I’ll argue that Hive is indispensable to people creating “data warehouses” with Hadoop, because it gives them a “similar” SQL interface to their data, making it easier to migrate skills and even apps from existing relational tools to Hadoop.
Dean Wampler

- Consultant for Typesafe.
- Big Data, Scala, Functional Programming expert.
- dean.wampler@typesafe.com
- @deanwampler
- Hire me!
Why Hive?
Since your team *knows* SQL and all your *Data Warehouse* apps are written in SQL, *Hive* minimizes the effort of migrating to *Hadoop.*
Hive

- Ideal for *data warehousing*.
- *Ad-hoc queries* of data.
- Familiar *SQL* dialect.
- Analysis of *large* data sets.
- Hadoop *MapReduce* jobs.

Hive is a killer app, in our opinion, for data warehouse teams migrating to Hadoop, because it gives them a familiar SQL language that hides the complexity of MR programming.
Hive

- Invented at Facebook.
- Open sourced to Apache in 2008.
  - http://hive.apache.org
A Scenario: Mining Daily Click Stream Logs
Ingest & Transform:

- **From**: file:///server1/var/log/clicks.log

  Jan  9 09:02:17 server1 movies[18]: 1234: search for “vampires in love”.

  ...

As we copy the daily click stream log over to a local staging location, we transform it into the Hive table format we want.
Ingest & Transform:

• **From:** file://server1/var/log/clicks.log
  
  **Jan 9 09:02:17 server1 movies[18]:**
  1234: search for “vampires in love”.

...
Ingest & Transform:

• **From:** file://server1/var/log/clicks.log

  Jan  9 09:02:17 **server1** movies[18]:
  1234: search for “vampires in love”.
...

The server
Ingest & Transform:

- **From:** file://server1/var/log(clicks.log

Jan 9 09:02:17 server1 movies[18]:
1234: search for “vampires in love”.

...
Ingest & Transform:

- **From:** file://server1/var/log/clicks.log

  Jan 9 09:02:17 server1 movies[18]:
  1234: search for “vampires in love”.
...

  Customer id
Ingest & Transform:

- **From:** file://server1/var/log/clicks.log

  Jan 9 09:02:17 server1 movies[18]:
  1234: search for “vampires in love”.

  ...
Ingest & Transform:

- **From:** file://server1/var/log/clicks.log

  Jan  9 09:02:17 server1 movies[18]:
  1234: search for “vampires in love”.
  ...

- **To:** /staging/2012-01-09.log

  09:02:17^Aserver1^Amovies^A18^A1234^Asearch for “vampires in love”.
  ...

As we copy the daily click stream log over to a local staging location, we transform it into the Hive table format we want.
Ingest & Transform:

- **To:** /staging/2012-01-09.log

09:02:17^Aserver1^Amovies^A18^A1234^Asearch for “vampires in love”.

...

- *Removed* month (*Jan*) and day (09).
- *Added* ^A as *field* separators (Hive convention).
- *Separated* process *id* from process *name*.

The transformations we made. (You could use many different Linux, scripting, code, or Hadoop–related ingestion tools to do this.)
Ingest & Transform:

• Put in HDFS:

```bash
hadoop fs -put /staging/2012-01-09.log \
/clicks/2012/01/09/log.txt
```

• (The final file name doesn’t matter…)

Here we use the hadoop shell command to put the file where we want it in the file system. Note that the name of the target file doesn’t matter; we’ll just tell Hive to read all files in the directory, so there could be many files there!
Back to Hive...

- Create an *external* Hive table:

```sql
CREATE EXTERNAL TABLE clicks (
    hms STRING,
    hostname STRING,
    process STRING,
    pid INT,
    uid INT,
    message STRING)
PARTITIONED BY (
    year INT,
    month INT,
    day INT);
```

You don’t have to use EXTERNAL and PARTITIONED together….

Now let’s create an “external” table that will read those files as the “backing store”. Also, we make it partitioned to accelerate queries that limit by year, month or day. (You don’t have to use external and partitioned together…)
Back to Hive...

• Add a *partition* for 2012-01-09:

```sql
ALTER TABLE clicks ADD IF NOT EXISTS PARTITION (year=2012, month=01, day=09) LOCATION '/clicks/2012/01/09';
```

• A *directory* in HDFS.

We add a partition for each day. Note the LOCATION path, which is the directory where we wrote our file.
Now, Analyze!!

- What’s with the *kids* and *vampires*??

```sql
SELECT hms, uid, message FROM clicks
WHERE message LIKE '%vampire%' AND
  year = 2012 AND
  month = 01 AND
  day = 09;
```

- After some *MapReduce* crunching...

...  
09:02:29 1234 search for “twilight of the vampires”  
09:02:35 1234 add to cart “vampires want their genre back”  
...

And we can run SQL queries!!
Recap

• *SQL analysis* with Hive.
• *Other tools* can use the data, too.
• *Massive scalability* with Hadoop.
Hive queries generate MR jobs. (Some operations don’t invoke Hadoop processes, e.g., some very simple queries and commands that just write updates to the metastore.)

CLI = Command Line Interface.
HWI = Hive Web Interface.
There is “early” support for using Hive with HBase. Other databases and distributed file systems will no doubt follow.
For production, you need to set up a MySQL or PostgreSQL database for Hive’s metadata. Out of the box, Hive uses a Derby DB, but it can only be used by a single user and a single process at a time, so it’s fine for personal development only.
Queries

• Most queries use **MapReduce** jobs.

Hive generates MapReduce jobs to implement all the but the simplest queries.
The high latency makes Hive unsuitable for “online” database use. (Hive also doesn’t support transactions and has other limitations that are relevant here…) So, these limitations make Hive best for offline (batch mode) use, such as data warehouse apps.
HDFS Storage

• **Benefits**
  • Horizontal scalability.
  • Data redundancy.

• **Drawbacks**
  • No *insert, update, and delete*!

You can generate new tables or write to local files. Forthcoming versions of HDFS will support appending data.
HDFS Storage

• *Schema on Read*

• Schema enforcement at *query* time, not *write* time.

Especially for external tables, but even for internal ones since the files are HDFS files, Hive can’t enforce that records written to table files have the specified schema, so it does these checks at query time.
Other Limitations

- No *Transactions*.
- Some *SQL* features not implemented (yet).
More on Tables and Schemas
Data Types

• The usual *scalar* types:
  • TINYINT, ..., BIGINT.
  • FLOAT, DOUBLE.
  • BOOLEAN.
  • STRING.
Data Types

• The unusual *complex* types:
  • STRUCT.
  • MAP.
  • ARRAY.

Structs are like “objects” or “c-style structs”. Maps are key-value pairs, and you know what arrays are ;)

Wednesday, May 14, 14
CREATE TABLE employees (  
    name STRING,  
    salary FLOAT,  
    subordinates ARRAY<STRING>,  
    deductions MAP<STRING,FLOAT>,  
    address STRUCT<  
        street:STRING,  
        city:STRING,  
        state:STRING,  
        zip:INT>  
);
Suppose our employees table has a custom format and field delimiters. We can change them, although here I’m showing all the default values used by Hive!
Select, Where, Group By, Join,...
Common SQL...

• You get most of the usual suspects for `SELECT, WHERE, GROUP BY` and `JOIN`.

We’ll just highlight a few unique features.
"User Defined Functions"

ADD JAR MyUDFs.jar;

CREATE TEMPORARY FUNCTION net_salary
AS 'com.example.NetCalcUDF';

SELECT name,
    net_salary(salary, deductions)
FROM employees;

Following a Hive defined API, implement your own functions, build, put in a jar, and then use them in your queries. Here we (pretend to) implement a function that takes the employee’s salary and deductions, then computes the net salary.
ORDER BY vs. SORT BY

- A total ordering - one reducer.

```
SELECT name, salary
FROM employees
ORDER BY salary ASC;
```

- A local ordering - sorts within each reducer.

```
SELECT name, salary
FROM employees
SORT BY salary ASC;
```

For a giant data set, piping everything through one reducer might take a very long time. A compromise is to sort “locally”, so each reducer sorts its output. However, if you structure your jobs right, you might achieve a total order depending on how data gets to the reducers. (E.g., each reducer handles a year’s worth of data, so joining the files together would be totally sorted.)
Inner Joins

• Only equality \( x = y \).

\[
\text{SELECT} \ldots \\
\text{FROM} \ clicks \ a \ \text{JOIN} \ clicks \ b \ \text{ON} \ (a.\text{uid} = b.\text{uid}, \ a.\text{day} = b.\text{day}) \\
\text{WHERE} \ a.\text{process} = 'movies' \\
\text{AND} \ b.\text{process} = 'books' \\
\text{AND} \ a.\text{year} > 2012;
\]

Note that the \( a.\text{year} > \ldots \) is in the WHERE clause, not the ON clause for the JOIN. (I’m doing a correlation query; which users searched for movies and books on the same day?) Some outer and semi join constructs supported, as well as some Hadoop-specific optimization constructs.
A Final Example of Controlling MapReduce...
Specify Map & Reduce Processes

- Calling out to external programs to perform map and reduce operations.
FROM ( 
    FROM clicks
    MAP message
    USING '/tmp/vampire_extractor'
    AS item_title, count
    CLUSTER BY item_title) it
INSERT OVERWRITE TABLE vampire_stuff
REDUCE it.item_title, it.count
USING '/tmp/thing_counter.py'
AS item_title, counts;

Note the MAP … USING and REDUCE … USING. We’re also using CLUSTER BY (distributing and sorting on “item_title”).
Example

FROM ( 
FROM clicks
MAP message
USING '/tmp/vampire_extractor'
AS item_title, count
CLUSTER BY item_title) it
INSERT OVERWRITE TABLE vampire stuff
REDUCE it.item_title, it.count
USING '/tmp/thing_counter.py'
AS item_title, counts;

Call specific map and reduce processes.

Note the MAP … USING and REDUCE … USING. We’re also using CLUSTER BY (distributing and sorting on “item_title”).
Note the MAP … USING and REDUCE … USING. We’re also using CLUSTER BY (distributing and sorting on “item_title”).
... And Also:

FROM (  
  FROM clicks  
  MAP message  
  USING '/tmp/vampire_extractor'  
  AS item_title, count  
  CLUSTER BY item_title) it  
INSERT OVERWRITE TABLE vampire_stuff  
REDUCE it.item_title, it.count  
USING '/tmp/thing_counter.py'  
AS item_title, counts;

How to populate an “internal” table.

Note the MAP … USING and REDUCE … USING. We’re also using CLUSTER BY (distributing and sorting on “item_title”).

Wednesday, May 14, 14
Hive:
Conclusions
Hive Disadvantages

• *Not a real SQL Database.*

• Transactions, updates, etc.

• … but features will grow.

• *High latency* queries.

• Documentation poor.
Hive Advantages

• Indispensable for SQL users.
• Easier than Java MR API.
• Makes porting data warehouse apps to Hadoop much easier.
Questions?

- dean.wampler@typesafe.com
- @deanwampler
- github.com/deanwampler/Presentations
- Hire me!