Quick Poll

Where do you use SQL?

A. Transactional Systems
B. Data Warehouse
C. BI / Analytics
Agenda

• The need for SQL on Hadoop
• Current popular open source options for SQL on Hadoop
• Feature review of Hive, SparkSQL, Drill, HAWQ, Phoenix, Splice machine and Trafodian
• Q&A
The Need for SQL
Transition to Modern Data Architecture - Legacy

Source: HDInsight Essentials (ISBN 1784396664)
Transition to Modern Data Architecture - New
SQL on Hadoop – the need

• Hadoop is a fit for **ETL offload** and/or Data warehouse offload

• But…it is not a cheap **replacement** to RDBMS; it’s a new **platform**

• SQL-on-Hadoop is an **abstraction** on HDFS and YARN

• SQL-on-Hadoop enables **ad-hoc** analysis on files, **ETL** and abstractions on complex data types

• SQL-on-Hadoop reduces the **learning curve** for existing DBA’s and developers
SQL on Hadoop – Patterns of Usage

- **Batch**
  - ETL jobs that run typically daily
  - Select, Insert, Update and Delete
  - Latency minutes to hours

- **Interactive**
  - Ad-hoc analysis and queries
  - Typically Select
  - Latency seconds to minutes

- **Transactional**
  - ACID transactional support
  - Select, Insert, Update and Delete
  - Latency milliseconds to second
Patterns of Usage & Popular Options

- **Batch**: Hive, Spark, SQL, Hawq
- **Interactive**: Apache Drill
- **Transactional**: Trafodion, Splice Machine, Apache Phoenix
General Architecture of SQL Engines

3 general architectures for SQL Engines

• **Leverage YARN and managed completely by Hadoop.** Examples Hive, SparkSQL

• **Separate process outside Hadoop;** does share metadata, HDFS and infrastructure. Example Hawq, Drill

• **Leverage Hbase**  Example Phoenix, Trafodian, Splicemachine
Hive Overview
Hive Leveraging YARN

Source: HDInsight Essentials (ISBN 1784396664)
Hive Key Features

- **Standard component** in all Hadoop distributions
- **Hcatalog** (Hive metadata) is standard metastore for several Hadoop components
- **Data warehouse solution** built on Hadoop for providing data summarization.
- Provides SQL-like query language called **HiveQL**.
- Minimal learning curve for SQL developers/business analysts.
- Ability to bring structure to various formats.
- Simple interface for **ad-hoc query**, summarization, joins on big data.
- Extensible via data types, functions and scripts.
- **JDBC driver** to connect with SQL IDE’s like SquirrelSQL, DBeaver
Notable New Features (Hive 1.2)

- Transactional support (ACID) limited to ORC formats
- Row level inserts, updates, or deletes
- ALTER SCHEMA DDL
- ANSI SQL compliance which is not 100%, but getting closer
- Cost Based Optimizer
- SQL Temporary Tables
CREATE TABLE docs (line STRING);

LOAD DATA INPATH 'docs.txt' OVERWRITE INTO TABLE docs;

CREATE TABLE word_counts AS
SELECT word, count(1) AS count FROM
(SELECT explode(split(line, '\s')) AS word FROM docs) w
GROUP BY word ORDER BY word;
Hive Architecture
But…..

• **Latency**; queries run typically in minutes and not seconds.
• **HiveQL** is not ANSI compliant; RDBMS queries will need rewrite
• **Update, Delete** support is **limited** (Transactions support only works on ORC)
• **No Primary Keys, Foreign Keys, Constraints**.

**Bottom line**… **this is not a replacement for transactional databases**
SparkSQL Overview
Spark & SparkSQL Key Features

- **Strong community support** - currently the **most popular** Hadoop component (2015, 2016)

- Can run as **YARN** application and runs in **memory and disk**

- Developers can use **Java, Scala, Python, R & SQL**

- Powerful SQL features like **Pivot** tables, efficient Joins

- Improved Query performance and caching with **Catalyst**

- Claims **100X** faster response time over MapReduce in memory, **10X** on disk.

- Supports **HiveQL** and leverages Hive metastore

- **JDBC, OBDC and beeline support**
Spark Architecture (2.0)

Spark SQL

Catalyst

ML Pipelines  |  Structured Streaming  |  GraphFrames

SQL  |  Dataframe / Dataset

Spark Core (Resilient Distributed Dataset)

Source: http://www.slideshare.net/SparkSummit/deep-dive-into-catalyst-apache-spark-20s-optimizer-63071120
Spark-SQL in action

> ./spark-shell // initialize spark shell

> sqlContext.sql("CREATE TABLE page_view(viewTime INT, page_url, STRING, user STRING") // create Hive table

> sqlContext.sql("LOAD DATA LOCAL INPATH '/path/to/file.txt' INTO TABLE page_view") // load data

> sqlContext.sql("SELECT * FROM page_view") // select data from table
Spark-SQL in action using thrift and beeline

[root@quickstart sbin]# ./start-thriftserver.sh --hiveconf hive.server2.thrift.port=12345
Spark assembly has been built with Hive, including Datanucleus jars on classpath

beeline> !connect jdbc:hive2://localhost:12345
scan complete in 7ms
Connecting to jdbc:hive2://localhost:12345
Enter username for jdbc:hive2://localhost:12345: cloudera
Enter password for jdbc:hive2://localhost:12345:
log4j:WARN No appenders could be found for logger (org.apache.thrift.transport.TSaslTransport).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Connected to: Hive (version 0.12.0)

0: jdbc:hive2://localhost:12345> show databases;
+-----------------+
| result          |
+-----------------+
| default         |
| training        |
+-----------------+
But....

- Latency (better than Hive): may not be sufficient for some applications.
- Queries are not ANSI compliant; RDBMS queries will need rewrite
- Update, Delete support is not supported
- No Primary Keys, Foreign Keys, Constraints..

Bottom line... maybe good for ad-hoc queries but not for transactions.
What's behind Catalyst
Spark References

• Spark SQL: http://spark.apache.org/docs/latest/sql-programming-guide.html

• Spark Scala API docs: http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.package

• Overview of DataFrames: http://xinhstechblog.blogspot.com/2016/05/overview-of-sparkdataframe-api.html
HAWQ Overview

(Incubator Project)
HAWQ Key Features

- **MPP database** that is native to Hadoop HDFS, YARN and Ambari managed
- **Robust ANSI SQL compliance**: SQL-92, SQL-99, SQL-2003, OLAP extension
- **Many times faster** than other Hive and Spark-SQL
- **Full transaction capability** and consistency guarantee: ACID
- Multi-language user defined function support: **Python, Perl, Java, C/C++, R**
- **Standard connectivity**: JDBC/ODBC
- **Support** multiple level partitioning, constraints, procedures.
- **Built on top of** **Greenplum DB engine**
HAWQ Architecture

- **HAWQ Master** is entry point for all client requests.
- **HAWQ master** hosts metadata in global system catalog.
- **Physical Segments** are units that process data in parallel. One per Hadoop Data Node.
- Each segment can start multiple **Query Executors (QE)** that act as virtual segments.

Source: [http://hdb.docs.pivotal.io/20/overview/HAWQArchitecture.html](http://hdb.docs.pivotal.io/20/overview/HAWQArchitecture.html)
Create Table and check list

CREATE TABLE product_dim
  (product_id integer,
   product_name varchar(200))
DISTRIBUTED BY (product_id) ;

gadmin=# \dt
List of relations
  Schema | Name    | Type   | Owner
-------------------------------
student0 | product_dim | table | gpadmin
(1 row)

gadmin=## \\
Query global catalog

gadmin=# select * from pg_tables where
  tablename='users';
- schemaname | tablename | tableowner | tablespace | hasindexes | hasrules | hstriggers
+---------------------------------------------
  student0 | product_dim | gpadmin | f | f | f
(1 row)

gadmin=## \d product_dim;
  Table “product_dim”
  Column | Type | Modifiers
---------------------------------------------
  product_id | integer | 
  product_name | character varying(200) |
But....

- Currently HAWQ is in **Apache incubation** state (was Pivotal earlier)

- For best performance, **data has to be stored in HAWQ format**; which is not usable by other tools in eco-system.

- **stored by HAWQ in HDFS** can only be used by HAWQ.

- **Additional processing/compute** is required to manage data intake from external files (like .csv) into Hawq.

- Does **not** work well with **complex data types**
Comparing GPFDIST and PXF

<table>
<thead>
<tr>
<th>GPFDIST</th>
<th>PXF</th>
</tr>
</thead>
<tbody>
<tr>
<td>For readable external tables, can read files present on a local file system</td>
<td>For readable external tables, reads from files present in hdfs</td>
</tr>
<tr>
<td>For writable external tables, stores files on local file system</td>
<td>For writable external tables, stores files on HDFS</td>
</tr>
<tr>
<td>supports plain text format only</td>
<td>supports binary format like AVRO and customized format.</td>
</tr>
<tr>
<td>gpfdist doesn’t support generating compressed file</td>
<td>while PXF supports compression (can specify compression codec used in Hadoop like org.apache.hadoop.io.compress.GzipCodec).</td>
</tr>
</tbody>
</table>
HAWQ References

http://hdb.docs.pivotal.io/20/overview/HAWQOverview.html
Drill Key Features

- Scale-Out MPP Query Engine with low latency
- **Schema-less** access to any data, any data source with SQL interface
- Supports ANSI SQL
- Scale in all dimensions
- **Excellent Query** response times; great for ad-hoc queries & data exploration
- **Row/Column** level controls
- Can query Hadoop and other NoSQL data stores
Drill in action

use dfs.yelp;

alter session set `store.format`='json';

>SELECT id, type, name, ppu FROM dfs.`/Users/brumsby/drill/donuts.json`;

+------------+------------+------------+------------+
| id         | type       | name       | ppu        |
+------------+------------+------------+------------+
| 0001       | donut      | Cake       | 0.55       |
+------------+------------+------------+------------+

1 row selected (0.248 seconds);

Source: https://drill.apache.org/docs/create-table-as-ctas/
But..

- Requires separate processes to run Drillbits
- Only read-only support
- Memory and CPU intensive; example for 128GB server, recommendations

  File system = 20G, HBase = 20G, OS = 8G, YARN = 60GB, Drill=20GB

Reference: https://drill.apache.org/docs/configuring-multitenant-resources/
Trafodion Overview
(Incubator Project)
Trafodian Key Features

- **OLTP and ODS** with ANSI support
- **Distributed ACID transaction** protection across multiple statements, tables and rows
- **Integrated with HBase and Hive**
- Support for nested loop, merge, hash joins
- **ODBC and JDBC** drivers
Trafodian Architecture - Components

Source: http://trafodion.apache.org/presentations/dtm-architecture.pdf
Sqlcl or trafcl

```sql
create schema test_sandbox_schema;
set schema test_sandbox_schema;
create table t (c1 int not null, c2 int not null, primary key (c1));
insert into t values (1,1);
insert into t values (2,3);
insert into t values (3,2);
begin work;
insert into t values (4,5);
insert into t values (5,2);
commit work;
insert into t values (7,3);
select * from t order by c2;
create index tix on t (c2);
create view tview as select c1, c2 from t where c2 > 3;
select * from tview where c2 < 3;
select * from tview where c2 > 2;
update statistics for table t on every column;
explain select * from t order by c2;
select * from t order by c2;
drop view tview;
drop table t;
drop schema test_sandbox_schema cascade;
```

Connect with HBase

Example

```sql
select * from hbase."_CELL_"."TRAFODION.MYSCH.MYTAB";
select * from hbase."_CELL_"."table_created_in_HBase";
```

Connect with Hive

```sql
hive> create database testdb;
hive> create table testdb.test1(col1 string, col2 string);
hive> insert into testdb.test1 values('Big','Data');
Esgyn, from terminal ... type
$ sqlcl
select count(*) from hive.testdb.test1;
```
Phoenix
Phoenix Features

- **SQL on Hbase** and lightweight.
- Converts SQL statements to series of Hbase API’s.
- Recently added **ACID support**
- Supports Schemas, **Composite Primary Key**, Parallel Scan, Joins (limited)
- Supports **Sequences**, Rowtimestamps
- Supports write to Hbase, read from Phoenix
- Supports **Statistics collection**
CREATE TABLE IF NOT EXISTS METRIC_RECORD (  
    METRIC_NAME VARCHAR,
    HOSTNAME VARCHAR,
    SERVER_TIME UNSIGNED_LONG NOT NULL
    METRIC_VALUE DOUBLE,
    ...
    CONSTRAINT pk PRIMARY KEY (METRIC_NAME, HOSTNAME, SERVER_TIME))
DATA_BLOCK_ENCODING='FAST_DIFF', TTL=604800, COMPRESSION='SNAPPY'
SPLIT ON ('a', 'k', 'm');

CREATE SEQUENCE my_schema.my_sequence START WITH 100 INCREMENT BY 2 CACHE 10;

UPSERT VALUES INTO my_table(id, col1, col2)
VALUES( NEXT VALUE FOR my_schema.my_sequence, 'foo', 'bar');
But..

- Phoenix *does not integrate with Hive*
- Phoenix *doesn’t support cross-row transactions yet.*
- *Query optimizer* and *join* mechanisms needs improvements
- *Secondary indexes* can get out of sync with the primary table
- *Multi-tenancy is constrained* as it uses a single HBase table.
Splice Machine
Overview
(not Apache but Open Source)
Splice Machine Key Features

- True ANSI SQL Engine with ACID support
- Offload OLTP and OLAP workloads
- Supports joins, transactions, constraints, stored procedures, window functions
- Leverages proven processing of Apache Derby
- Simplifies Lambda architecture using one interface
- Auto Workload isolation, OLAP goes to SPARK, OLTP to HBase
Simplified Lambda architecture
High Concurrency Support

Data Volume

- Data Warehouses
- MPP Databases
- In-Memory Analytics DBs

Concurrent Reads/Writes

- NoSQL DBs
- Oracle RAC DBs
- In-Memory OLTP DBs
- MySQL
# Splice Machine key commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Describe</td>
<td>Displays a description of a table or view.</td>
<td>splice&gt; describe myTable;</td>
</tr>
<tr>
<td>Elapsedtime</td>
<td>Enables or disables display of elapsed time for command execution.</td>
<td>splice&gt; elapsedtime on;</td>
</tr>
<tr>
<td>Execute</td>
<td>Executes an SQL prepared statement or SQL command string.</td>
<td>splice&gt; execute 'insert into myTable(id, val) values(?,?);'</td>
</tr>
<tr>
<td>Exit</td>
<td>Causes the command line interface to exit.</td>
<td>splice&gt; exit;</td>
</tr>
<tr>
<td>Export</td>
<td>Exports query results to CSV files.</td>
<td>splice&gt; EXPORT('/my/export/dir', null, null, null, null, null) SELECT a,b,sqrt(c) FROM join2 on t1.a=t2.a;</td>
</tr>
<tr>
<td>Help</td>
<td>Displays a list of the available commands.</td>
<td>splice&gt; help;</td>
</tr>
<tr>
<td>MaximumDisplayWidth</td>
<td>Sets the maximum displayed width for each column of results displayed by the command line interpreter.</td>
<td>splice&gt; maximumdisplaywidth 30;</td>
</tr>
<tr>
<td>Prepare</td>
<td>Creates a prepared statement for use by other commands.</td>
<td>splice&gt; prepare seeMenu as 'SELECT * FROM menu';</td>
</tr>
<tr>
<td>Remove</td>
<td>Removes a previously prepared statement.</td>
<td>splice&gt; remove seeMenu;</td>
</tr>
<tr>
<td>Run</td>
<td>Runs commands from a file.</td>
<td>splice&gt; run myCmdFile;</td>
</tr>
</tbody>
</table>

So let’s conclude
# SQL on Hadoop Feature Matrix

<table>
<thead>
<tr>
<th>Feature</th>
<th>Hive / HiveQL</th>
<th>Spark SQL</th>
<th>Drill</th>
<th>HAWQ</th>
<th>Phoenix</th>
<th>Trafodian</th>
<th>Splice Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key Use Cases</td>
<td>Batch / ETL / Long-running jobs</td>
<td>Batch / ETL / advanced analytics / ad-hoc queries</td>
<td>Self-Service Data Exploration Interactive BI / Ad-hoc queries</td>
<td>Interactive BI / Ad-hoc queries</td>
<td>OLTP on Hadoop</td>
<td>OLTP on Hadoop</td>
<td>OLTP and OLAP on Hadoop</td>
</tr>
<tr>
<td>SQL Support</td>
<td>HiveQL</td>
<td>HiveQL</td>
<td>ANSI SQL</td>
<td>ANSI SQL</td>
<td>ANSI SQL</td>
<td>ANSI SQL</td>
<td>ANSI SQL</td>
</tr>
<tr>
<td>ACID Support</td>
<td>Limited</td>
<td>None</td>
<td>None</td>
<td>ACID Support</td>
<td>ACID Support (new)</td>
<td>ACID Support</td>
<td>ACID Support</td>
</tr>
<tr>
<td>Metastore</td>
<td>Hcatalog</td>
<td>Hcatalog</td>
<td>Hcatalog</td>
<td>Hcatalog</td>
<td>HBase</td>
<td>Hcatalog + Custom</td>
<td>Hcatalog + Custom</td>
</tr>
<tr>
<td>Native to Hadoop</td>
<td>Yes, YARN + HDFS</td>
<td>Yes, YARN + HDFS</td>
<td>Requires Drillbits as separate process</td>
<td>YARN + HDFS (special format)</td>
<td>HBase</td>
<td>HBase</td>
<td>HBase &amp; SPARK</td>
</tr>
<tr>
<td>Latency</td>
<td>High</td>
<td>Medium (in memory)</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>Data Types Supported</td>
<td>Relational &amp; Limited complex types</td>
<td>Relational &amp; Limited complex types</td>
<td>Relational &amp; excellent schema less support</td>
<td>Relational only</td>
<td>Relational &amp; Limited complex types</td>
<td>Relational &amp; Limited complex types</td>
<td>Relational &amp; Limited complex types</td>
</tr>
</tbody>
</table>
Q&A

rnadipalli@zaloni.com
@ranadipa