Hive and Pig

Juliana Freire
New York University

Some slides from J. Lin
Need for High-Level Languages

• Hadoop is great for large-data processing!
• But writing Java programs for everything is verbose and slow
• Not everyone wants to (or can) write Java code
• Solution: develop higher-level data processing languages
  • Hive: HQL is like SQL
  • Pig: Pig Latin is a bit like Perl
Hive and Pig

- Hive: data warehousing application in Hadoop
  - Query language is HQL, variant of SQL
  - Tables stored on HDFS as flat files
  - Developed by Facebook, now open source
- Pig: large-scale data processing system
  - Scripts are written in Pig Latin, a dataflow language
  - Developed by Yahoo!, now open source
  - Roughly 1/3 of all Yahoo! internal jobs
- Common idea:
  - Provide higher-level language to facilitate large-data processing
  - Higher-level language “compiles down” to Hadoop jobs
Hive: Background

- Started at Facebook
- Data was collected by nightly cron jobs into Oracle DB
- “ETL” via hand-coded python
- Grew from 10s of GBs (2006) to 1 TB/day new data (2007), now 10x that
Hive Components

- **Shell:** allows interactive queries
- **Driver:** session handles, fetch, execute
- **Compiler:** parse, plan, optimize
- **Execution engine:** DAG of stages (MR, HDFS, metadata)
- **Metastore:** schema, location in HDFS, SerDe
## Data Model

- **Tables**: analogous to tables in RDBMS
- **Typed columns** (int, float, string, boolean)
- **Structs**: `{a INT; b INT}`.
- **Also, list, arrays**: map (for JSON-like data)
- **Partitions**
- **For example, range-partition tables by date**
- **Buckets**
- **Hash partitions within ranges** (useful for sampling, join optimization)

Partitions - Each table can have one or more partitions which determine the distribution of data within sub-directories of the table directory. Suppose data for table T is in the directory `/wh/T`. If T is partitioned on columns `ds` and `ctry`, then data with a particular `ds` value 20090101 and `ctry` value US, will be stored in files within the directory `/wh/T/ds=20090101/ctry=US`.

[Thusoo et al., VLDB 2009]
Metastore

- Database: namespace containing a set of tables
- Holds table definitions (column types, physical layout)
- Holds partitioning information
- Can be stored in Derby, MySQL, and many other relational databases
Physical Layout

- Warehouse directory in HDFS
  - E.g., /user/hive/warehouse
- Tables stored in subdirectories of warehouse
  - Partitions form subdirectories of tables
  - Each table has a corresponding HDFS directory
- Actual data stored in flat files
  - Users can associate a table with a serialization format
    - Control char-delimited text, or SequenceFiles
    - With custom SerDe, can use arbitrary format
Hive: Example

- Hive looks similar to an SQL database
- Relational join on two tables:
  - Table of word counts from Shakespeare collection
  - Table of word counts from the bible

```
SELECT s.word, s.freq, k.freq FROM shakespeare s
JOIN bible k ON (s.word = k.word) WHERE s.freq >= 1 AND k.freq >= 1
ORDER BY s.freq DESC LIMIT 10;
```

<table>
<thead>
<tr>
<th></th>
<th>the</th>
<th>I</th>
<th>and</th>
<th>to</th>
<th>of</th>
<th>a</th>
<th>you</th>
<th>my</th>
<th>in</th>
<th>is</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>25848</td>
<td>62394</td>
<td>19671</td>
<td>38985</td>
<td>16700</td>
<td>34654</td>
<td>14170</td>
<td>8057</td>
<td>10797</td>
<td>12445</td>
</tr>
<tr>
<td></td>
<td>23031</td>
<td>8854</td>
<td>18038</td>
<td>13526</td>
<td>12702</td>
<td>2720</td>
<td>11297</td>
<td>4135</td>
<td>8882</td>
<td>6884</td>
</tr>
</tbody>
</table>
Hive: Another Example

LOAD DATA LOCAL INPATH '/logs/status_updates'
INTO TABLE status_updates PARTITION (ds='2009-03-20')

FROM (SELECT a.status, b.school, b.gender
       FROM status_updates a JOIN profiles b
       ON (a.userid = b.userid and
           a.ds='2009-03-20')
     ) subq1
INSERT OVERWRITE TABLE gender_summary
     PARTITION(ds='2009-03-20')
SELECT subq1.gender, COUNT(1) GROUP BY subq1.gender

INSERT OVERWRITE TABLE school_summary
     PARTITION(ds='2009-03-20')
SELECT subq1.school, COUNT(1) GROUP BY subq1.school
Hive: Another Example

- HiveQL provides MapReduce constructs
  REDUCE subq2.school, subq2.meme, subq2.cnt
    USING ‘top10.py’ AS (school, meme, cnt)
FROM (SELECT subq1.school, subq1.meme, COUNT(1) AS cnt
     FROM (MAP b.school, a.status
          USING ‘meme-extractor.py’ AS (school, meme)
          FROM status_updates a JOIN profiles b
            ON (a.userid = b.userid)
     ) subq1
     GROUP BY subq1.school, subq1.meme
     DISTRIBUTUE BY school, meme
     SORT BY school, meme, cnt desc
  ) subq2;
Example Data Analysis Task

Find users who tend to visit “good” pages.

<table>
<thead>
<tr>
<th>Visits</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>url</td>
</tr>
<tr>
<td>Amy</td>
<td><a href="http://www.cnn.com">www.cnn.com</a></td>
</tr>
<tr>
<td>Amy</td>
<td><a href="http://www.crap.com">www.crap.com</a></td>
</tr>
<tr>
<td>Amy</td>
<td><a href="http://www.myblog.com">www.myblog.com</a></td>
</tr>
<tr>
<td>Amy</td>
<td><a href="http://www.flickr.com">www.flickr.com</a></td>
</tr>
<tr>
<td>Fred</td>
<td>cnn.com/index.htm</td>
</tr>
</tbody>
</table>

Pig Slides adapted from Olston et al.
Conceptual Dataflow

Load Visits(user, url, time)

Canonicalize URLs

Load Pages(url, pagerank)

Join url = url

Group by user

Compute Average Pagerank

Filter avgPR > 0.5
System-Level Dataflow

Visits
- load
- canonicalize
- join by url
- group by user
- compute average pagerank
- filter
- the answer

Pages
- load
- join by url

Pig Slides adapted from Olston et al.
public class Mapper {  
  public void map(Text key, Text value, OutputCollector<Text, Text> oc, Reporter reporter) throws IOException {  
    // Find the url  
    String key = line.substring(firstComma, secondComma);  
    String value = line.substring(firstComma + 1);  
    for (String s2 : second) {  
      oc.collect(outKey, new LongWritable(1L));  
    }  
  }  
}
Visits = load '/data/visits' as (user, url, time);
Visits = foreach Visits generate user, Canonicalize(url), time;

Pages = load '/data/pages' as (url, pagerank);

VP = join Visits by url, Pages by url;
UserVisits = group VP by user;
UserPageranks = foreach UserVisits generate user, AVG(VP.pagerank) as avgpr;
GoodUsers = filter UserPageranks by avgpr > '0.5';

store GoodUsers into '/data/good_users';
Java vs. Pig Latin

1/20 the lines of code

Hadoop | Pig
---|---
180 | 20
160 |
140 |
120 |
100 |
80 |
60 |
40 |
20 |
0 |

1/16 the development time

Hadoop | Pig
---|---
300 | 30
250 |
200 |
150 |
100 |
50 |
0 |

Performance on par with raw Hadoop!

Pig Slides adapted from Olston et al.
Pig takes care of...

- Schema and type checking
- Translating into efficient physical dataflow
  - (i.e., sequence of one or more MapReduce jobs)
- Exploiting data reduction opportunities
  - (e.g., early partial aggregation via a combiner)
- Executing the system-level dataflow
  - (i.e., running the MapReduce jobs)
- Tracking progress, errors, etc.
References

• Getting started with Pig: http://pig.apache.org/docs/r0.11.1/start.html
• Pig Tutorial: http://pig.apache.org/docs/r0.7.0/tutorial.html
• Hive Tutorial: https://cwiki.apache.org/confluence/display/Hive/Tutorial
Questions