Leveraging Hadoop to Augment MySQL Deployments
About me

- Sarah Sproehnle
- Former MySQL instructor
- Joined Cloudera in March 2010
- sarah@cloudera.com
What is Hadoop?

- An open-source framework for storing and processing data on a cluster of computers
- Built-in high availability
- Scales linearly (proven to scale to 1000s of nodes)
- Designed for batch processing
- Optimized for streaming reads
Why Hadoop?

- Lots of data (TB+)
- Need to scan, process or transform *all* data
- Complex and/or unstructured data
Use cases for Hadoop

- **Recommendation engine**
  - Netflix recommends movies
  - last.fm recommends music

- **Ad targeting, log processing, search optimization**
  - ContextWeb, eBay and Orbitz

- **Machine learning and classification**
  - Yahoo! Mail's spam detection

- **Graph analysis**
  - Facebook and LinkedIn suggest connections
  - eHarmony matches people
# Hadoop vs. MySQL

<table>
<thead>
<tr>
<th></th>
<th>MySQL</th>
<th>Hadoop</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data capacity</td>
<td>TB+ (may require sharding)</td>
<td>PB+</td>
</tr>
<tr>
<td>Data per query</td>
<td>GB?</td>
<td>PB+</td>
</tr>
<tr>
<td>Read/write</td>
<td>Random read/write</td>
<td>Sequential scans, Append-only</td>
</tr>
<tr>
<td>Query language</td>
<td>SQL</td>
<td>Java MapReduce, scripting languages, HiveQL</td>
</tr>
<tr>
<td>Transactions</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Indexes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Latency</td>
<td>Sub-second (hopefully)</td>
<td>Minutes to hours</td>
</tr>
<tr>
<td>Data structure</td>
<td>Structured</td>
<td>Structured or un-structured</td>
</tr>
</tbody>
</table>
How does Hadoop work?

- Spreads your data onto tens, hundreds or thousands of machines using the **Hadoop Distributed File System (HDFS)**
  - Built-in redundancy (replication) for fault-tolerance
    - Machines will fail!
    - HDD MTBF 1000 days, 1000 disks = 1 failure every day

- Ability to read and process with **MapReduce**
  - Processing is sent to the data
  - Many "map" tasks each work on a slice of the data
  - Failed tasks are automatically restarted on another node
Example - word count

map(key, value)

foreach (word in value)

output (word, 1)

Key and value represent a row of data: key is the byte offset, value is a line

Intermediate output:
- the, 1
- cat, 1
- in, 1
- the, 1
- hat, 1
Reduce

reduce(key, list)
  sum the list
output(key, sum)

Hadoop aggregates the keys and calls reduce for each unique key:
  the, (1,1,1,1,1,1,...1)
  cat, (1,1,1)
  in, (1,1,1,1,1,1,1) ...

Final result:
  the, 45823
  cat, 1204
  in, 2693
  ...

Why MapReduce?

- By constraining computation to “map” and “reduce” phases, the tasks can be split and run in parallel
- Scales horizontally
- Programmer is isolated from individual failed tasks
  - Tasks are restarted on another node
- However, many computational tasks will require a series of map/reduce phases
The problem with MapReduce

- The developer has to worry about a lot of things besides the analysis/processing logic (Job setup, InputFormat, custom key/value classes)
- Typically written in Java
- The data is schema-less
- Even simple things may require several MapReduce passes
- Would be more convenient to use constructs such as "filter", "join", "aggregate"
Facebook's story

- Facebook collects TBs of data each day, coming from various places (MySQL, web logs, etc)

- Uses for Hadoop:
  - Log processing
  - Text mining
  - Document indexing
  - BI/analytics

- Desired:
  - Command-line interface
  - Easier development environment that supported ad hoc queries
  - A "schema" for the data
So Hive was born

- Since many at Facebook were familiar with SQL, Hive was created to use a similar interface

- Hive provides:
  - HiveQL, an SQL-like language for formulating your queries
  - A metastore which stores schema information (typically a MySQL server)

- Hive translates HiveQL to MapReduce code

- Today Facebook runs over 7500 Hive queries/day and scans more than 100TB
## HiveQL vs. SQL

<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 but <strong>not</strong> 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>
<tr>
<td><strong>Data per query</strong></td>
<td>GBs?</td>
<td>PBs</td>
</tr>
</tbody>
</table>
Getting started is easy!

1. Install Hadoop (on a single machine or a cluster)
   – Or download Cloudera's free VM!

2. Create Hive tables
   – CREATE TABLE t (col1 INT, col2 STRING)…

3. Load data into Hive
   – LOAD DATA INPATH '/path-in-hdfs' INTO TABLE t;
   – LOAD DATA LOCAL INPATH '/local-path' INTO TABLE t;

4. SELECT data
   – SELECT sum(col1) AS total, col2
     FROM t
     WHERE col2 LIKE 'foo%'
     GROUP BY col2
     ORDER BY total;
Under the hood

- Hive converts HiveQL to a series of MapReduce jobs and submits code to the Hadoop cluster
  - WHERE => map
  - GROUP/ORDER BY => reduce
  - JOIN => map or reduce depending on optimizer

- **EXPLAIN** shows the MapReduce plans
Example

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

- **Single MapReduce required:**
  - WHERE clause translates to a “map”
  - Map outputs order_date as key
  - Single reducer collects sorted rows
Extra features

- Partitioning
- UDF/UDAs
- Support for "sampling"
- JDBC and ODBC interfaces
  - BI vendor support coming soon!
- Integration with HBase
  - For an introduction to HBase, see Tom Hanlon's talk @ 4:25
It gets even easier!

- **Sqoop = SQL-to-Hadoop**
- Open source product from Cloudera
- Parallel import of data from many databases to Hadoop
- Parallel export of data from Hadoop to databases
- Integrates with Hive

...Via auto-generated datatype definitions

Custom MapReduce programs reinterpret data
"Sqooping" your tables into Hadoop

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

- Features supported:
  - Parallelized import and export
  - Choose rows (--where) or columns to import
  - Optimized for MySQL (uses parallel mysqldump commands)
  - LOBs can be inline or a separate file
  - Incremental loads (with TIMESTAMP or AUTO_INCREMENT col)
Example data pipeline

1. Use MySQL for real-time read/write data access
2. Cron job occasionally "sqoops" data into Hive
3. Use HiveQL to transform data, run batch analysis, join with other data (e.g., Apache logs), etc
4. Export the transformed results to OLAP or OLTP environment
Summary

Web tier

Log aggregator (Flume)

Hadoop cluster

Existing MySQL servers

Ad-hoc Hive queries

Nightly processing

Import with sqoop
Thanks!
sarah@cloudera.com