What is Hive?

- Subproject of Apache Hadoop
- Originally created at Facebook
- Provides an easier and faster way to analyze data in Hadoop (compared to MapReduce in Java, Python, C++, etc)
The problem with MapReduce

• The developer has to worry about a lot of things besides the analysis/processing logic
• Often requires several MapReduce passes to accomplish a final

• The data is schema-less
• Would be more convenient to use constructs such as "filter", "join", "aggregate"
What does Hive provide?

- A parser/optimizer/compiler that translates HiveQL to MapReduce code
- A Metastore (usually a MySQL server) that stores the "schema" information
  - Table name, column names, data types
  - Partition information
  - Storage location, row format (SerDe), storage format (Input and OutputFormat)
Despite SQL-like language, Hive is NOT an RDBMS

<table>
<thead>
<tr>
<th></th>
<th>RDBMS</th>
<th>Hive</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Language</strong></td>
<td>SQL-92 standard (maybe)</td>
<td>Subset of SQL-92 plus Hive-specific extension</td>
</tr>
<tr>
<td><strong>Update Capabilities</strong></td>
<td>INSERT, UPDATE, and DELETE</td>
<td>INSERT OVERWRITE; no UPDATE or DELETE</td>
</tr>
<tr>
<td><strong>Transactions</strong></td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td><strong>Latency</strong></td>
<td>Sub-second</td>
<td>Minutes or more</td>
</tr>
<tr>
<td><strong>Indexes</strong></td>
<td>Any number of indexes, very important for performance</td>
<td>No indexes, data is always scanned (in parallel)</td>
</tr>
<tr>
<td><strong>Data size</strong></td>
<td>TBs</td>
<td>PBs</td>
</tr>
</tbody>
</table>
Getting started is easy!

1. Setup Hadoop (NameNode, DataNodes, JobTracker and TaskTrackers)
2. Create Hive tables
3. Load data into Hive
4. SELECT data
Creating a table in Hive

- CREATE TABLE tablename
  (col1 INT, col2 STRING)
  ROW FORMAT DELIMITED FIELDS TERMINATED BY ' \t ' STORED AS TEXTFILE;

Describes the format of the file
Load data into the table

- \texttt{LOAD DATA LOCAL INPATH \\
  '/myhd/file\_or\_dir' \\
  INTO TABLE <tablename>}

- \texttt{LOAD DATA INPATH \\
  '/hdfs/file\_or\_dir' \\
  INTO TABLE <tablename>}

- File format should match the CREATE TABLE definition!

Copies data from the client's filesystem to HDFS

Moves data to /user/hive/warehouse

Copyright 2010 Cloudera - Do not distribute
Schema on read, not write

- Data is not checked during the load
  - Loads are very fast
- Parsing errors would be found at query time
- Possible to have multiple schemas for the same data (using `EXTERNAL` tables)
Sqoop: SQL to Hadoop

```bash
$ sqoop --connect jdbc:mysql://foo.com/corp \
  --table employees \
  --hive-import \
  --fields-terminated-by '\t' \
  --lines-terminated-by '\n'
```

Copyright 2010 Cloudera - Do not distribute
Query the data with SELECT

• Similar to SQL:
  • SELECT..
  • FROM..
  • WHERE..
  • GROUP BY..
  • ORDER BY..

• Inner join, outer join, full outer join
• Subqueries (in the FROM clause)
• Built-in functions such as `round`, `concat`, `substr`, `max`, `min`, `sum`, `count`
Hive converts to a series of MapReduce phases

- **WHERE** => map
- **GROUP BY/ORDER BY** => reduce
- **JOIN** => map or reduce depending on optimizer

**Example:**

```
SELECT * FROM purchases
WHERE cost > 40
ORDER BY order_date DESC;
```

- **Single MapReduce required:**
  - WHERE clause translates to a “map”
  - Mapper outputs order_date as key
  - Single reducer collects sorted rows
EXPLAIN - the "map" (slide 1/3)

- EXPLAIN SELECT * FROM purchases
  WHERE cost > 40
  ORDER BY order_date DESC;

- ...

STAGE PLANS:
Stage: Stage-1
Map Reduce
  Alias -> Map Operator Tree:
    purchases
    TableScan
      alias: purchases
    Filter Operator
      predicate:
        expr: (cost > UDFToDouble(40))
        type: boolean
EXPLAIN - the "shuffle and sort" (slide 2/3)

• Select Operator
  expressions:
    expr: custid
type: int
    expr: order_date
type: string
    expr: cost
type: double

outputColumnNames: _col0, _col1, _col2

Reduce Output Operator
key expressions:
  expr: _col1
type: string
EXPLAIN - the "reduce" (slide 3/3)

- value expressions:
  expr: _col0
type: int
expr: _col1
type: string
expr: _col2
type: double

Reduce Operator Tree:
Extract
File Output Operator

Output of reducers

Copyright 2010 Cloudera - Do not distribute
Optimizations

• Some operations use direct HDFS access
  • SELECT * FROM table LIMIT 10;

• Number of MapReduce phases is minimized if possible

• Map-side join via MAPJOIN hint
  • SELECT /*+ MAPJOIN(t1) */ t1.col, t2.col
    FROM t1 JOIN t2
    ON (t1.col = t2.col)
Hive extension: multi-table insert

FROM (  
    SELECT username, accessdate  
    FROM logs WHERE url LIKE '%.cloudera.com'
) clicks

INSERT OVERWRITE DIRECTORY 'count'  
    SELECT count(1)

INSERT OVERWRITE DIRECTORY 'list_users'  
    SELECT DISTINCT clicks.username;
Invoking custom map script

- ADD FILE /tmp/map.py;

- INSERT OVERWRITE TABLE results_table
  SELECT transform(logdata.*)
  USING './map.py' as (output)
  FROM
  (SELECT * FROM logs) logdata;

Map.py receives the log records as tab-separated list and returns output.
Partitioning and bucketing

• Divide data into subsets of rows

• Benefits:
  • Better performance
  • Easier data management (add or delete a portion of a table)
  • Sampling of data
Partitioning

• CREATE TABLE logs (url STRING, user STRING) PARTITIONED BY (d STRING);

• LOAD DATA LOCAL INPATH '/tmp/new_logs.txt' INTO TABLE logs PARTITION (d='2010-04-01');
Bucketing

• CREATE TABLE tablename (columns)
  CLUSTERED BY (col) INTO N BUCKETS;
• SET hive.enforce.bucketing = true;
• INSERT OVERWRITE TABLE target
  SELECT * FROM helper;
Sampling

• Reading a subset of the data is very efficient if the table is bucketed

• `SELECT * FROM tablename TABLESAMPLE (BUCKET 1 OUT OF 4 ON col)`
Summary of Hive

• Enables easy analysis of data without a lot of setup
• Adds features that Hadoop lacks via the metastore
• Quite full-featured with lots of development work ongoing
Thanks!

Sarah Sproehnle
Educational Services, Cloudera Inc.