
    outcaslib=casuser;  
    load casdata="gas"  
    casout="gas"  
    replace;
```

```
run;
```

```
76  proc casutil  
77  incaslib=splib  
78  outcaslib=casuser;  
NOTE: The UUID 'b75390d7-065c-9240-806f-2dff63b13e77' is connected using  
session MYSESSION.  
79  
79  ! load casdata="gas"  
80  casout="gas"  
81  replace;  
NOTE: Performing parallel LoadTable action using SAS Data Connect  
Accelerator for Hadoop.  
NOTE: SAS Embedded Process tracking URL:  
NOTE: Job Status .....: SUCCEEDED  
NOTE: Job ID .....:
```

```

NOTE: Job Name .....: SAS CAS/DC Input [in: default.gas]

NOTE: File splits..... : 0
NOTE: Input records ...: 0
NOTE: Input bytes .....: 0
NOTE: Output records ..: 0
NOTE: Output bytes .....: 0
NOTE: Transcode errors : 0
NOTE: Truncations .....: 0
NOTE: Map Progress .....: 0.00%
NOTE: Cloud Analytic Services made the external data from gas available as
table GAS in caslib CASUSER(demo).
NOTE: The Cloud Analytic Services server processed the request in 16.61905
seconds.
82      run;

```

Output 3. SAS Log Output from PROC CASUTIL LOAD data CAS Action Statement

The screenshot shows the Hadoop 'All Applications' dashboard. On the left is a navigation menu with options like 'Cluster', 'About Nodes', 'Node Labels', 'Applications', and 'Scheduler'. The main area displays 'Cluster Metrics' with a table of application statistics. Below this, there are 'Scheduler Metrics' and a table of application entries. The table includes columns for ID, User, Name, Application Type, Queue, Application Priority, Start Time, Finish Time, State, Final Status, Running Containers, Allocated CPU V-Cores, Allocated Memory MB, and QoS.

Cluster Metrics													
Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	V-Cores Used	V-Cores Total	V-Cores Reserved	Active Nodes	Decommissioned Nodes	
327	0	0	327	0	0 B	150 GB	0 B	0	38	0	2	0	

ID	User	Name	Application Type	Queue	Application Priority	StartTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU V-Cores	Allocated Memory MB	QoS
application_1531011404729_0339	hive	SAS CAS/DC Input [in: default.gas]	SPARK	default	0	Mon Feb 25 16:17:25 -0500 2019	Mon Feb 25 16:17:53 -0500 2019	FINISHED	SUCCEEDED	N/A	N/A	N/A	0

Display 1. Load Data from Hadoop to CAS Using Spark

The PROC CASUTIL can be used to call many CAS actions to process data. In this case, the table named "gas" was loaded to the CAS in-memory server, which was made possible using the LOAD CAS action.

INCASLIB and OUTCASLIB are input and output CAS libraries to read and write data respectively. "splib" in INCASLIB corresponds to the CAS library created earlier using CASLIB statement. "casuser" in OUTCASLIB corresponds to the default CAS library of the user in SAS® Viya.

From the log file, Data Connect Accelerator for Hadoop was used to move data in parallel to CAS. Display 1 shows that the YARN application executed the work as a Spark job. This was possible because the CASLIB statement had Platform= Spark option specified. The data movement direction, in this case Hadoop to CAS can be identified using the Spark job name, "SAS CAS/DC Input," where *Input* is data loaded into CAS.

SAVING DATA FROM CAS TO HADOOP USING SPARK

Data can be saved back to Hadoop from CAS at many stages of the analytic life cycle. For example, data in CAS can be used to prepare, blend, visualize, and model. Once the data meets the business use case, if you want to share it with other part of the organization, data can be saved in parallel to Hadoop using Spark jobs. When a data transfer job is initiated, Procedure CAS calls SAVE CAS action to move data. Based on the licensed transfer

mechanism, in this case SAS Data Connect Accelerator to Hadoop initiates a parallel Embedded Process transfer from CAS worker nodes to Hadoop data nodes.

Here is an example of using the SAVE CAS action to move data to Hadoop using Spark:


```
proc cas;
session mysession;

    table.save /
    caslib="splib"
    table={caslib="casuser", name="gas"},
    name="gas.sashdat"
    replace=True;

quit;
```

```
76   proc cas;
77     session mysession;
78     table.save /
79     caslib="splib"
80     table={caslib="casuser", name="gas"},
81     name="gas.sashdat"
82     replace=True;
83   quit;
NOTE: Active Session now mysession.
NOTE: Performing parallel SaveTable action using SAS Data Connect
Accelerator for Hadoop.
NOTE: SAS Embedded Process tracking URL:
NOTE: Job Status .....: SUCCEEDED
NOTE: Job ID .....:
NOTE: Job Name .....: SAS CAS/DC Output [out: default.gas]
NOTE: File splits..... : 0
NOTE: Input records ...: 0
NOTE: Input bytes .....: 0
NOTE: Output records ..: 0
NOTE: Output bytes ....: 0
NOTE: Transcode errors : 0
NOTE: Truncations .....: 0
NOTE: Map Progress ....: 0.00%
NOTE: Cloud Analytic Services saved the file gas2 in caslib SPLIB.
{caslib=SPLIB,name=gas}
NOTE: PROCEDURE CAS used (Total process time):
      real time          12.67 seconds
      cpu time           0.38 seconds
```

Output 4. SAS Log Output from PROC CASUTIL SAVE Data CAS Action Statement



All Applications

<ul style="list-style-type: none"> Cluster About Nodes Node Labels Applications NEW SAVING SUBMITTED ACCEPTED RUNNING FINISHED FAILED KILLED Scheduler Tools 	Cluster Metrics										
	Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved
	329	0	0	329	0	0 B	150 GB	0 B	0	38	0
	Scheduler Metrics										
	Scheduler Type			Scheduling Resource Type				Minimum Allocation			
	Capacity Scheduler			[MEMORY]				<memory:2048, vCores:6>			
	Show 20 entries										
	ID	User	Name	Application Type	Queue	Application Priority	StartTime	FinishTime	State	FinalStatus	Run Cont
	application_1531011404729_0341	hive	SAS CAS/DC Output [out: default.gas]	SPARK	default	0	Mon Feb 25 18:04:32 -0500 2019	Mon Feb 25 18:04:57 -0500 2019	FINISHED	SUCCEEDED	N/A

Display 2. Save Data from CAS to Hadoop Using Spark

Data from CAS is saved as a Hadoop table using Spark as the execution platform. As SAS Data Connect Accelerator for Hadoop is used to transfer data in parallel, individual Spark executors in each of the Spark executor nodes handles data execution for that specific Hadoop cluster node.

Display 2 shows the SAVE data execution as a Spark job. The Spark job named "SAS CAS/DC Output" specifies that the data was moved from CAS to Hadoop.

IN-DATABASE SCORING USING SPARK

The integration of the SAS Embedded Process and Hadoop allows scoring code to be run directly on Hadoop. Both DS2 and DATA step models can be published and scored inside Hadoop. Scoring models in Hadoop can be run with either MapReduce or the Spark2 engine. DS2 supports Apache Spark and JDBC-compliant Hadoop data sources. You can access the Spark data through the SAS Workspace Server or the SAS Compute Server by using SAS/ACCESS to Hadoop. You can access the Spark data from the CAS server by using SAS data connectors.

SCORING DATA FROM CAS USING SPARK

PROC SCOREACCEL provides an interface to the CAS server for DATA step and DS2 model publishing and scoring. Model code can be published from CAS to Spark and then executed there via the SAS Embedded Process.

PROC SCOREACCEL supports a file interface for passing the model components (model program, format XML, and analytic stores). The procedure reads the specified files and passes their contents on to the model-publishing CAS action. In this case, the files must be visible from the SAS client.

Here is an example in which the CAS Publishmodel and Runmodel actions are used to publish and execute score data in Spark:

```
%let CLUSTER="/opt/sas/viya/config/data/hadoop/lib:
/opt/sas/viya/config/data/hadoop/lib/spark:/opt/sas/viya/config/data/hadoop/conf";
proc scoreaccel sessref=mysess1;
  publishmodel
    target=hadoop
    modelname="simple01"
    modeltype=DS2
```

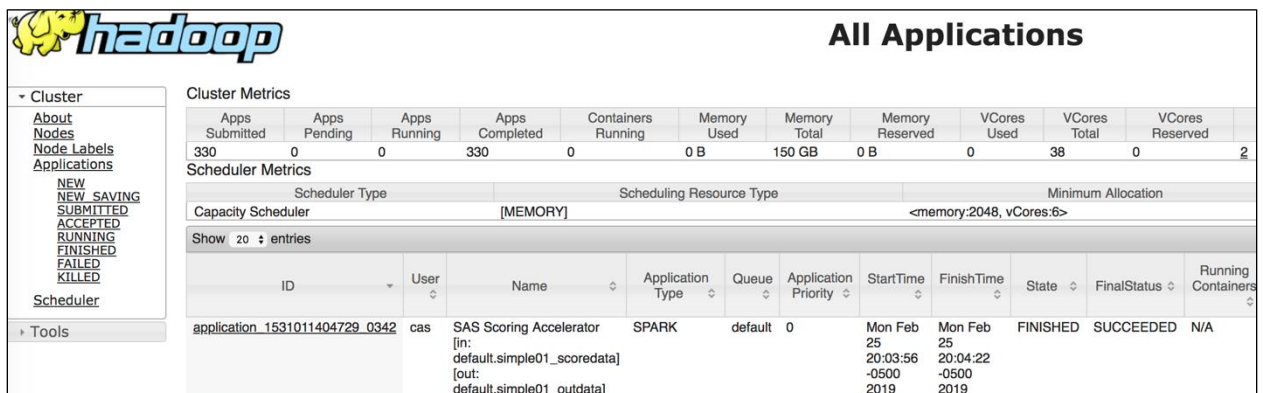
```
/*
  filelocation=local */
programfile="/demo/code/simple.ds2"
username="cas"
modeldir="/user/cas"
classpath=&CLUSTER.
; runmodel
  target=hadoop
  modelname="simple01"
  username="cas"
  modeldir="/user/cas"
  server=hadoop.server.com'
  intable="simple01_scoredata"
  outtable="simple01_outdata"
  forceoverwrite=yes
  classpath=&CLUSTER.
  platform=SPARK
;
quit;
```

```

76  proc scoreaccel sessref=mysess1;
NOTE: Added action set 'modelPublishing'.
NOTE: Added action set 'ds2'.
77  publishmodel
78  target=hadoop
79  modelname="simple01"
80  modeltype=DS2
81  /*    filelocation=local */
82  programfile="/demo/code/simple.ds2"
83  username="cas"
84  modeldir="/user/cas"
85  classpath=&CLUSTER.
86  ;
NOTE: Running 'modelPublishing' action set with 2 workers.
NOTE: Model 'simple01' has been successfully published to the external
database.
87  runmodel
88  target=hadoop
89  modelname="simple01"
90  username="cas"
91  modeldir="/user/cas"
92  server='hadoop.server.com'
93  intable="simple01_scoredata"
94  outtable="simple01_outdata"
95  forceoverwrite=yes
96  classpath=&CLUSTER.
98  platform=SPARK
99  ;
NOTE: Running 'modelPublishing' action set with 2 workers.
NOTE: Job Status .....: SUCCEEDED
NOTE: Job Name .....: SAS Scoring Accelerator [in:
default.simple01_scoredata] [out: default.simple01_outdata]
NOTE: Execution of model 'simple01' succeeded.
100 quit;
NOTE: PROCEDURE SCOREACCEL used (Total process time):
      real time           34.10 seconds
      cpu time            0.32 seconds

```

Output 5. SAS Log Output from SAS Scoring Accelerator from CAS



The screenshot shows the Hadoop web interface with the 'All Applications' page. On the left is a navigation menu with options like 'Cluster', 'About Nodes', 'Node Labels', 'Applications', and 'Scheduler'. The main content area displays 'Cluster Metrics' and 'Scheduler Metrics'. Below these is a table of applications. The table has columns for ID, User, Name, Application Type, Queue, Application Priority, Start Time, Finish Time, State, Final Status, and Running Containers. One application is listed with ID 'application_1531011404729_0342', User 'cas', Name 'SAS Scoring Accelerator', Application Type 'SPARK', Queue 'default', Application Priority '0', Start Time 'Mon Feb 25 20:03:56 -0500 2019', Finish Time 'Mon Feb 25 20:04:22 -0500 2019', State 'FINISHED', Final Status 'SUCCEEDED', and Running Containers 'N/A'.

ID	User	Name	Application Type	Queue	Application Priority	StartTime	FinishTime	State	FinalStatus	Running Containers
application_1531011404729_0342	cas	SAS Scoring Accelerator [in: default.simple01_scoredata] [out: default.simple01_outdata]	SPARK	default	0	Mon Feb 25 20:03:56 -0500 2019	Mon Feb 25 20:04:22 -0500 2019	FINISHED	SUCCEEDED	N/A

Display 3. Running Model Score in Spark Using SAS Scoring Accelerator from CAS

In this PROC SCOREACCEL example, a simple DS2 model is published to Hadoop and executed there with Spark. The CLASSPATH statement specifies a link to the Hadoop cluster. The input and output tables, *simple01_scoredata* and *simple01_outdata*, already exist on the Hadoop cluster. Display 3 shows that SAS Scoring Accelerator was used to score the model in Spark, and the Spark job name reflects the input and output tables.

SCORING DATA FROM MVA SAS USING SPARK

To run a scoring model in Hadoop, follow these steps:

1. Create a traditional scoring model by using SAS Enterprise Miner or an analytic store scoring model, generated using SAS Factory Miner HPFOREST or HPSVM components.
2. Start SAS.
3. Specify the Hadoop connection attributes:

```
%let indconn= user=myuserid;
```

The INDCONN macro variable is used to provide credentials to connect to the Hadoop HDFS and MapReduce. You must assign the INDCONN macro variable before you run the %INDHD_PUBLISH_MODEL and the %INDHD_RUN_MODEL macros.

4. Run the %INDHD_PUBLISH_MODEL macro.

With traditional model scoring, the %INDHD_PUBLISH_MODEL performs multiple tasks using some of the files that are created by the SAS Enterprise Miner Score Code Export node. Using the scoring model program (score.sas file), the properties file (score.xml file), and (if the training data includes SAS user-defined formats) a format catalog, this model performs all the following tasks:

- translates the scoring model into the sasscore_*modelname*.ds2 file, which is used to run scoring inside the SAS Embedded Process
- takes the format catalog, if available, and produces the sasscore_*modelname*_ufmt.xml file. This file contains user-defined formats for the scoring model that is being published.
- uses SAS/ACCESS Interface to Hadoop to copy the sasscore_*modelname*.ds2 and sasscore_*modelname*_ufmt.xml scoring files to HDFS

5. Run the %INDHD_RUN_MODEL macro.

The %INDHD_PUBLISH_MODEL macro publishes the model to Hadoop, making the model available to run against data that is stored in HDFS.

The %INDHD_RUN_MODEL macro starts a Spark job that uses the files generated by the %INDHD_PUBLISH_MODEL to execute the DS2 program. The Spark job stores the DS2 program output in the HDFS location that is specified by either the OUTPUTDATADIR= argument or by the <outputDir> element in the HDMD file. Here is an example:

```
option
set=SAS_HADOOP_CONFIG_PATH="/opt/sas9.4/Config/Lev1/HadoopServer/conf";
option
set=SAS_HADOOP_JAR_PATH="/opt/sas9.4/Config/Lev1/HadoopServer/lib:/opt/sas9.4
/Config/Lev1/HadoopServer/lib/spark";
%let scorename=m6sccode;
%let scoredir=/opt/code/score;
option sastrace=',,,d' sastraceloc=saslog;
option set=HADOOPPLATFORM=SPARK;
```

```

%let indconn = %str(USER=hive HIVE_SERVER='hadoop.server.com');
%put &indconn;

%INDHD_PUBLISH_MODEL( dir=&scoredir.,
  datastep=&scorename..sas,
  xml=&scorename..xml,
  modeldir=/sasmodels,
  modelname=m6score,
  action=replace);

%INDHD_RUN_MODEL(inputtable=sampleddata,
  outputtable=sampleddata9score,
  scorepgm=/sasmodels/m6score/m6score.ds2,
  trace=yes,
  platform=spark);

```

The screenshot shows the Hadoop web interface for 'All Applications'. On the left is a navigation menu with options like 'Cluster', 'About Nodes', 'Node Labels', 'Applications', and 'Scheduler'. The main area displays 'Cluster Metrics' with a table of application statistics:

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes
343	0	0	343	0	0 B	150 GB	0 B	0	38	0	2

Below this, 'Scheduler Metrics' are shown, including 'Capacity Scheduler' and 'Scheduling Resource Type' as '[MEMORY]'. A table of application entries is visible, with one entry highlighted:

ID	User	Name	Application Type	Queue	Application Priority	StartTime	FinishTime	State	FinalStatus	Running Containers	All
application_1531011404729_0355	hive	SAS Scoring Accelerator [in: default.sampledata] [out: default.sampledata9score]	SPARK	default	0	Thu Feb 28 18:55:23 -0500 2019	Thu Feb 28 18:55:47 -0500 2019	FINISHED	SUCCEEDED	N/A	N/

Display 4. Model Scoring in Spark Using SAS Scoring Accelerator from MVA SAS

To execute the job in Spark, either set the HADOOPPLATFORM= option to *SPARK* or set PLATFORM= to *SPARK* inside the INDHD_RUN_MODEL macro. SAS Scoring Accelerator uses SAS Embedded Process to execute the Spark job with the job name containing the input table and output table.

EXECUTING USER-WRITTEN DS2 CODE USING SPARK

User-written DS2 programs can be complex. When running inside a database, a code accelerator execution plan might require multiple phases. By generating Scala programs that integrate with the SAS Embedded Process program interface to Spark, the many phases of a Code Accelerator job can be comprised of one single Spark job.

IN-DATABASE CODE ACCELERATOR

SAS In-Database Code Accelerator on Spark is a combination of generated Scala programs, Spark SQL statements, HDFS files access, and DS2 programs. SAS In-Database Code Accelerator for Hadoop enables the publishing of user-written DS2 thread or data programs to Spark, where they can be executed in parallel, exploiting Spark's massively parallel processing power. Examples of DS2 thread programs include large transpositions, computationally complex programs, scoring models, and BY-group processing. For more information about DS2 BY-group processing, consult the SAS In-Database product documentation.

To use Spark as the execution platform, the DS2ACCEL option in the PROC DS2 statement must be set to *YES* or the DS2ACCEL system option must be set to *ANY*; and the HADOOPPLATFORM system option must be set to *SPARK*. In addition, the Hive table or HDFS file that is used as input must reside on the cluster, and SAS Embedded Process must be installed on all the nodes of the Hadoop cluster that can run a Spark Executor.

There are six different ways to run the code accelerator inside Spark. They are called Cases. The generation of the Scala program by the SAS Embedded Process Client Interface depends on how the DS2 program is written. In the following example, we are looking at Case 2, which is a thread and a data program, neither of them with a BY statement:

```
proc ds2 ds2accel=yes;
thread work.workthread / overwrite=yes;
    method run();
        set hdplib.cars;
        output;
end; endthread; run;

data hdplib.carsout (overwrite=yes); dcl thread work.workthread m;
dcl double count;
keep count make model;
method run(); set from m; count+1; output;

end; enddata; run; quit;
```

The entire DS2 program runs in two phases. The DS2 thread program runs during Phase One, and its tasks are executed in parallel. The DS2 data program runs during Phase Two using a single task.

CONCLUSION

Hadoop plays an essential role in acquiring data as a data lake store in the ever-growing stores of a data-driven world. Managing and processing that data in an efficient manner is key to deriving business insights. With SAS® Data Management and SAS® Advanced Analytics, you can prepare data, model data, and score data, which can open many previously unknown possibilities . Combining Hadoop and SAS creates a powerful solution to achieve business goals.

Moving and processing data using Spark elevates the performance of the overall Analytics solution that SAS offers. With parallel data movement between CAS and Spark using SAS Data Connect Accelerator for Hadoop in Spark, scoring modeled data using SAS Scoring Accelerator for Hadoop and Spark, and most importantly, giving users the power and flexibility to write DS2 and DATA Step code to be executed in Spark using SAS In-Database Code Accelerator for Hadoop, you are now ready to collect, store, and process data with confidence using the power of SAS.

This paper has enabled you to explore all three of these areas using Apache Spark as the execution platform. The code samples that we have provided have prepared you to execute either the individual components or a combined analytic life cycle.

REFERENCES

Ghazaleh, David. 2016. "Exploring SAS® Embedded Process Technologies on Hadoop." Proceedings of the SAS Global Forum 2016 Conference. Cary, NC: SAS Institute Inc. Available <http://support.sas.com/resources/papers/proceedings16/SAS5060-2016.pdf>.

DeHart, C., Maher, S., and Kemper, B. 2017. "Introduction to SAS® Data Connectors and SAS® Data Connect Accelerators on SAS® Viya®." *Proceedings of the SAS Global Forum 2017 Conference*. Cary, NC: SAS Institute Inc. Available <https://support.sas.com/resources/papers/proceedings17/SAS0331-2017.pdf>.

Apache Spark Documentation. Available <https://spark.apache.org/docs/latest/configuration.html>

CONTACT INFORMATION

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

Kumar Thangamuthu
SAS Institute Inc.
kumar.thangamuthu@sas.com
www.sas.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.