Paper SAS1905-2015

Tips and Techniques for Efficiently Updating and Loading Data into SAS® Visual Analytics

Kerri L. Rivers and Christopher Redpath, SAS Institute Inc., Cary, NC

ABSTRACT

So you have big data and need to know how to quickly and efficiently keep your data up-to-date and available in SAS® Visual Analytics? One of the challenges that customers often face is how to regularly update data tables in the SAS® LASR™ Analytic Server, the in-memory analytical platform for SAS Visual Analytics. Is appending data always the right answer? What are some of the key things to consider when automating a data update and load process? Based on proven best practices and existing customer implementations, this paper provides you with answers to those questions and more, enabling you to optimize your update and data load processes. This ensures that your organization develops an effective and robust data refresh strategy.

INTRODUCTION

SAS Visual Analytics is an easy-to-use, web-based solution that leverages the SAS LASR Analytic Server and empowers organizations to explore small to large volumes of data very quickly. Your organization might be about to begin your SAS Visual Analytics journey or it might have already started down that path. The SAS Visual Analytics interface allows you to perform basic data operations. The tips and techniques presented in this paper enable you to move beyond the basics to a more advanced, effective, and robust data refresh strategy to meet your specific needs.

The examples in this paper are based on SAS LASR Analytic Server 2.5 and SAS Visual Analytics 7.1.

SAS® LASR™ ANALYTIC SERVER OVERVIEW

The SAS LASR Analytic Server is an analytic platform that provides a secure, multi-user environment for concurrent access to data that is loaded into memory across a non-distributed or distributed computing environment. By loading tables into memory for analytic processing, the SAS LASR Analytic Server enables business analysts to explore data and discover relationships in data at the speed of RAM.

Because the SAS LASR Analytic Server is an in-memory analytic platform, as opposed to an in-memory database, it is designed with a high-performance, multi-threaded, analytic code that processes client requests at high speeds. In contrast, an in-memory database invariably uses a query language primarily focused on extracting data before manipulating it. You ask a database to select rows from a table; you ask an analytic platform to produce a box plot or a heat map overlaid with a regression line.

The SAS LASR Analytic Server in a distributed environment supports the Hadoop Distributed File System (HDFS) for co-located storage of data with replication capabilities. The SAS implementation of HDFS offers some key benefits that include storing files in blocks that results in balanced memory utilization across the server and reduced execution time.

An alternative to co-located data storage is using one of several asymmetric data providers. SAS supports five possible asymmetric data providers: Cloudera CDH, Hortonworks, Oracle Exadata, Pivotal Greenplum, or Teradata. In asymmetric configuration, the source system containing the data has a SAS Embedded Process deployed within it. This process allows for parallel loading from the source system into the SAS Visual Analytics environment.

The topology for SAS Visual Analytics encompasses the client, middle, and server tiers. These tiers interact with the SAS LASR Analytic Server to enable client access to data for performing analytic tasks. Any actions initiated by clients are channeled by the middle tier and the server tier before they are sent to the SAS LASR Analytic Server process. In a distributed environment, the LASR root node sends the actions to the LASR worker nodes before results of actions are channeled back to the clients through the middle tier.

LASR actions can be single-pass or multi-pass through the data. For example, a simple summary action to aggregate data does a single pass through the data. In contrast, a correlation does multiple passes through the data. This could be particularly important in deciding how to structure your data table for exploration and reporting.

DATA TABLE STRUCTURE

SAS LASR Analytic Server supports two types of data structures: (1) analytical base table and (2) star schema. The underlying structure you use is transparent to the end user when interacting with explorations or reports. In both cases, the table structure appears to be a single analytical base table.

ANALYTICAL BASE TABLES

The structure of an analytical base table is as follows:

- Flat, fully materialized (that is, pre-joined)
- Atomic (that is, at the lowest level of granularity for reporting)
- Dimensions, facts, and measure variables all in one table

This is the standard format for analytics.

STAR SCHEMAS

A star schema is a simple representation of a data mart where relations are defined between a fact table and one or more dimension tables. The schema derives its name from picturing the dimension tables arranged around the fact table in a star (see Figure 1).

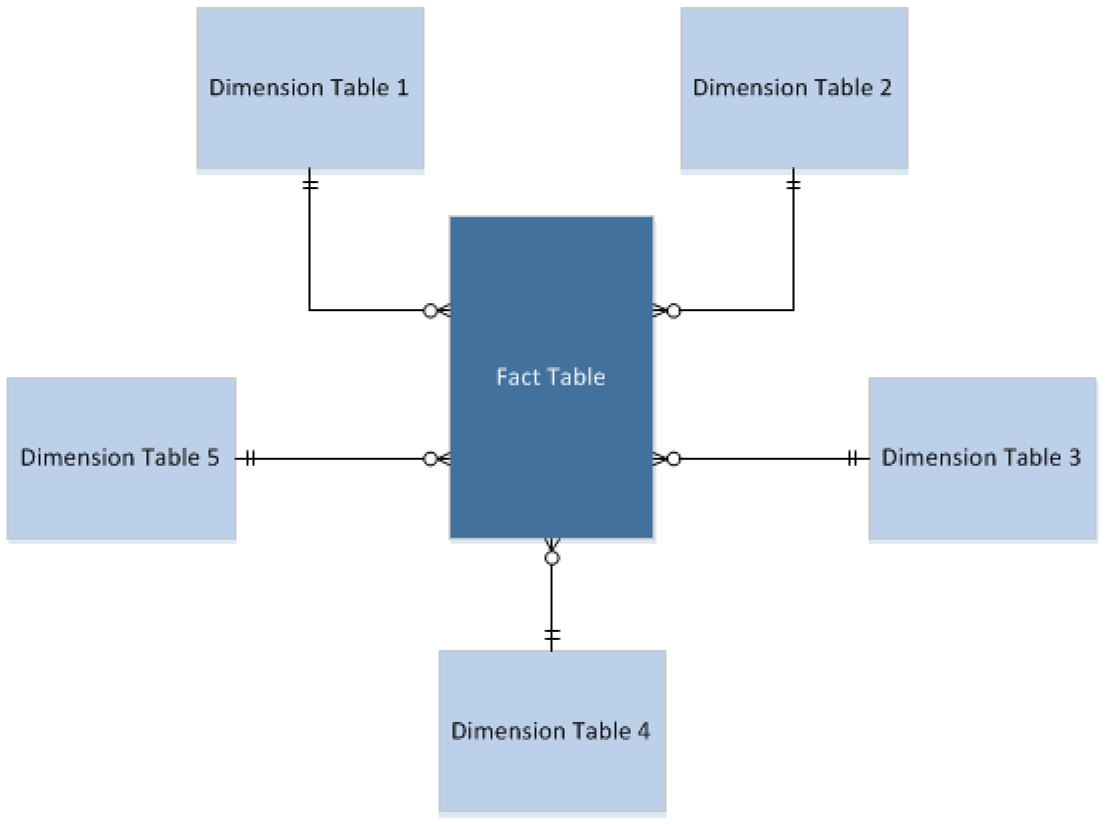

Figure 1. Star Schema Design

LASR supports simple star schema data structures that can be used either to create a LASR Star Schema View or to materialize a full LASR physical table in an analytical base table structure.

The SAS LASR Analytic Server makes the following assumptions in relation to schema structures:

- LASR star schemas only support one fact table.
- Dimension tables do not have repeat values for the dimension keys.
- The relation between the fact table and the dimension table is expressed by one pair of keys—that is, one variable in the fact table defines the relation to one variable in the dimension table. Note that you can use LASR table-computed columns as the keys.
- The keys in the fact table do not have to have the same name as the keys in the dimension tables and vice-versa.
- Partitioning and order-by information is preserved when the schema is created. However, only partitioning of the fact table is taken into account and the resulting table or view is partitioned by the same variables as the fact table.
- When generating a view, the relations are resolved upfront when the schema is created to make subsequent passes through the data more efficient.

WHICH STRUCTURE WHEN

Each data structure has advantages and disadvantages.

	Analytical Base Table	**Star Schema View**
Advantages	• Best performance • Easy to append, update, and delete rows	• Smaller memory footprint • Reuse multiple dimension tables across schema views
Disadvantages	• Larger memory footprint • Hierarchy data item replication across analytical base tables	• Performance overhead due to execution of joins at action time • Schema must be reinitialized to reflect appended, updated, and/or deleted rows (see the following hot swap example)

Table 1. Advantages and Disadvantages of Different Table Structures

It is recommended that you do some performance testing to determine what structure works best for your specific use case. Even if the performance overhead of a star schema view is above acceptable levels, you might choose to use the star schema structure to minimize data transfer from source data systems. If you materialize your analytical base table at the source, you have to transfer the fully materialized table across your network. In contrast, the component fact and dimension tables are smaller in volume and, as such, take less time to transfer across a network. Using the star schema functionality in LASR, you can opt to create a physical table instead of a view. This might be faster than transferring the fully joined table from the source.

METHODS FOR LOADING AND INTERACTING WITH DATA TABLES

To enable explorations and reporting in SAS Visual Analytics, you first need to load data into the SAS LASR Analytic Server. Typically, your first load of data into LASR is a large one. Subsequent loads are often smaller, performing appends, updates, and/or deletes to keep the data current.

SAS® VISUAL DATA BUILDER

SAS® Visual Data Builder, a web-based component of SAS Visual Analytics, enables you to perform basic data operations:

- Self-service data imports
- Data queries to perform joins
- Structure the data for analytics and reporting before loading into any available SAS library (including the SAS LASR Analytic Server or the HDFS libraries)

The interface is primarily a drag-and-drop utility, but it is supplanted by wizards for common LASR data tasks:

- Create a LASR star schema based on tables already loaded into LASR memory
- Append one LASR table to another
- Save a LASR table to HDFS

SAS Visual Data Builder could be a useful starting point in piecing together a wider data management process. The remainder of this section focuses on the underlying mechanisms for loading data into LASR.

There are three methods for loading data into LASR:

- LASR procedure
- DATA step
- HPDS2 procedure

The following sections discuss the capabilities and limitations of each of these methods. Your data load and update process is likely to use a combination of these methods.

LASR PROCEDURE

In a distributed environment, when you need to load data from HDFS to LASR, use PROC LASR. When you submit PROC LASR, the LASR root node directs the LASR worker nodes to read data from HDFS. PROC LASR performs memory mapping from HDFS blocks directly to LASR blocks, so the data do not actually pass through the LASR head node (see Figure 2). The memory mapping is a local disk to memory action. Because it performs memory mapping, PROC LASR does not use physical memory until the table is accessed.

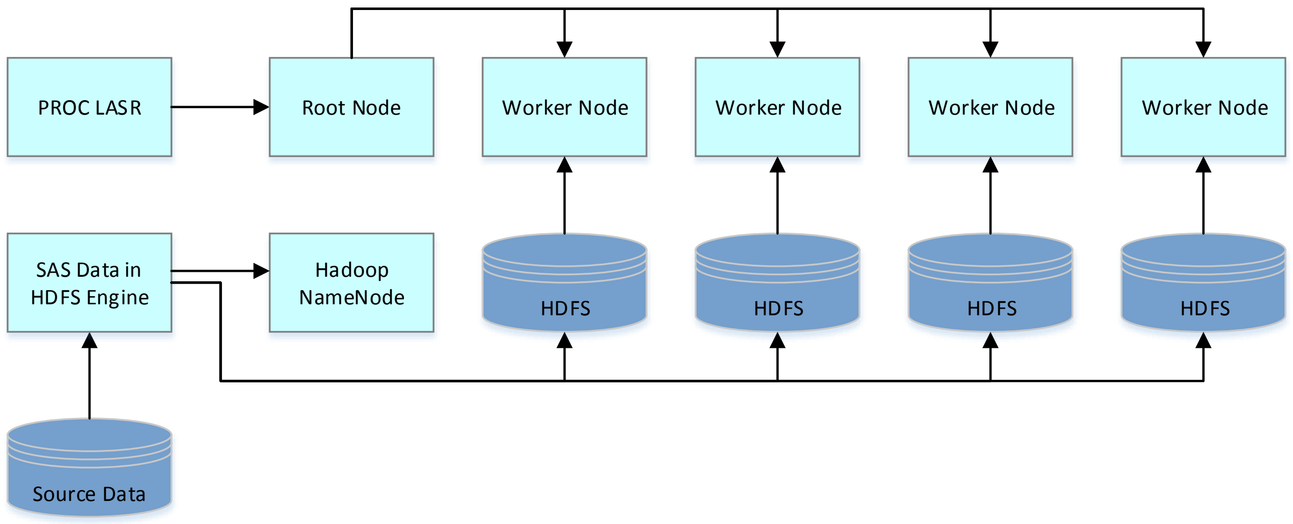

Figure 2. Communication Between LASR Worker Nodes and Hadoop Nodes

This is basic PROC LASR code to load a table from HDFS to LASR:

```
LIBNAME hps SASHDAT PATH="/SASGF"
    SERVER = "sasserver01.sas.com"
    INSTALL = "/opt/sas/TKGrid”;

PROC LASR add
            data = hps.prdsale
            port=10010
            noclass;
performance host="sasserver01.sas.com";
run;
```

The LIBNAME statement defines the Hadoop library. Key points about the PROC LASR statement are:

- ADD indicates you are loading a table to LASR.
- DATA specifies the data table to load.
- PORT specifies the port number that corresponds to the LASR server.
- NOCLASS indicates that character fields should not be implicitly treated as classification variables. This is used to improve performance. Using the CLASS option increases the size of the in-memory table and load time. Furthermore, you cannot append data to a table that was loaded using a CLASS option.

Finally, use the PERFORMANCE statement and HOST option because you are working in a distributed environment and need to specify the grid host to use for the server. Note that if the table you loaded is not already registered in the LASR library, you can add PROC METALIB code to register the table, making it visible in SAS Visual Analytics:

```
LIBNAME valibla SASIOLA
    TAG=HPS
    PORT=10010
    HOST="sasserver01.sas.com"
    SIGNER="http://sasserver01.sas.com:7980/SASLASRAuthorization";
```

```
PROC METALIB;
   omr(library="/Products/SAS Visual Analytics Administrator/Visual
    Analytics LASR");
   folder="/Shared Data/SASGF/03 LASR Data";
   select=("PRDSALE");
run;
```

PROC LASR does have some constraints. The procedure cannot create a partitioned table. However, if the table is already partitioned in HDFS, the partitioning will be maintained in LASR. Also, you cannot filter the data from HDFS before you load it into LASR using this procedure. If you need to filter the data from HDFS, use the HPDS2 procedure described here. Finally, if the table already exists in memory, you must unload it before you can reload it using PROC LASR. This makes the table unavailable to end users until the table reload is complete.

DATA STEP

You can use a straightforward DATA step to load data from a non-Hadoop source into LASR:

```
LIBNAME valibla SASIOLA
   TAG=HPS
   PORT=10010
   HOST="sasserver01.sas.com"
   SIGNER="http://sasserver01.sas.com:7980/SASLASRAuthorization";

DATA VALIBLA.PRDSALE;
   set sashelp.prdsale;
run;
```

The DATA step is the primary method for loading data into LASR in non-distributed SAS Visual Analytics environments because you cannot use PROC LASR.

There are some things to keep in mind when using this method. When you load data using Base SAS®, all of the data pass through the main LASR root node, so it is important that you ensure that the SAS workspace server resources are adequate to handle the data load. Because the data pass through the LASR root node, this method is not necessarily recommended when you are updating and deleting big data. In those cases, the IMSTAT procedure is recommended. This paper discusses PROC IMSTAT in a later section. Finally, a load using Base SAS is not as fast as a load from Hadoop.

HPDS2 PROCEDURE

This method uses multi-threaded distributed processing from HDFS to LASR through the LASR head node. It is useful when there is a large amount of intensive, row-independent computations to be performed. It is also useful if you need to filter the data in HDFS and load a subset into LASR:

```
OPTION SET=GRIDINSTALLLOC="/opt/sas/TKGrid";
OPTION SET=GRIDHOST="sasserver01";

LIBNAME SASGF BASE "/opt/sas/data";
LIBNAME hps sashdat path="/hps";
LIBNAME VALIBLA sasiola port=10010 tag="hps";

/*Load complete table to HDFS*/
DATA HPS.PRDSALE(replace=yes);
   set sashelp.prdsale;
run;

/*Load subsetted table to LASR*/
PROC HPDS2 in=HPS.prdsale out=VALIBLA.prdsale_US;
```

```
      DATA DS2GTF.OUT;
         method run();
            set DS2GTF.in;
            if country = 'U.S.A.';
         end;
      enddata;
run;

/*Register LASR table*/
PROC METALIB;
omr(library=
   /Products/SAS Visual Analytics Administrator/Visual Analytics LASR");
folder="/Shared Data/SASGF/03 LASR Data";
select=("PRDSALE_US");
run;
```

The three LIBNAME statements from the preceding code block define the source library, the Hadoop library, and the LASR library. After distributing the data across Hadoop, PROC HPDS2 is used to load LASR. Here are key points about the PROC HPDS2 syntax:

- IN= specifies the Hadoop source table.
- OUT= specifies the name for the new table being loaded into LASR.
- DS2GTF.OUT is constant syntax that refers to the target LASR table name. This syntax is not customizable.
- DS2GTF.IN is constant syntax that refers to the Hadoop table. This syntax is not customizable.
- The IF statement is used to filter the data from SASHDAT to LASR. Any DS2 logic can be used here and is performed through threaded distributed processing. The WHERE statement is not valid for filtering.

As with the other methods for loading data, there are some things to keep in mind about PROC HPDS2. This method does not support partitioning or appends. It also does not use memory mapping, and because processing is coordinated through the head node, it does not perform as well as PROC LASR.

TIPS FOR UPDATING DATA

Loading a data table into LASR is rarely a one-time event. Typically, you need to add new rows, perhaps delete old rows, and/or update existing rows in cases where historical data has changed. There are essentially two ways to refresh data in LASR: a complete overwrite with new data or an append/update/delete process. A complete overwrite is a simple and easy solution, while an append/update/delete process is more complicated.

One of the main considerations when choosing a data refresh strategy is whether you have a batch window. If you have a sufficient batch window, doing a full overwrite is the simplest way to refresh your data. If you do not have an adequate batch window, you should use the functionality available in the IMSTAT procedure.

IMSTAT PROCEDURE

To interact with data once in LASR, you use the IMSTAT procedure. PROC IMSTAT was written specifically to interact with the SAS LASR Analytic Server from a traditional SAS code session.

The IMSTAT procedure is divided into two sets of statements. The first is a set of statements relating to data and server management of LASR data, and the second is another set of statements aligned toward executing analytics. With a standard SAS Visual Analytics license, you get full access to all the data and server management statements. The analytics statements are available with SAS® In-Memory Statistics.

The IMSTAT procedure can be executed in both distributed and non-distributed SAS Visual Analytics environments. It is an interactive procedure and therefore executes collections of statements within run blocks before terminating. It terminates when it encounters the start of a DATA step, another PROC invocation, or the QUIT statement.

One key feature of PROC IMSTAT is that the existing, in-memory table being updated remains available to end users with no down time. Specifically, if an end user is using the existing table in an exploration or a report when PROC IMSTAT executes, the user continues to see data from the original table. When PROC IMSTAT terminates, the changes are committed to the existing table. Those changes are visible when the end user refreshes the report on receipt of the notification in the interface that the underlying data has changed.

The following examples show how you can use PROC IMSTAT to refresh data on a rolling basis using dates and a flag field. In this example, the DATE and FLAG fields were created prior to distributing the data across HDFS and loading into LASR:

```
LIBNAME valibla SASIOLA
   TAG=HPS
   PORT=10010
   HOST="sasserver01.sas.com"
   SIGNER="http://sasserver01.sas.com:7980/SASLASRAuthorization";

LIBNAME hps SASHDAT PATH="/SASGF"
   SERVER = "sasserver01.sas.com"
   INSTALL = "/opt/sas/TKGrid”;

/*Update existing flag field based on date to mark old rows for deletion*/
proc imstat;
   table valibla.prdsale;
   where Date="01JAN1993"d;
   update flag = 1;
run;
   table valibla.prdsale;
   where Date NE "01JAN1993"d;
   update flag = 0;
run;
quit;
```

In the preceding code, the WHERE clause applies to all of the statements below it until a QUIT statement is encountered. Note that it crosses RUN boundaries.

```
/*Load into memory new data to be appended*/
proc lasr add
   data = hps.prdsale_append
   port=10010
   noclass;
performance host="sasserver01.sas.com";
run;

/*Append new data to existing and drop small append table from memory*/
proc imstat;
  table valibla.prdsale;
  set prdsale_append / drop;
run;
quit;

/*Delete old rows based on the flag field*/
```

```
proc imstat
  data=valibla.prdsale;
  where flag=1;
  deleterows / purge;
run;
quit;
```

In the preceding code block, the DELETEROWS statement marks for deletion all rows from the in-memory table that meet the WHERE clause. The PURGE option actually deletes the rows. Without using the PURGE option, the rows marked for deletion would remain in the data table, but they would not be used in calculations.

```
/*Save newly updated table in HDFS*/
proc imstat data=valibla.prdsale;
   save
   path="/SASGF"
   replace;
run;
quit;
```

GENERAL TIPS

COMPRESSION

LASR supports compression for in-memory tables and tables stored in HDFS. Both character and numeric variables are compressed, and all compression is performed by the LASR server. Note that for data tables containing rows with many long character variables consisting mostly of blanks, the compression ratio can be quite high.

Like a star schema view, compression adds performance overhead when interacting with the table. As LASR accesses the table to run an action, it uncompresses the data to execute the computations and then discards the uncompressed block. This operation is performed at an individual data block level, so there is negligible inflation of data on the wider LASR cluster. The performance overhead comes from the block level uncompression, which must occur at every pass of the data.

Compression is probably most appropriate in a SAS Visual Analytics scenario where you have to maintain in LASR memory several small to low-end medium-sized tables and want to decrease their memory footprint. It can also be very useful for maximizing HDFS usage to store a lot more history in HDFS than is active in memory at any one time. Data in HDFS can be stored in compressed format and then, for maximum performance, you can uncompress through a coded process once loaded into memory.

These code blocks show various examples of using compression:

```
LIBNAME valibla SASIOLA
   TAG=HPS
   PORT=10010
   HOST="sasserver01.sas.com"
   SIGNER="http://sasserver01.sas.com:7980/SASLASRAuthorization";

/*Save from LASR to HDFS and compress data in HDFS*/
proc imstat data=valibla.data_table;
save path="/hps/data_table"
   copies=1
   blocksize=32M
   compress
   replace
```

```
      fullpath;
   run;
   quit;
```

In the preceding block of code, the COPIES option specifies the number of replications to make beyond the original blocks. Use the BLOCKSIZE option to specify the block size to use to distribute the data across HDFS. BLOCKSIZE is ignored when the table in memory is partitioned. In those cases, the partition size determines the block size. Finally, the FULLPATH option indicates that the path includes the table name (without the SASHDAT extension).

```
   /*Compress in-memory table in place in LASR*/
   proc imstat;
      table valibla.data_table;
      compress;
   run;
   quit;

   /*Uncompress in-memory table in place in LASR*/
   proc imstat;
      table valibla.data_table;
      uncompress;
   run;
   quit;
```

COMPUTED COLUMNS

Using computed columns at data load time offers a couple of key advantages. First, computed columns are virtual and exist as code that executes when the table is accessed. In effect, this does not increase your data footprint in HDFS or in the data warehouse. Second, using computed columns in this way may provide you more flexibility in changing the calculation(s) if and when business rules change.

Use PROC IMSTAT to add a permanent computed column to an in-memory table:

```
   proc imstat;
      table lasr.cars;
      compute computed1 "computed1=mpg_city/mpg_highway";
      run;
   quit;
```

It is worth noting that adding calculated columns at load time presents all users with the same calculation and report designers do not need to calculate the measures for every report.

FORMATS AND LENGTHS

To minimize your data footprint, you may want to change the format of character fields before loading data into LASR. Several large character fields in a data table can increase the data footprint and consume more memory. To minimize that effect, reduce the length of character fields where possible and appropriate. For example, if the maximum length of a field is actually 30 characters, but the field has a length of 100, that increases the size of your data table unnecessarily.

Format encoding of columns is a way to minimize the field length, but be aware of some of its shortcomings. It could make your data refresh strategy more complex, particularly if you need to maintain an append process, because you need to append correctly encoded variables. In addition, custom formats have a performance overhead that can be quite high on high cardinality fields. Finally, picture statement formats are not supported.

LABELS

By default in explorations and reports, SAS Visual Analytics displays the labels that are attributed to each column in a data table. If no labels have been applied, SAS Visual Analytics displays the column name, which is often not user-friendly. PROC LASR does not support labels, so if you use PROC LASR to load data, apply labels before you invoke the LASR procedure.

PARTITIONING

By default, data is distributed across nodes in a round-robin fashion to more evenly balance the workload. Partitioning the data, on the other hand, distributes data according to your partition key. For example, assume data analysis frequently subsets the data table by account. If you partition the data table by account, you speed up processing in an exploration or report because the server can direct all of the work to the one machine containing the relevant account. However, if the data table is partitioned by something that is not frequently used for analysis, performance can be negatively impacted.

Partitioning can only be done using a DATA step. If you have a distributed environment, you can use a DATA step with the PARTITION= option to distribute the data across Hadoop before loading the data into LASR. Even though PROC LASR cannot *create* a partitioned table, if the table is already partitioned in HDFS using a DATA step, the partitioning will be maintained in LASR. In a non-distributed environment, you can partition a data table, but all partitions are kept on a single machine because there is no distributed computing environment.

READAHEAD

When using PROC LASR, there is a tradeoff between load time and first table access performance. The default PROC LASR statement makes the load very fast, but the first access of the table takes more time than subsequent accesses. Using the READAHEAD option tells the server to be more aggressive in reading memory pages during memory mapping. This makes the first access of the table faster, but it increases the load time.

```
LIBNAME hps SASHDAT PATH="/SASGF"
   SERVER = "sasserver01.sas.com"
   INSTALL = "/opt/sas/TKGrid”;

PROC LASR readahead
          add
          data = hps.prdsale
          port=10010
          noclass;
performance host="sasserver01.sas.com";
run;
```

SQL VIEWS

Using PROC SQL views to specify column length, labels, and formats before the data is loaded into HDFS or LASR is especially useful when you are loading data from a database source. An SQL view does not physically create an intermediary table in the SAS compute server before it goes into HDFS or LASR, but it allows you to set the appropriate attributes up front:

```
LIBNAME td TERADATA SERVER=tdprod  SCHEMA=td_schema;

/*Set length, formats, and labels in a view*/
proc sql;
   create view work.database_data as
      select
         dbt.person_name length=30 format=$30. label="Person Name"
        ,dbt.region length=15 format=$15. label="Home Region"
        ,dbt.city length=15 format=$15. label="Home City"
        ,dbt.balance length=8 format=dollar14.2 label="Account Balance"
   from
      td.database_table dbt;
quit;

LIBNAME hps SASHDAT
   PATH="/hps"
   SERVER=" sasserver01.sas.com"
   INSTALL="/opt/sas94/TKGrid";

/*Distribute data from view across HDFS*/
DATA hps.hadoop_data (blocksize=16M replace=yes);
   set work.database_data;
run;
```

CONCLUSION

This paper reviewed several ways to load data into LASR to make it available for exploration and reporting in SAS Visual Analytics. Typically, that is only the beginning, and you need to consider various options for keeping the data current. Clearly there are lots of possibilities and many things to consider when designing a data load and refresh process. This paper is not meant to outline all possible scenarios and decision points, but rather to introduce some of the key points to think through as you move beyond the basics with SAS Visual Analytics.

REFERENCES

Herrmann, Gregor, and Chitale, Anand. 2015. “Access, Modify, Enhance: Self-Service Data Management in SAS® Visual Analytics.” *Proceedings of the SAS Global Forum 2015 Conference.* Cary, NC. SAS Institute Inc.

Mehler, Gary, and Bennett, Donna. 2014. “Big Data Everywhere! Easily Loading and Managing Your Data in the SAS® LASR™ Analytic Server.” *Proceedings of the SAS Global Forum 2014 Conference.* Cary, NC. SAS Institute Inc. Available at http://support.sas.com/resources/papers/proceedings14/SAS347-2014.pdf.

Otto, Greg, and Weber, Tom. 2013. “Adaptive In-Memory Analytics with Teradata: Extending Your Existing Data Warehouse to Enable SAS® High-Performance Analytics.” *Proceedings of the SAS Global Forum 2013 Conference.* Cary, NC. Teradata Corporation and SAS Institute Inc. Available at http://support.sas.com/resources/papers/proceedings13/076-2013.pdf.

SAS® LASR™ Analytic Server 2.5: Reference Guide
(http://support.sas.com/documentation/onlinedoc/va)

Base SAS® 9.4 Procedures Guide: High Performance Procedures, Second Edition
(http://support.sas.com/documentation/cdl/en/prochp/66704/PDF/default/prochp.pdf)

Adaptive In-Memory Analytics with Teradata: Extending Your Existing Data Warehouse to Enable SAS® High-Performance Analytics
(http://support.sas.com/resources/papers/proceedings13/076-2013.pdf)

ACKNOWLEDGMENTS

We would like to acknowledge the great work of the SAS® LASR™ Analytic Server team and the Global Enablement & Learning team at SAS.

CONTACT INFORMATION

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

Kerri L. Rivers
100 SAS Campus Drive
Cary, NC 27513
SAS Institute Inc.
kerril.rivers@sas.com
http://www.sas.com

Christopher Redpath
100 SAS Campus Drive
Cary, NC 27513
SAS Institute Inc.
christopher.redpath@sas.com
http://www.sas.com