Other M/R Interfaces: PIG II

EN 600.420
Instructor: Randal Burns
16 April 2018
PG Language Constructs

- **FOREACH**: allows parallel processing (in a mapper) for all inputs in a data set
- **FILTER**: discard unwanted data (in either mapper or reducer)
- **GROUP/CO-GROUP**: put related data together using the shuffle process.

- These constructs allow for database-style query optimization.
PIG Data Model

- Atom: simple value
- Tuple: sequence of values
- Bag: multiset with duplicates
  - flexible schema for elements
  
  \[
  \{ ('alice', 'lakers') \\
  ( 'alice', ( 'iPod', 'apple' )) \}
  \]

- Map: key/value data structure
  - Keys must be atoms for efficiency

\[
\begin{align*}
\text{fan of} & \rightarrow \{ ('lakers') \\
& \{ ('iPod') \} \\
'age' & \rightarrow 20
\end{align*}
\]
PIG Expressions

- Simple set that have to be parallelizable

Let fields of tuple $t$ be called $f_1$, $f_2$, $f_3$

<table>
<thead>
<tr>
<th>Expression Type</th>
<th>Example</th>
<th>Value for $t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>‘bob’</td>
<td>Independent of $t$</td>
</tr>
<tr>
<td>Field by position</td>
<td>$0$</td>
<td>‘alice’</td>
</tr>
<tr>
<td>Field by name</td>
<td>$f_3$</td>
<td>‘age’ → 20</td>
</tr>
<tr>
<td>Projection</td>
<td>$f_2.0$</td>
<td>(‘lakers’)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(‘iPod’)</td>
</tr>
<tr>
<td>Map Lookup</td>
<td>$f_3#’age’</td>
<td>20</td>
</tr>
<tr>
<td>Function Evaluation</td>
<td>SUM($f_2.1$)</td>
<td>1 + 2 = 3</td>
</tr>
<tr>
<td>Conditional Expression</td>
<td>$f_3#’age’&gt;18?$</td>
<td>‘adult’</td>
</tr>
<tr>
<td></td>
<td>‘adult’: ‘minor’</td>
<td></td>
</tr>
<tr>
<td>Flattening</td>
<td>FLATTEN($f_2$)</td>
<td>‘lakers’, 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘iPod’, 2</td>
</tr>
</tbody>
</table>

Table 1: Expressions in Pig Latin.
Bags and Co-Groupings

- Pig programming uses the pattern of co-grouping data, applying aggregates, and then flattening the results
  - Allows SQL-like functionality in sequenced programming
  - It’s not super-intuitive (see the paper)

Figure 2: COGROUP versus JOIN.
Compiling to MR

- Each PIG program compiles to several MR programs and is run in Hadoop!

**Figure 3: Map-reduce compilation of Pig Latin.**
Compiling to MR (ii)


![Diagram showing Pig Latin and Logical Plan]
Compiling to MR (ii)

Hive

- Hive: data model and system for data warehousing in map/reduce systems.
- HiveQL: SQL programming for Map/Reduce
  - Not SQL 92 complete
  - No transactions, no materialized views, limited subquery support
- The definitive Hive paper
Hive Example: Status Meme

- Table schema:
  
  ```
  status_updates(userid int, status string, ds string)
  ```

- Load log files daily:
  
  ```
  LOAD DATA LOCAL INPATH '../logs/status_updates'
  INTO TABLE status_updates PARTITION (ds='2009-03-20')
  ```
Daily Statistics

- Join logs with profiles and figure out the number of tweets from men/women and by school

```
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
```
How do it go?

- Hive puts tables on HDFS as files and runs queries as Hadoop! jobs

Figure 1: Hive Architecture
Resulting Query Plan (part 1)

- You don’t need to understand. These are the MR jobs generated by the example.
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
Take Aways

- Other ways to program M/R
  - More concise, easier to maintain
  - Particularly for data processing tasks that result in multi-stage map/reduce programs

- Ethos: take the best from DBs
  - Declarative languages and optimization
  - Ad-hoc queries

- Ethos: and leave behind the stuff that’s not parallel
  - Indexes, nested sub-queries
The (Un)Reasonable Debate

- Imperative programming
  - How humans think, step by step
  - Program encodes execution instructions

- Declarative programing
  - What! (Not how.)
  - Allows system to optimize execution
  - Non-intuitive (for many)
  - SQL != declarative programming. It is a specific instance that some love and some hate.

- PIG notable for trying to strike a happy balance
  - DB guys don’t see the upside here