AGENDA

• Review of the “FROM, IN & WITHIN” Hadoop integration patterns
• Deployment patterns for SAS HPA/LASR with Hadoop
• SAS/Access and SPDE on HDFS
• DS2 Basics for Hadoop
FROM:  
Moving the data out of Hadoop

Pulling data back to a SAS environment for processing

IN:  
Moving the SAS workload to the data

Run SAS logic in the cluster—process big data with the MapReduce frameworks

WITHIN:  
Moving the SAS application to the data

SAS advanced analytics running natively inside Hadoop under the YARN resource management framework
SAS WITHIN THE HADOOP ECOSYSTEM

Tools
- SAS® Studio
- SAS® Enterprise Guide/Microsoft Office
- SAS® Enterprise Miner
- SAS® Data Management Advanced
- SAS® Data Loader for Hadoop
- SAS® HP Data Mining / Statistics
- SAS® Visual Analytics/Statistics

Metadata
- SAS Metadata

Data Access
- Base/SAS & SAS/ACCESS® to Hadoop™, IMPALA™, HAWQ™

Security
- KNOX/RANGER/SENTRY

Workload management
- YARN

Data Processing, Ingestion & Advanced analytics
- HAWQ (HDP)
  - Oozie
  - Pig
  - Hive/Tez
  - HCATALOG
- IMPALA (CDH)
  - SQOOP
  - MAPREDUCE/SPARK
  - SAS® Embedded Process Accelerators
  - SAS® High Performance Analytics Procedures
  - SAS® LASR™ Analytic Server
  - SAS® Grid™ Manager for Hadoop
  - SAS® Event Stream processing™

Distributed File System
- HDFS
THE SAS LASR® ANALYTIC SERVER

“It is an in-memory engine specifically engineered for the demands of interactive and iterative analytics”

• In-memory = Fast, sub-second responses
• Multi-User = Hundreds of concurrent users
• Stateless = Don’t pre-compute things
• Interactive = Instantly visualise analytical output
• Deployment = MPP on HDFS (distributed) or SMP (single machine)
SAS HIGH PERFORMANCE ANALYTICS (HPA)

Single / Multi-threaded
Not aware of distributed computing environment
Computes locally / where called
Fetches Data as required
Memory still a constraint

Massively Parallel (MPP)
Two degrees of Parallelism
Uses distributed computing environment
Computes in massively distributed mode
Work is co-located with data
In-Memory Analytics
40 nodes x 96GB almost 4TB of memory

```
proc logistic data=HDP.mydata;
  class A B C;
  model y(event='1') = A B B*C;
run;
```

```
proc hplogistic data=HDP.mydata;
  class A B C;
  model y(event='1') = A B B*C;
run;
```
## LASR VS HPA

<table>
<thead>
<tr>
<th></th>
<th>LASR</th>
<th>HPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Model</td>
<td>Public</td>
<td>Private</td>
</tr>
<tr>
<td></td>
<td>(Data persisted in-memory and shared)</td>
<td>(each execution of the proc creates own</td>
</tr>
<tr>
<td></td>
<td></td>
<td>copy of the data in-memory. Data is not</td>
</tr>
<tr>
<td></td>
<td></td>
<td>persisted)</td>
</tr>
<tr>
<td>Concurrent users</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Key SAS</td>
<td>• SAS Visual Analytics/Statistics</td>
<td>• SAS High Performance Data</td>
</tr>
<tr>
<td>Products</td>
<td>• SAS In-memory statistics</td>
<td>Mining (via Enterprise Miner)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SAS High Performance statistics (via</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EG or SAS Studio)</td>
</tr>
</tbody>
</table>
THREE DEPLOYMENT OPTIONS FOR HPA/LASR

Symmetric
SAS TKGrid on name and all data nodes

Asymmetric
(separate SAS cluster)

Asymmetric
(collocated subset)
SAS TKGrid on subset of data nodes – YARN manages resources
1. Load data from SASHDAT if available (fastest)
2. Load data in parallel from Hadoop cluster via SAS EP
3. Serial loads via SAS/Access ok for small tables
/*libname to local HDFS storing SASHDAT data*/
libname sashdfs sashdat path="/hps/user" HOST="xxxxxx-01.suk.sas.com" install="/opt/TKGrid";

<table>
<thead>
<tr>
<th>option</th>
<th>notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>sashdat</td>
<td>The SAS engine which refers to HDFS</td>
</tr>
<tr>
<td>Path=</td>
<td>The hdfs path</td>
</tr>
<tr>
<td>Host=</td>
<td>The hostname of the TKGrid head node</td>
</tr>
<tr>
<td>Install=</td>
<td>The path where the SAS TKGrid binaries are installed</td>
</tr>
</tbody>
</table>
SASHDAT IS A UNIDIRECTIONAL ENGINE

You can create data using the SASHDAT engine but you cannot re-read it. E.g.

NOTE: Libref SASHDFS was successfully assigned as follows:
    Engine: SASHDAT
    Physical Name: Directory '/hps/user' of HDFS cluster on host 'ukva1-01.suk.sas.com'

60
61
62 data sashdfs.hmeq_new;
63 set sashdfs.hmeq;
64 run;

ERROR: The SASHDAT engine is a uni-directional engine. Data flows from the SAS client to the Hadoop Distributed File System. The engine cannot be used to fetch data from HDFS.
USE PROC HPDS2 TO MANIPULATE SASHDAT DATA

Proc HPDS2 can be used to create a new sashdat files from a sashdat file

```sas
proc hpds2
   in = sashdfs.simdata_large
   out = sashdfs.simdata_large_blocksize(blocksize=128m);
data D52GTF.out;
dcl double avgx;
method run();
set D52GTF.in;
   sumx=sum(x1,x2,x3);
   end;
enddata;
run;
```

NOTE: The HPDS2 procedure is executing in the distributed computing environment with 7 worker nodes.
NOTE: The data set SASHDFS.SIMDATA_LARGE_BLOCKSIZE has 199000000 observations and 10 variables.
NOTE: PROCEDURE HPDS2 used (Total process time):
   real time        49.46 seconds
   cpu time         5.81 seconds

```
-rw-r--r--  sukdmg  supergroup  11.88 GB  2  2 MB  simdata_large.sashdat
-rw-r--r--  sukdmg  supergroup  14.88 GB  2 128 MB simdata_large_blocksize.sashdat
```
SAS/ACCESS AND SPDE ON HDFS
SAS/ACCESS® TO HADOOP

- Uses Existing SAS Interfaces
- Standard Libname syntax
- PROC HADOOP
- Datastep and Proc SQL translated to Hive
- Filename support
- Execute Pig Scripts and MapReduce
- Push-down of certain procedures
- Custom SerDe support
- SPDE formats
SAS/ACCESS TO HADOOP

- HIVE Data types (avoid strings, use VARCHAR for character fields)
- Use native Hadoop file formats (ORC, PARQUET etc.) and partition data where appropriate
- Make use of supported In-database SAS procedures
  - FREQ, MEANS, REPORT, SUMMARY/MEANS, TABULATE

Data integration:
- Use the standard SQL transformations in DI
- Generate explicit pass-through
- Create and manage SASHDAT and LASR tables using the DI transformations
MAKING USE OF YARN QUEUES

• Setting Hive Queue:
  • PROPERTIES= option can be added to the LIBNAME statement to add properties, like `mapreduce.job.queuename`, to the library connection.
    (http://support.sas.com/documentation/cdl/en/acreldb/68028/HTML/default/viewer.htm#p0ly2onqqpbys8n1j9lra8qa6q20.htm)

Libname hivetez hadoop server="gbrhadoop1-01" USER=sasdemo PASSWORD="{SAS002}1D57933958C580064BD3DCA81A33DFB2"
port=10000 PROPERTIES='mapreduce.job.queuename=sas_user_queue'
DBCREATE_TABLE_OPTS='STORED AS PARQUET';
• Avoid joining SAS data with Hive Data. It is recommended to move the SAS dataset into Hive and execute the join inside Hadoop to leverage distributed processing.
• Avoid using SAS functions that will bring back Hadoop data on the SAS Server because the function does not exist in HIVE. E.g. datepart
• Use SASTRACE option to see the communication between SAS and Hadoop.
SPDE ON HDFS

SAS Data Set SAN

- Metadata
- Data
- Index

SAS Scalable Performance Data Engine and or Server Table SAN and HDFS

- Metadata
- Data 1
- Data 2
- Data 3
- Data 4

- Hybrid Index
- Bitmap, B-tree Indexes
**SPDE ON HDFS**

**Analytical Base Table**

Meant to support VERY wide tables for Predictive Analytics, Visualization, Dashboards, Self Service Reporting
SPDE ON HDFS

Can sometimes be faster than HIVE access when working with SAS:
• Depending on the queries (no need to deal with Hive, direct access via HDFS)
• Can be faster than HIVE when used as input to SAS HPA procedures

SPDE also provide some of the traditional SAS features as:
• Encryption
• File compression
• Member-level locking
• SAS indexes
• SAS password
• Special missing values
• Physical ordering of returned observations
• User-defined formats and informats
LEVERAGING HADOOP USING SPD ENGINE

1. Use PROC HADOOP to create the path on HDFS:

   ```
   proc hadoop
     username='Hadoop_userid'
     password='Hadoop_password'
   verbose;
   hdfs mkdir='/user/sasss1/spde';
   run;
   ```

2. SPD Engine LIBNAME statement:

   ```
   LIBNAME MYSPDE SPDE '/user/sasss1/spde'
   HDFSHOST=DEFAULT
   PARALLELWRITE=YES
   PARALLELREAD=YES
   ACCELWHERE=YES;
   ```

1. MYSPDE is the libref we reference in our SAS code to process the SPD Engine data stored on HDFS.
2. SPDE is the engine SPD Engine uses to process SPD Engine tables.
3. '/user/sasss1/spde' is the path on HDFS where our SPD Engine data is stored.
4. HDFSHOST=DEFAULT To connect to the Hadoop cluster, Hadoop configuration files must be copied from the specific Hadoop cluster to a physical location that the SAS client machine can access. The SAS environment variable SAS_HADOOP_CONFIG_PATH must be defined and set to the location of the Hadoop configuration files. For complete instructions, see the SAS Hadoop Configuration Guide for Base SAS and SAS/ACCESS.
5. PARALLELWRITE=YES tells SPD Engine to use parallel processing to write data to HDFS. **Note: data must be uncompressed.**
6. PARALLELREAD=YES tells SPD Engine to use parallel processing to read data stored in HDFS. Note: data can be uncompressed, compressed or encrypted.
7. ACCELWHERE=YES tells SPD Engine, when possible, to push all WHERE clauses down to Hadoop as MapReduce.
## HADOOP AND SAS FILE FORMATS

<table>
<thead>
<tr>
<th>Engine</th>
<th>Typically, use for</th>
<th>3rd party Hadoop access</th>
</tr>
</thead>
</table>
| HIVE       | • Data that needs to be available for processing by the broader Hadoop ecosystem  
            • Data to be processed by pushdown SQL queries or SAS DS2.                                                                                                                                             | yes                     |
| (ORC - HDP) (Parquet - CDH) (AVRO) |                                                                                                                                                                                                                   |                         |
| SASHDAT    | • Persisting data on HDFS and for the fast, parallel loading of data into LASR/HPA                                                                                                                             | No                      |
| (only supported for symmetric deployments) |                                                                                                                                                                                                                   |                         |
| SPDE on HDFS | • Migrating SAS data sets onto HDFS without code modification  
            • Optimised data retrieval back to SAS.  
            • Input to LASR/HPA (faster than HIVE)  
            • Very wide analytical base tables                                                                                                             | Yes – read only access via SAS supplied SerDe |
|            |                                                                                                                                                                                                                   |                         |
## OVERVIEW OF PROCESSING OPTIONS

<table>
<thead>
<tr>
<th>SAS Programming Method</th>
<th>SPDE</th>
<th>HIVE</th>
<th>SASHDAT</th>
<th>LASR (SASIOLA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proc SQL implicit</td>
<td>Yes</td>
<td>Yes – via SAS/Access</td>
<td>No</td>
<td>No*</td>
</tr>
<tr>
<td>Proc SQL explicit</td>
<td>No</td>
<td>Yes – via SAS/Access</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Data Step</td>
<td>Yes - via SAS EP</td>
<td>Yes - via SAS EP</td>
<td>No**</td>
<td>Yes</td>
</tr>
<tr>
<td>Proc DS2</td>
<td>Yes - via SAS EP</td>
<td>Yes - via SAS EP</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Proc HPDS2</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

*Would work but will pull data to SAS client for processing
**Can be used to create new SASHDAT datasets but not to modify data
THE SAS EMBEDDED PROCESS: A WORD ON THE TECHNOLOGY

A portable, lightweight execution container for SAS code that makes SAS portable and deployable on a variety of platforms

```sas
proc ds2;
/* thread ~ equiv to a mapper */
  thread map_program;
  method run(); set dbmslib.intab;
  /* program statements */
  end;
  endthread;
/* program wrapper */
data hdf.data_reduced;
  dcl thread map_program map_pgm;
  method run();
  set from map_pgm threads=N;
  /* reduce steps */ end; enddata;
  run; quit;
```

1. Data Lifting
2. Data Preparation
3. Data Quality
4. Scoring
RUNFASTER.RUN EMBEDDED

• Efficient way to process data.
• Runs inside Hadoop’s MPP architecture.
• Moves the computation to the data.
• Eliminates data movement.
• Decreases overall processing times.
• Procedural programming language.
• Mainly focused around parallel execution.
• Supports ANSI SQL data types.
• Allows Embedded SQL as input to the program.
• Allows modular programming: Scope and Methods.
• Supports Packages and Threads.
SAS EMBEDDED PROCESS FOR HADOOP

- Lightweight execution container for DS2.
- Written in C and Java.
- Runs inside a MapReduce task.
- Orchestrated by Hadoop MapReduce framework.
- Resource allocation managed by YARN.
WHAT IS DS2?

- Does not replace the DATA step language
- DATA step DNA is clearly visible
  - DATA and SET statements
  - IF...THEN...ELSE, DO loops
  - Expressions and Functions
  - Arrays
- A new species of DATA step
  - No INFILE, INPUT, MERGE, UPDATE, MODIFY statements
  - Methods, Packages, and Scoping – oh my!
## DS2: WHAT IS DS2? DATA TYPES

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIGINT</td>
<td>INTEGER</td>
</tr>
<tr>
<td>BINARY(n)</td>
<td>NCHAR(n)</td>
</tr>
<tr>
<td>CHAR(n)</td>
<td>NVARCHAR(n)</td>
</tr>
<tr>
<td>DATE</td>
<td>REAL</td>
</tr>
<tr>
<td></td>
<td>SMALLINT</td>
</tr>
<tr>
<td></td>
<td>TIME(p)</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>NUMERIC((p,s))</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>TINYINT</td>
</tr>
<tr>
<td>FLOAT(p)</td>
<td>VARBINARY(n)</td>
</tr>
</tbody>
</table>
DS2 DATA STEP SIMILARITIES/DIFFERENCES

Other obvious differences between BASE SAS and DS2

- Many DATA step functions are implemented in DS2
- But, many are not

<table>
<thead>
<tr>
<th>Analysis of Function Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA Step</td>
</tr>
<tr>
<td>-----------</td>
</tr>
<tr>
<td>468</td>
</tr>
</tbody>
</table>

User-defined functions (PROC FCMP) can be executed in DS2
Other obvious differences between BASE SAS and DS2:

- Over half of the DATA step statements are not implemented.

<table>
<thead>
<tr>
<th>Analysis of Programming Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data step</td>
</tr>
<tr>
<td>-----------</td>
</tr>
<tr>
<td>73</td>
</tr>
</tbody>
</table>
RUNNING SAS DATA STEP & DS2 IN HADOOP THROUGH THE CODE ACCELERATOR

• Key SAS options: DSACCELL=ANY and DS2ACCELL=ANY
• DS2 in Hadoop supports both HIVE and SAS SPDE tables
  • Use proc HPDS2 to manipulate SASHDAT tables
1. Hadoop libname
2. SAS Options
3. Create thread program
4. DS2 logic
5. Call thread program
proc ds2 ds2accel=yes;
thread compute;
   method run();
      set hdfs.emp_donations;
      total = sum(jan--dec);
   end;
endthread;
data hdfs.totals;
dcl thread compute t;
   method run();
      set from t;
   end;
enddata;
run; quit;
proc ds2 ds2accel=yes;
thread compute;
   method run();
      set hdfs.emp_donations;
      by region;
      if first.region then total = 0;
      total + sum(jan--dec);
      if last.region then output;
   end;
endthread;
data hdfs.totals;
dcl thread compute t;
   method run();
      set from t;
   end;
enddata;
data test;
  input i j x;
datalines;
1 1 123
1 1 3245
1 2 23
1 2 543
1 2 87
1 3 90
2 1 88
2 1 86
;

/* When the first observation in each BY-Group is read, the variables JSUB and */
/* FREQ are initialized to zero and with each subsequent observation in the */
/* BY-Group, FREQ is incremented by one and JSUB is incremented by the value of */
/* X. When the last observation in the BY-Group is read, AVER is created by */
/* dividing JSUB by FREQ to determine the average value for the group. */

data jsbtot (keep=i j freq aver);
  set test;
  by i j;
  retain jsub freq;
  if first.j then do;
    jsub=0;
    freq=0;
  end;
  jsub + x;
  freq + 1;
  if last.j then do;
    aver=jsub/freq;
    output;
  end;
run;

proc print;
run;
proc ds2;
    thread compute / overwrite=yes;
    declare double jsub freq aver;
    retain jsub freq;
    keep i j freq aver;
    method run();
    set hdp.test;
    by i j;
    if first.j then do;
        jsub=0;
        freq=0;
    end;
    jsub + x;
    freq + 1;
    if last.j then do;
        aver=jsub/freq;
        output;
    end;
    end;
endthread;
data hdp.jsubtot (overwrite=yes);
    declare thread compute t;
    method run();
    set from t;
    end;
enddata;
run;
quit;
DS2 EQUIVALENT FOR HADOOP

```
proc ds2;
    thread compute / overwrite=yes;
    declare double jsub freq aver;
    retain jsub freq;
    keep i j freq aver;
    method run();
    set hdp.test;
    by i j;
    if first.j then do;
        jsub=0;
        freq=0;
    end;
    jsub + x;
    freq + 1;
    if last.j then do;
        aver=jsub/freq;
        output;
    end;
endthread;
data hdp.jsubtot (overwrite=yes);
    declare thread compute t;
    method run();
    set from t;
    end;
enddata;
run;
quit;
```
To run in-database, a thread program must be used. The SAS Code Accelerator enables you to publish a DS2 thread program and execute that thread program in parallel inside Hadoop.
Unlike Base/SAS, DS2 enables you to explicitly declare variables using the DECLARE statement. Here it is declared outside of a method so its scope is GLOBAL.
DS2 EQUIVALENT FOR HADOOP

```sas
proc ds2;
    thread compute / overwrite=yes;
    declare double jsuh freq aver;
    retain jsuh freq;
    keep i j freq aver;
    method run();
    set hdp.test;
    by i j;
    if first.j then do;
        jsuh=0;
        freq=0;
    end;
    jsuh + x;
    freq + 1;
    if last.j then do;
        aver=jsuh/freq;
        output;
    end;
endthread;

data hdp.jsuh tot (overwrite=yes);
    declare thread compute t;
    method run();
    set from t;
    end;
enddata;
run;
quit;
```

DS2 has new data types, more akin to an RDBMS, and should be explicitly declared. E.g. VARCHAR, DOUBLE, INT, BIGINT etc.
DROP/KEEP/RETAIN/RENAME are only valid in global scope. i.e. outside of a method programming block.
**DS2 EQUIVALENT FOR HADOOP**

Method `run()` is a system method – will execute in an implicit loop for every row of the input data. Other system methods are `init()` & `term()`.
This block of code is identical to the original data step program.
DS2 EQUIVALENT FOR HADOOP

```
proc ds2;
  thread compute / overwrite=yes;
  declare double jsup freq aver;
  retain jsup freq;
  keep i j freq aver;
  method run();
  set hdp.test;
  by i j;
  if first.j then do;
    jsup=0;
    freq=0;
    end;
    jsup + x;
    freq + 1;
  if last.j then do;
    aver=jsup/freq;
    output;
  end;
  endthread;
  data hdp.jsubtot (overwrite=yes);
    declare thread compute t;
    method run();
    set from t;
    end;
  enddata;
run;
quit;
```

A BY statement is required to generate Hadoop REDUCE tasks. Without a BY statement, only MAP tasks are generated.
End statement to close the run() method.
DS2 EQUIVALENT FOR HADOOP

proc ds2;
   thread compute / overwrite=yes;
      declare double jsub freq aver;
      retain jsub freq;
      keep i j freq aver;
      method run();
      set hdp.test;
      by i j;
      if first.j then do;
         jsub=0;
         freq=0;
      end;
      jsub + x;
      freq + 1;
      if last.j then do;
         aver=jsub/freq;
         output;
      end;
   endthread;
   data hdp.jsubtot (overwrite=yes);
      declare thread compute t;
      method run();
      set from t;
      end;
   enddata;
run;
quit;
Now we reference the output dataset to be created on Hadoop

```sas
proc ds2;
  thread compute / overwrite=yes;
    declare double jsup freq aver;
    retain jsup freq;
    keep i j freq aver;
  method run();
  set hdp.test;
  by i j;
  if first.j then do;
    jsup=0;
    freq=0;
  end;
  jsup + x;
  freq + 1;
  if last.j then do;
    aver=jsup/freq;
    output;
  end;
endthread;
data hdp.jsubtot (overwrite=yes);
  declare thread compute t;
  method run();
  set from t;
  end;
enddata;
run;
quilt;
```
Explicitly declare the thread program and specify a name that identifies an instance of the thread.
Use method `run()` to allow the program to read from the thread program.
proc ds2;
   thread compute / overwrite=yes;
      declare double jsub freq aver;
      retain jsub freq;
      keep i j freq aver;
      method run();
      set hdp.test;
      by i j;
      if first.j then do;
         jsub=0;
         freq=0;
      end;
      jsub + x;
      freq + 1;
      if last.j then do;
         aver=jsub/freq;
      output;
      end;
   endthread;
   data hdp.jsubtot (overwrite=yes);
      declare thread compute t;
      method run();
      set from t;
      end;
      enddata;
run;
quit;
End statement to close the run() method.
The enddata statement marks the end of a data statement.
The RUN statement submits the DS2 statements
DS2 EQUIVALENT FOR HADOOP

As DS2 is a SAS procedure we must explicitly quit it.
proc ds2;
NOTE: Connection string:
NOTE: (DRIVER=SQL;CONCPTS= (DRIVER=SQL;CATALOG=work;SCHEMA=*
(NAME=work;PRIMARYPATH=/tmp/SAS_work458850004F88_ukval-01.suk.sas.com;SAS_work415E00004F88_ukval-01.suk.sas.com)); (DRIVER=HIVE;SERVER=gr8hadoop-01.suk.sas.com;DB=ukcmd;Pw=D70HT18001;SUBPROTOCOL=Hivev2;Hadoop:CATALOG=work;SCHEMA=*
(NAME=work;PRIMARYPATH=/home/sukcmd/workwork)); (DRIVER=base;CATALOG=SASDATA;SCHEMA=*
(NAME=SASDATA;PRIMARYPATH=/data/SAS/config/Level1/SASapp/Data/)); (DRIVER=base;CATALOG=СПДСЖ;SCHEMA=*
(NAME=СПДСЖ;PRIMARYPATH=/data/SAS/config/Level1/SASapp/Data/альб)); (DRIVER=base;CATALOG=МАПС;SCHEMA=*
(NAME=МАПС;PRIMARYPATH=/data/SAS/software/SASFoundation/9.4/maps/)); (DRIVER=base;CATALOG=МУПЖ;SCHEMA=*
(NAME=МУПЖ;PRIMARYPATH=/data/SAS/software/SASFoundation/9.4/maps/)); (DRIVER=base;CATALOG=SASUSER;SCHEMA=*
(NAME=SASUSER;PRIMARYPATH=/home/sukcmd/sasuser.v94/))));
thread compute / overwrite=yes;
declare double jsub freq aver;
isub = freq = 0;
method run();
set hcp.test;
by i j;
if first.i.j then do;
  jsub = 0;
  freq = 0;
end;
jsub + x;
freq = 1;
if last.i j then do;
  aver = jsub/freq;
output;
end;
end;
endthread;
data hdo;subtot (overwrite=yes);
declare thread compute t;
method run();
set from t;
end;
output;
enddata;
nrun;
NOTE: Created thread compute in data set work.compute.
NOTE: Running THREAD program in-database.
NOTE: Running DATA program in-database.
NOTE: Execution succeeded. No rows affected.
quit;
NOTE: PROCEDURE DS2 used (Total process time):
### WHAT'S HAPPENING ON THE HADOOP CLUSTER?

<table>
<thead>
<tr>
<th>Logs</th>
<th>ID</th>
<th>Name</th>
<th>Status</th>
<th>User</th>
<th>Maps</th>
<th>Reduces</th>
<th>Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1431102899342_0270</td>
<td>SAS Map/Reduce Job</td>
<td>RUNNING</td>
<td>sukdmg</td>
<td>50%</td>
<td>50%</td>
<td>root.sukdmg</td>
</tr>
</tbody>
</table>