Paper SAS5244-2016

Unleashing High-Performance Risk Data With the Hadoop Custom File Reader

Stacey Christian, Michael Whitcher, Don McAlister, Phil Hanna

SAS Institute Inc.

ABSTRACT

SAS® High-Performance Risk distributes financial risk data and big data portfolios with complex analyses across a networked Hadoop Distributed File System (HDFS) grid to support rapid in-memory queries for hundreds of simultaneous users. This data is extremely complex and must be stored in a proprietary format to guarantee data affinity for rapid access. However, customers still desire the ability to view and process this data directly. This paper demonstrates how to use the SAS® High-Performance Risk Custom File Reader to directly access risk data in Hadoop MapReduce jobs, using the DS2 procedure, and the LASR procedure.

INTRODUCTION

Until now, accessing SAS High-Performance risk data by products outside of the SAS High-Performance Risk space has not been possible. The HPRISK procedure creates risk cubes in the form of a highly compressed, proprietary set of non-relational data files. This data is distributed across the customer's Hadoop Distributed File System (HDFS). The procedure enables high-performance analytical computations that scale according to the number of computers in the cluster (or grid). However, it also means that the high-performance risk data is unreadable by other SAS® products that operate on more traditional row and column (relational) tables. Sure, you can use PROC HPRISK to create a rectangular data set alongside the risk cube, but now you have two copies of a very large amount of data. There is now a better way.

Welcome the SAS High-Performance Risk Custom File Reader for High-Performance Risk Data (HPR Reader). It was created to enable PROC HPRISK users to leverage the native Hadoop file reader interface available in MapReduce and to transform a High-Performance Risk cube into a relational table view without a second copy of the data. This is done on-the-fly and in-memory. Once presented in a tabular form by the HPR Reader, the data is directly accessible to virtually all other SAS products. Specifically, we will demonstrate using the HPR Reader technology with the SAS In-Database Code Accelerator for Hadoop and with PROC LASR to transform and load high-performance risk data directly into SAS Visual Analytics for reporting. We will also include code examples to quickly get you started, and supply best practices for configuring and using the technology.

THE BUSINESS PROBLEM

One of the best ways to understand a new technology is to place it in the context of an actual problem. Throughout this paper, coding examples are used to answer the following business questions:

ANALYZING A PORTFOLIO OF CORPORATE BONDS

A portfolio of corporate bonds can lose money when the credit rating of an issuing company drops. A default of a company certainly triggers loss, but there is also loss when a company is downgraded since replacing the original bond costs more than the current bond is worth. To analyze the potential loss of our portfolio, we can run a simulation to see how bond values vary under many market states over time.

Using PROC HPRISK, we run a simulation. A complete simulation may include 100,000 market states, 1M bonds, and in our analysis we compute 8 output variables (for example, loss, cumulative loss, etc.) over the course of five years. The resulting cube would be roughly 6 terabytes of price data. PROC

HPRISK offers a great exploration tool to look at this data and calculate various quantile statistics like Value at Risk.

However, we sometimes want to perform other ad hoc calculations that the HPRISK procedure knows nothing about. Specifically, we want to do post processing of our HPRISK procedure results to isolate the simulated states where credit-related losses occurred. At the same time, we also want to prepare our data for reporting in SAS Visual Analytics.

Specifically, we want to answer these three questions:

1. What is my expected loss from bonds that migrated from AAA to AA over the course of five years?
2. Of my total loss over five years, how much comes from bond default?
3. Of my total loss over five years, how much comes from bonds getting downgraded?

The code example on the following pages will calculate the expected loss, as the mean of the simulated distribution, as well as the maximum and minimum losses for each of the three cases. At the same time, we will use standard data management techniques, which are available in DS2 to transpose the results from row-wise to column-wise formation.

If we had to export the data from the cube before performing these calculations, we would first have to create a 6-terabyte copy. The HPR Reader makes it easy to perform such data manipulations and calculations by directly reading the original risk cube data.

HOW THE SAS HIGH-PERFORMANCE RISK READER TECHNOLOGY WORKS

UNDERSTANDING THE HPR READER, DS2, AND THE SAS® IN-DATABASE CODE ACCELERATOR FOR HADOOP

Using the HPR Reader with DS2 is the simplest configuration possible. The HPR Reader is a Java JAR file that is installed on your SAS Workspace server tier. The SAS options, SAS_HADOOP_JAR_PATH and SAS_HADOOP_CONFIG_PATH, enable you to configure the use of PROC DS2 and the HPR Reader for your system. PROC DS2 uses these options to locate the HPR Reader JAR file, as well as your other Hadoop client-side JAR files. PROC DS2 can also find related Hadoop client-side configuration information. Together, these allow PROC DS2 to initiate the compilation of DS2 code for execution in-database in Hadoop.

Using the SAS In-Database Code Accelerator for Hadoop also means that the HPR Reader JAR will be automatically deployed in-database on the Hadoop cluster just prior to in-database execution. How does this work? The DS2 code runs inside a framework container called the SAS® Embedded Process. The SAS Embedded Process provides the run-time environment for the execution of DS2 code on each data node of the Hadoop cluster. The first time PROC DS2 invokes the Code Accelerator, it will copy and deploy needed JARs to the cache on each node. This JAR cache is where the HPR Reader JAR is deployed just in time to allow the DS2 code running inside the EP to use it. Once the Embedded Process for the SAS session is terminated, the JAR cache for that user is deleted. See Figure 1 below.

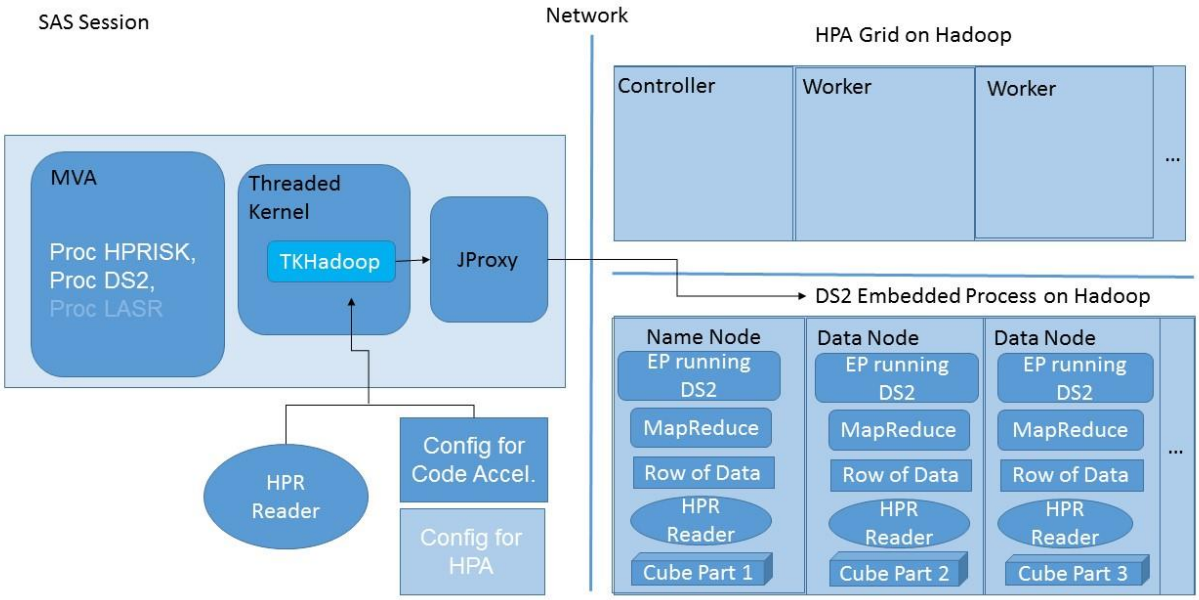

Figure 1. HPR Reader with the PROC DS2 Code Accelerator

UNDERSTANDING THE HPR READER WITH SAS® HIGH-PERFORMANCE ANALYTICS PROCEDURES

On the other hand, when using the HPR Reader directly with any SAS High-Performance Analytics procedure (including PROC LASR), the HPR Reader JAR must already be accessible on the grid. For this setup, the HPR Reader JAR must be copied to each node running on your grid. The *TKGrid_REP/tkmpirsh.sh* script is what is invoked for each SAS High-Performance Analytics grid request. Inside `tkmpirsh.sh`, there is a definition for SAS_HADOOP_JAR_PATH. By adding the local directory path to your HPR Reader JAR to this Java JAR path, Hadoop can locate the necessary HPR Reader classes when the MapReduce job is invoked by LASR. See Figure 2.

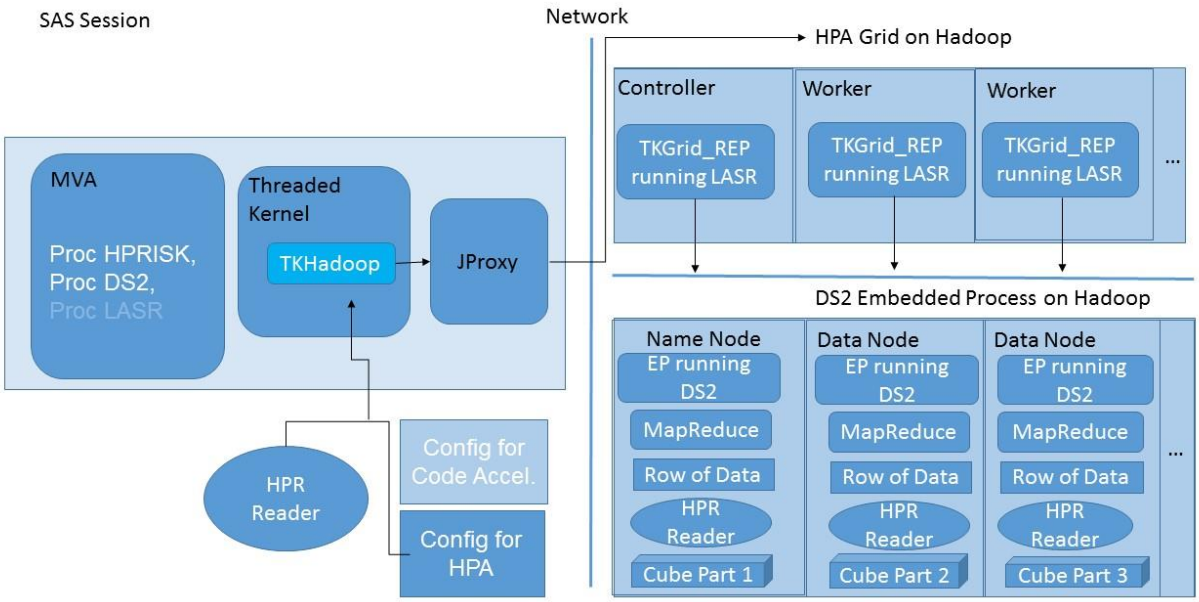

Figure 2. HPR Reader with the LASR Analytics Server

To ease configuration and maintenance, a best practice is to copy the HPR Reader JAR into the same directory on each TKGrid_REP node. For more details, see the section on "Best Practices for Using the High-Performance Risk Reader."

UNDERSTANDING THE RELATIONSHIP BETWEEN THE HPR READER AND HADOOP MAPREDUCE

The SAS Embedded Process obtains data from the HPR Reader by interfacing with the Apache Hadoop input format class. The HPR Reader extends this class by overriding the `RecordReader` method with a corresponding HPR Reader class that understands the format of the risk cube data, and is able to return row-wise data to the caller.

Whether using PROC DS2 or using HPA procedures, the SAS language runs on each data node of the Hadoop cluster inside a container called the SAS Embedded Process. The SAS Embedded Process provides the run-time environment to allow execution of complex data management operations in-database. When DS2 requests data, the HPR reader is invoked to supply the next row of data. Running inside the Map task of the Hadoop MapReduce job on each node of the cluster, the HPR Reader reads the MapReduce split information, and positions the start of its processing to the HDFS file offset for its partition of the data. The HPR Reader knows how to read the cube file format and reconstruct the data row-by-row as the calling process requests it. Only the partition of the data located on the node where the HPR Reader is running will be returned. Thus each HPR Reader operates on a different partition of the HDFS data that is local to that node.

UNDERSTANDING THE PROC HPRISK ENHANCEMENTS

A new statement has been added to PROC HPRISK to support the HPR Reader. The CUBETOTABLE statement allows users to register a risk cube along with the HPR Reader in any PROC HPRISK task. Registration can be at the time of cube creation, or later when a cube is queried. Behind the scenes, the

CUBETOTABLE statement uses PROC HDMD syntax to describe the risk data as a relational table. PROC HPRISK generates and stores the SASHDMD meta-information into HDFS as a file. DS2 uses the SASHDMD file to discover how the HPR Reader lays out the fixed row of data for the risk cube. Additional meta-information contained within the SASHDMD file lets the DS2 process know the encoding for character data, and the endianness for numeric data.

Required input arguments for the CUBETOTABLE statement include name of the cube, and the destination data store for the resulting relational table description. Here is an example CUBETOTABLE statement syntax that will register a SASHDMD table description, called PriceTable, in HDFS. A subsequent PROC DS2 or PROC LASR step will then be able to read the risk data and process the rows being returned from the HPR Reader.

```
libname myrskdat hadoop
   server="<HadoopNameNodeHost>"
   user="<yourUser>" password="<yourPW>"
   HDFS_DATADIR="/local/data/example"
   HDFS_TEMPDIR="/tmp"
   HDFS_METADIR="/local/data/example";

options set=SAS_HADOOP_JAR_PATH="/pathTo/client/Hadoop/…HPRreader/jar";
options set=SAS_HADOOP_CONFIG_PATH="/pathTo/client/Hadoop/config";
options set=GRIDHOST=”host.domain.com”;
options set=GRIDINSTALLLOC=”/pathTo/risk/install/location/TKGrid_REP”;

proc hprisk
  task=query
  cube=”/pathTo/cube/descriptor/MyCube”;
  cubetotable table=myrskdat.PriceTable  content=price;
run;
```

Note: The values for HDFS_DATADIR, HDFS_TEMPDIR and HDFS_METADIR depend on your particular Hadoop installation. Contact your Hadoop system administrator for the values.

The TABLE= option can be to any SAS/ACCESS® Interface to Hadoop library.table name. When the CUBETOTABLE statement executes, the HPRISK cube descriptor is opened to gather information about the cube. CUBETOTABLE will then generate and store the SASHDMD table descriptor into HDFS. Note that no actual data is accessed by this step. It simply sets up the definition of your risk cube view within HDFS.

Now that we have looked at an example, here is the full syntax for the CUBETOTABLE statement.

```
CUBETOTABLE TABLE=<HadoopLibrary>.<TableName>
            [REPLACE|NOREPLACE]
            [CONTENT = (PRICES | POSITIONS)];
```

When you are using the REPLACE option, any previous version of the SASHDMD file descriptor is deleted before creating the new one. The default is NOREPLACE, which will instead give an error if a SASHDMD file by the same name already exists.

The CONTENT= option enables you to access different types of data within the risk cube. CONTENT=PRICES is the default. It instructs the HPR Reader to return the prices (values) for the output variables specified in the risk cube along with the classification information (dimension variables). CONTENT=POSITIONS is a variation that returns instrument information for all the positions in the risk cube (that is, the portfolio that was analyzed). PRICES and POSITIONS map to different Java classes found within the HPR Reader. When the Hadoop Metadata (SASHDMD) is written, the name of which class to use is stored in the SASHDMD file. Thus, multiple CUBETOTABLE registrations of the same cube using different TABLE= names can be made, each specifying a different CONTENT= value. Also,

note that while both the Price cubes and a Stress cubes created in HPRISK can be used with CUBETOTABLE, Join cubes are not yet supported.

Once CUBETOTABLE registration is complete, the virtual risk table is ready for use.

USING THE SAS HIGH-PERFORMANCE RISK READER WITH DS2

Once the CUBETOTABLE statement has been run to create the SASHDMD file in HDFS, you are ready to execute DS2 program. In this section, we return to our portfolio for the corporate bonds example, and show how to write DS2 to compute the results to answer our earlier questions.

Here is a DS2 program that will compute the results. It is written as a SAS macro, so we can use the same program to answer all three of our questions stated earlier. Note that this code is a subset of the original code (which was too long to place within the paper). The full example can be downloaded from our website.

```
/*-----------------------------------------------------------------------
 * Using DS2
 * Calculate Incremental Risk Charge for a given downgrade criteria.
 *-----------------------------------------------------------------------
*/
%macro Incremental_Risk_Charge(DOWNGRADE_CRITERIA=,
                               OUT_TABLE=,
                               DATA=);

proc ds2 ds2accel=yes;

/* For each thread reading from the Hadoop MapReduce job */
thread th_pgm / overwrite=yes;

dcl int EnableOutput;

/* Compute Expected Losses for each Instrument having 5 Horizons */
vararray double expected_loss[5] expected_loss_1yr expected_loss_2yr
               expected_loss_3yr expected_loss_4yr expected_loss_5yr;
vararray double cnt_loss[5] cnt_loss_1yr cnt_loss_2yr  cnt_loss_3yr
                           cnt_loss_4yr cnt_loss_5yr;

retain EnableOutput
          expected_loss_1yr expected_loss_2yr expected_loss_3yr
          expected_loss_4yr expected_loss_5yr
          cnt_loss_1yr cnt_loss_2yr cnt_loss_3yr cnt_loss_4yr
          cnt_loss_5yr;

keep Industry InstID InstType
         expected_loss_1yr expected_loss_2yr
         expected_loss_3yr expected_loss_4yr expected_loss_5yr;

method run();
    dcl int i;
    set &DATA; by InstId StateNumber;

    if ( FIRST.InstID ) then do;
       /* re-initialize stat arrays at the start of each group */
       do i = 1 to dim(expected_loss);
          expected_loss[i] = 0.0;
          cnt_loss[i] = 0.0;
       end;
```

```
        EnableOutput = 0;
    end;

    /* Note: Exclude the base case from the calculation
             And apply passed in criterion. */
    if ( ( missing(simulationtime) eq 0) and
           &DOWNGRADE_CRITERIA and
           cumulative_credit_migration_loss gt 0.0 ) then do;
       expected_loss[simulationtime] = expected_loss[simulationtime] +
             cumulative_credit_migration_loss;
       cnt_loss[simulationtime] = cnt_loss[simulationtime] + 1.0;
       EnableOutput = 1;
    end;

    if ( LAST.InstID /* Last obs in this group */ ) then do;
       if (EnableOutput) then do;
          if (cnt_loss[simulationtime] gt 0.0) then do;
             expected_loss[simulationtime] =
                 expected_loss[simulationtime] /
                 cnt_loss[simulationtime];
          end;
          output;
      end;
      EnableOutput = 0;
    end;
end; /* method run() */

endthread;
run;

data &OUT_TABLE.;
   dcl thread th_pgm m;

   /* Note: Additional post thread computational logic can go here */
   method run();
      set from m;
   end;
enddata;

run; quit;

%mend Incremental_Risk_Charge;
```

Here again are the questions we would like to answer concerning our bond portfolio. By applying a different DS2 filter to our program we are able to answer all three.

1. What is my expected loss from bonds that migrated from AAA to AA over the course of 5 years?

```
%let AAA_to_AA=bond_initial_rating eq 1 and end_rating eq 2;
   %Incremental_Risk_Charge(downgrade_criteria= &AAA_to_AA,
                            out_table=  myrskdat.AAA_to_AA,
                            data= myrskdat.incrementalriskcharge);
```

2. Of my total loss over 5 years, how much comes from bond default?

```
%let BondsDefaulting=bond_initial_rating<8 and end_rating=8;
   %Incremental_Risk_Charge_DS2(downgrade_criteria= &BondsDefaulting,
                                out_table= myrskdat.Bonds_to_Default,
                                data= myrskdat.incrementalriskcharge);
```

3. Of my total loss over 5 years, how much comes from bonds getting downgraded?

```
%let BondsDowngrading=bond_initial_rating<end_rating and end_rating ne 8;
%Incremental_Risk_Charge(downgrade_criteria= &BondsDowngrading,
                         out_table= myrskdat.Bonds_DownGrading,
                         data= myrskdat.incrementalriskcharge);
```

When the PROC DS2 step is submitted from your SAS session (or SAS Workspace Server), DS2 constructs an underlying Hadoop MapReduce job. The DS2 program is sent over to the Embedded Process where it compiles the code for each node on the cluster. The DS2 thread routine is connected to the Mapper portion of the MapReduce job and begins to read records from the HPR Reader component. DS2 is able to run in the SAS Embedded Process on each node of the Hadoop cluster, and the HPR Reader manages the reading and transforming of risk cube information into relational rows that are supplied to DS2 for processing. Results from the DS2 program can either be stored back into HDFS as a delimited text file or the rows can be returned to the SAS session.

The full breath of in-database DS2 language syntax is available for use with the HPR Reader. Refer to the online version of the *SAS® 9.4 DS2 Language Reference, Fifth Edition*, available at support.sas.com, for syntax and additional DS2 coding examples.

USING THE SAS HIGH-PERFORMANCE RISK READER WITH LASR ANALYTICS SERVER

After running the above code against our risk cube using the HPR Reader, we would like to load the resultant data into an in-memory table in the SAS LASR Analytics Server. Once loaded into LASR, it is available for further analysis and reporting by SAS® Visual Analytics Designer. For an example, see Figure 3.

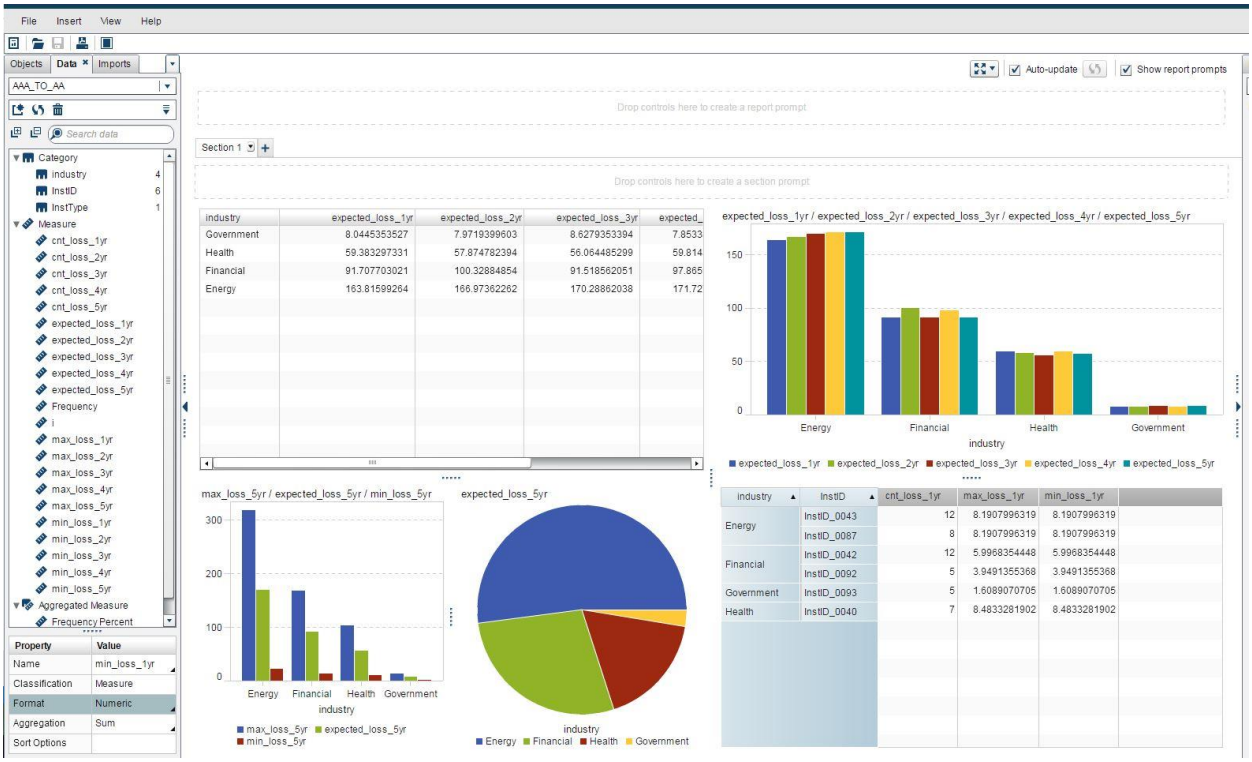

Figure 3 - SAS Visual Analytics Designer showing the expected losses due to AAA to AA rating downgrades

In this report the expected loss is the average of the simulated states that resulted in an AAA bond dropping to an AA rating. We can compare this across different industries. Similarly, we can compare the maximum loss from the simulated states and the minimum loss. We can also observe how these values change over the five years of the simulation.

The default output table format from an in-database PROC DS2 step is <cntl-A> delimited text file. PROC LASR is able to load this data directly from HDFS into the LASR Analytics Server.

A common question is whether the data is brought back into the SAS Workspace Server before it is loaded into the SAS LASR Analytics Server. Since the SAS LASR Analytics Server runs alongside the Hadoop Server on the same grid, the answer is no. When reading delimited text output from DS2, PROC LASR is able to leverage a native Hadoop reader to load the data.

Note that PROC LASR could also be used to read data from the cube directly if no transformations were necessary. PROC LASR is also able to leverage the HPR Reader running inside of MapReduce, to directly load the data from Hadoop into the SAS LASR Analytics Server. The result is bulk-loading of your large data into LASR.

Once your risk cube data has been transformed and loaded into the SAS LASR Analytics Server, it takes the form of a standard SASHDAT table containing rows and columns. This means it is now accessible from other SAS applications, such as SAS Visual Analytics. By using the SASIOLA access engine, your relational table in LASR is also accessible from the SAS Workspace Server session. This makes the data accessible to other SAS procedures. Here is an example:

```
/* Connect to a running LASR server */
libname lasrlib sasiola port=12345 tag=myrskdat;

/* Upload a DS2 results table into LASR */
proc lasr add data= myrskdat.AAA_to_AA
          port=12345
          verbose;
run;
```

When PROC LASR loads the data, the value of the TAG= option is the name of the SAS LIBRARY. In the examples throughout this paper we have been using the myrskdat SAS LIBRARY, so an implicit TAG=myrskdat will be associated with the SAS LASR table. Therefore, when you define your LIBNAME statement, lasrlib, specify TAG=myrskdat to allow the SASIOCA SAS Access Engine to be able to resolve the two part LASR table name.

BEST PRACTICES FOR USING THE SAS HIGH-PERFORMANCE RISK READER

Begin with the end in mind. This is standard advice given to all computer science students, and never is it more useful than when configuring a multi-tiered software system. At its root, success relies on an understanding of how the final production system will be used. What are the problems being solved and the steps (workflows) needed to solve them? Which SAS procedures or other techniques will be used to accomplish the workflows? Because there are often multiple ways to accomplish the same task, which one(s) perform best? Knowing this information affects configuration.

In this section, we outline some of the tricks of the trade used to configure and run the HPR Reader. They are not meant to be all inclusive, but are offered as time savers to those in the field who are working with the technology.

HPR READER CONFIGURATION FOR THE SAS IN-DATABASE CODE ACCELERATOR FOR HADOOP

Let us look at a hypothetical workflow involving the use of the HPR Reader, PROC DS2, and PROC LASR to read and process risk cube data. The assumption is that the final results will reside on a SAS LASR Analytics Server, and SAS Visual Analytics will be used explore the data and display reports. But first there are one or more data management steps that begin with using the HPR Reader to extract rows

of data from an HPRISK cube, perform data prep, and write the results to a delimited file in HDFS. This workflow is the union of Figure 1 and Figure 2, so it makes it a good place to begin.

When you configure Figure 1 for use with PROC DS2 and the SAS Code Accelerator, the HPR Reader JAR must be accessible from the machine where the SAS Session and PROC DS2 step is running. When assigning the SAS library that will be used with PROC DS2, make sure you edit the SAS_HADOOP_JAR_PATH environment variable to include the directory where the HPR Reader JAR is located. A UNIX version might look like this.

```
options
set=SAS_HADOOP_JAR_PATH="/pathTo/client/Hadoop/jar;/pathTo/HPRreader/jar";
```

In addition, PROC DS2 must also know the location of the client-side Hadoop configuration files.

```
options set=SAS_HADOOP_CONFIG_PATH="/pathTo/client/Hadoop/config";
```

A listing of the configuration directory will show several XML files, including `mapred-site.xml`. This is the file that SAS uses to configure an in-database connection to Hadoop. There is a wealth of additional properties that can be used to configure and tune your system. Since there are too many to present in this paper, so we will focus on the ones related to configuring the Code Accelerator for use with PROC DS2 and the HPR Reader.

Output data from the HPR Reader returns fixed length records. Particularly when running SAS on a Microsoft® Windows system, set the following property.

```
<property>
   <name>sas.ep.output.record.format</name>
   <value>fixed</value>
</property>
```

When PROC DS2 is executed with DS2ACCEL=YES, a Hadoop MapReduce job is created in which to launch the HPR Reader. The MapReduce job must know the location of the HPR Reader JAR. Using the *mapreduce.job.jar property*, the Code Accelerator will copy the JAR to each node of the grid. These underlying components are then able to locate the JAR. Here is what this might look like for a Windows system.

```
<property>
   <name>mapreduce.job.jar</name>
   <value>c:\HPRreader\jars\sas.analytics.riskhp.cubereader.jar</value>
</property>
```

Note the path on a Windows machine must begin with a drive letter, as shown, and the path must be accessible from your SAS session or SAS Workspace Server. When running on a UNIX system, the path will be a standard UNIX directory.

These are the only two required properties. But to troubleshoot problems, another property can be set in the mapred-site.xml file to enable additional tracing information. It is called *sas.ep.server.trace.level*.

```
<property>
    <name>sas.ep.server.trace.level</name>
    <value>10</value>
</property>
```

For more information about this property, see the section entitled “Adjusting the SAS Embedded Process Performance” in the *SAS® 9.4 In-Database Products: Administrator’s Guide, Sixth Edition.*

The HPR Reader supports log4j tracing, and is inherited by the configuration passed from the SAS Embedded Process. So by enabling tracing for the SAS Embedded Process, tracing is also enabled for the HPR Reader.

Once the DS2/MapReduce job has completed, the Hadoop job history can be consulted for the messages. By default, Hadoop configures the job history to the following URL:

```
http://host.domain:19888/jobhistory
```

HPR READER CONFIGURATION FOR SAS HIGH-PERFORMANCE ANALYTICS PROCEDURES

SAS High-Performance Analytics procedures are configured to run inside a threaded-kernel, TKGrid framework alongside your Hadoop system. There is no implicit copying of JARs to the cluster as is done by the SAS Code Accelerator. Therefore, before you can run an HPA procedure with the HPR Reader, the HPR Reader JAR must first be accessible from the TKGrid and from the SAS session.

The HPA setup for the SAS session mirrors the configuration for the SAS Code Accelerator. Set your SAS_HAOOP_JAR_PATH environment variable to include the directory location of sas.analytics.riskhp.cubereader.jar. Also, edit the Hadoop client-side file, mapred-site.xml, and add the mapreduce.job.jar property to point to the location on the Hadoop cluster where the HPR Reader is copied. TIP: If working with a Windows client, it might be useful to maintain two sets of client-side configuration directories, one used by the SAS Code Accelerator and a second set of configuration files used by the HPA procedures.

When an HPA procedure is invoked, it is launched in-database on the TKGrid by using a script, *`TKGrid/tkmpirsh.sh`*. For the TKGrid to be aware of the HPR Reader, the `tkmpirsh.sh` script must be edited to include its location. So be sure to edit the `tkmpirsh.sh` script and include directory location of the HPR Reader on the SAS_HADOOP_JAR_PATH environment variable.

If using PROC LASR, your SAS Install Depot should include the LASRANLT package. During installation it will create a TKGrid_REP directory alongside the TKGrid directory with additional artifacts, including its own `tkmpirsh.sh` script, needed by the LASR Analytics Server. If using LASR, be sure to edit the `TKGrid_REP/tkmpirsh.sh` script to have SAS_HADOOP_JAR_PATH point to the location of the HPR Reader JAR.

TIPS TO USING PROC HPRISK TO REGISTER RISK CUBE DATA WITH THE HPR READER

If you are already familiar with using PROC HPRISK, this section will be very natural. We have already covered the use of the new CUBETOTABLE statement to register a risk cube for use with the HPR Reader. Remember that there are four types of cubes create by HPRISK: Price, Stress, and Aggregate Joins and Comparative Joins. Joins are not support by the CUBETOTABLE statement. Both Price cubes and Stress cubes containing Result Sets can be used with CUBETOTABLE with the PRICES option to access pricing data. All Stress cubes can be used with the POSITIONS option to access the portfolio data. However, Price cubes must be created with the `DistributePositions=YES` option to be used by the CUBETOTABLE with the POSITIONS option.

The HPR reader reads risk cube data that resides in the Hadoop Distributed File System. So when creating a risk cube that is to be read by the HPR reader, be sure to specify the `cube_store_type=hdfs` option on PROC HPRISK to direct the procedure to store the risk cube in HDFS.

The CUBETOTABLE statement can be combined with other HPRISK statements to build and register your cube all in one step. However, you can also make the CUBETOTABLE registration a separate step from cube creation. For example,

```
proc hprisk task=query cube="/pathTo/cube/descriptor/MyCube";
  cubetotable table=myrskdat.PriceTable  content=price;
run;
```

When reading and analyzing risk data, it is helpful to see the variables available to you. By turning on PROC HPRISK's debug feature, you can also see the underlying SASHDMD that is generated by the CUBETOTABLE step to perform the registration.

```
options mprint;
proc hprisk debug…;
```

The following HDMD tracing was produced using a combination of the `mprint` SAS system option and the PROC HPRISK debug option when we registered our example risk cube:

```
byte_order = LITTLEENDIAN
data_file = 'incrementalriskcharge_demo_1771760957.sasrdat'
encoding = wlt1
file_type = CUSTOM
input_class = 'com.sas.analytics.riskhp.cubereader.mapreduce.PricesInputFormat'
name = MYRSKDAT.INCREMENTALRISKCHARGE
record_length = 384;
column industry char(32) bytes=32 ctype=char offset=0;
column InstType char(32) bytes=32 ctype=char offset=32;
column InstID char(256) bytes=256 ctype=char offset=64;
column StateNumber double bytes=8 ctype=double offset=320;
column SimulationTime double bytes=8 ctype=double offset=328 ;
column SimulationReplication double bytes=8 ctype=double offset=336;
column VALUE double bytes=8 ctype=double offset=344;
column CUMULATIVE_CREDIT_MIGRATION_LOSS double bytes=8 ctype=double
                                                 offset=352;
column BOND_INITIAL_RATING double bytes=8 ctype=double offset=360;
column END_RATING double bytes=8 ctype=double offset=368;
column ReturnedValue double bytes=8 ctype=double offset=376;
```

This information is quite helpful as it spells out exactly what columns you should expect when using the HPR Reader with this cube as well as the type, length, and order for each column.

Also keep in mind that how you created your cube in HPRISK effects how your HPR Reader returns data on the grid. In our example we create the Price Cube using the PriceBy=Positions option. This means that each instance of the HPR Reader will receive all Simulation Replications for a subset of the positions on each node. On the other hand, if we had used the PriceBy=States option when creating the cube, each HPR Reader instance would have received a subset of the Simulation Replications, but for all positions.

TIPS TO USING THE HPR READER WITH DS2

As mentioned in an earlier section, when using PROC DS2, the full power and capability of in-database computational programming is available to you along with the HPR Reader. However, if you are a long time DATA step user, when it comes to working with HPRISK data, there are some differences between DATA step and DS2 that are worth mentioning.

PROC DS2 and the Code Accelerator supply the framework by which the HPR Reader can be executed inside the Map task of a Hadoop MapReduce job. This is accomplished by associating the DS2 thread with the Map task. This match up creates the following DS2 code pattern from which all of your in-database programs will start.

```
proc ds2 ds2accel=yes iptrace trace=all;
thread th_pgm / overwrite=yes;
  method run();
     set myrskdat.riskcube;
  end;
endthread;
run;
```

```
data myrskdat.riskcube_out;
dcl thread th_pgm m;
method run();
   set from m;
end;
enddata;
run;
quit;
```

Notice that the Code Accelerator is enabled by setting DS2ACCEL=YES, and that the thread program is overwritten from one invocation to the next. Additional tracing can be enabled by using `iptrace` and `trace=all`.

Another difference between DS2 and the DATA step is with BY processing. BY processing is a very powerful way to gather and group information that is distributed across a Hadoop cluster according to some other layout algorithm. However, the DATA step formats the groups based on the raw value of the data. DS2 formats the groups based on the formatted values of the data. In general this is not an issue for character data, but can be an issue when working with numeric data. Referring back to the DS2 code for our bond example, notice how we used `FIRST.InstID` and `LAST.InstID` to locate the beginning and ending of each group. Using FIRST. and LAST. is a best practice for DS2.

With DS2, both the THREAD and the DATA sections are available to pass down in-database. However, not every piece of DS2 syntax will run in-database. When faced with using statements or options that cannot run in-database, a best practice is to place these items in the DATA section, if possible. By continuing to allow the THREAD program to run on each node in the cluster you maximize the performance of the step. Of course, if your DATA section cannot run in-database, it will be executed in the SAS Session, so be aware of just how many rows are being returned to the SAS Session.

Other differences between DS2 and the DATA step exist. To learn more, see the section entitled "Differences between DS2 and the DATA step" in *SAS® 9.4 DS2 Language Reference, Fifth Edition*.

The DS2 code for our bond examples did not require use of the DS2 *init()* or *term()* methods, but these along with DS2 packages, functions, formats, and many other features are available for use in-database.

TIPS TO USING THE HPR READER WITH LASR ANALYTICS SERVER

After your DS2 data management and data prep step(s) complete, you are ready to load the data into the LASR Analytics Server. By default, the output from DS2 is a <cntl-A> delimited file, and the file should be in HDFS prior to loading it into LASR. PROC LASR uses the native Hadoop delimited file reader to read the data, so no additional setup is required. The PROC LASR ADD syntax is also straightforward.

```
proc lasr add data=myrskdat.riskcube_out
          port=12345
          verbose;
run;
```

The `verbose` option enables additional tracing information. Once the data is loaded into the SAS LASR Analytics Server, the SAS Visual Analytics Administrator is used to perform the LASR table metadata registration. The data is then useable by all the applications in the SAS Visual Analytics Hub.

CONCLUSION

SAS offers a suite of power software tools and solutions for analyzing and viewing information. Until now, access to High-Performance Risk Data was limited to the HPRISK procedure. By using the High-Performance Risk Cube Reader, users are now able to access, transform, and process HPRISK cube data directly from a variety of SAS procedures and store results into an HDFS relational table. This

opens the door for the full breath of SAS data management, computational analysis, and reporting using DS2, LASR, SAS Visual Analytics, and many others.

ACKNOWLEDGMENTS

The authors would like to acknowledge the following people for their insights and expertise on Hadoop, DS2, and SAS Hadoop Custom File Readers.

Robert Ray, Director, Data Management, SAS Institute Inc.

David Ghazaleh, Senior Software Developer, Data Management, SAS Institute Inc.

Alex Fang, Senior Development Tester, Quality User-driven Enterprise Software Testing, SAS Institute Inc.

RECOMMENDED READING

- *SAS® 9.4 In-Database Products: User's Guide, Fifth Edition*
- *SAS® 9.4 In-Database Products: Administrator's Guide, Sixth Edition*
- *SAS® 9.4 DS2 Language Reference, Fifth Edition*
- *SAS5060-2016: Exploring SAS® Embedded Process Technologies in Hadoop®*
- *Base SAS® 9.4 Procedures Guide, Fifth Edition*
- *SAS/ACCESS® 9.4 for Relational Databases: Reference, Seventh Edition*
- *SAS® High-Performance Risk 3.6: Procedures Guide*
- *Hadoop®: The Definitive Guide, Fourth Edition*, by Tom White

CONTACT INFORMATION

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

Michael Whitcher
SAS
(919) 531-7936
mike.whitcher@sas.com
https://www.linkedin.com/in/mike-whitcher-17a9138