Introduction to Hive

Do The Math

Chicago, IL
Bangalore, India
www.mu-sigma.com

December, 2013

Agenda

- Introduction to Hive
- Data Types and Data Units
- Hive - Data Definition Language
- Hive - Data Manipulation Language
- Executing Hive
- Functions- Windowing, Analytics and UDFs
- Compression and Indexing
- Hive Optimizations and Tuning
- Exercises

Introduction to Hive

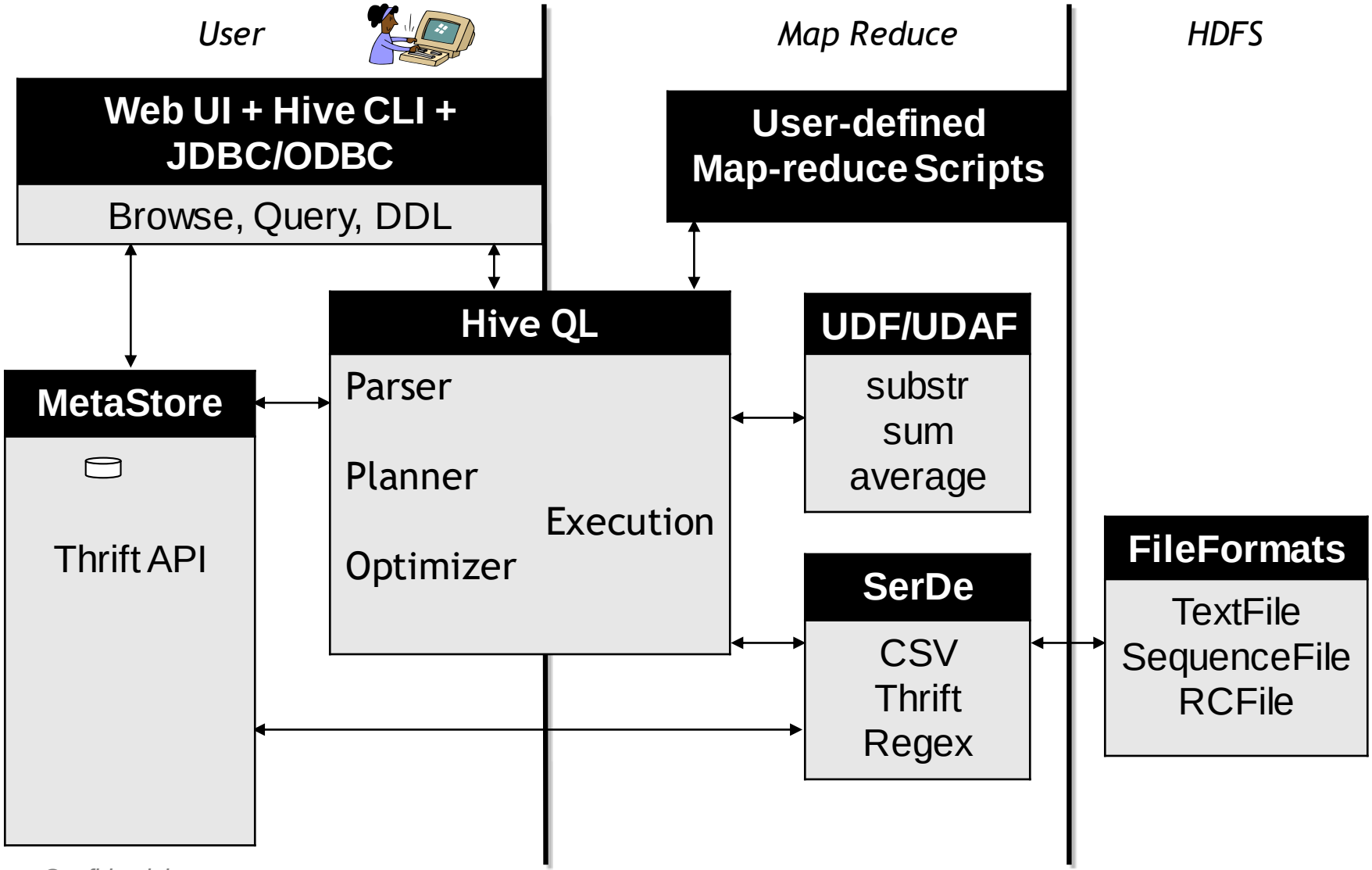
- Apache Hive is a data warehouse infrastructure built on top of Hadoop for providing data summarization, query, and analysis.
- While initially developed by Facebook, Apache Hive is now used and developed by other companies such as Netflix and Amazon
- Main features:
  - Apache Hive supports analysis of large datasets stored in Hadoop's HDFS and compatible file systems.
  - It provides an SQL-like language called HiveQL while maintaining full support for map/reduce. To accelerate queries, it provides indexes, including bitmap indexes.
  - By default, Hive stores metadata in an embedded Apache Derby database, and other client/server databases like MySQL can optionally be used.
  - Currently, in Hive 0.12.0(Oct , 2013) there are four file formats supported in Hive, which are TEXTFILE, SEQUENCEFILE, ORC and RCFILE.

Why do you need Hive?

- Hadoop is the future of enterprise data management and Hive is the gateway for business intelligence and visualization tools integrated with Hadoop.

- Here are some advantages of Hive:
  - Hundreds of unique users can simultaneously query the data using a language familiar to SQL users.
  - Familiar JDBC and ODBC drivers allow many applications to pull Hive data for seamless reporting. Hive allows users to read data in arbitrary formats, using SerDes and Input/Output formats.
  - Operating on compressed data stored into Hadoop ecosystem, algorithm including gzip, bzip2, snappy, etc.
  - Built-in user defined functions (UDFs) to manipulate dates, strings, and other data-mining tools. Hive supports extending the UDF set to handle use-cases not supported by built-in functions

How Hive relates to the Hadoop ecosystem



Hive Architecture

- **Metastore**: stores system catalog
- **Driver**: manages life cycle of HiveQL query as it moves through HIVE; also manages session handle and session statistics
- **Query compiler**: Compiles HiveQL into a directed acyclic graph of map/reduce tasks
- **Execution engines**: The component executes the tasks in proper dependency order; interacts with Hadoop
- **HiveServer**: provides Thrift interface and JDBC/ODBC for integrating other applications.
- **Client components**: CLI, web interface, jdbc/odbc inteface
- **Extensibility interface** include SerDe, User Defined Functions and User Defined Aggregate Function.

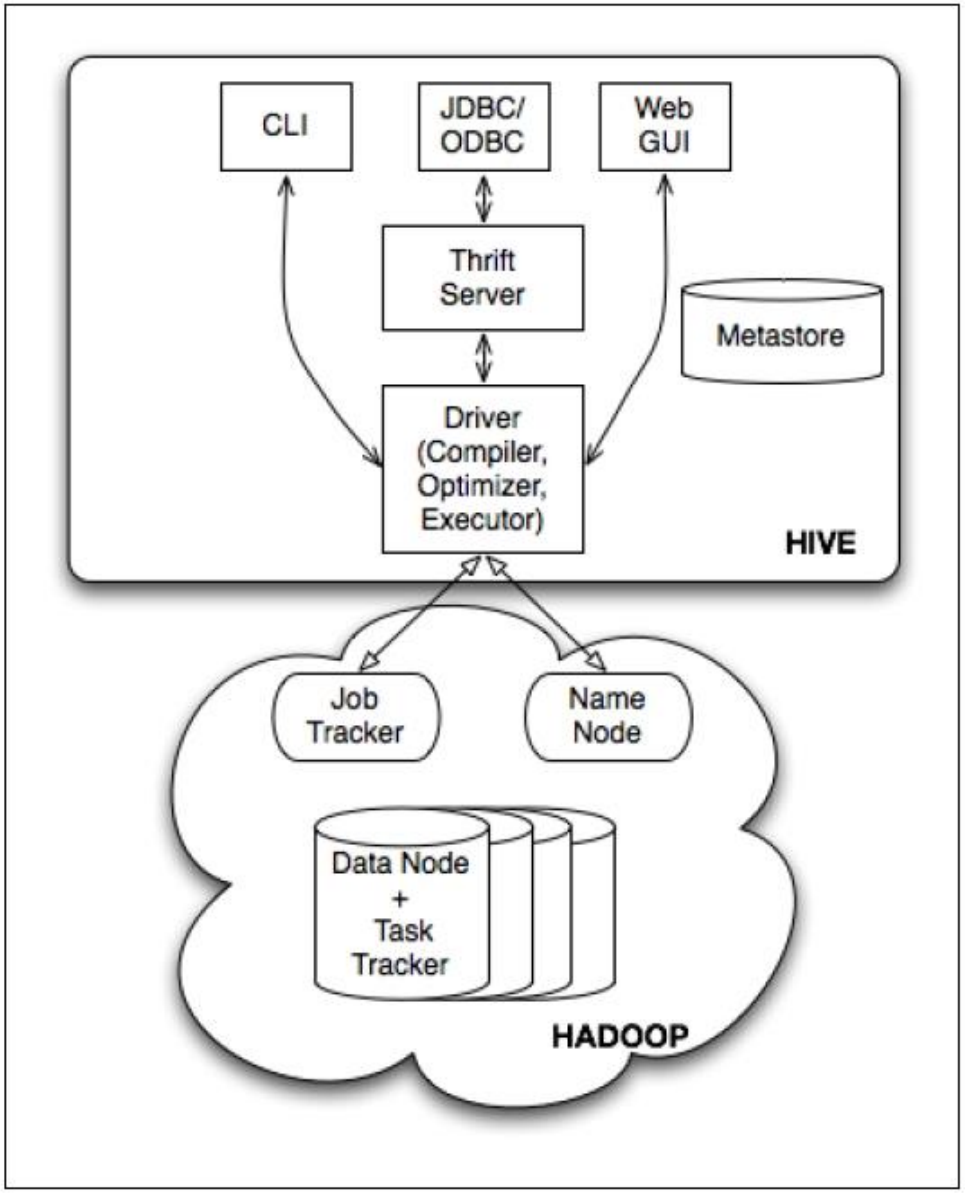


Image Courtesy: www.Cubrid.org

Hive Use cases

- **Log processing**
  - Call logs, web logs, machine logs – Big, repetitively collected Data
- **Text mining**
  - Unstructured textual data with a convenient structure overlaid and analyzed with map-reduce
- **Reporting and Adhoc Analysis**
  - Query transaction history, payment history – Querying historic data
  - Microstrategy reports
- **Machine Learning (Assembling training data)**
  - Ad Optimization – User engagement as a function of user attributes

Agenda

- Introduction to Hive
- **Data Types and Data Units**
- Hive - Data Definition Language
- Hive - Data Manipulation Language
- Executing Hive
- Functions- Windowing, Analytics and UDFs
- Compression and Indexing
- Hive Optimizations and Tuning
- Exercises

Hive Data Types

Hive structures data into well-understood database concepts such as tables, rows, columns, partitions

1. **Numeric Types**

   TINYINT, SMALLINT, INT, BIGINT, FLOAT, DOUBLE, DECIMAL

2. **Date/Time Types**

   TIMESTAMP, DATE

3. **String Types**

   STRING, VARCHAR

4. **Misc Type**

   BOOLEAN, BINARY

5. **Complex Types**

   - arrays: ARRAY<data_type>
   - maps: MAP<primitive_type, data_type>
   - structs: STRUCT<col_name : data_type [COMMENT col_comment], ...>
   - union: UNIONTYPE<data_type, data_type, ...>

Data Units in Hive

- In the order of granularity - Hive data is organized into:
  - **Databases**: Namespaces that separate tables and other data units from naming conflicts.
  - **Tables**: Homogeneous units of data which have the same schema.
    - » An example of a table could be page_views table, where each row could comprise of the following columns (schema):
      - timestamp - which is of INT type that corresponds to a unix timestamp of when the page was viewed.
      - userid - which is of BIGINT type that identifies the user who viewed the page.
      - page_url - which is of STRING type that captures the location of the page.
  - **Partitions**: Each Table can have one or more partition Keys which determines how the data is stored. Partitions - apart from being storage units - also allow the user to efficiently identify the rows that satisfy a certain criteria.
    - » For example, a date_partition of type STRING and country_partition of type STRING. Each unique value of the partition keys defines a partition of the Table.
  - **Buckets** (or Clusters): Data in each partition may in turn be divided into Buckets based on the value of a hash function of some column of the Table.
    - » For example the page_views table may be bucketed by userid, which is one of the columns, other than the partitions columns, of the page_view table. These can be used to efficiently sample the data.

Agenda

- Introduction to Hive
- Data Types and Data Units
- Hive - Data Definition Language
- **Hive - Data Manipulation Language**
- Executing Hive
- Functions- Windowing, Analytics and UDFs
- Compression and Indexing
- Hive Optimizations and Tuning
- Exercises

Hive Data Definition Language – CREATE TABLE

- The CREATE TABLE with all the options are shown
  -
```
CREATE [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]table_name
[(col_name data_type [COMMENT col_comment], ...)]   [COMMENT table_comment]
[PARTITIONED BY (col_name data_type [COMMENT col_comment], ...)]
[CLUSTERED BY (col_name, col_name, ...)
[SORTED BY (col_name [ASC|DESC], ...)] INTO num_buckets BUCKETS]
[SKEWED BY (col_name, col_name, ...) ON ([(col_value, col_value, ...),
...|col_value, col_value, ...])]
[    [ROW FORMAT row_format] [STORED AS file_format]    | STORED BY
'storage.handler.class.name' [WITH SERDEPROPERTIES (...)]  ]
[LOCATION hdfs_path]
[TBLPROPERTIES (property_name=property_value, ...)]
[AS select_statement]
```

- The important options in CREATING a table will be discussed in the coming slides.

Create table options

- **Skewed Tables**
  - This feature can be used to improve performance for tables where one or more columns have skewed values. By specifying the values that appear very often (heavy skew) Hive will split those out into separate files automatically and take this fact into account during queries so that it can skip (or include) whole files if possible.
  - `CREATE TABLE list_bucket_single (key STRING, value STRING) SKEWED BY (key) ON (1,5,6);`

- **Create Table As Select (CTAS)**
  - Tables can also be created and populated by the results of a query in one create-table-as-select (CTAS) statement. The table created by CTAS is atomic, meaning that the table is not seen by other users until all the query results are populated.
  - `CREATE TABLE new_key_value_store ROW FORMAT SERDE "org.apache.hadoop.hive.serde2.columnar.ColumnarSerDe" STORED AS RCFile AS SELECT (key % 1024) new_key, concat(key, value) key_value_pair FROM key_value_store SORT BY new_key, key_value_pair;`

- **Create Table Like**
  - The LIKE form of CREATE TABLE allows you to copy an existing table definition exactly (without copying its data).
  - `CREATE [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]table_name LIKE existing_table_or_view_name [LOCATION hdfs_path]`

Row Format and SerDe

- row_format
  - `DELIMITED [FIELDS TERMINATED BY char [ESCAPED BY char]] [COLLECTION ITEMS TERMINATED BY char]`
    `[MAP KEYS TERMINATED BY char] [LINES TERMINATED BY char]`

- *SerDe* is short for serializer/deserializer. A SerDe encapsulates the logic for converting the unstructured bytes in a *record*, which is stored as part of a file, into a record that Hive can use. SerDes are implemented using Java. Hive comes with several built-in SerDes and many other third-party SerDes are available.
  - `SERDE serde_name [WITH SERDEPROPERTIES (property_name=property_value, property_name=property_value, ...)]`

  - For reference:https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL

Data Storage Formats

- File formats are specified at the table (or partition) level. You can specify the ORC file format with Hive QL statements such as these:
  - `CREATE TABLE ... STORED AS <File_format>`
  - `ALTER TABLE ... [PARTITION partition_spec] SET FILEFORMAT <File_format>`
  - `SET hive.default.fileformat=<File_format>`

- **Text Files**
  - Normal storage takes place as Text by default the other formats are discussed below.

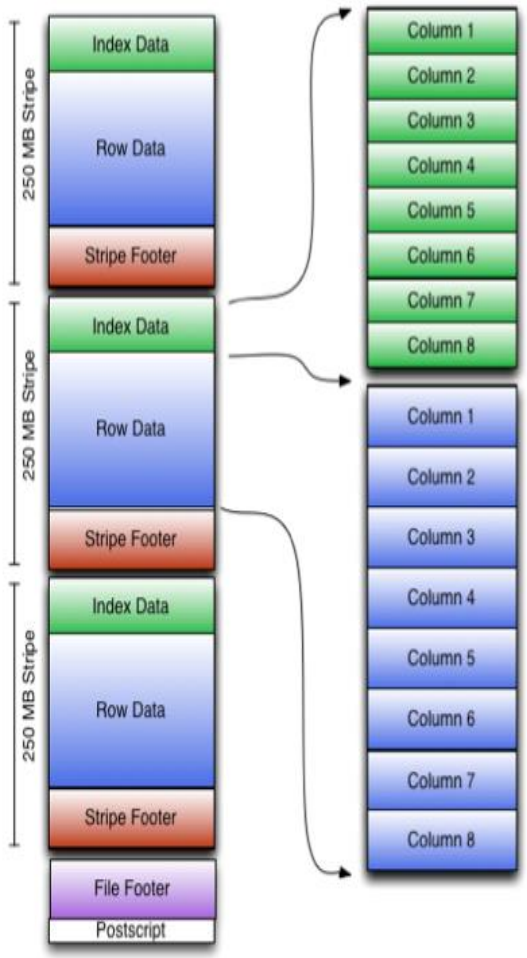
- **Sequence files**
  - They are flat files consisting of binary key-value pairs.
  - Sequence files have three different compression options: NONE, RECORD, and BLOCK.

```
<property>
<name>mapred.output.compression.type</name>
<value>BLOCK</value>
<description>If the job outputs are to compressed as SequenceFiles,how should
they be compressed? Should be one of NONE, RECORD or BLOCK.</description>
</property>
```

Data Storage Formats contd…

- **RC File**
  - Stores columns of a table in a record columnar way.
  - It first partitions rows horizontally into row splits. and then it vertically partitions each row split in a columnar way. RCFile first stores the meta data of a row split, as the key part of a record, and all the data of a row split as the value part.

- **ORC File**
  - The Optimized Row Columnar (ORC) file format provides a highly efficient way to store Hive data.
  - It was designed to overcome limitations of the other Hive file formats. Using ORC files improves performance when Hive is reading, writing, and processing data.



ORC File Format

Image Courtesy: https://cwiki.apache.org

Data Storage Formats contd…

- **Parquet File**
  - Parquet is a column-oriented binary file format intended to be highly efficient for the types of large-scale queries. Parquet is especially good for queries scanning particular columns within a table, for example to query "wide" tables with many columns, or to perform aggregation operations such as SUM() and AVG() that need to process most or all of the values from a column.
  - Queries against a Parquet table can retrieve and analyze these values from any column quickly and with minimal I/O.

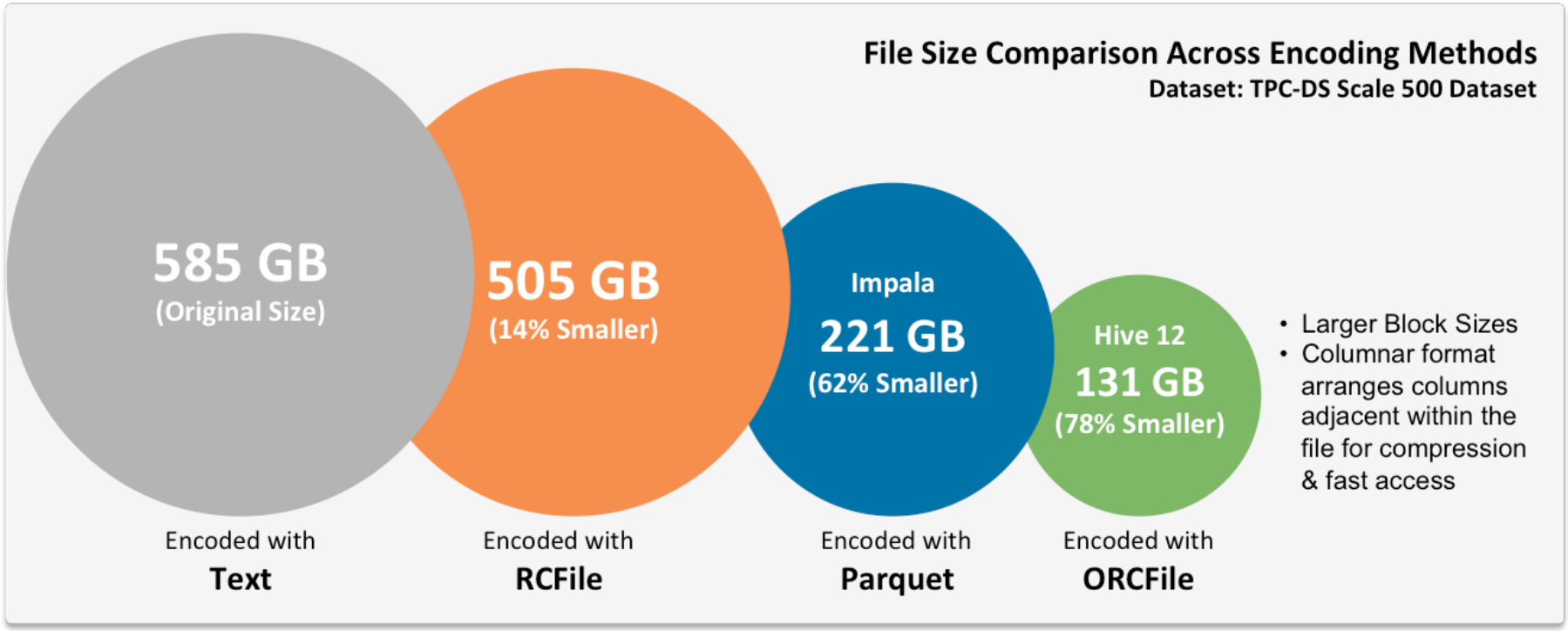


Image Courtesy: www.hortonworks.com

Hive Data Model – Partitions

- Analogous to dense indices on partition columns
- Nested sub-directories in HDFS for each combination of partition column values
- Partition column **must not already exist** in the table – can be used as a pseudo-column to query on
- `CREATE TABLE foo (id INT, msg STRING) PARTITIONED BY (dt STRING, ctry STRING);`
  - » Example
  - » Partition columns: `ds, ctry`
  - » HDFS subdirectory for ds = 20090801, ctry = US – `/wh/pvs/ds=20120201/ctry=US`
  - » HDFS subdirectory for ds = 20090801, ctry = CA – `/wh/pvs/ds=20120201/ctry=CA`
  - » Queries with partition columns in WHERE clause will scan through only a subset of the data that which exists in the specified partition

Hive Data Model - Dynamic Partitions

- Dynamic Partition (DP) columns are specified the same way as it is for Static Partitioned (SP) columns – in the partition clause.
- The only difference is that DP columns do not have values, while SP columns do. In the partition clause, we need to specify all partitioning columns, even if all of them are DP columns.
- In INSERT ... SELECT ... queries, the dynamic partition columns must be specified last among the columns in the SELECT statement and in the same order in which they appear in the PARTITION() clause.
- Examples:
  - ```
    INSERT OVERWRITE TABLE T PARTITION (ds, hr)
    SELECT key, value, ds, hr FROM srcpart WHERE ds is not null and hr>10;
    ```
  - ```
    CREATE TABLE T (key int, value string) PARTITIONED BY (ds string, hr int) AS
    SELECT key, value, ds, hr+1 hr1 FROM srcpart WHERE ds is not null and hr>10;
    ```

Hive Data Model – Buckets

- Split data based on hash of a column – mainly for parallelism
- One HDFS file per bucket within partition sub-directory
- `CREATE TABLE user_info_bucketed(user_id BIGINT, firstname STRING, lastname STRING) COMMENT 'A bucketed copy of user_info' PARTITIONED BY(ds STRING, ctry STRING) CLUSTERED BY(user_id) INTO 20 BUCKETS;`
- Example
  - Bucket column `user_id` into 20 buckets
  - HDFS file for `user_id` hash bucket 0 – `/wh/pvs/ds=20120201/ctry=US/part-00000`
  - HDFS file for `user_id` hash bucket 20 – `/wh/pvs/ds=20120201/ctry=US/part-00020`

External tables

- Point to existing data directories in HDFS
- Can create tables and partitions – partition columns just become annotations to external directories
- Example: create external table with partitions
  -
```
CREATE EXTERNAL TABLE pvs(userid int, pageid int, ds string, ctry string)
PARTITIONED ON (ds string, ctry string)
STORED AS textfile
LOCATION '/path/to/existing/table'
```
- Example: add a partition to external table
  -
```
ALTER TABLE pvs ADD PARTITION (ds='20120201', ctry='US')
LOCATION '/path/to/existing/partition'
```

Hbase Tables from Hive

- HBase tables from Hive
  - Use the HBaseStorageHandler to register HBase tables with the Hive metastore. You can optionally specify the HBase table as EXTERNAL, in which case Hive will not create to drop that table directly – you'll have to use the HBase shell to do so.
  - Schema mapping-As part of that registration, you also need to specify a column mapping. This is how you link Hive column names to the HBase table's rowkey and columns. Do so using the hbase.columns.mapping SerDe property.

- ```
  CREATE [EXTERNAL] TABLE hive_table(...)
  STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
  TBLPROPERTIES ('hbase.table.name' = 'hbase_table');
  CREATE TABLE foo(rowkey STRING, a STRING, b STRING)
  STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
  WITH SERDEPROPERTIES ('hbase.columns.mapping' = ':key,f:c1,f:c2')
  TBLPROPERTIES ('hbase.table.name' = 'bar');
  ```
  - The above statement registers the HBase table named hbase_table in the Hive metastore, accessible from Hive by the name hive_table.
  - The values provided in the mapping property correspond one-for-one with column names of the hive table. HBase column names are fully qualified by column family, and you use the special token :key to represent the rowkey.
  ```

Hive Data Definition Language contd…

- Browsing through Tables
  - `hive> SHOW TABLES;`
  - `hive> SHOW TABLES '.*s';`
- Altering and Dropping Tables
  - Table names can be changed and columns can be added or replaced:
  - `hive> ALTER TABLE events RENAME TO Days;`
  - `hive> ALTER TABLE pokes ADD COLUMNS (new_col INT);`
- Dropping tables:
  - `hive> DROP TABLE pokes;`

Hive Data Manipulation Language.

- Loading data from flat files into Hive:
  - `hive> LOAD DATA LOCAL INPATH './examples/files/kv1.txt' OVERWRITE INTO TABLE pokes;`
- The two LOAD statements below load data into two different partitions of the table invites. Table invites must be created as partitioned by the key ds for this to succeed.
  - `hive> LOAD DATA LOCAL INPATH './examples/files/kv2.txt' OVERWRITE INTO TABLE invites PARTITION (ds='2008-08-15');`
  - `hive> LOAD DATA LOCAL INPATH './examples/files/kv3.txt' OVERWRITE INTO TABLE invites PARTITION (ds='2008-08-08');`
- The above command will load data from an HDFS file/directory to the table.
  - `hive> LOAD DATA INPATH '/user/myname/kv2.txt' OVERWRITE INTO TABLE invites PARTITION (ds='2008-08-15');`

Hive Data Manipulation Language.

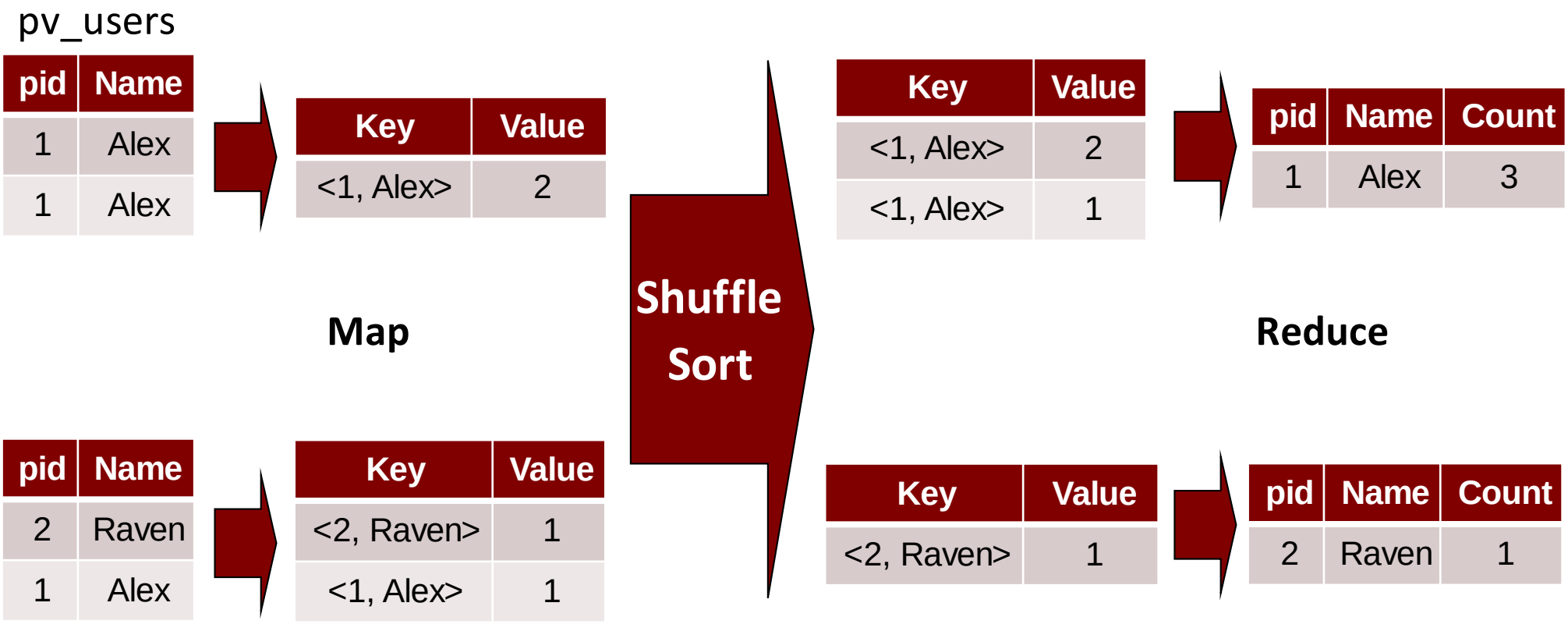
- Inserting Data into Tables from Queries
  - The INSERT statement lets you load data into a table from a query.
  - ```
    INSERT OVERWRITE TABLE employees PARTITION (country = 'US', state = 'OR')
    SELECT * FROM staged_employees se
    WHERE se.cnty = 'US' AND se.st = 'OR';
    ```
- Exporting Data
  - If the data files are already formatted the way you want, then it's simple enough to copy the directories or files:
  - ```
    hadoop fs -cp source_path target_path
    ```
  - Otherwise, you can use INSERT ... DIRECTORY ..., as in this example:
  - ```
    INSERT OVERWRITE LOCAL DIRECTORY '/tmp/ca_employees' SELECT name, salary,
    address
    FROM employees
    WHERE se.state = 'CA';
    ```
- SerDe: API for serialization and deserialization to be used to move data in and out of tables
    ```

Hive query language – Group by

- Efficient execution plans based on:
  - Data skew:
    - how evenly distributed is data across a number of physical nodes
    - bottleneck vs. load balance
- Partial aggregation:
  - Group the data with the same group by value as soon as possible
    - In memory hash-table for mapper
    - Earlier than combiner
- Example:

```
SELECT pid, name, count(1)
FROM pv_users
GROUP BY pid, name;
```

Hive query language – Group by illustration



Hive query language – Joins

- Traditional Map-Reduce Join
  - Data from each table is read into the mapper which outputs rows
  - Mapper output keys are the values of the join key
  - Reducers collects values associated with each join key
- Early Map-side Join – very efficient for joining a small table with a large table
  - Keep smaller table data in memory first
  - Join with a chunk of larger table data each time
  - Space complexity for time complexity
- Example:

```
INSERT OVERWRITE TABLE pv_users
SELECT pv.pid, u.name
FROM page_views pv
JOIN user u
ON (pv.uid = u.id);
```

Hive query language – Joins illustration

page_view

pid	uid	ctry
1	111	UK
2	111	US
1	222	India

user

id	Name	gender
111	Alex	female
222	Raven	male

Map

key	Value
111	<**1**, 1>
111	<**1**, 2>
222	<**2**, 1>

key	Value
111	<**1**, Alex>
222	<**2**, Raven>

Shuffle Sort

key	Value
111	<**1**, 1>
111	<**1**, 2>
111	<**1**, Alex>

key	Value
222	<**2**, 1>
222	<**2**, Raven>

Reduce

pv_users

pid	Name
1	Alex
2	Alex

pid	Name
1	Raven

Join Optimisation

- Consider the query

```
SELECT a.ymd, a.price_close, b.price_close , c.price_close
FROM stocks a JOIN stocks b ON a.ymd = b.ymd
              JOIN stocks c ON a.ymd = c.ymd
WHERE a.symbol = 'AAPL' AND b.symbol = 'IBM' AND c.symbol = 'GE';
```

- Most of the time, Hive will use a separate MapReduce job for each pair of things to join. In this example, it would use one job for tables a and b, then a second job to join the output of the first join with c.

- Hive can apply an optimization where it joins all three tables in a single MapReduce job. If every ON clause uses the same join key, a single MapReduce job will be used.

- Hive also assumes that the last table in the query is the largest. It attempts to buffer the other tables and then stream the last table through, while performing joins on individual records. Therefore, you should structure your join queries so the largest table is last.

Join Optimisations - Map side joins

- If all but one table is small, the largest table can be streamed through the mappers while the small tables are cached in memory. Hive can do all the joining map-side, since it can look up every possible match against the small tables in memory, thereby eliminating the reduce step required in the more common join scenarios.
- Even on smaller data sets, this optimization is noticeably faster than the normal join. Not only does it eliminate reduce steps, it sometimes reduces the number of map steps, too.
- However, you still have to set a property, `hive.auto.convert.join`, to true before Hive will attempt the optimization
- you can also configure the threshold size for table files considered small enough to use this optimization. Here is the default definition of the property (in bytes):
  - `hive.mapjoin.smalltable.filesize=25000000`

Agenda

- Introduction to Hive
- Data Types and Data Units
- Hive - Data Definition Language
- Hive - Data Manipulation Language
- **Executing Hive**
- Functions- Windowing, Analytics and UDFs
- Compression and Indexing
- Hive Optimizations and Tuning
- Exercises

Executing Hive – Command-line interface*

- Start a terminal and run:

```
$ hive
```

- You should see a prompt like:

```
hive>
```

- Set a Hive or Hadoop conf prop:

```
hive> set propkey=value;
```

- List all properties and values:

```
hive> set -v;
```

- Add a resource to the DCache:

```
hive> add [ARCHIVE|FILE|JAR]
    filename;
```

- Run a HiveQL script:

```
hive> source hive_script.ql;
```

***A web UI exists to run Hive scripts, but not mature for enterprise use**

Executing Hive – Command-line interface

- List tables:

```
hive> show tables;
```

- Describe a table:

```
hive> describe <tablename>;
```

- More information:

```
hive> describe extended <tablename>;
```

- List Functions:

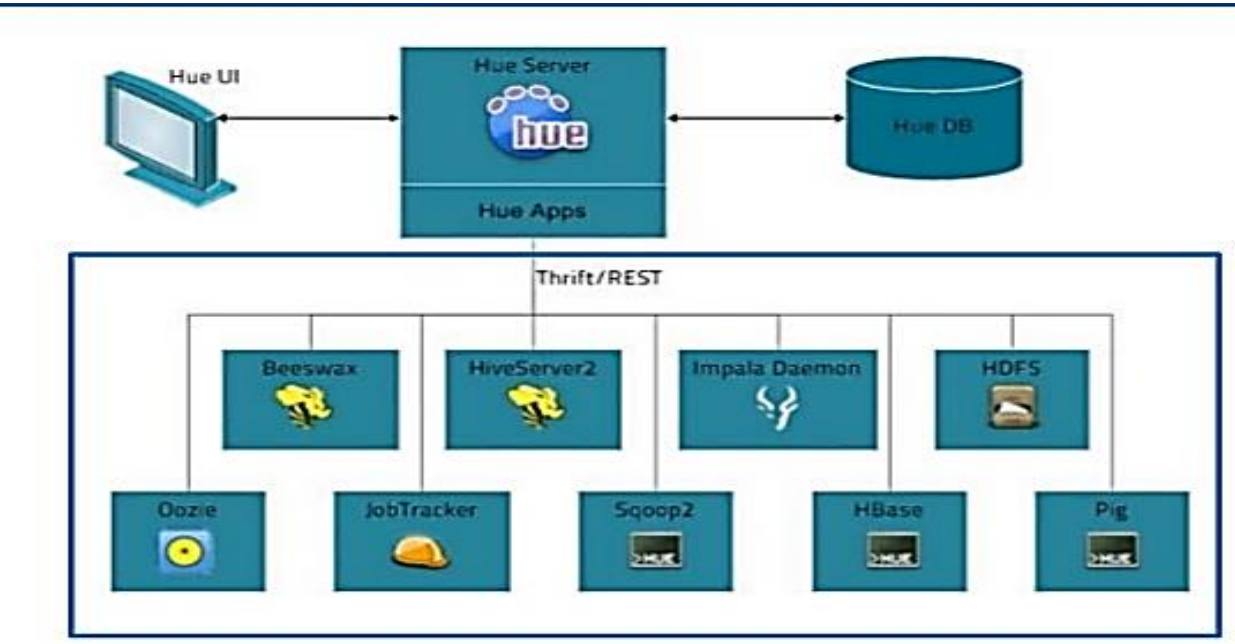
```
hive> show functions;
```

- More information:

```
hive> describe function <functionname>;
```

Hue, the open source Apache Hadoop UI

- Hue is an open-source Web interface that supports Apache Hadoop and its ecosystem
- Hue features a File Browser for HDFS, a Job Browser for MapReduce/YARN, an HBase Browser, query editors for Hive, Pig, Cloudera Impala and Sqoop2.
  - It also ships with an Oozie Application for creating and monitoring workflows, a Zookeeper Browser and a SDK.
- Beeswax Hive UI
  - The Beeswax application enables you to perform queries on Apache Hive. You can create Hive tables, load data, create, run, and manage queries and download the results in a Microsoft Office Excel worksheet file or a comma-separated values file.



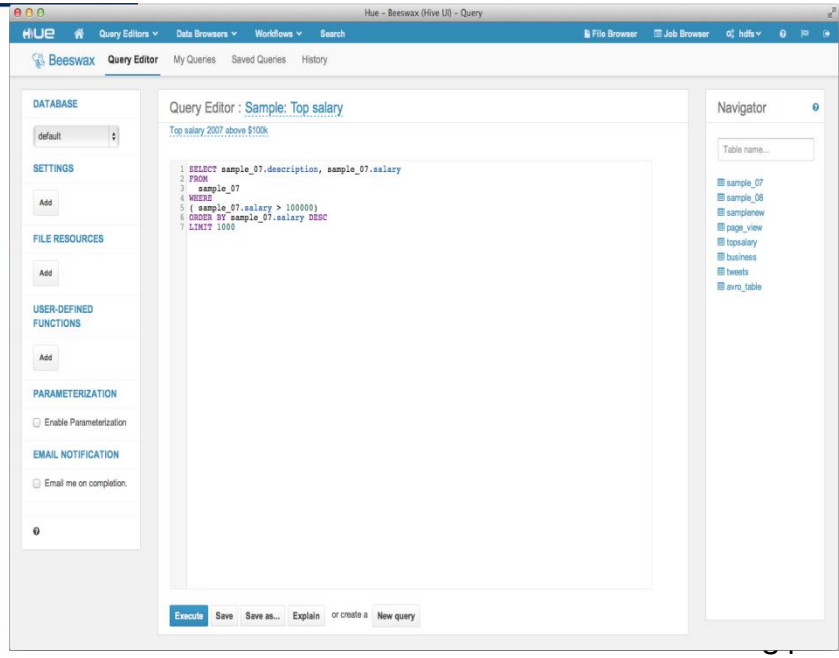


Image Courtesy: http://cloudera.github.io/hue/

Executing Hive Scripts – script to execute

```
-- hive_replace_null_values.ql - Load data from CSV and replace NULL Values
-- Create the Customer table
    CREATE TABLE cust(id INT, name STRING, amount DOUBLE) ROW FORMAT DELIMITED
    FIELDS TERMINATED BY ',' STORED AS TEXTFILE;
-- Load the data from the local csv file (will overwrite any existing values)
    LOAD DATA LOCAL INPATH
"/home/hadoop/Documents/Hive/IntroductionToHive/Datasets/HiveCustData.csv"
OVERWRITE INTO TABLE cust;

-- Create a table to store the mean value of the amount
    CREATE TABLE cust_mean(mean DOUBLE, key_id INT);
-- Compute the mean value and store in the table
    INSERT OVERWRITE TABLE cust_mean SELECT avg(amount),1 FROM cust;
-- Add key_id column to the cust table also
    ALTER TABLE cust  ADD COLUMNS(key_id INT);
-- Insert the key_id value for all the rows in the table
    INSERT OVERWRITE TABLE cust  SELECT cust.id, cust.name, cust.amount, 1 FROM
    cust;
-- Now find the NULL values and replace them
    INSERT OVERWRITE TABLE cust SELECT cust.id, cust.name, CASE WHEN cust.amount
    IS NULL THEN cust_mean.mean ELSE cust.amount END AS amount, 1 FROM cust  JOIN
    cust_mean ON (cust_mean.key_id = cust.key_id);
```

Windowing and Analytics Functions

- Windowing functions
  - LEAD
  - LAG
- The OVER clause
  - OVER with standard aggregates: COUNT, SUM, MIN, MAX, AVG
  - OVER with a PARTITION BY statement with one or more partitioning columns of any primitive datatype.
  - OVER with PARTITION BY and ORDER BY with one or more partitioning and/or ordering columns of any datatype.
  - OVER with a window specification. Window specifications support these standard options:
    - » SELECT a, AVG(b) OVER (PARTITION BY c ORDER BY d ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING) FROM T;
- Analytics functions
  - RANK
  - ROW_NUMBER
  - DENSE_RANK
  - CUME_DIST
  - PERCENT_RANK
  - NTILE

User-defined functions in Hive

- UDFs can be defined in any language
  - However, UDFs in languages other than Java are implemented using Hadoop streaming and aren't as efficient
- Hive supports implicit type conversions
  - However, it is more efficient to use Hadoop writables
- UDF can be overloaded (can be defined for different input types)
- UDF can take arguments of variable length, just as in Java
  - For example, a UDF to sum or multiply should be able to do so for any number of numbers

A simple UDF example – log transformation

```
add jar hive-udfs.jar;
CREATE TEMPORARY FUNCTION logval AS
'com.mu_sigma.hive.udf.UDFLogValue';
SELECT logval(3.14) FROM src;
SELECT logval(5) FROM src;
DROP TEMPORARY FUNCTION logval;
```

- UDFLogValue.java:

```
package com.mu_sigma.hive.udf;
public class UDFLogValue extends UDF {
    /* Evaluate log of Double values */
    public Double evaluate(Double s) {
            if (s == null)return null;
            return java.lang.Math.log(s);
    }
    /* Evaluate log of Integer values */
    public Double evaluate(Integer s) {
            if (s == null) return null;
            return java.lang.Math.log(Double.valueOf(s.toString()));
    }
}
```

Compression in Hive

- Using compression has the advantage of minimizing the disk space required for files and the overhead of disk and network I/O. Compression is best used for I/O-bound jobs, where there is extra CPU capacity, or when disk space is at a premium.

- The codecs available are in a comma-separated list named io.compression.codec:
  - `# hive -e "set io.compression.codecs"`
    `io.compression.codecs=org.apache.hadoop.io.compress.GzipCodec,`
    `org.apache.hadoop.io.compress.DefaultCodec,`
    `org.apache.hadoop.io.compress.BZip2Codec,`
    `org.apache.hadoop.io.compress.SnappyCodec`

- Enabling Intermediate Compression
  - hive.exec.compress.intermediate
    - » This controls whether intermediate files produced by Hive between multiple map-reduce jobs are compressed. The compression codec and other options are determined from hadoop config variables mapred.output.compress

- Final Output Compression
  - hive.exec.compress.output
    - » This controls whether the final outputs of a query (to a local/hdfs file or a Hive table) is compressed. The compression codec and other options are determined from hadoop config variables mapred.output.compress

Indexing in Hive

- CREATE INDEX creates an index on a table using the given list of columns as keys
- Indexing is to improve the speed of query lookup on certain columns of a table. Without an index, queries with predicates like 'WHERE tab1.col1 = 10' load the entire table or partition and process all the rows. But if an index exists for col1, then only a portion of the file needs to be loaded and processed.
- Hive has limited indexing capabilities. There are no keys in the usual relational database sense, but you can build an index on columns to speed some operations. The index data for a table is stored in another table.index on the country partition only:
  - ```
    CREATE INDEX employees_index
    ON TABLE employees (country)
    AS 'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler'
    WITH DEFERRED REBUILD
    IDXPROPERTIES ('creator = 'me', 'created_at' = 'some_time')
    IN TABLE employees_index_table
    PARTITIONED BY (country, name)
    COMMENT 'Employees indexed by country and name.';
    ```
  - ```
    DROP INDEX [IF EXISTS] index_name ON table_name
    ```

Hive Optimisations

- Avoid Nulls in Joins
- Use Map Side Joins
- CLUSTER BY
  - Using DISTRIBUTE BY ... SORT BY or the shorthand CLUSTER BY clauses is a way to exploit the parallelism of SORT BY, yet achieve a total ordering across the output files.
  - `hive> SELECT s.ymd, s.symbol, s.price_close FROM stocks s CLUSTER BY s.symbol;`
- Replicating RANK() over PARTITION BY
- Optimise GROUP Bys
  - `hive.optimize.groupby` -Whether to enable the bucketed group by from bucketed partitions/tables.
- ORDER BY and SORT BY
  - The ORDER BY clause is familiar from other SQL dialects. It performs a total ordering of the query result set. This means that all the data is passed through a single reducer, which may take a long time.
  - Hive adds an alternative, SORT BY, that orders the data only within each reducer, thereby performing a local ordering, where each reducers output will be sorted. Better performance is traded for total ordering.

Optimize Auto Join Conversion

- When auto join is enabled, there is no longer a need to provide the map-join hints in the query. The auto join option can be enabled with two configuration parameters:
  - ```
    set hive.auto.convert.join.noconditionaltask = true;
    set hive.auto.convert.join.noconditionaltask.size = 10000;
    ```
  - The default for hive.auto.convert.join.noconditionaltask is false which means auto conversion is disabled.
  - The size configuration enables the user to control what size table can fit in memory. This value represents the sum of the sizes of tables that can be converted to hashmaps that fit in memory.
    ```

Auto Conversion to SMB Map Join

- Sort-Merge-Bucket (SMB) joins can be converted to SMB map joins as well. SMB joins are used wherever the tables are sorted and bucketed.

- The join boils down to just merging the already sorted tables, allowing this operation to be faster than an ordinary map-join. However, if the tables are partitioned, there could be a slow down as each mapper would need to get a very small chunk of a partition which has a single key.

- The following configuration settings enable the conversion of an SMB to a map-join SMB:
  -
```
set hive.auto.convert.sortmerge.join=true;
set hive.optimize.bucketmapjoin = true;
set hive.optimize.bucketmapjoin.sortedmerge = true;
set hive.auto.convert.sortmerge.join.noconditionaltask=true;
```

Queries that Sample Data

- For very large data sets, sometimes you want to work with a representative sample of a query result, not the whole thing. Hive supports this goal with queries that sample tables organized into buckets.

- In the following example, assume the numbers table has one number column with values 1-10.
  - `hive> SELECT * from numbers TABLESAMPLE(BUCKET 3 OUT OF 10 ON rand()) s;`
    `2`
    `4`
  - `hive> SELECT * from numbers TABLESAMPLE(BUCKET 3 OUT OF 10 ON rand()) s;`
    `7`
    `10`

Tuning – Controlling File and Split Size

- Determining the optimal number of mappers and reducers depends on many variables, such as the size of the input and the operation being performed on the data
- In order to change the average load for a reducer (in bytes):
  - `set hive.exec.reducers.bytes.per.reducer=<number>`
- In order to limit the maximum number of reducers:
  - `set hive.exec.reducers.max=<number>`
- In order to set a constant number of reducers:
  - `set mapred.reduce.tasks=<number>`
- Hive determines the number of reducers from the input size. This can be confirmed using the `dfs -count` command
- It is a good idea to set this value in your `$HIVE_HOME/conf/hive-site.xml`. A suggested formula is to set the no of reducers to the result of calculation:
  - (Total Cluster Reduce Slots * 1.5) / (avg number of queries running)
    - » The 1.5 multiplier is a fudge factor to prevent underutilization of the cluster.

Tuning - Limit

- Hive has a configuration property to enable sampling of source data for use with LIMIT:

```
<property>
<name>hive.limit.optimize.enable</name>
<value>true</value>
<description>Whether to enable to optimization to try a smaller subset of data
for simple LIMIT first.</description>
</property>
```

- Once the hive.limit.optimize.enable is set to true, two variables control its operation,
  - `hive.limit.row.max.size`: When trying a smaller subset of data for simple LIMIT ,how much size we need to guarantee each row to have at least.
  - `hive.limit.optimize.limit.file`: When trying a smaller subset of data for simple LIMIT ,maximum number of files we can sample

Tuning – Parallel Execution

- If a job is running more stages in parallel, it will increase its cluster utilization
- Hive converts a query into one or more stages. Stages could be a MapReduce stage, a sampling stage, a merge stage, a limit stage, or other possible tasks Hive needs to do.
- By default, Hive executes these stages one at a time. However, if more stages are run simultaneously, the job may complete much faster.
- Setting the property `hive.exec.parallel` to true enables parallel execution.

```
<property>
<name>hive.exec.parallel</name>
<value>true</value>
<description>Whether to execute jobs in parallel</description>
</property>
```

- You can use another property `hive.exec.parallel.thread.number` to control how many jobs at most can be executed in parallel.

Tuning – Dynamic Partitioning

- It is good to set the dynamic partition mode to strict in your hive

```
<property>
<name>hive.exec.dynamic.partition.mode</name>
<value>strict</value>
<description>In strict mode, the user must specify at least one static partition
in case the user accidentally overwrites all partitions.</description>
</property>
```

- Then, increase the other relevant properties to allow queries that will create a large number of dynamic partitions
  - `hive.exec.max.dynamic.partitions`- Maximum number of dynamic partitions allowed to be created in total
  - `hive.exec.max.dynamic.partitions.pernode`-Maximum number of dynamic partitions allowed to be created in each mapper/reducer node

Future work in Hive

- **Stinger, Interactive query for Apache Hive**

- The Stinger Initiative is a broad, community-based effort to drive the future of Apache Hive, delivering 100x performance improvements at petabyte scale with familiar SQL semantics.

- Project Goals
  - **Speed**: Deliver interactive query through performance increases as compared to Hive 10.
  - **Scale**: The only SQL interface to Hadoop designed for queries that scale from Terabytes to Petabytes.
  - **SQL**: Support the broadest array of SQL semantics for analytic applications running against Hadoop.

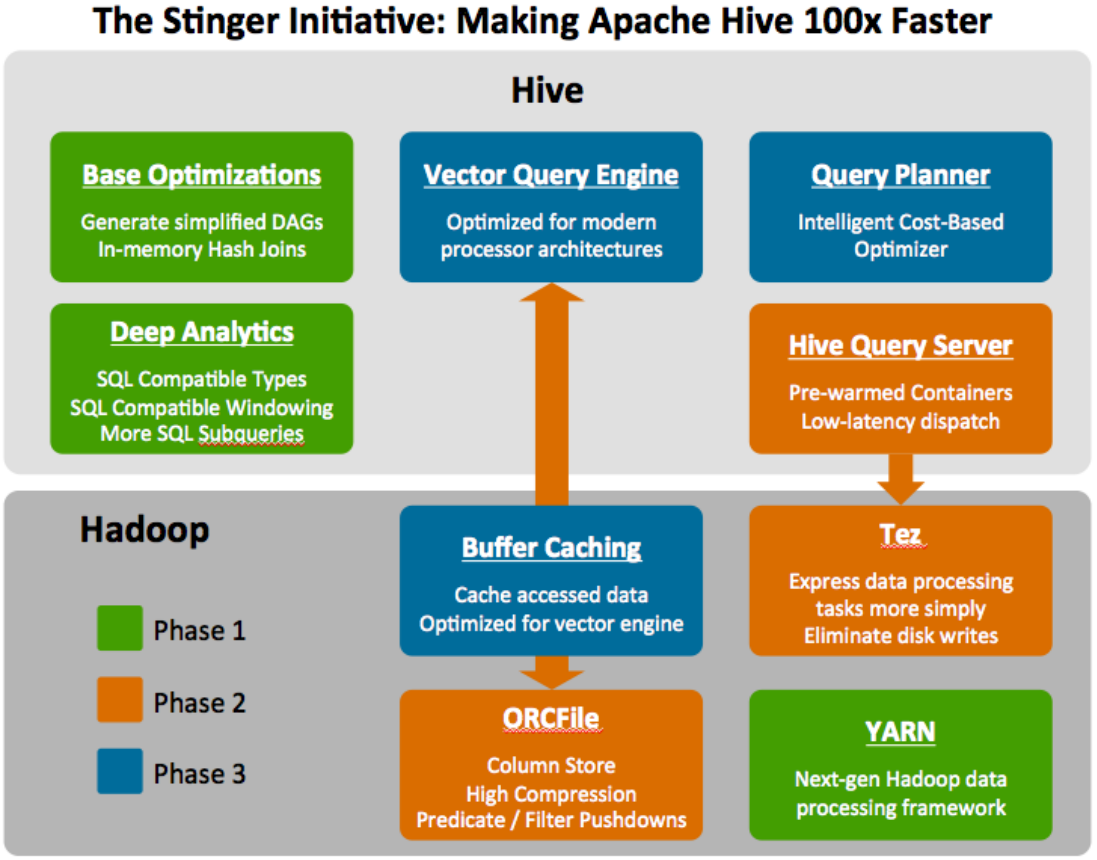
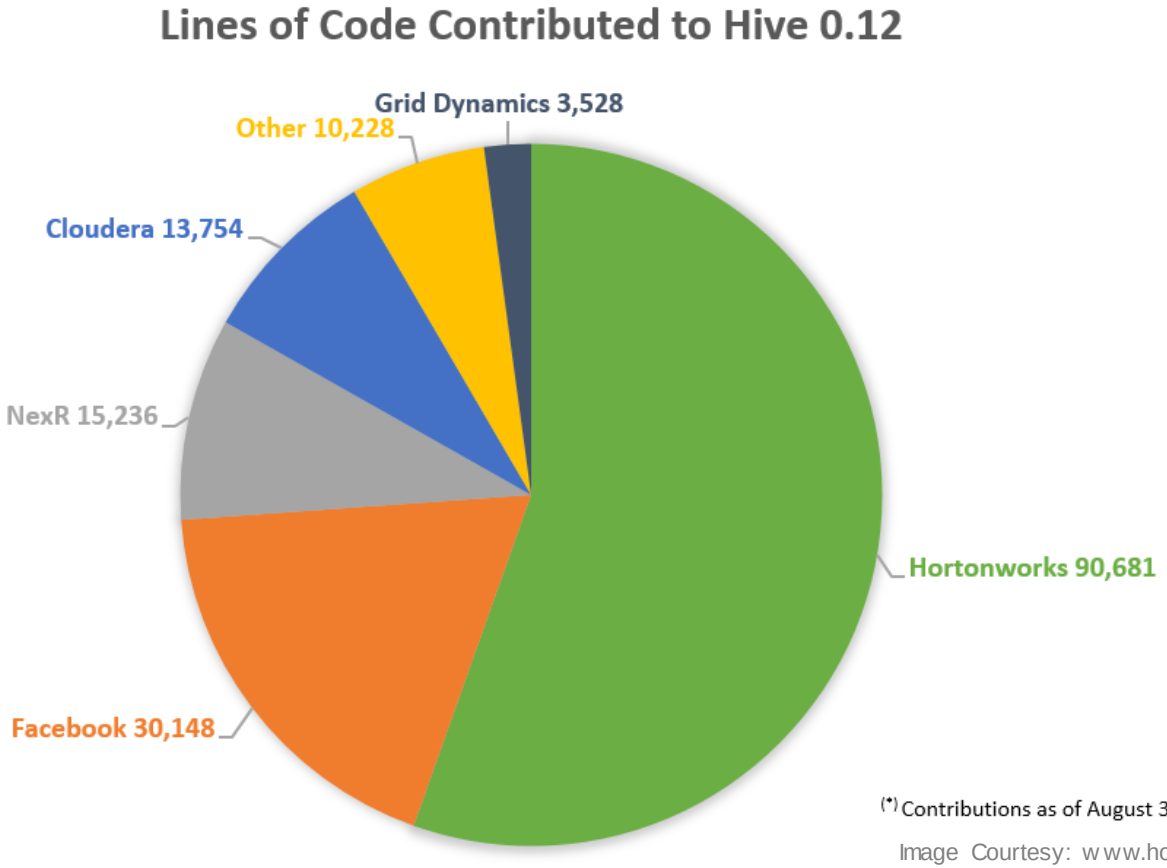


Image Courtesy: www.hortonworks.com

The Hive Ecosystem: Stronger Than Ever

- The philosophy is to innovate SQL in Hadoop with 100% vendor-neutral Apache Foundation software that welcomes contributions from anyone.
- Although Hive 12 is not officially released yet, a lot of work has been flowing in from more than 60 developers from many contributors in the community.



(*) Contributions as of August 30, 2013

Image Courtesy: www.hortonworks.com

Exercises – Basic Hive querying

- Using the datasets provided create the `page`, `user`, and `page_views` tables in Hive
  - The `user` dataset has the fields `id`, `name`, and `email`
  - The `page` dataset has the fields `id`, and `name`
  - The `page_views` dataset has the fields `user_id`, `page_id`, `date`, and `country`
- Using `JOIN` and `GROUP BY` clauses, find the total number of users who has visited all the pages from each country
- The output should be in the following format:

```
Canada      46
Belgium     54
…….         ….
```

Exercises – Hive UDFs

- Try to implement UDF for sine, cosine, tangent of a value (both integer and double)
  - Hint: Use `java.math.Math` for the functions
- Create a sample table by loading data from a CSV and use the above UDF on its values

Appendix

- Hive Data Types and Semantics

Hive Data Types and Semantics

Hive SQL Datatypes
INT
TINYINT/SMALLINT/BIGINT
BOOLEAN
FLOAT
DOUBLE
STRING
TIMESTAMP
BINARY
DECIMAL
ARRAY, MAP, STRUCT, UNION
DATE
VARCHAR
CHAR

Hive SQL Semantics
SELECT, INSERT
GROUP BY, ORDER BY, SORT BY
JOIN on explicit join key
Inner, outer, cross and semi joins
Sub-queries in FROM clause
ROLLUP and CUBE
UNION
Windowing Functions (OVER, RANK, etc)
Custom Java UDFs
Standard Aggregation (SUM, AVG, etc.)
Advanced UDFs (ngram, Xpath, URL)
Sub-queries for IN/NOT IN, HAVING
INTERSECT / EXCEPT
Expanded JOIN Syntax

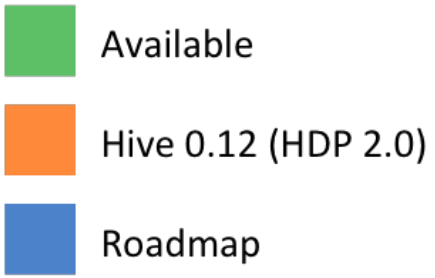


Image Courtesy: www.hortonworks.com

Thank You

Chicago, IL
Bangalore, India
December, 2013
www.mu-sigma.com