Paper TT-03-2015

Leveraging Hadoop from the Comfort of SAS®

Emily Hawkins, UnitedHealthcare, Minnetonka, MN
Lal Puthenveedu Rajanpillai, UnitedHealthcare, Minnetonka, MN

ABSTRACT

Using Hadoop for distributed computing is rapidly becoming the most talked about, most sought after solution, and at times confusing ecosystem for big data. With a vast range of capabilities from data access tools to provisioning and governance tools, one can quickly become overwhelmed in a sea of funny names. However, by understanding a little bit about Hadoop the average SAS programmer can comfortably merge the Power of SAS with the distributed computing capabilities of Hadoop. This paper provides a basic introduction to Hadoop from a SAS programmer's perspective. It will explain why many companies see Hadoop as imperative to adopt into their enterprise solution. We we'll cover topics on connecting to HDFS using both LIBNAME and PROC SQL pass-through, as well as moving data in and out of HDFS with Pig using PROC HADOOP. One of the most important aspects of using Hadoop with SAS is pushing processing to the distributed cluster so we will also cover how to ensure that you are leveraging the power of Hadoop to run efficient programs.

INTRODUCTION

Hadoop is a distributed file system used commonly with storage and processing of big data. The core of Hadoop is commodity hardware running HDFS (Hadoop Distributed File System) and MapReduce. The commodity hardware allows for scalable inexpensive growth and MapReduce provides a simple distributed programming model. For these reasons more and more companies are turning towards Hadoop for distributed computing. The proficiencies of Hadoop compliment nicely the Power of SAS. SAS Enterprise Guide® users are able to use in-database processing with SAS/ACCESS® to Hadoop by using the SQL skills they already possess, running Pig scripts or working with explicit Hive Query Language (HQL). Hive is a service that allows users to write HQL, which is quite similar to SQL and then translates the code into a MapReduce job. MapReduce jobs are written in Java a language which pure SAS programmers may not be familiar with or want to learn. SAS and SQL programmers can easily begin using Hadoop by creating and querying Hive tables. The examples in this paper are written from the perspective of SAS users to work with data in HDFS and Hive.

In this paper we will discuss some strategies to incorporate moving data to Hadoop for storage and processing speed all while using SAS Enterprise Guide. Specifically we will look at how to connect to Hadoop, and then get into storage options with Hive to consider. Finally, we will touch upon working directly with HDFS from SAS using PROC HADOOP and Pig scripting. Consider there are hundreds of TB of historical data sitting in a SAS server. This data is infrequently queried, large, and static, yet for a number of reasons is necessary to be kept and accessible. Storage like this on traditional relational databases and servers can be costly to retain and maintain. This is the use case in which we have developed examples to work with throughout this paper. One thing we will cover throughout this paper is how to avoid bringing more data than necessary into SAS.

CONNECTION OPTIONS TO HIVE

As with relational databases SAS provides different options to connect to and run code in Hadoop. Both the concept of LIBNAME statements and PROC SQL can be used to access data in Hive.

The Hadoop data access from SAS is achieved using the SAS/ACCESS module for Hadoop. SAS/ACCESS works in conjunction with Hive in the Hadoop eco system. SAS/ACCESS will allow you to interact with Hadoop data similar to SAS datasets. You will be able to consume the data from familiar SAS tools and abstract the complexities and intricacies of working directly with Hadoop.

Taking advantage of the SASTRACE options in your code when working with SAS/ACCESS will greatly increase your understanding of the way SAS is processing queries. The option looks like this:

```
options sastrace=',,,d' sastraceloc=saslog nostsuffix;
```

These two options set at the beginning of your program will return a detailed log file of all the SQL statements that are sent to the DBMS. SASTRACELOC gives the location for the information; in this case (and is the default) the detail of the SQL will write to the SAS log. This is especially important when trying to maximize the efficiency of code by pushing as much of the processing to where the data resides.

PROC SQL PASS-THROUGH

PROC SQL is the common PROCEDURE that a SAS user uses to interact with a relational databases or SAS data sets. You can execute SQL statements to access data from Hive tables. The query will get executed in the Hadoop server and the resultant data will be available in SAS. During the execution SAS will determine whether or not to pull all or part of the data to the SAS environment.

The pass-through feature will allow the users to run HQL queries. The queries can be for data definition (DDL) or data manipulation (DML). Users can process queries such as joins and aggregations in Hive and bring back only the resultant data to SAS for any data analysis or reporting.

An example of a Hive query with pass-through to join two hive tables and get the aggregate values back to SAS is given below. The query will create a hive table and load the data for a HDFS file.

```
proc sql;

  connect to hadoop (user="user" password="Password"
             server=dbsld0032 port=10000 schema=default subprotocol=hive2
             cfg="/sas/hadoop/cfg-site.xml");
   execute
       (create table  if not exists  ledger_2_0
                  (  contract string,
                     customer string,
                     fiscal_date date,
                     found_contract_pbp string,
                     idb_date date,
                     idb_year string,
                     legal_entity string,
                     location string,
                     operating_unit string,
                     pbp string,product string,
                     project string,
                     reinsurance string,
                     source_sys string,
                     account string,
                     amount float  )
       row format delimited fields terminated by ',')
   by hadoop;

   execute
       (LOAD data inpath '/datalake/LEDGER.csv'
             overwrite into table ledger_2_0 )
       by hadoop ;
quit;
```

It is important to make sure that you are using the pass-through feature especially while running queries to join or extract or aggregate data from large datasets.

LIBNAME

SAS/ACCESS for Hadoop will allow users to define libraries that connect to the Hive server. The Hive tables in the library will look like SAS datasets. SAS will try to push the execution to Hadoop. This is also referred to as implicit pass-through.

```
libname hivelib hadoop server=servername port=10000 schema=default
     user="username" password="password" subprotocol=hive2
     cfg="/sas/hadoop/cfg-site.xml";
```

The hive configuration file and server details can be configured in SAS in the metadata. In such cases the configuration and authentication can be done without explicitly providing the configuration file or the server credentials.

Hadoop has its inherent limitations in appending data and inserting values to hive tables in an update mode Users should be aware of these limitations when they try to update tables in Hive. Loading data to hive tables are essentially copy or move of the source files to the Hive storage location

While executing a hive query SAS/ACCESS to Hadoop decides whether the data needs be pulled in to SAS environment. In situations where the data is to be brought back to Hadoop, SAS/ACCESS can directly interact with HDFS data and pull the data back to SAS in an efficient manner. This is transparent to the user.

PROS AND CONS OF CONNECTION TYPES

For any PROC SQL (with or without pass-through) SAS will convert the query to a Hive Query Language (HQL) and send it to Hive. The query will ultimately get executed in Hadoop as a MapReduce job.

The PROC SQL pass-through will allow the use of explicit HQL. Thus all the features that are supported by Hive can be utilized for processing Hive data. Only the final resultant data if any will be brought in to SAS. Users should be familiar with HQL queries.

For PROC SQL without pass-through users can use standard SQL queries. SAS will try to push the job to Hadoop environment whenever possible. But if the query is having statements or functions that are not compatible with the HQL language then the entire table will be pulled to SAS before processing. This will be inefficient and time consuming.

MOVING SAS DATA SETS TO HIVE

When choosing data to move from SAS to Hive storage, consider data that is infrequently accessed, yet necessary to keep for reporting, data that has high velocity, volume and variety or data that is unstructured. As with any database that you have write access to, SAS/ACCESS will write tables using a libname statement. Assuming the LIBNAME statement has already been run we can easily move SASHELP.CARS to Hive by running a DATA step.

```
data hivedb.cars_to_hive;
set sashelp.cars;
run;
```

The log that follows shows that the data set was created and the string variables retained their formatted length. The metadata from the tables like the variable labels, formats and lengths were not written to the Hive table.

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

HADOOP_87: Executed: on connection 3
```

```
CREATE TABLE `CARS_TO_HIVE` (`Make` STRING,`Model` STRING,`Type`
STRING,`Origin` STRING,`DriveTrain` STRING,`MSRP` DOUBLE,`Invoice`
DOUBLE,`EngineSize` DOUBLE,`Cylinders` DOUBLE,`Horsepower` DOUBLE,`MPG_City`
DOUBLE,`MPG_Highway` DOUBLE,`Weight`
DOUBLE,`Wheelbase` DOUBLE,`Length` DOUBLE) ROW FORMAT DELIMITED FIELDS
TERMINATED BY '\001' LINES TERMINATED BY '\012' STORED AS
TEXTFILE TBLPROPERTIES ('SAS OS Name'='Linux','SAS
Version'='9.04.01M1P12042013','SASFMT:Make'='CHAR(13)','SASFMT:Model'='CHAR(40)
','SASFMT:Type'='CHAR(8)','SASFMT:Origin'='CHAR(6)','
SASFMT:DriveTrain'='CHAR(5)')

NOTE: There were 428 observations read from the data set SASHELP.CARS.
NOTE: The data set HIVEDB.CARS_TO_HIVE has 428 observations and 15 variables.
```

Other notable comments in the log are that the Hive table was written as a TEXTFILE and the file was delimited by '\001'. Text file storage is the default for Hive, but there are several different options depending on the type of data you want to store. The CARS_TO_HIVE table that was created in Hive is a managed table. This means that if the data is dropped from Hive then the physical file in HDFS will also get deleted. If the table is created as an external table in Hive, when the data is dropped it will still be available outside of the Hive warehouse. This is helpful if there are a number of developers and analysts who want to use the data and don't necessarily use SAS. The option to create an external table is DBCREATE_TABLE_EXTERNAL=YES.

```
data hivedb.cars_to_hive (DBCREATE_TABLE_EXTERNAL=YES);
     set sashelp.cars;
     where make = 'BMW';
run;
```

And the converted SQL

```
HADOOP_246: Executed: on connection 3
CREATE EXTERNAL TABLE `CARS_TO_HIVE` (`Make` STRING,`Model` STRING,`Type`
STRING,`Origin` STRING,`DriveTrain` STRING,`MSRP`
DOUBLE,`Invoice` DOUBLE,`EngineSize` DOUBLE,`Cylinders` DOUBLE,`Horsepower`
DOUBLE,`MPG_City` DOUBLE,`MPG_Highway` DOUBLE,`Weight`
DOUBLE,`Wheelbase` DOUBLE,`Length` DOUBLE) ROW FORMAT DELIMITED FIELDS
TERMINATED BY '\001' LINES TERMINATED BY '\012' STORED AS
TEXTFILE TBLPROPERTIES ('SAS OS Name'='Linux','SAS
Version'='9.04.01M1P12042013','SASFMT:Make'='CHAR(13)','SASFMT:Model'='CHAR(40)
','SASFMT:Type'='CHAR(8)','SASFMT:Origin'='CHAR(6)','
SASFMT:DriveTrain'='CHAR(5)')
```

To change the storage type in the Hive table, use the data step option DB_CREATE_TABLE_OPTS=. Defining the value of the available storage types in Hive is outside the scope of this paper, but it is important to note that options are available to control the way the Hive table is created.

HIVE QUERIES

If you've worked with SAS long enough you know that not all code is created equal just because it returns the same output. This is the case with SAS/ACCESS to Hadoop as well. The best way to ensure all code is being run in-database is to use a PROC SQL pass-through and write explicit Hive SQL. SAS has useful functions that will translate but if there is one function or statement that cannot be translated to Hive SQL the whole of the data will be returned to SAS and then the function or statement will be performed.

EXAMPLE 1

Here is a basic query to the SASHELP.CARS data which has already been moved into a Hive table.

```
proc sql;
```

```
          create table sample as
          select make, model
          from hivedb.cars;
        quit;
```

With the SASTRACE options set the log will return with descriptive verbiage of the SQL that was run in Hadoop. This is the goal; that all processing would complete in Hadoop.

```
HADOOP_123: Prepared: on connection 4
CREATE TABLE sasdata_22_38_57_471_00002 ROW FORMAT DELIMITED FIELDS TERMINATED
BY '1' LINES TERMINATED BY '10' STORED AS TEXTFILE
LOCATION '/tmp/sasdata_22_38_57_471_00002' AS SELECT  `CARS`.`make`,
`CARS`.`model`  FROM `CARS`

HADOOP_124: Prepared: on connection 4
SELECT * FROM sasdata_22_38_57_471_00002  --/* describe columns */

HADOOP_127: Executed: on connection 4
DROP TABLE sasdata_22_38_57_471_00002

NOTE: Table WORK.SAMPLE created, with 428 rows and 2 columns.

33         quit;
NOTE: PROCEDURE SQL used (Total process time):
      real time           22.98 seconds
      user cpu time       0.02 seconds
      system cpu time     0.02 seconds
      memory              3271.34k
      OS Memory           23964.00k
```

In the first section, the SQL is translated to Hive and Hive created a temporary table in stored as a text file which is the default way to store tables. The temporary table is then selected to write to the SAS data set and the temporary table was dropped in Hive. In this example the query executed as expected and was efficient because it was able to complete in Hadoop and return the result to SAS.

EXAMPLE 2

This example shows what happens when functions are submitted that will not translate to a Hive query.

```
proc sql;
create table transform as
select  sum(amount)*1.2           as balance,
        substr(store,4,2)         as state,
        intnx('month',rpt_period,2) as forecast
from hivedb.account_balance
where upper(category) = 'toys'
group by location, calculated forecast
;quit;

ACCESS ENGINE:  SQL statement was not passed to the DBMS, SAS will do the
processing.
HADOOP_20: Prepared: on connection 1

HADOOP_74: Prepared: on connection 3
CREATE TABLE sasdata_20_06_08_332_00004 ROW FORMAT DELIMITED FIELDS TERMINATED BY
'44' LINES TERMINATED BY '10' STORED AS TEXTFILE
LOCATION '/tmp/sasdata_20_06_08_332_00004' AS SELECT  `account_balance`.`amount`, `
account_balance `.`location`, ` account_balance `.`rpt_period`,
` account_balance `.`category`  FROM ` account_balance `  WHERE  (UPPER( `category`)
= 'toys' )
```

```
HADOOP_75: Prepared: on connection 3
SELECT * FROM sasdata_20_06_08_332_00004  --/* describe columns */


HADOOP_76: Prepared: on connection 3
DESCRIBE FORMATTED LEDGER_2_0


HADOOP_78: Executed: on connection 3
DROP TABLE sasdata_20_06_08_332_00004

NOTE: Table WORK.TRANSFORM created, with 52 rows and 3 columns.
NOTE: PROCEDURE SQL used (Total process time):
      real time           26.26 seconds
      user cpu time       0.03 seconds
      system cpu time     0.04 seconds
      memory              5823.37k
      OS Memory           25732.00k
      Timestamp           09/16/2015 08:06:30 PM
```

SAS documentation for SAS/ACCESS 9.4 has a complete list of functions that will pass to Hadoop for processing. In this case, the function INTNX is not supported by Hadoop, and thus the log shows that it was not passed to Hadoop and SAS will do the processing. This is easy to see in the SQL that was submitted to SAS in the CREATE TABLE statement. The PROC SQL was converted to a basic CREATE TABLE with the where clause.

The simplest way to be certain that Hadoop will be doing the heavy lifting in any query is to use a pass-through query. In this, explicit Hive code is used and the developer knows that all processing will be done in-database. Since Hive SQL is so similar to SQL it is preferential to use PROC SQL pass-through whenever data needs to be filtered from Hadoop and brought into SAS for analysis. In the event that a SAS data set needs to be joined with a Hive table, consider moving the SAS data set into Hive temporarily to do the join and filter the results in Hadoop.

WORKING DIRECTLY WITH HADOOP

Direct interaction with the Hadoop cluster from SAS becomes necessary when you need to execute any HDFS commands for creating folders or any HDFS file operations as part of data ingestion or other requirements. Also you may need to analyze data that is residing in HDFS that is not loaded to any Hive table.

SAS can interact directly with the Hadoop cluster using PROC HADOOP which can use a variety of tools available to Hadoop such as Pig, MapReduce, and HDFS commands already mentioned. PROC HADOOP needs a configuration file that has the details of the Hadoop environment. The file will be specified in the OPTIONS statement. The login credentials also need to be provided.

```
proc hadoop options="/sas/hadoop/bdpaas_dev_4_0_2/fin360/cfg-site.xml"
username="user" password="password" verbose;
```

PROC HADOOP interfaces with the Hadoop cluster and can execute HDFS commands directly. It also enables the user to execute the MapReduce tasks from SAS.

HDFS COMMANDS

A limited set of HDFS commands can be directly executed from SAS using PROC HADOOP. The following options are available:

- Copy data to HDFS from local file system
- Copy data from local file system to HDFS
- Delete/Rename files in HDFS
- Create/Delete Directories in HDFS

The HDFS operations are particularly helpful when a local file needs to be ingested to HDFS for storage or for further processing.

Example for copying a file to HDFS

```
proc hadoop options="/sas/hadoop/cfg-site.xml"
    username="user" password="password" verbose;
    hdfs copytolocal='/landing/Claims_feed.dat'
         out='/datalake/project/process/' overwrite;run;
```

MAPREDUCE JOBS

Users can submit a MapReduce job which is developed and available as a set of jar files. The map and reduce functionalities need to be implemented as required by the use case. The MAPREDUCE statement in PROC HADOOP will submit the statements from SAS in to the Hadoop cluster.

Typically most developers tend to utilize higher level languages like PIG or HQL than using low level programming paradigms like Map Reduce. Development, debugging and maintenance MapReduce jobs tend to be very time and effort intensive.

PIG SCRIPTS

Apache Pig is part of the HADOOP ecosystem which is used for analysis of large data sets. It uses a high level data flow and scripting language called Pig Latin. Pig Latin scripts can process the data in a distributed mode in the Hadoop cluster.

PROC HADOOP can be used to submit a script that is the written. The script can either be provided inline or from a script file to the Pig statement.

EXAMPLE 1:

Data is available in a HDFS file and the SAS user need to filter the records with a selection criteria and write the selected records to a file. The code is given below.

```
filename pscript1 "/users/lputhenv/Pigscript1.sas" ;

data _null_;
   file pscript1 ;
   put "LEDG2 = LOAD '/datalake/LLEDGER.csv' using PigStorage(',')
           AS (contract:chararray, customer:chararray, fiscal_date:chararray,
          ...
          project:chararray, reinsurance:chararray, source_sys:chararray,
          account:chararray,
          amount:float) ;" ;
    put "LEDG3 = FILTER LEDG2 BY customer != '8100000' ;" ;
    put "STORE LEDG3 INTO '/datalake/LEDGER_FILTERED.csv' using PigStorage(',');" ;
run;

proc hadoop options="/sas/hadoop/cfg-site.xml"
     username="user" password="password" verbose;
      PIG code=pscript1
     ;
run;
```

In the code above, we have created a script pscript1 in the DATA step. The Script has a LOAD statement which loads the data into native Pig storage, then a FILTER statement which will filter the records. Finally we use a STORE statement which will write the records to a output file. The PROC HADOOP procedure will take the pscript1 file and execute it in the Hadoop cluster.

EXAMPLE 2:

Summary data is available in a SAS data set. The user needs to get the details from a HDFS file for a selected number of records. In this case the data can be filtered from SAS and write out to an HDFS file. A Pig script can be used to join the HDFS file with the filtered data. The resulting details can be read in to a SAS data set and used for further analysis or reporting.

Filter SAS data and load to Hadoop:

```
proc sql;
   create table account_sel as
   select distinct account from ledger
   where substr(account,1,1) = '7' ;
  quit;
run;

filename accfile hadoop '/datalake/acclist.txt'
         options="/sas/hadoop/cfg-site.xml"
         username="user" password="password";

data _NULL_ ;
   set account_sel ;
   file accfile;
   put @1 account $5;
run;
```

Pig script to join tables:

Script Pigscript2 (`/hpsasfin/Pigscript2.sas)`:

```
LEDG2 = LOAD '/datalake/optum/optuminsight/dfarchpoc/lputhenv/LALTEST_LEDGER.csv'
using PigStorage(',')
           AS (contract:chararray, customer:chararray, fiscal_date:chararray,
                                ..
                             account:chararray,amount:float) ;
ACCT1 = LOAD '/datalake/optum/optuminsight/dfarchpoc/lputhenv/acclist.txt' using
PigStorage(',')
          AS (taccount:chararray) ;
LEDG3 = JOIN LEDG2 BY account , ACCT1 by taccount;
STORE LEDG3 INTO '/datalake/filtered_ledger.csv' using PigStorage(',');
```

Excecute Pigscript2:

```
filename pscript2 "/hpsasfin/Pigscript2.sas" ;

proc hadoop options="/sas/hadoop/cfg-site.xml"
     username="user" password="password" verbose;
    PIG code=pscript2
     ;
run;
```

Read the results from HDFS to SAS:

```
filename accfile hadoop "/datalake/filtered_ledger.csv " ;

data ledger_result ;
  infile accfile delimiter = ',' MISSOVER DSD ;
  input
        contract:$CHAR5.  customer:$CHAR7. fiscal_date:MMDDYY10.
        ..
        account:$CHAR5. amount:10.2 ;
  ;
run;
```

Pig scripts are very effective in analyzing large data. You can analyze data in HFDS without loading the data in to Hive or bringing to SAS.

Users need to have access to the Pig environment (Grunt shell) for development. This is more relevant when there is a need to develop complex scripts or for debugging. The Pig logs available in the Grunt shell are much more detailed than the logs returned in the SAS environment.

CONCLUSION

- The use of SAS/Access to Hadoop and Hive allows the user to access HDFS data using SAS clients in the same way as they access SAS datasets.
- Accessing Hadoop data from SAS is not as seamless as accessing data from other RDBMS database. Hive is basically high latency data storage. Many of the functions and operations in SAS are not supported in Hive and SAS end up pulling huge amount of data from Hadoop to SAS. This will have the significant performance impact.
- Hive performs well for larger data size. It is a great choice to store history or details data which are required for data analysis and data mining.
- The most efficient way of working with Hadoop data from a SAS environment is use a blend of SAS and Hadoop functionality. For achieving this users need to have the knowhow on the Hadoop and SAS environment. They also need access to the Hadoop cluster to develop and debug HQL / Pig scripts. Using the SASTRACE options to get detail logs while running Hive queries from SAS will provide valuable insight when creating implicit Hive queries.
- Hadoop technologies are fast evolving. Various distributions of Hadoop are also available in market. Compatibility of SAS to Hadoop depend on the version of SAS Installation and that of Hadoop components such and Hive , Pig, YARN, Tez etc. (We used SAS Version 9.4.1. and used Hive 0.13. for all the examples).

REFERENCES

- Capriolo, Edward, Dean Wampler, Jason Rutherglen. *Programming Hive*. Sebastopol: O'Reilly Media, Inc., 2012. Print.
- White, Tom. *Hadoop: The Definitive Guide*. 3rd Edition. Sebastopol: O'Reilly Media Inc., 2012. Print.
- "SAS/ACCESS(R) 9.4 for Relational Databases: Reference, Seventh Edition." *SAS/ACCESS(R) 9.4 for Relational Databases: Reference, Seventh Edition*. SAS, n.d. Web. 16 Sept. 2015.

CONTACT INFORMATION

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

Emily Hawkins
UnitedHealthcare
E-mail: emily_p_hawkins@uhc.com

Lal Puthenveedu Rajanpillai
UnitedHealthcare
E-mail: lal_rajanpillai@uhc.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.