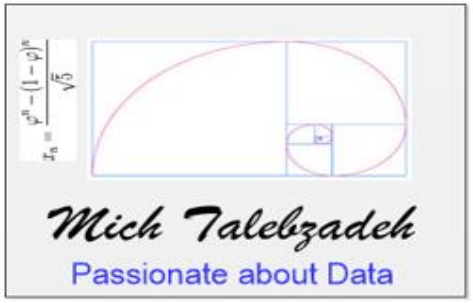


Query Engines for Hive: MR, Spark, Tez with LLAP – Considerations!

Presentation: Future of Data
Organised by Hortonworks
London July 20, 2016

Author

This presentation was prepared by:

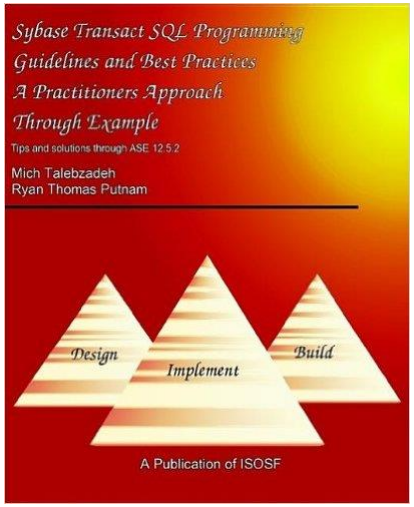
Mich Talebzadeh

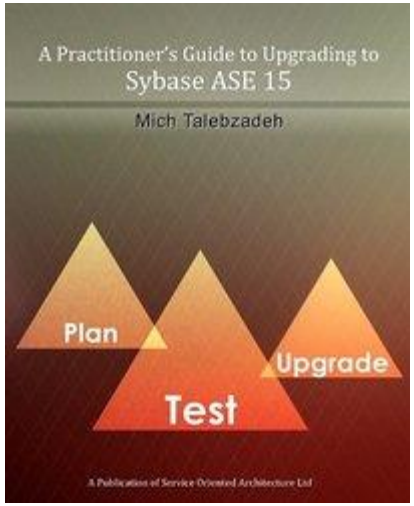
Big Data and RDBMS Senior Technical Architect

E-mail: mich@Peridale.co.uk

https://www.linkedin.com/in/mich-talebzadeh-ph-d-5205b2?

https://talebzadehmich.wordpress.com/





The Big Picture

- Hive and Spark are both extensively used in Big Data Space
- In a nutshell, with Hive on Spark engine, one gets the Hive optimizer and Spark query engine.
- With Spark using Hive context, Spark does both the optimization (using Catalyst) and query engine (Spark).
- Although on the face of it there are distinct advantages for each case, in my opinion these are two tools that can **complement each other** in numerous ways *and we should leverage both where needed as practitioners including using Hive metastore for Spark.*
- Do not know if there is necessarily a universal preferred way for how to use Spark as an execution engine or indeed if Spark is necessarily the best execution engine for any given Hive job.

The Big Picture

- The reality is that once you start factoring in the numerous tuning parameters of the systems and jobs there probably is not a clear answer. For some queries, the Catalyst optimizer may do a better job. For others it may not be.
- Spark as yet does not have a Cost Based Optimizer (CBO) although there are plans for it. Please follow this JIRA, which suggests that it is planned for the future:
  https://issues.apache.org/jira/browse/SPARK-16026
  We Consider the approaches with Spark using Hive Context and Hive metastore and Hive using Spark as its execution engine.

Spark Affinity with Hive

- Spark has Spark SQL that provides similar functionality and syntax as Hive SQL
- Spark has a complete fork of Hive inside it. Spark SQL is a sub-set of Hive SQL
- In Spark 2.0, they a are writing (some) DDL functionality within Spark. Looks like they are reducing dependency on Hive.
- Beware not all Hive features are supported!
- Spark Thrift Server (STS) allows JDBC client access to Spark SQL. It uses Hive Driver
- Can run multiple STS on different nodes, but essentially can run multiple STS on the same node, different ports

Spark Affinity with Hive

- In some cases using STS will give you the benefit of using Hive SQL with the added advantage of Spark in-memory performance.
- Transactional support was added to Hive for ORC tables.
- No transactional support with Spark SQL on ORC tables yet.
- Locking and concurrency (as used by Hive) with Spark app running a Hive context. I am not convinced this works

Case for Hive

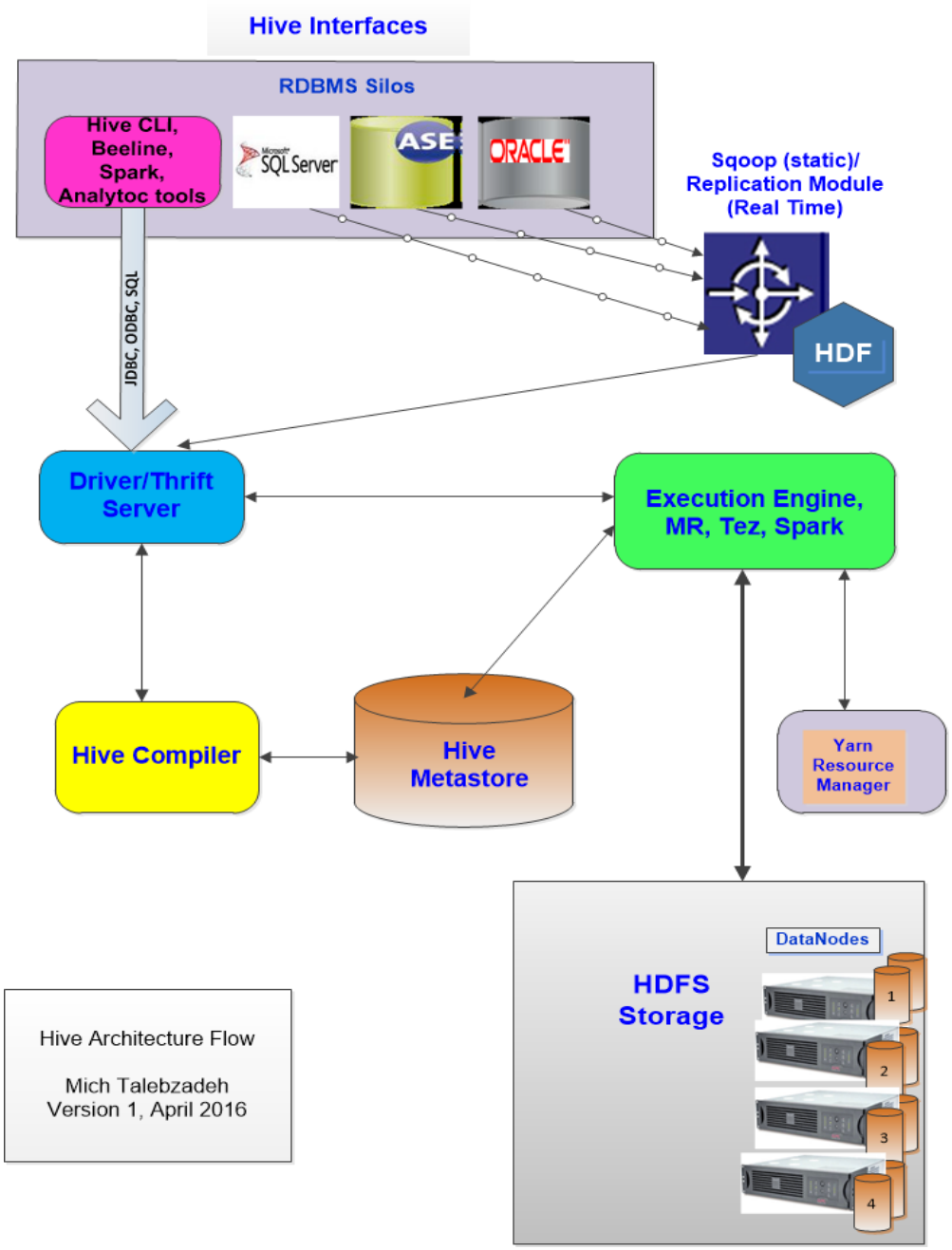
- Hive was billed as a Data Warehouse (DW) on HDFS.
- Hive is the most versatile and capable of the many SQL or SQL-like ways of accessing data on Hadoop
- You can set up your copy of your RDBMS table in Hive in no time and use Sqoop to get the table data into Hive table practically in one command. For many this is the great attraction of Hive that can be summarised as:

Case for Hive

- Leverage existing SQL skills on Big Data.
- You have a choice of metastore for Hive including MySQL, Oracle, SAP ASE and others.
- You have a choice of plug ins for your engine (Map-Reduce, Spark, Tez)
- Ability to do real time analytics on Hive by sending real time transactional movements from RDBMS tables to Hive via the existing replication technologies. This is very handy. Today, organizations are struggling to achieve real-time integration between RDBMS silos and Big Data.
- Fast decision-making depends on real-time data movement that allows businesses to gather data from multiple locations into Big Data as well as conventional data warehouses.
- Hive offers a convenient mechanism for this but crucially requires a faster engine, We will come to that

Case for Hive

- Hive is not the correct tool for every problem. Like anything else you need to use the tool that makes the most sense for your problem and your experience. Your mileage varies.
- One of the main hindrance/reluctance of Hive deployment has been its perceived slow performance because of its reliance on the map-reduce execution engine. You put in your query, go to lunch, and hope it is done by the time you get back. Not exactly interactive.
- Need low latency/interactive capability for Hive


Hive Interfaces
RDBMS Silos
Hive CLI, Beeline, Spark, Analytoc tools
SQL Server
ASE
ORACLE
Sqoop (static)/ Replication Module (Real Time)
HDF
JDBC, ODBC, SQL
Driver/Thrift Server
Execution Engine, MR, Tez, Spark
Hive Compiler
Hive Metastore
Yarn Resource Manager
HDFS Storage
DataNodes
1
2
3
4
Hive Architecture Flow
Mich Talebzadeh
Version 1, April 2016

Options considered for Hive Engine

- Hive on Map-reduce. Default out of the box
- Hive on Tez essentially Map-reduce with DAG
- Hive on Tez plus LLAP, a great novel alternative
- Hortonworks provide Hive on Tez + LLAP as a distro
- Hive on Spark uses DAG and Spark's in-memory capabilities
- Used Hive on Spark. Tried few releases of Spark from the source. Managed to make it work kind of war of attrition!
- Not for everyone but does work

Deploying Hive on Spark Execution Engine

- Need to build your spark-assembly.jar without Hive and copy it to Hive's /lib directory.
- You shouldn't use the one you downloaded from Spark installation for Hive to use. The downloaded one has Hive classes possibly of a different version. They clash with existing classes
- Use the jar built from source code (spark version 1.3.1) -> spark-assembly-1.3.1-hadoop2.4.0.jar
- The stack used to build the jar file
  - `Maven 3.3.9`
  - `Java version: 1.8.0_77`
  - `OS name: "linux", version: "2.6.18-92.el5", arch: "amd64", family: "unix"`
  - `Spark version 1.3.1`
  - `Hive version 1.2.1`
  - `Hadoop version 2.6`

Deploying Hive on Spark Execution Engine

- Hive on Spark, current challenges:
  - Use the jar built from source code (spark version 1.3.1) -> spark-assembly-1.3.1-hadoop2.4.0.jar
  - `./make-distribution.sh --name "hadoop2-without-hive" --tgz "-Pyarn,hadoop-provided,hadoop-2.4,*.jar'`

```
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary:
[INFO]
[INFO] Spark Project Parent POM ........................... SUCCESS [ 19.250 s]
[INFO] Spark Project Networking ........................... SUCCESS [ 20.643 s]
[INFO] Spark Project Shuffle Streaming Service ............ SUCCESS [  4.461 s]
[INFO] Spark Project Core ................................. SUCCESS [02:22 min]
[INFO] Spark Project Bagel ................................ SUCCESS [  7.642 s]
[INFO] Spark Project GraphX ............................... SUCCESS [ 22.831 s]
[INFO] Spark Project Streaming ............................ SUCCESS [ 39.353 s]
[INFO] Spark Project Catalyst ............................. SUCCESS [ 41.270 s]
[INFO] Spark Project SQL .................................. SUCCESS [01:03 min]
[INFO] Spark Project ML Library ........................... SUCCESS [01:14 min]
[INFO] Spark Project Tools ................................ SUCCESS [  6.224 s]
[INFO] Spark Project Hive ................................. SUCCESS [01:40 min]
[INFO] Spark Project REPL ................................. SUCCESS [ 14.469 s]
[INFO] Spark Project YARN ................................. SUCCESS [ 10.835 s]
[INFO] Spark Project Assembly ............................. SUCCESS [ 59.374 s]
[INFO] Spark Project External Twitter ..................... SUCCESS [ 14.482 s]
[INFO] Spark Project External Flume Sink .................. SUCCESS [  6.524 s]
[INFO] Spark Project External Flume ....................... SUCCESS [ 10.195 s]
[INFO] Spark Project External MQTT ........................ SUCCESS [ 12.407 s]
[INFO] Spark Project External ZeroMQ ...................... SUCCESS [ 11.234 s]
[INFO] Spark Project External Kafka ....................... SUCCESS [ 16.516 s]
[INFO] Spark Project Examples ............................. SUCCESS [02:42 min]
[INFO] Spark Project YARN Shuffle Service ................. SUCCESS [ 14.564 s]
[INFO] Spark Project External Kafka Assembly .............. SUCCESS [ 31.014 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 15:07 min
[INFO] Finished at: 2016-07-06T20:25:34+01:00
[INFO] Final Memory: 81M/1035M
[INFO] ------------------------------------------------------------------------
```

Deploying Hive on Spark Execution Engine

- Hive on Spark, current challenges:
  - You just need this by-product
  - `spark-1.3.1-bin-hadoop2-without-hive.tgz`
  - Unzip and untar it and you will get
  - `spark-1.3.1-bin-hadoop2-without-hive/lib/`**`spark-assembly-1.3.1-hadoop2.4.0.jar`**
  - Take this jar file and place it under directory $HIVE_HOME/lib
  - Before starting Hive do
  - unset $SPARK_HOME
  - create a new environment variable to indicate that you want to use Spark as the execution engine for Hive:
  - `export HIVE_ON_SPARK='Y'`

Deploying Hive on Spark Execution Engine

- Hive on Spark, current challenges:
  - `# Add Spark assembly jar to the classpath for Hive on Spark engine as a work-around! Set HIVE_ON_SPARK='Y' outside of this shell`
  - `if [[ -n "$HIVE_ON_SPARK" ]]`
  - `then`
  - `  sparkAssemblyPath=`ls ${HIVE_HOME}/lib/spark-assembly-*.jar``
  - `  CLASSPATH="${CLASSPATH}:${sparkAssemblyPath}"`
  - `fi`

  I noticed that they have now incorporated above into hive 2 script itself but use different environment variable ☺

```
# add Spark assembly jar to the classpath
if [[ -n "$SPARK_HOME" && !("$HIVE_SKIP_SPARK_ASSEMBLY" =
"true") ]]
then
  sparkAssemblyPath=`ls ${SPARK_HOME}/lib/spark-assembly-*.jar`
  CLASSPATH="${CLASSPATH}:${sparkAssemblyPath}"
fi
```

Deploying Hive on Spark Execution Engine

- Hive on Spark, current challenges:
  - Modify hive-site.xml to use Spark

```
<property>
    <name>hive.execution.engine</name>
    <value>spark</value>
    <description>
      Expects one of [mr, tez, spark].
      Chooses execution engine. Options are: mr
(Map reduce, default), tez, spark. While MR
      remains the default engine for historical
reasons, it is itself a historical engine
      and is deprecated in Hive 2 line. It may
be removed without further warning.
    </description>
  </property>
```

Deploying Hive on Spark Execution Engine

- Hive on Spark, current challenges:
  - Modify hive-site.xml and provide Spark binary location

```
<property>
  <name>spark.home</name>
  <value>/usr/lib/spark-1.3.1-bin-hadoop2.6</value>
  <description>Directory where Spark binaries are installed to be used as Hive execution engine</description>
</property>
```

Deploying Hive on Spark Execution Engine

- Hive on Spark, current challenges:
  - Modify hive-site.xml and provide Spark Run mode

```
<property>
  <name>spark.master</name>
  <value>yarn-client</value>
  <description>something</description>
</property>
<property>
  <name>spark.eventLog.enabled</name>
  <value>true</value>
  <description>something</description>
</property>
```

Deploying Hive on Spark Execution Engine

- Hive on Spark, current challenges:
  - Modify hive-site.xml, other parameters etc

```
<property>
    <name>spark.eventLog.dir</name>
  <value>/work/hadoop/tmp/spark/logs</value>
    <description>something</description>
 </property>
 <property>
    <name>spark.executor.memory</name>
    <value>512m</value>
    <description>something</description>
 </property>
```

Deploying Hive on Spark Execution Engine

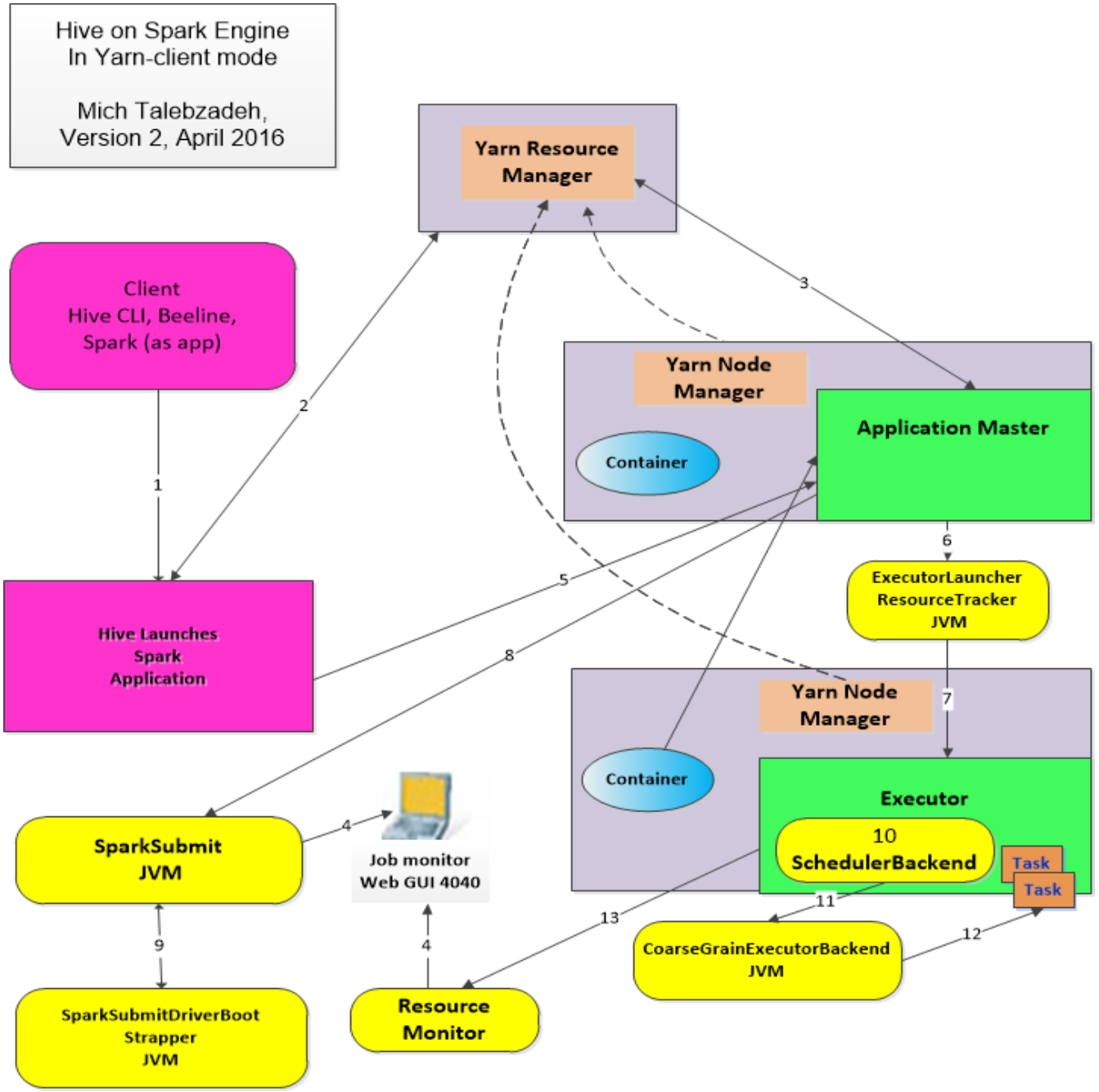
- Hive on Spark, current challenges:
  - Modify hive-site.xml, Check other options in hive-site.xml.
- Otherwise
  - You can put these in an initialisation file
  - `hive_on_spark_init.hql`
  - What is in that file
  - `set spark.home=/usr/lib/spark-1.3.1-bin-hadoop2.6;`
  - `--set spark.home=/usr/lib/spark-2.0.0-preview-bin-hadoop2;`
  - `set hive.execution.engine=spark;`
  - `set spark.master=yarn;`
  - `set spark.deploy.mode=cluster;`
  - `set spark.executor.memory=3g;`
  - `set spark.driver.memory=3g;`
  - `set spark.executor.cores=8;`
  - `set spark.ui.port=7777;`

Deploying Hive on Spark Execution Engine

- Otherwise
  - If I want to use MR as my execution engine.
  - Only one liner!
  - `set spark.execution.engine=mr`
  - Case in point, which engine am I using in this session?
  - `hive>  set hive.execution.engine;`
  - `hive.execution.engine=spark`
  - `hive> CREATE TEMPORARY TABLE tmp AS SELECT * FROM test.mytest WHERE 1 = 2;`
  - **`Starting Spark Job`** `= eac0453b-c259-4c9a-b3e0-fd9951aa8222`
  - **`Query Hive on Spark job[0] stages:`**

Deploying Hive on Spark Execution Engine

- Now switch the execution engine to MR
  - `hive> set hive.execution.engine=mr;`
  - `hive> CREATE TEMPORARY TABLE tmp2 AS SELECT * FROM test.mytest WHERE 1 = 2;`
  - `WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.`
  - `Starting Job = job_1468226887011_0010, Tracking URL =`


Hive on Spark Engine
In Yarn-client mode
Mich Talebzadeh,
Version 2, April 2016
Yarn Resource Manager
Client
Hive CLI, Beeline,
Spark (as app)
Yarn Node Manager
Container
Application Master
Hive Launches
Spark
Application
ExecutorLauncher
ResourceTracker
JVM
Yarn Node Manager
Container
Executor
10
SchedulerBackend
Task
Task
SparkSubmit
JVM
Job monitor
Web GUI 4040
CoarseGrainExecutorBackend
JVM
SparkSubmitDriverBoot
Strapper
JVM
Resource Monitor
1
2
3
4
5
6
7
8
9
10
11
12
13

Hive Optimizer and Spark

- Does Hive account for what engine it is using?
- In Hive, command EXPLAIN can be used to show the execution plan of a query.
- For Hive on Spark, this command itself is not changed. It behaves the same as before. It still shows the dependency graph, and plans for each stage.
- Note that if the engine is set to “spark”, it shows the execution plan with the Spark execution engine, instead of the default (“mr”) MapReduce execution engine.
-

Hive Optimizer and Spark

- This one with Spark
- `EXPLAIN SELECT MAX(id) from dummy_parquet;`

```
Explain
STAGE DEPENDENCIES:
  Stage-1 is a root stage
  Stage-0 depends on stages: Stage-1

STAGE PLANS:
  Stage: Stage-1
    Spark
      Edges:
        Reducer 2 <- Map 1 (GROUP, 1)
      DagName: hduser_20160713120252_0dad2071-9f7b-4902-b8f6-9c2778f1d77b:3
      Vertices:
        Map 1
            Map Operator Tree:
                TableScan
                  alias: dummy_parquet
                  Statistics: Num rows: 100000000 Data size: 700000000 Basic stats: COMPLETE Column stats: NONE
                  Select Operator
                    expressions: id (type: int)
                    outputColumnNames: id
                    Statistics: Num rows: 100000000 Data size: 700000000 Basic stats: COMPLETE Column stats: NONE
                    Group By Operator
                      aggregations: max(id)
                      mode: hash
                      outputColumnNames: _col0
                      Statistics: Num rows: 1 Data size: 4 Basic stats: COMPLETE Column stats: NONE
                      Reduce Output Operator
                        sort order:
                        Statistics: Num rows: 1 Data size: 4 Basic stats: COMPLETE Column stats: NONE
                        value expressions: _col0 (type: int)
        Reducer 2
            Reduce Operator Tree:
              Group By Operator
                aggregations: max(VALUE._col0)
                mode: mergepartial
                outputColumnNames: _col0
```

Hive Optimizer and Spark

- And this one with map reduce

```
Explain
STAGE DEPENDENCIES:
  Stage-1 is a root stage
  Stage-0 depends on stages: Stage-1

STAGE PLANS:
  Stage: Stage-1
    Map Reduce
      Map Operator Tree:
          TableScan
            alias: dummy_parquet
            Statistics: Num rows: 100000000 Data size: 700000000 Basic stats: COMPLETE Column stats: NONE
            Select Operator
              expressions: id (type: int)
              outputColumnNames: id
              Statistics: Num rows: 100000000 Data size: 700000000 Basic stats: COMPLETE Column stats: NONE
              Group By Operator
                aggregations: max(id)
                mode: hash
                outputColumnNames: _col0
                Statistics: Num rows: 1 Data size: 4 Basic stats: COMPLETE Column stats: NONE
                Reduce Output Operator
                  sort order:
                  Statistics: Num rows: 1 Data size: 4 Basic stats: COMPLETE Column stats: NONE
                  value expressions: _col0 (type: int)
      Reduce Operator Tree:
        Group By Operator
          aggregations: max(VALUE._col0)
          mode: mergepartial
          outputColumnNames: _col0
```

Hive Optimizer and Spark

- Dependency Graph
  - Dependency graph shows the dependency relationship among stages.
  - For Hive on Spark, there are Spark stages instead of Map Reduce stages. There is no difference for other stages.
  - For most queries, **there is just one Spark stage since many map and reduce works can be done in one Spark work.** Therefore, for the same query, with Hive on Spark, there may be less number of stages. For some queries, there are multiple Spark stages, for example, queries with map join, skew join, etc..

Hive Optimizer and Spark

- Dependency Graph
  - One thing should be pointed out that here a stage means a Hive stage. It is very different from the stage concept in Spark. A Hive stage could correspond to multiple stages in Spark. In Hive, a stage contains a list of operations that can be processed in one job
- Spark Stage Plan
  - The plans for each stage are shown by command EXPLAIN, besides dependency graph. For Hive on Spark, the Spark stage is new. It replaces the Map Reduce stage for Hive on MapReduce. The Spark stage shows the Spark work graph, which is a DAG (directed acyclic graph). It contains:
    - ● DAG name, the name of the Spark work DAG;
    - ● Edges, that shows the dependency relationship among works in this DAG;
    - ● Vertices, that shows the operator tree of each work.

Hive Cost Based Optimizer

- Spark Stage Plan
  - For each individual operator tree, there is no change for Hive on Spark. The difference is dependency graph.
  - For MapReduce, you cannot have a reducer without a mapper.
  - For Spark, that is not a problem. **Therefore, Hive on Spark can optimize the plan and get rid of those mappers not needed.**
  - The edge information is new for Hive on Spark. There is no such information for MapReduce.
  - Different edge type indicates different shuffle requirement. For example, Check this pdf
  - *https://cwiki.apache.org/confluence/download/attachments/.../hos_explain.pdf?...*

How does Hive on Spark fare?

- Simple test. Not a proof that in all cases Hive on Spark is going to be faster but taken as an indication. As ever you need to test it for yourself. Back to your mileage varies statement

Small test ride

First using Hive 2 on Spark 1.3.1 to find max(id) for a 100million rows parquet table

```
hive> select max(id) from oraclehadoop.dummy_parquet;

Starting Spark Job = a7752b2b-d73a-45de-aced-ddf02810938d
Query Hive on Spark job[1] stages:
2
3
Status: Running (Hive on Spark job[1])
Job Progress Format
CurrentTime StageId_StageAttemptId: SucceededTasksCount(+RunningTasksCount-FailedTasksCo
2016-07-11 17:41:52,386 Stage-2_0: 0(+8)/24     Stage-3_0: 0/1
2016-07-11 17:41:55,409 Stage-2_0: 1(+8)/24     Stage-3_0: 0/1
2016-07-11 17:41:56,420 Stage-2_0: 8(+4)/24     Stage-3_0: 0/1
2016-07-11 17:41:58,434 Stage-2_0: 10(+2)/24    Stage-3_0: 0/1
2016-07-11 17:41:59,440 Stage-2_0: 12(+8)/24    Stage-3_0: 0/1
2016-07-11 17:42:01,455 Stage-2_0: 17(+7)/24    Stage-3_0: 0/1
2016-07-11 17:42:02,462 Stage-2_0: 20(+4)/24    Stage-3_0: 0/1
2016-07-11 17:42:04,476 Stage-2_0: 23(+1)/24    Stage-3_0: 0/1
2016-07-11 17:42:05,483 Stage-2_0: 24/24 Finished       Stage-3_0: 1/1 Finished
Status: Finished successfully in 14.12 seconds
OK
100000000
Time taken: 14.38 seconds, Fetched: 1 row(s)
```

How does Hive on Spark fare?

Simply switch the engine in hive to MR

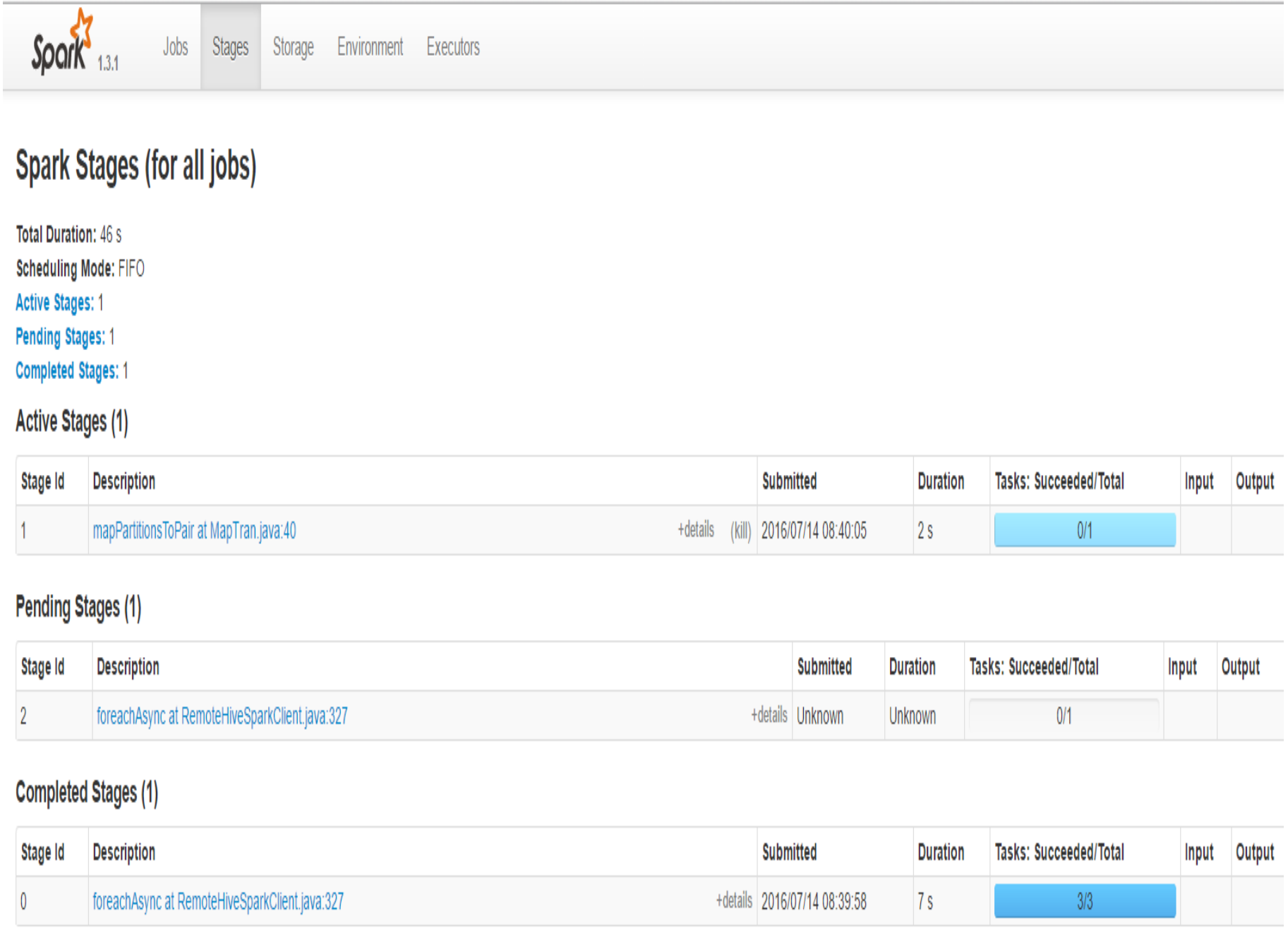
```
hive> set hive.execution.engine=mr;
Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions.
Consider using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.

hive> select max(id) from oraclehadoop.dummy_parquet;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions.
Consider using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.
Starting Job = job_1468226887011_0005, Tracking URL = http://rhes564:8088/proxy/application_1468226887011_0005/
Kill Command = /home/hduser/hadoop-2.6.0/bin/hadoop job  -kill job_1468226887011_0005
Hadoop job information for Stage-1: number of mappers: 24; number of reducers: 1
2016-07-11 17:42:46,904 Stage-1 map = 0%,  reduce = 0%
2016-07-11 17:42:56,328 Stage-1 map = 4%,  reduce = 0%, Cumulative CPU 31.76 sec
2016-07-11 17:43:05,676 Stage-1 map = 8%,  reduce = 0%, Cumulative CPU 61.78 sec
2016-07-11 17:43:16,091 Stage-1 map = 13%,  reduce = 0%, Cumulative CPU 95.44 sec
2016-07-11 17:43:24,419 Stage-1 map = 17%,  reduce = 0%, Cumulative CPU 121.6 sec
----------------------------------------------------------------------------------
2016-07-11 17:44:14,222 Stage-1 map = 42%,  reduce = 0%, Cumulative CPU 286.21 sec
2016-07-11 17:44:22,502 Stage-1 map = 46%,  reduce = 0%, Cumulative CPU 310.34 sec
2016-07-11 17:44:32,923 Stage-1 map = 50%,  reduce = 0%, Cumulative CPU 346.26 sec
2016-07-11 17:46:02,962 Stage-1 map = 88%,  reduce = 0%, Cumulative CPU 632.38 sec
2016-07-11 17:46:13,316 Stage-1 map = 92%,  reduce = 0%, Cumulative CPU 666.45 sec
2016-07-11 17:46:23,656 Stage-1 map = 96%,  reduce = 0%, Cumulative CPU 693.72 sec
2016-07-11 17:46:31,919 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 714.71 sec
2016-07-11 17:46:36,060 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 721.83 sec
MapReduce Total cumulative CPU time: 12 minutes 1 seconds 830 msec
Ended Job = job_1468226887011_0005
MapReduce Jobs Launched:
Stage-Stage-1: Map: 24  Reduce: 1   Cumulative CPU: 721.83 sec   HDFS Read: 400442823 HDFS Write: 10 SUCCESS
Total MapReduce CPU Time Spent: 12 minutes 1 seconds 830 msec
OK
100000000
Time taken: 239.532 seconds, Fetched: 1 row(s)
```

How does Hive on Spark fare?

- Did the test on an ORC table using data from the Parquet table
- Won't repeat the details just summarize them all

Table	MR/sec	Spark/sec
Parquet	239.532	14.38
ORC	202.333	17.77

- Still I would use Spark if I had a choice and I agree that on VLT (very large tables), the limitation in available memory may be the overriding factor in using Spark.
- As always the devil is in the detail.
- You can also check the progress of work on Spark UI


Spark 1.3.1
Jobs
Stages
Storage
Environment
Executors
Spark Stages (for all jobs)
Total Duration: 46 s
Scheduling Mode: FIFO
Active Stages: 1
Pending Stages: 1
Completed Stages: 1
Active Stages (1)
Stage Id | Description | Submitted | Duration | Tasks: Succeeded/Total | Input | Output
1 | mapPartitionsToPair at MapTran.java:40 +details (kill) | 2016/07/14 08:40:05 | 2 s | 0/1
Pending Stages (1)
Stage Id | Description | Submitted | Duration | Tasks: Succeeded/Total | Input | Output
2 | foreachAsync at RemoteHiveSparkClient.java:327 +details | Unknown | Unknown | 0/1
Completed Stages (1)
Stage Id | Description | Submitted | Duration | Tasks: Succeeded/Total | Input | Output
0 | foreachAsync at RemoteHiveSparkClient.java:327 +details | 2016/07/14 08:39:58 | 7 s | 3/3

What About Alternatives?

- Hortonworks Distro offers Hive on Tez and LLAP!
- http://www.slideshare.net/HadoopSummit/llap-subsecond-analytical-queries-in-hive-63959757