Paper SAS488-2017

SAS® and Hadoop: The Sixth Annual State of the Union

Paul Kent, SAS Institute Inc.

ABSTRACT

The fourth maintenance release for SAS® 9.4 and the new SAS® Viya™ platform bring even more progress with respect to the interoperability between SAS® and Hadoop (the industry standard for big data). This talk brings you up-to-date with where we are: more distributions, more data types, more options. And then there is the cloud. Come and learn about the exciting new developments for blending your SAS processing with your shared Hadoop cluster.

INTRODUCTION

Hadoop and the era of using several computers harnessed together as a team as a team to solve your problems promised to change analytics back in 2010, if not 2006. (Hadoop has just celebrated its tenth birthday.) Similar waves of change have had a significant change on SAS software, and this time around is no different.

This paper introduces some basic Hadoop concepts and describes the ways that a SAS programmer can interact with data stored in a Hadoop Distributed File System (HDFS). In fact, 2016 is the first year I can recall where I didn't see a headline in the run-up to SAS Global Forum asking, "Are YOU Hadoop-ing Yet?" In 2017, the conversation has matured to "Hadoop, the sustaining eco-system" and has moved beyond "Hadoop – map reduce all your data."

After a short introduction to Hadoop, I'll detail several ways that a SAS programmer interfaces with Hadoop. At the very most basic level, it is just another source of raw data. Hadoop uses hive as a flexible approach toward defining a tabular shape for data. Hive is the next tier of integration with SAS. Finally, there are SAS approaches for full-on adoption of Hadoop divide-and-conquer principles of operation.

In the talk, I liken these stages to those of human courtship. At first, the attraction might be driven by curiosity. Introductions are made, and, before long, the parties are dating, becoming engaged, and possibly marrying.

As Hadoop has matured and more enterprise data has been stored in HDFS, security has become a very important topic.

HADOOP INTRODUCED

Hadoop is a very successful (and visible) open-source project, like Linux itself and the Apache Web Server project that helped to establish the "Apache way" at the Apache Software Foundation – whose mission is "to provide software for the public good". (See https://hadoop.apache.org.)

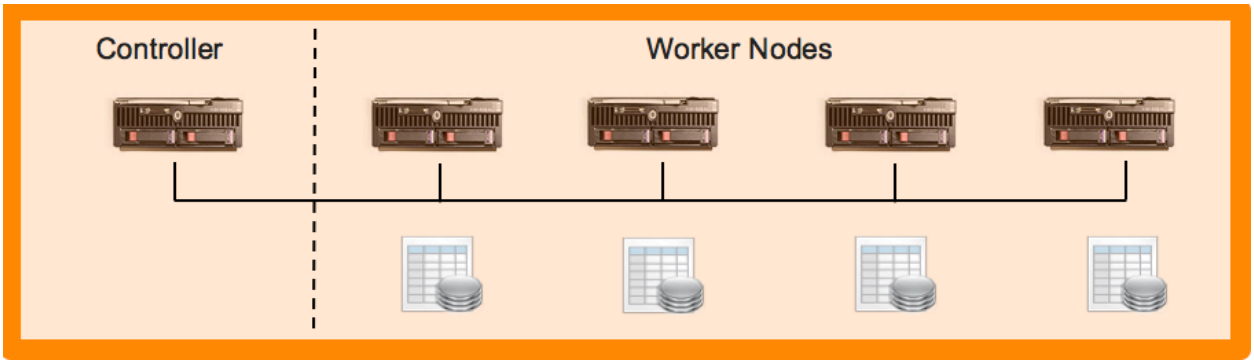
The Apache™ Hadoop® project develops open-source software for reliable, scalable, distributed computing.

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, thus delivering a highly available service on top of a cluster of computers, each of which might be prone to failures.

Apache Hadoop YARN is a relative newcomer to the project. YARN enables enterprises to build large shared clusters that they can share widely across their constituent groups. YARN makes it possible to administer access to a shared resource that is often oversubscribed. YARN is also important because it

represents the transition to Hadoop as an ecosystem for data and analytics. The Hadoop project is very open-minded about new approaches being built on the underlying infrastructure. Apache Spark and SAS Viya are two examples of this.

The essential idea behind Hadoop is to use a number of computers to solve a problem. You use them in such a way that you can divide and conquer the workload by sharing it across all the workers in the cluster. It is typical to have some form of management node that is the coordinator and that controls the work distributed to the workers.



There are two beautiful ideas in the original Hadoop:

1. I will spread your data over many computers to keep it safe.
2. I will facilitate a way to send the work to the data, instead of asking that you gather all the data together for processing.

IDEA #1: HADOOP NEVER FORGETS

A file in HDFS is represented as blocks, and each block is sent to a different computer:

Head Node	Data 1	Data 2	Data 3	Data 4…
MYFILE.TXT				
..block1 ->	block1copy1			
..block2 ->		block2copy1		
..block3 ->			block3copy1	

This in and of itself doesn't increase the chances that your file is available. Now you are at the mercy of five computers instead of just one (and usually a more reliable one). Hadoop compensates for this by storing redundant copies of the same blocks on different nodes in the cluster:

Head Node	Data 1	Data 2	Data 3	Data 4…
MYFILE.TXT				
..block1 ->	block1copy1		block1copy2	
..block2 ->		block2copy1		block2copy2
..block3 ->	block3copy2		block3copy1	

Even in this small cluster, you are now protected against a single data node failure. Suppose that the Data 3 node develops a hardware fault. Without redundancy, you could not locate block3. But with even a single level of redundancy, you can find the alternate copy of block3 on the Data 1 node.

Head Node	Data 1	Data 2	Data 3	Data 4…
MYFILE.TXT				
..block1 ->	block1copy1		bloc…2	
..block2 ->		block2copy1		block2copy2
..block3 ->	block3copy2		blo…1	

A Hadoop cluster is typically operated with copies=2 or a redundancy factor of 3. This makes the data in files very resilient. They are available even in the face of several failures. Hadoop is proactive about keeping the availability high. Ten minutes after a block is missing (so that there are only two instead of the required three copies), Hadoop will commission yet another copy of the block elsewhere in the cluster to maintain the copies= number of safe copies.

IDEA #2: SEND THE WORK TO THE DATA

Imagine that you are at a wedding reception. To keep the numbers workable, the room contains 10 round tables, and there are 10 people at each table. The controller (Master of Ceremonies) is warming up the crowd and wants to find the youngest person in the room. (In other words, calculate the MIN statistic over the data set of 100 records.)

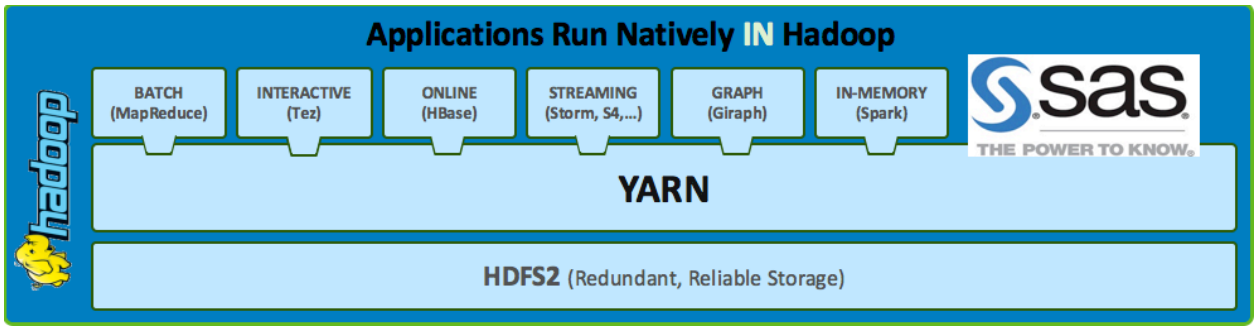
The default approach would have the MC visit each table in turn, ask each person their age, and keep a running tally of the youngest person thus far. That is, the MC creates a sequential scan of the data set.

The Hadoop approach engages the crowd. The MC asks the youngest person at each table to come to the center of the room, where a quick selection can be made as to the youngest of the youngest. This is distributed, or parallel computing. It does not rely on one computer making a sequential scan of the entire data set. In fact, the problem (or solution rather) scales nicely as the wedding grows. Either add 5 more tables (grow the cluster), or add 5 people to each table (grow the capacity of each node)

Idea #2 isn't very useful without Idea #1 first having spread the data over many computers.

IDEA #3: MAKE IT EASY TO SHARE THE RESOURCES OF THE CLUSTER

The first two ideas were sufficient to get Hadoop adopted by the early web-scale computing centers like Google, Yahoo, Facebook, and LinkedIn. But we needed a good way to share the resources of the cluster. This resulted in traditional enterprises being interested too. The cost dynamics of Hadoop were one and sometimes two orders of magnitude better than the previous approach.



YARN is the component of Hadoop that interposes itself between the distributed, reliable storage subsystem (HDFS) and the applications that want to process that data. Applications make resource requests to the YARN controller, who allocates them, possibly after waiting for the requested resource to become available again after a different application is finished using it.

SAS 9.4 AND HADOOP – GETTING CONNECTED

The SAS FILEREF procedure has traditionally been used to refer to external data. Initially, it was used to reference one of the DD cards in the JCL of the job that you submitted to your mainframe. But more recently, a fileref has been seen referring to a path on the operating system, to a directory of an FTP server, and to a page (URL) of a website. The SAS fileref has been extended to enable you to specify a file in HDFS.

```
FILENAME paul HADOOP
   '/users/kent/mybigfile.txt'
   CONFIG='/etc/hadoop.cfg' USER='kent' PASS='sekrit';

DATA MYFILE;
   INFILE paul;
   INPUT name $ age sex $ height weight;
   RUN;
```

Many details about the Hadoop cluster that you want to connect to are not repeated in every program. You simply refer to a configuration file for the cluster. Your system admin folks will have set this file up for you, so ask around until you find a colleague with one that works and copy that approach ☺

At this point, your SAS program can read or write data to a Hadoop cluster. You are not yet enjoying all the possible benefits that Hadoop can bring. This is especially because you are still copying the data between the SAS server and the Hadoop cluster. This might be just the thing for an ETL job where you are adding the next hours' worth of website visits to the collection. But you certainly don't want to copy the past 10 years of website visits out of the cluster to do some analytics!

The HADOOP fileref allows your SAS program to connect with data in Hadoop. PROC HADOOP is the counterpart that enables you to interface with the control of the Hadoop cluster.

```
PROC HADOOP CFG=CFG … [VERBOSE];
   HDFS <hdfs commands>;
   MAPREDUCE <mapreduce options>;
   PIG <pig options>;
   RUN;

 /* file and directory manipulation */
   hdfs mkdir='/tmp/rick/mydir';
   hdfs copyfromlocal='myfile.txt' out='/tmp/rick/mydir/myfile.txt';
   hdfs copytolocal='/tmp/rick/mydir/myfile.txt' out='myfile2.txt';
   hdfs delete='/tmp/rick/mydir/myfile.txt';

 /* new with SAS 94m3 */
   hdfs ls='/tmp/rick';
   hdfs ls='/tmp/rick' out=lsfile;
   hdfs cat='/tmp/rick/testfile.txt';
   hdfs cat='/tmp/rick/*.txt out=catfile;
```

SAS VIYA AND HADOOP – GETTING CONNECTED

HDFS can store plain-text files, and these are often further defined to Hive with a schema. More likely, data is saved into Hadoop in CSV (comma-separated) files or as JSON payloads. These files are often stored with a date-time stamp in their filenames, or in their encompassing folder structure. This naturally establishes a data set over time. This week, Jan2017, Last Quarter, and Previous 3 years are all simple wildcard expressions for a series of files containing the desired time range.

SAS Viya can load CSV files from HDFS in parallel, and can load other files using the SAS® Embedded Process. This capability also facilitates parallel loading. Parallel is the operative word. Different workers can process a different slice of the total file, thus loading the data much more quickly than having to wait for a single worker (or the controller) to process the entire file.

SAS AND HADOOP – DATING

Relational databases like Oracle, DB2, Teradata, SQL Server, and to a large extent SAS itself have spoiled us to expect more from our data sources than the typical CSV (comma-separated values) file. We have grown accustomed to a richer description of the data elements in the collection: the variable names, the data type (is it a date?), and even formatting niceties like a Label and DisplayFormat.

We process these tables with languages like the SAS DATA step, Standards-Based SQL and open-source concepts like the data frame. And we expect the table metadata to be durable and the information it represents to be transferred into the language that we are using.

Apache Hive is the Hadoop subsystem responsible for managing and presenting tabular style metadata about files stored in HDFS. SAS/ACCESS® Interface to Hadoop interfaces to Hive so that these Hadoop tables appear with high fidelity as SAS tables in your SAS session. Hive operates using well known SQL principles. There is a pair of vendor implementations that are specific to Hadoop. These are SQL and the corresponding SAS/ACCESS products to interoperate with them: SAS/ACCESS® to Impala and SAS/ACCESS to Apache HAWQ.

You use the SAS LIBNAME statement to reference the collection of tables in Hive.

```
LIBNAME olly HADOOP
  SERVER=olly.mycompany.com
  USER='kent' PASS='sekrit';

PROC DATASETS LIB=OLLY;
  RUN;
```

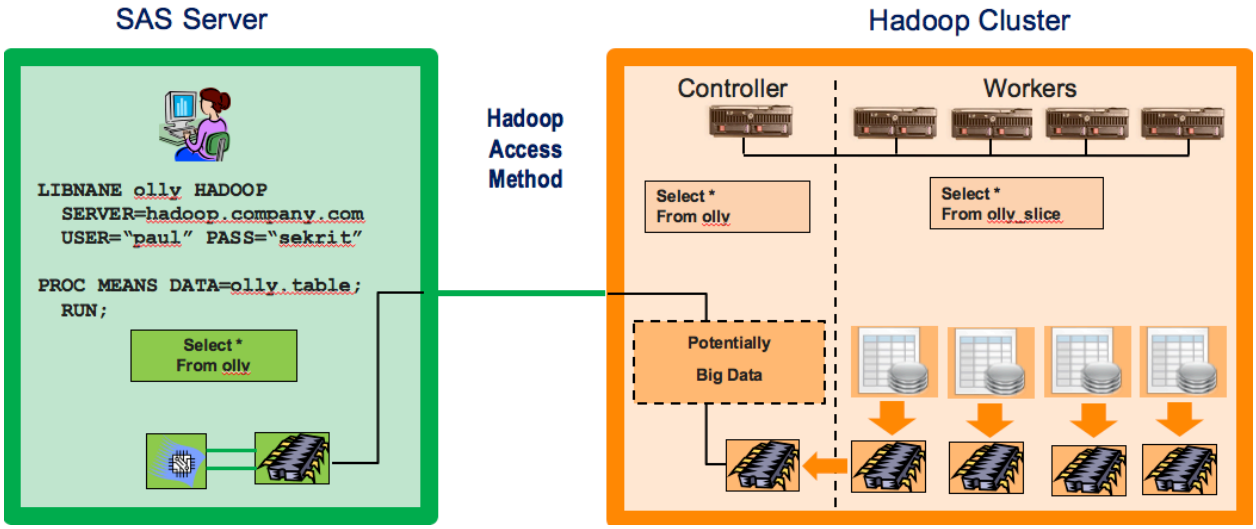
One of the reasons that I refer to this as dating is that more information is exchanged across the boundary between SAS and Hadoop, where lies all that rich metadata regarding the contents of the files. This is also the stage where we can see some of the work being transferred into the cluster rather than the rows of data being extracted over to the SAS server.

SQL PASS-THROUGH – ASK THE DATA SOURCE TO DO THE WORK

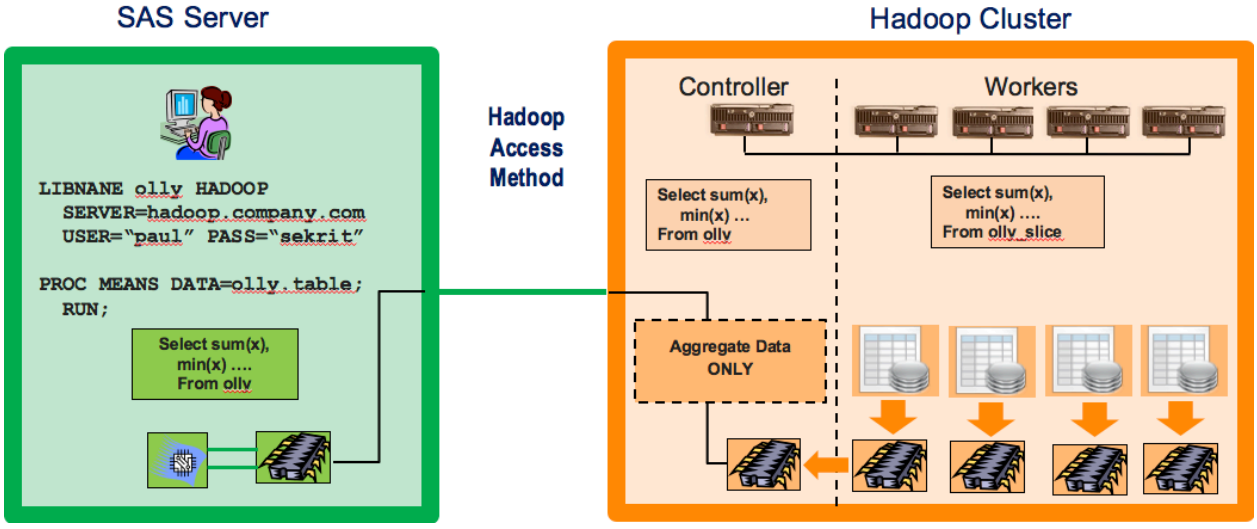
The MEANS procedure calculates simple descriptive statistics. Standard SQL has the SUM(), COUNT(), and other functions to calculate the same statistics. What if we could teach PROC MEANS to ask the data source to do the heavy lifting, rather than just access the raw records and perform the calculations itself? We have come to call the former SQL pass-through. It is the same concept as the MC at the wedding asking the groups at the tables to determine the youngest at each table. You're transferring a high-level specification of the work rather than requesting all the raw materials be delivered to your doorstep!

SAS/ACCESS to Hive implements this concept. As these two diagrams demonstrate, it can have a great impact on the number of rows of data that must be transferred between the Hadoop server and the SAS session.

In the default scenario, PROC MEANS must request all the rows from the table for which you require the descriptive statistics. It doesn't matter that Hive can do a nice job of partitioning the work and make cluster-local request for the data slice that lives on that cluster. Eventually, all rows in the table have to be sent over the connection back to SAS.



In the SQL pass-through scenario, things are a lot better. The request to PROC MEANS is understood and translated to the equivalent SQL asking for those statistics. The Hive optimizer can distribute the work optimally across the cluster, and a small set or records is returned to the SAS server for the result. We can take advantage of Hadoop Idea #2 and move the work to the data!



This SQL pushdown can get quite sophisticated. Here is an example from PROC RANK against an Impala data source.

```
options sqlgeneration=dbms;
```

```
proc rank data=x.cake out=order descending ties=low;
   var present taste;
   ranks PresentRank TasteRank;
run;
```

You will see the generated SQL in your SAS log:

```
IMPALA_25: Prepared: on connection 2
SELECT `table0`.`name`, `table0`.`present`, `table0`.`taste`,
`table1`.`rankalias0` AS `PresentRank`, `table2`.`rankalias1` AS
`TasteRank` FROM ( SELECT `name` AS `name`, `present` AS `present`, `taste`
AS `taste` FROM `cake` ) AS `table0` LEFT JOIN ( WITH subquery0 AS (SELECT
`present`, `tempcol0` AS `rankalias0` FROM ( SELECT `present`, MIN(
`tempcol1` ) OVER ( PARTITION BY `present` ) AS `tempcol0` FROM( SELECT
`present`, CAST( ROW_NUMBER() OVER ( ORDER BY `present` DESC ) AS DOUBLE )
AS `tempcol1` FROM ( SELECT `name` AS `name`, `present` AS `present`,
`taste` AS `taste` FROM `cake` ) AS `subquery2` WHERE ( ( `present` IS NOT
NULL ) ) ) AS `subquery1` ) AS `subquery0` ) SELECT DISTINCT `present`,
`rankalias0` FROM subquery0 ) AS `table1` ON ( (`table0`.`present` =
`table1`.`present` ) ) LEFT JOIN ( WITH subquery3 AS (SELECT `taste`,
`tempcol2` AS `rankalias1` FROM ( SELECT `taste`, MIN( `tempcol3` ) OVER (
PARTITION BY `taste` ) AS `tempcol2` FROM( SELECT `taste`, CAST(
ROW_NUMBER() OVER ( ORDER BY `taste` DESC ) AS DOUBLE ) AS `tempcol3` FROM
( SELECT `name` AS `name`, `present` AS `present`, `taste` AS `taste` FROM
`cake` ) AS `subquery5` WHERE ( ( `taste` IS NOT NULL ) ) ) AS `subquery4`
) AS `subquery3` ) SELECT DISTINCT `taste`, `rankalias1` FROM subquery3 )
AS `table2` ON ( ( `table0`.`taste` = `table2`.`taste` ) )
```

Thank goodness SAS takes care of all the nasty details of translating your innocent-looking PROC RANK request into the appropriate SQL that takes care of the correct ranking even in the face of ties and missing values.

SPDE – ASK HADOOP TO STORE SAS DATA SETS

Hadoop has HDFS, which is great at storing files and keeping them safe. Can I store my SAS data sets there?

There is a flavor of SAS data set that is well suited to storing in Hadoop. It's the Scalable Performance Data Engine format.

```
libname spdat spde '/user/dodeca' hdfshost=default;
```

This enables you to save your SAS data sets in the Hadoop cluster. Your CFO will like this, because storage costs inside the Hadoop cluster are typically an order of magnitude less than what SAN storage costs. This feature is new as of SAS 9.4, maintenance release 3. It does the following:

- Supports the SAS code accelerator.
- Allows for enhanced WHERE pushdown: AND, OR, NOT, parenthesis, range operators, and in-lists.
- Parallel write support can improve write performance up to 40%.
- You have the option to use Apache Curator or Apache Zookeeper as a distributed lock server. No more physical lock files.
- Provides a Hadoop SerDe for direct READ access to SPD Engine data from the Hadoop ecosystem.

- Provides a Java tool to populate the Hive metastore with SPD Engine metadata.

The second level of connectivity between SAS and Hadoop is SAS/ACCESS software and its connection to Hive, Impala, and Apache HAWQ. It facilitates the transfer of richer metadata regarding tables, as well as encourages the "Hadoop way." That is, move the work to the data by means of SQL pass-through. The second level also includes the SPD Engine format for Hadoop, which enables you to save your SAS data sets in Hadoop

SAS VIYA AND HADOOP – DATING

The SAS® Cloud Analytic Server (CAS) is the component of SAS Viya that implements the high-performance distributed computing logic of SAS software. The CAS server can share the same hosts as the Hadoop cluster, or it can be allocated its own set of servers, effectively running alongside the Hadoop cluster. So, while SAS Viya does not require a Hadoop instance, it co-exists very well with Hadoop if that is where your data is stored. (SAS Viya is also happy to read data from other distributed data stores such as Amazon S3.)

SAS 9.4 AND HADOOP, SAS VIYA, AND HADOOP – GETTING ENGAGED

Expressing work as SQL and pushing that down to the data source is well and good, but taking it to the next level requires that most of the SAS syntax to be executed on the cluster. There are three patterns for this:

1. The SAS® In-Database Code Accelerator for Hadoop, which is an embedded process that can run DS2 code.
2. The SAS® High Performance software (and the SAS® LASR™ Analytic Server behind SAS Visual Analytics).
3. The SAS CAS server – the distributed computational engine that executes the actions requested by SAS Viya – from SAS Visual Analytics, from next-generation SAS procedures, and from other Languages like Python, Lua, and Java (and soon also R).

THE SAS CODE ACCELERATOR

You can think of the SAS Code Accelerator as the run-time engine for processing SAS DS2 code. This engine exists on every data node in the cluster. So the engine can process requests with the same degree of parallelism as the cluster itself. The source code for programs run by the Code Accelerator might be models published in SAS® Analytics software, routine data management tasks as generated by SAS® Data Loader for Hadoop, or they might be specialty tasks for the data quality transformations.

New as of SAS 9,4, maintenance release 3, is support for the MERGE statement. DS2 generates Hive SQL to emulate the classic DATA step merge. Your data flows into the DS2 program data vector in the traditional BY Group style, so your programs have access to IN= as well as FIRST. And LAST. functionality.

The SAS Code Accelerator for Hadoop can take its input from several sources:

- SAS SPD Engine format data sets stored in HDFS
- Hive tables defined to Hadoop

- ORC, Parquet, and Avro tables defined via the HCATALOG

The DS2 Language supported by the SAS Code Accelerator for Hadoop supports a rich variety of input processing

- Arbitrary SQL in the SET statement

```
set { select * from t1 inner join t2 on t1.id = t2.id };
```

- Multiple tables in the SET statement are emulated with HIVE UNION ALL syntax
- MERGE statement is supported by generating custom HIVE SQL to handle the grouping as well as FIRST. and LAST. indicator variables.

The SAS Code Accelerator for Hadoop runs inside a Hadoop MapReduce job. As of SAS 9.4, maintenance release 3, it is fully integrated with the YARN resource manager.

THE SAS HIGH-PERFORMANCE ARCHITECTURE PROCEDURES

As of SAS 9.4, maintenance release 3, a significant set of SAS procedures has been rewritten to run is a distributed fashion on a cluster.

The second generation of SAS distributed computing is embodied in the LASR Analytic Server behind SAS Visual Analytics and other SAS software. This server holds your data set in the distributed memory of the cluster, and contains a set of actions that can be executed against the data sets. You can well imagine there must be actions that correspond to the tasks in SAS Visual Analytics, such as regression, correlation, and forecasting. You can get a feel for the actions more closely from the language of the SAS® In-Memory Statistics software.

THE SAS CAS SERVER

SAS Viya software includes the CAS Server as the distributed computing engine. This is the third-generation implementation of SAS Massively Parallel (MP) logic. The concepts of SAS® High-Performance Analytics procedures and SAS Visual Analytics actions in a LASR Analytic Server have merged into a single server capable of handling the work needed to support both the logic underlying SAS procedures as well as the actions required to implement SAS Visual Analytics and SAS Visual Statistics.

SAS AND HADOOP – GETTING MARRIED

Here are the three levels of commitment described in this paper so far:

1. Exchanging raw files with HDFS
2. Exchanging table-level metadata and using SQL to push work into the cluster
3. The scoring accelerator and SAS high-performance architecture for calculating the results of SAS procedures in a distributed (massively parallel) fashion

These are significant building blocks that allow SAS to build complete products that are based on and in a Hadoop cluster. Hadoop is simply one of the requirements. You can install the SAS product on your existing cluster, or you might use a purpose-built cluster and the Hadoop distribution included with SAS to support your application.

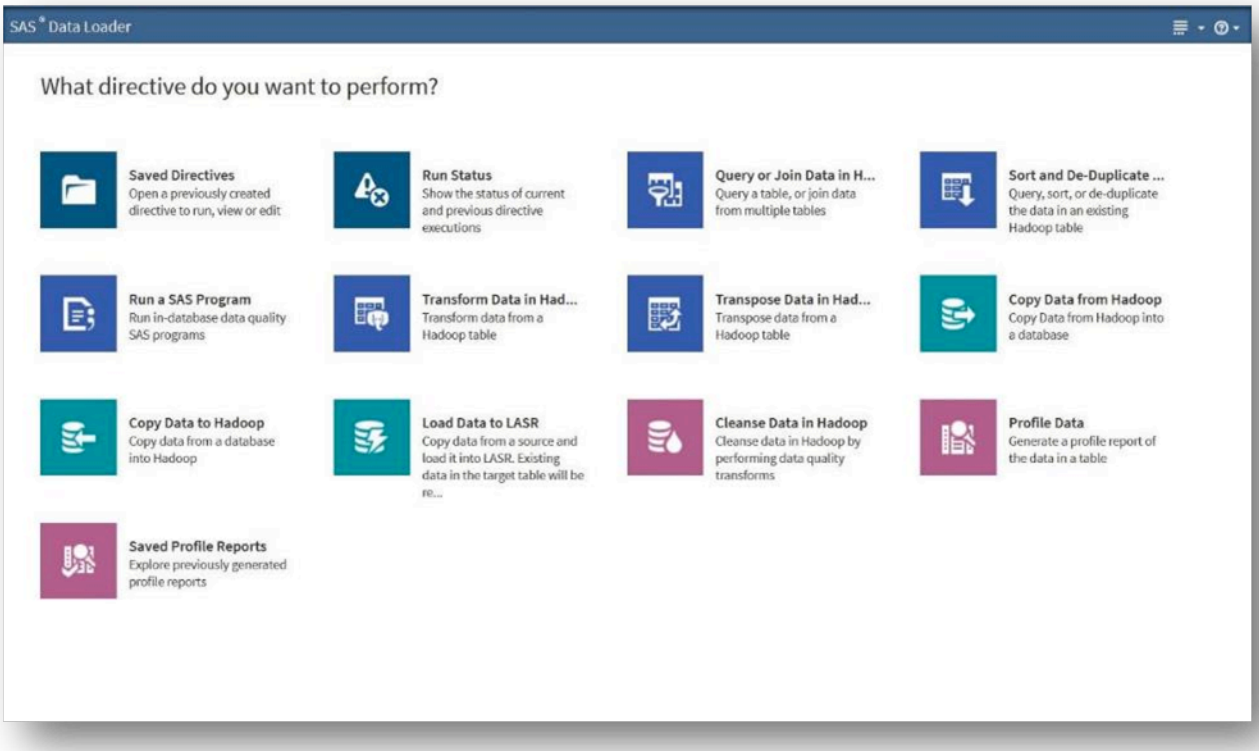
Examples of SAS Software built on this exciting new architecture are as follows:

1. SAS Data Loader for Hadoop

2. SAS Visual Analytics and SAS Visual Statistics
3. Mathematics – Statistics, Forecasting, and Optimization implemented as the following:
    a. SAS® In-Memory Statistics (built on the LASR Analytic Server in SAS 9.4)
    b. SAS actions that are addressable from many languages, including SAS syntax, Python, Lua, Java, and soon R (built on SAS Viya CAS).
4. SAS® Grid Manager for Hadoop

There are many other papers with details about these products from SAS. I include a screen shot or code snippet here to give you a flavor of what is possible as we begin to build on top of this new modern analytical platform.

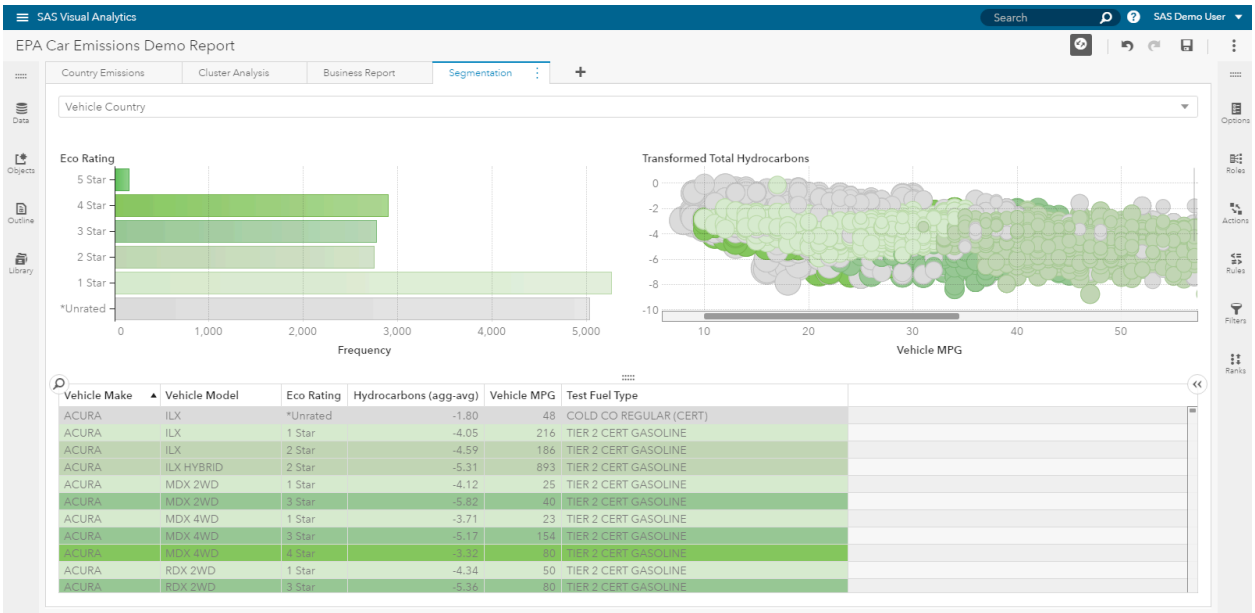
SAS DATALOADER FOR HADOOP



SAS VISUAL ANALYTICS

The 7.x generation of SAS Visual Analytics is implemented using Adobe Flash as the graphical user interface.

SAS Visual Analytics 8.x (the first releases supported by SAS Viya) are modernized to use HTML 5 as the user interface technology.

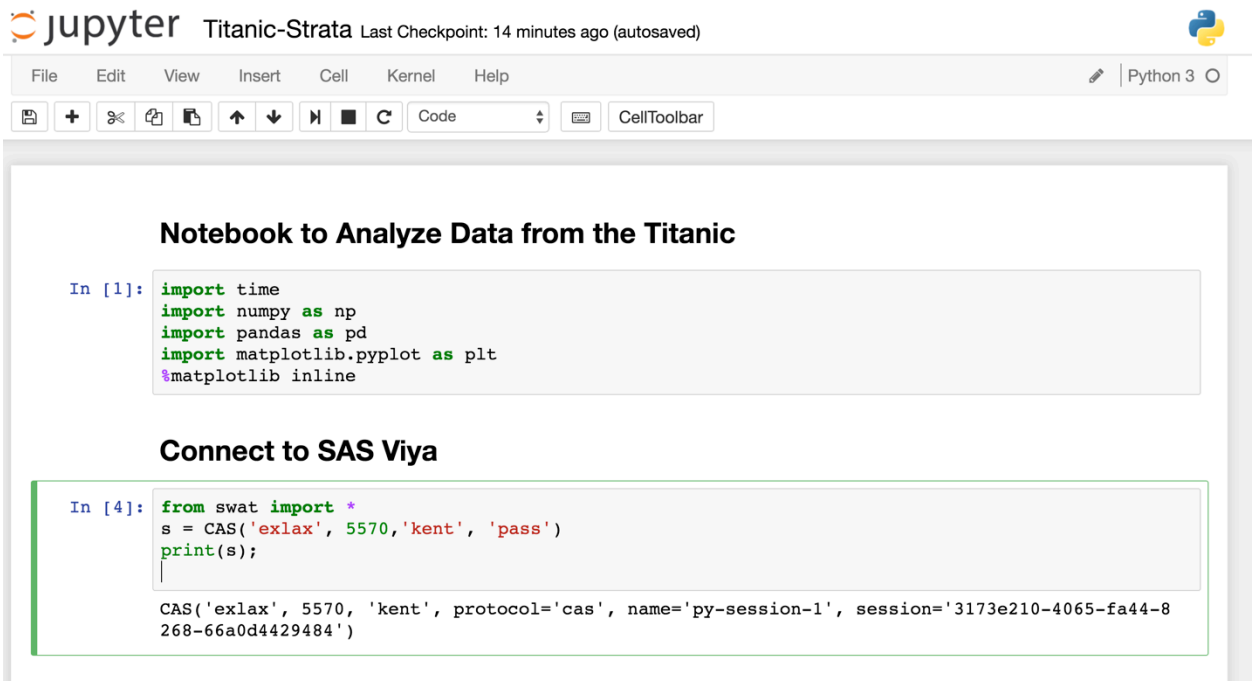


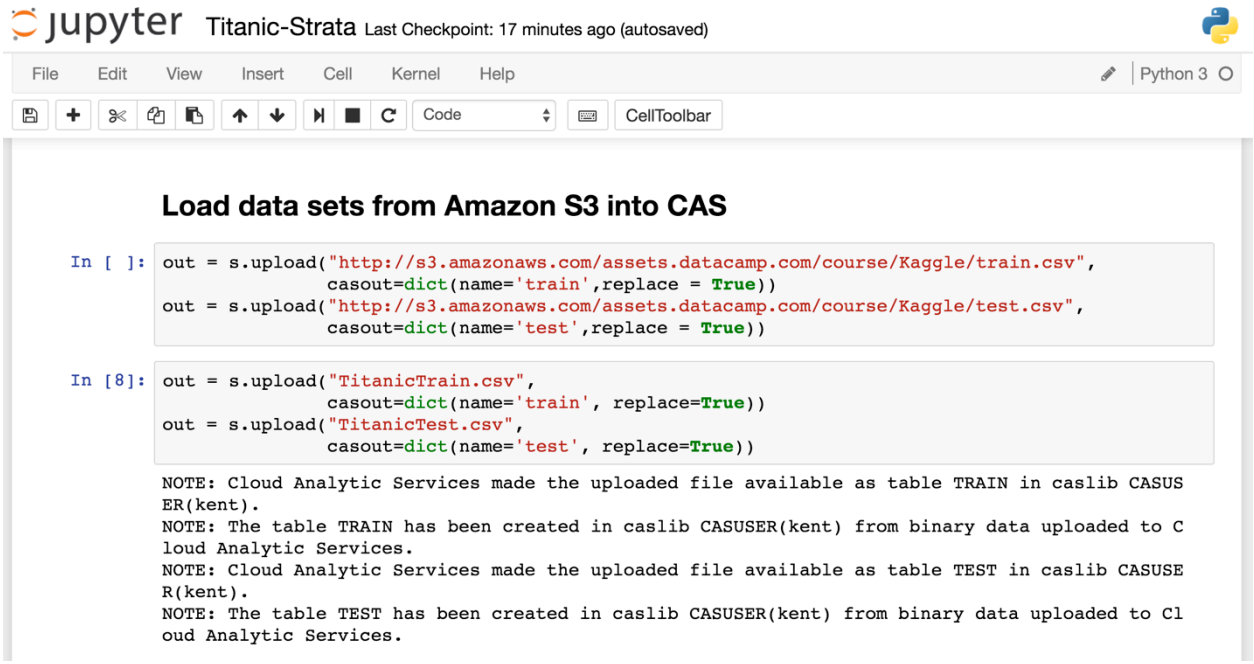
SAS IN-MEMORY STATISTICS

```
proc imstat;
    /*-- list tables in the server -----------------*/
    tableinfo / save=tmpcontent;
    store tmpcontent(2,3) = numObs;
run;
    /*-- print the last ten records from carinfo ---*/
    table lasr.carinfo;
    fetch / from=%eval(&numObs.-9) to=&numObs;
run;
    /*-- working on the fact table (cardata) -------*/
    /*-- data exploration using different actions --*/
    table lasr.cardata;
         distributioninfo;
         distinct _all_;
         boxplot mmrcurrentauctionaverageprice
                 mmrcurrentauctioncleanprice
                 mmrcurrentretailaverageprice
                 mmrcurrentretailcleanprice /
        groupby=auction;
         frequency isbadbuy;
         crosstab isbadbuy*vehyear / groupby=auction;
run;
    table lasr.cardata(tempnames=(avgOdo));
         summary  avgOdo
           / tempnames=avgOdo
             tempexpress="avgOdo = vehodo / (year(purchdate)-vehyear);";
run;

proc recommend port=&lasrport;
  add / item=movieid
  user=userid
  rating=rating;
  addtable mylasr.ml100k / type=rating vars=(movieid userid rating);

  method svd / factors=10 maxiter=20 maxfeval=100 tech=lbfgs seed=1234
               function=L2 lamda=0.1 label="svdlbfgs" details;

  method knn / similarity=pc positive k=20 label="knn1";

  method ensemble / details label="em1" maxiter=10 seed=4321;
run;
```

SAS VIYA ACTIONS CALLED FROM PYTHON USING A JUPYTER NOTEBOOK

These are successive screen shots of a Jupyter notebook using Python to connect to a SAS Viya server, load data from S3 (and locally too), and then perform some elementary data discovery on the data set.





jupyter Titanic-Strata Last Checkpoint: 19 minutes ago (autosaved)

File Edit View Insert Cell Kernel Help Python 3

Code CellToolbar

1. Get to know your data (what is target? what are variables?)

In [10]:
```
dtrain.columninfo()
```

Out[10]: **§ ColumnInfo**

	Column	ID	Type	RawLength	FormattedLength	NFL
0	PassengerId	1	double	8	12	0
1	Survived	2	double	8	12	0
2	Pclass	3	double	8	12	0
3	Name	4	varchar	82	82	0
4	Sex	5	varchar	6	6	0
5	Age	6	double	8	12	0
6	SibSp	7	double	8	12	0
7	Parch	8	double	8	12	0

pandas 0.18.1 documentation »

Table Of Contents

- What's New
- Installation
- Contributing to pandas
- Frequently Asked Questions (FAQ)
- Package overview
- 10 Minutes to pandas
  - Object Creation
  - Viewing Data
  - Selection
    - Getting
    - Selection by Label
    - Selection by Position
    - Boolean Indexing
    - Setting
  - Missing Data
  - Operations
    - Stats
    - Apply

10 Minutes to pandas

This is a short introduction to pandas, geared mainly *Cookbook*

Customarily, we import as follows:

```
In [1]: import pandas as pd
In [2]: import numpy as np
In [3]: import matplotlib.pyplot as plt
```

Object Creation

See the *Data Structure Intro section*

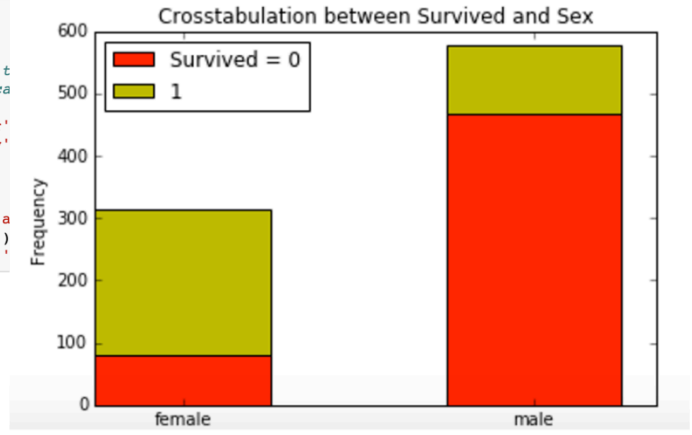
jupyter Titanic-Strata Last Checkpoint: 27 minutes ago (autosaved)

File Edit View Insert Cell Kernel Help

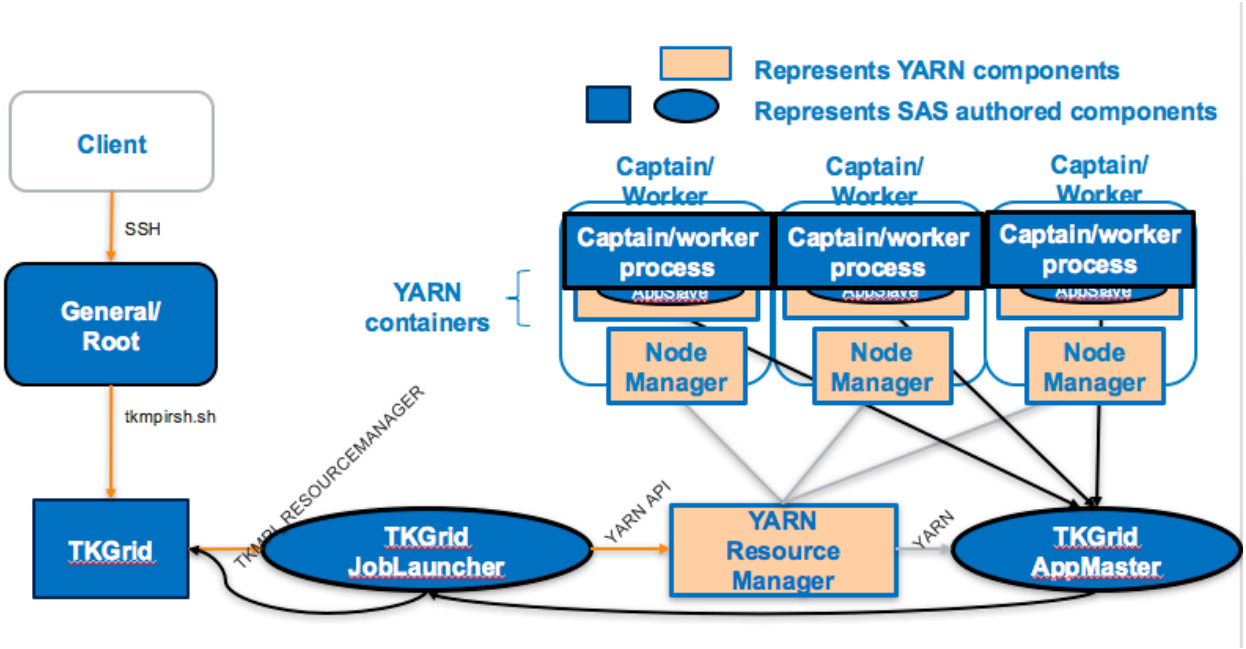
Code CellToolbar

2. Gender vs Survived

In [13]:
```
xtab_out = dtrain.crosstab(row = "Survived",
survived0=xtab_out["Crosstab"].loc[0][1:3]
survived1=xtab_out["Crosstab"].loc[1][1:3]
ind = np.arange(2)    # the x locations for t
width = 0.5       # the width of the bars: ca

p1 = plt.bar(ind, survived0, width, color='r'
p2 = plt.bar(ind, survived1, width, color='y'

plt.ylabel('Frequency')
plt.xlabel('Sex')
plt.title('Crosstabulation between Survived a
plt.xticks(ind + width/2., ('female', 'male')
plt.legend((p1[0], p2[0]), ('Survived = 0', '
```



SAS GRID MANAGER FOR HADOOP



HADOOP, SAS, AND SECURITY

The Hadoop eco-system has evolved its security posture incrementally. In the early days, a secure perimeter and some reasonable user separation were sufficient. This is not enough for the current day and age. Kerberos is used to authenticate people and processes. Sentry and Knox are used to define and manage set of rules that determine who can access what (that is, role-based access control). Cloudera has implemented Record Service, which is a secured way to "enable fine-grained access control to structured data that is consistently enforced across multiple compute frameworks."

As Hadoop has matured along this dimension, SAS has added support for these technologies.

RELEASE NUMBER SOUP

The Hadoop eco-system releases new versions of the many sub-projects that make up a Hadoop distribution at a rapid and sometimes alarming pace. This can often be a challenge for a software vendor like SAS. We keep track of the supported versions of various distributions on this web page:

http://support.sas.com/resources/thirdpartysupport/v94/hadoop/hadoop-distributions.html

http://support.sas.com/en/install-center/sas-viya.html -- See SAS Viya 3.1 System Requirements

CONCLUSION

Hadoop has most definitely changed the analytics landscape. Building models using a large number of computers using distributed computing concepts has become commonplace, as has scoring those models in an “as close to real time” fashion.

SAS software has evolved (and will keep evolving!) to keep up with these trends.

ACKNOWLEDGMENTS

Many people across SAS Institute contribute toward the features that support interacting with Hadoop.

CONTACT INFORMATION

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

Paul Kent
SAS Institute, Inc.

paul.kent@sas.com