Importing and Exporting Data Between Hadoop and MySQL

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
- Based on Google's whitepapers of the Google File System and MapReduce
- Scales linearly (proven to scale to 1000s of nodes)
- Built-in high availability
- 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
  - financial companies: identify fraud, credit risk
- **Graph analysis**
  - Facebook, LinkedIn and eHarmony suggest connections

Hadoop vs. MySQL

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

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
- **Read and process data 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

Why MapReduce?

- **By constraining computation to “map” and “reduce” phases, the tasks can be split and run in parallel**
- **Able to process huge amounts of data over thousands of machines**
- **Scales linearly**
- **Programmer is isolated from individual failed tasks**
  - Tasks are restarted on another node

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)**
- **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"**
- **Solution: Hive**
  - SQL-like language on top of MapReduce

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

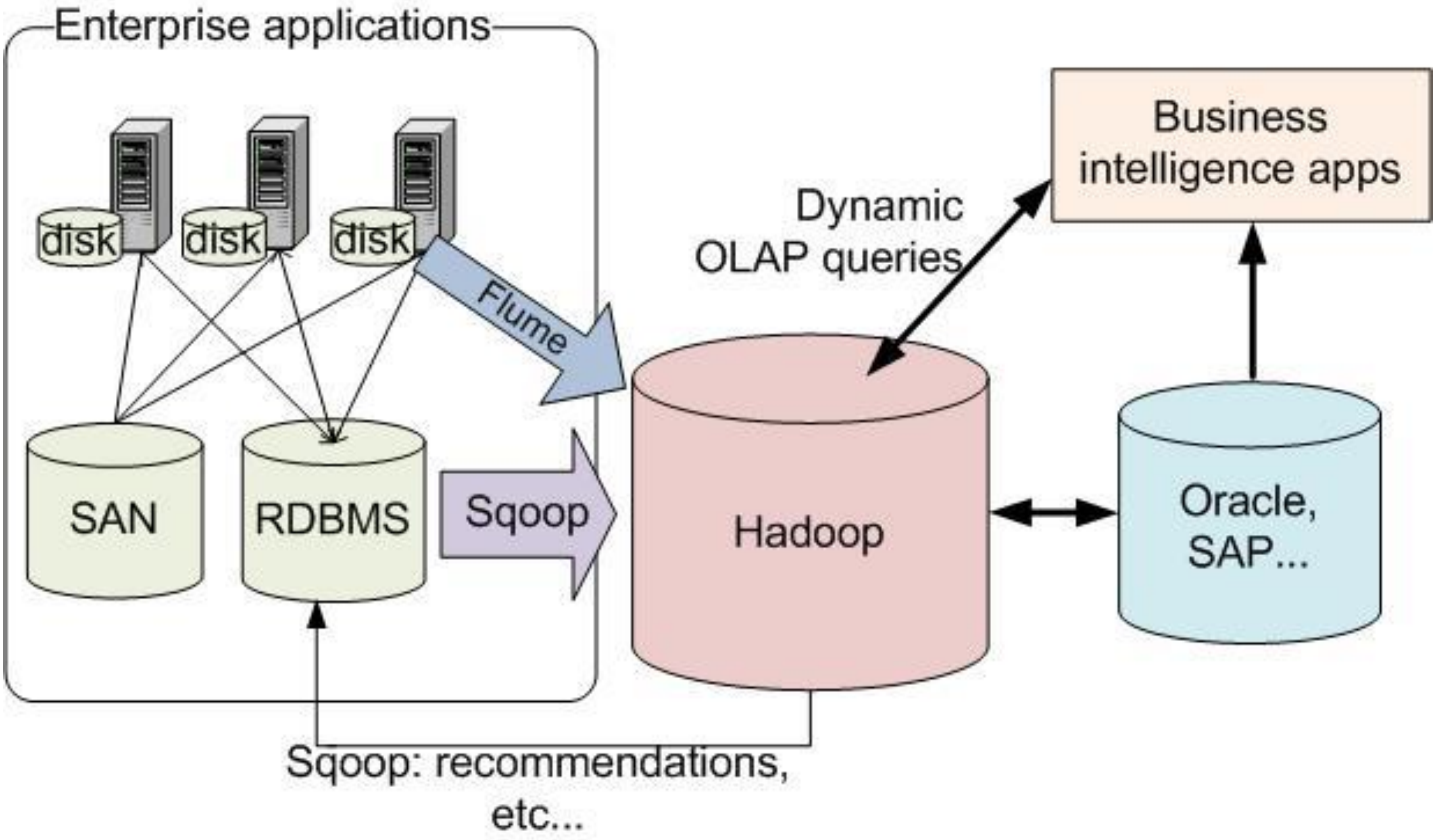
```
reduce(key, list)
    sum the list
    output(key, sum)
```

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) ...

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

So where does this fit in?



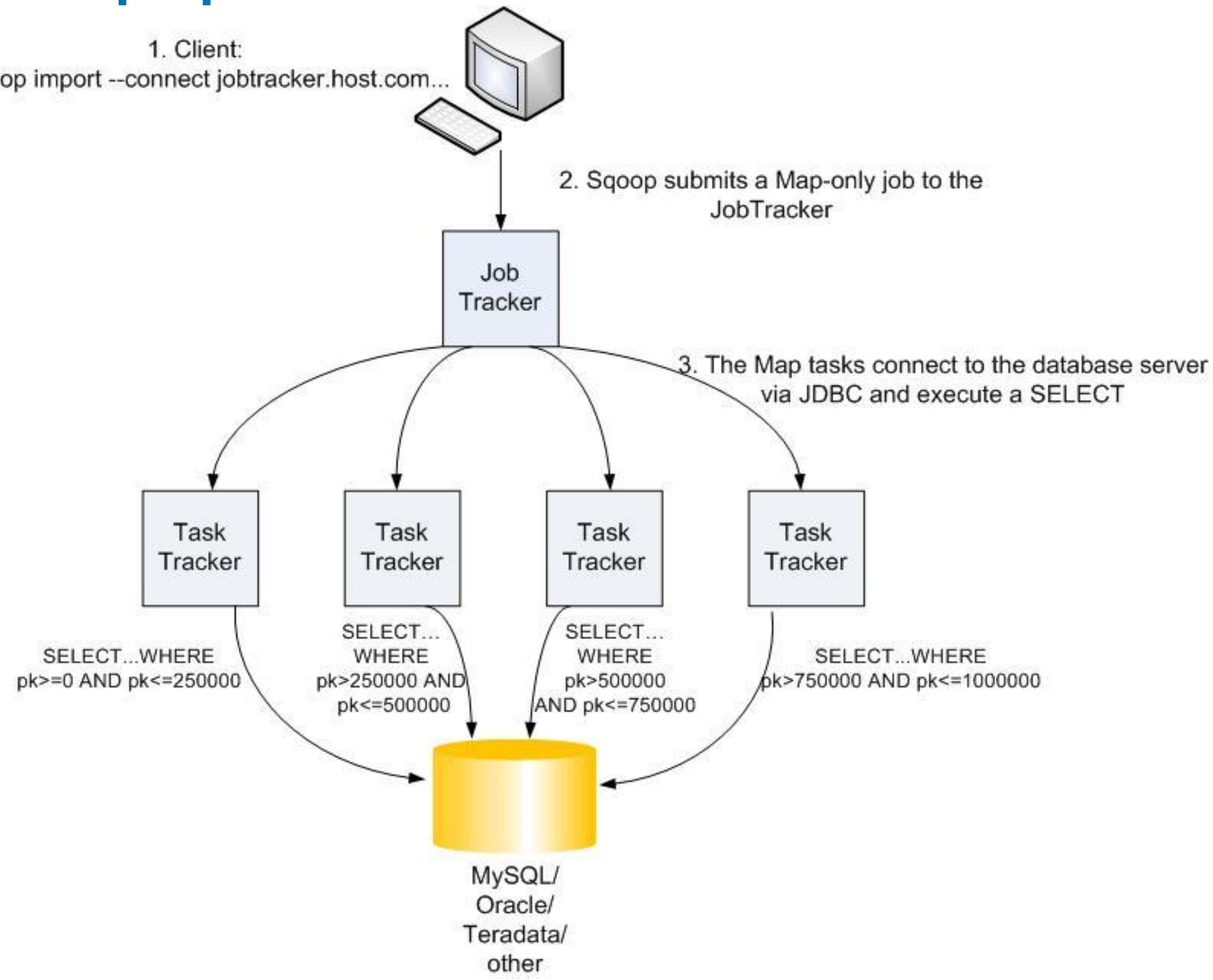
Example data pipeline

1. Use MySQL for real-time read/write data access (e.g., websites)
2. Cron job occasionally "Sqoops" data into Hadoop
3. Flume aggregates web logs and loads them into Hadoop
4. Use MapReduce to transform data, run batch analysis, join data, etc
5. Export the transformed results to OLAP or OLTP environment

Sqoop: SQL-to-Hadoop

- **Open source software**
- **Parallel import/export between Hadoop and various RDBMSes**
- **Default implementation is JDBC-based**
- **In Cloudera's Distribution including Apache Hadoop (CDH)**
- **Optimized for MySQL (built-in)**
- **Optimized connectors for Oracle, Netezza, Teradata (others coming)**
  - Freely available at cloudera.com

How Sqoop works



1. Client:
$ sqoop import --connect jobtracker.host.com...

2. Sqoop submits a Map-only job to the JobTracker

3. The Map tasks connect to the database server via JDBC and execute a SELECT

SELECT...WHERE pk>=0 AND pk<=250000

SELECT... WHERE pk>250000 AND pk<=500000

SELECT... WHERE pk>500000 AND pk<=750000

SELECT...WHERE pk>750000 AND pk<=1000000

"Sqooping" your tables into Hadoop

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

- This command will submit a Hadoop job that queries your MySQL server at foo.com and reads rows from db.orders
- Resulting TSV files are stored in Hadoop's Distributed File System

Other features

- **Other features :**
  - Choose which tables, rows (--where) or columns to import
  - Configurable parallelization
    - Specify the number of connections (--num-mappers)
    - Specify the column to split on (--split-by)
  - MySQL optimization (uses parallel mysqldump commands)
  - LOBs can be inline or a separate file
  - Incremental loads (with TIMESTAMP or AUTO_INCREMENT)
  - Integration with Hive and HBase

Exporting data

```
$ sqoop export --connect jdbc:mysql://foo.com/db
    --table bar
    --export-dir /hdfs_path/bar_data
```

- **Target table must already exist**
- **Assumes comma-separated fields**
  - Use `--fields-terminated-by` and `--lines-terminated-by`
- **Can use a staging table (`--staging-table`)**
  - Failed jobs can have unpredictable results otherwise

Sqoop + Hive

```
$ sqoop import --connect jdbc:mysql://foo.com/db
    --table orders
    --hive-import
```

By including --hive-import, Sqoop will create a Hive table definition for the data

- **Hive is a component that sits on top of Hadoop to provide:**
  - A command-line interface for submitting HiveQL
  - A metastore for putting table definitions on Hadoop data

Hive

```
$ hive

  hive> SHOW TABLES;

   ...
  hive> SELECT locationid, sum(cost) AS sales
        FROM orders
        GROUP BY locationid
        ORDER BY sales
        LIMIT 100;
```

Demo

```
$ sqoop import --connect jdbc:mysql://localhost/world
    --username root --table City --hive-import

$ sqoop import --connect jdbc:mysql://localhost/world
    --username root --table Country --hive-import

$ hadoop fs -ls /user/hive/warehouse

$ hadoop fs -cat /user/hive/warehouse/city/*0 | more

$ hive

SHOW TABLES;

SELECT * FROM city LIMIT 10;

SELECT countrycode, sum(population) as people FROM city WHERE
  population > 100000 GROUP BY countrycode
  ORDER BY people DESC LIMIT 10;

SELECT code, sum(ci.population) as people FROM city ci JOIN
  country co ON ci.countrycode = co.code WHERE continent =
  'North America'
  GROUP BY code ORDER BY people desc LIMIT 10;
```

Thanks!

sarah@cloudera.com