Let’s see how we can create complex MapReduce workflows by programming in a high-level language.

The Pig System

- Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, Andrew Tomkins: Pig Latin: a not-so-foreign language for data processing. SIGMOD Conference 2008: 1099-1110
- Several slides courtesy Chris Olston and Utkarsh Srivastava
- Open source project under the Apache Hadoop umbrella

Overview

- Design goal: find sweet spot between declarative style of SQL and low-level procedural style of MapReduce
- Programmer creates Pig Latin program, using high-level operators
- Pig Latin program is compiled to MapReduce program to run on Hadoop

Why Not SQL or Plain MapReduce?

- SQL difficult to use and debug for many programmers
- Programmer might not trust automatic optimizer and prefers to hard-code best query plan
- Plain MapReduce lacks convenience of readily available, reusable data manipulation operators like selection, projection, join, sort
- Program semantics hidden in “opaque” Java code – More difficult to optimize and maintain

Example Data Analysis Task

Find the top 10 most visited pages in each category

<table>
<thead>
<tr>
<th>Visits</th>
<th>Url Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>Url</td>
</tr>
<tr>
<td>Amy</td>
<td>cnn.com</td>
</tr>
<tr>
<td>Amy</td>
<td>bbc.com</td>
</tr>
<tr>
<td>Amy</td>
<td>flickr.com</td>
</tr>
<tr>
<td>Fred</td>
<td>cnn.com</td>
</tr>
</tbody>
</table>

Data Flow

- Load visits
- Group by url
- Foreach url generate count
- Load url info
- Join on url
- Group by category
- Foreach category generate top10 urls
In Pig Latin

visits = load '/data/visits' as (user, url, time);
gVisits = group visits by url;
visitCounts = foreach gVisits generate url, count(visits);

urlInfo = load '/data/urlInfo' as (url, category, pRank);
visitCounts = join visitCounts by url, urlInfo by url;
gCategories = group visitCounts by category;
topUrls = foreach gCategories generate top(visitCounts,10);
store topUrls into '/data/topUrls';

Pig Latin Notes

• No need to import data into database
  — Pig Latin works directly with files
• Schemas are optional and can be assigned dynamically
  — Load '/data/visits' as (user, url, time);
• Can call user-defined functions in every construct like Load, Store, Group, Filter, Foreach
  — Foreach gCategories generate top(visitCounts,10);

Pig Latin Data Model

• Fully-nestable data model with:
  — Atomic values, tuples, bags (lists), and maps
    `yahoo, finance, email, news`
  — More natural to programmers than flat tuples
  — Can flatten nested structures using FLATTEN
  — Avoids expensive joins, but more complex to process

Pig Latin Operators: LOAD

• Reads data from file and optionally assigns schema to each record
• Can use custom deserializer

queries = LOAD 'query_log.txt' USING myLoad() AS (userID, queryString, timestamp);

Pig Latin Operators: FOREACH

• Applies processing to each record of a data set
• No dependence between the processing of different records
  — Allows efficient parallel implementation
• GENERATE creates output records for a given input record

expanded_queries = FOREACH queries
GENERATE userId, expandQuery(queryString);

Pig Latin Operators: FILTER

• Remove records that do not pass filter condition
• Can use user-defined function in filter condition

real_queries = FILTER queries BY userId neq 'bot';
Pig Latin Operators: COGROUP

- Group together records from one or more data sets

<table>
<thead>
<tr>
<th>queryString</th>
<th>url</th>
<th>rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lakers</td>
<td>nba.com</td>
<td>1</td>
</tr>
<tr>
<td>Lakers</td>
<td>espn.com</td>
<td>2</td>
</tr>
<tr>
<td>Kings</td>
<td>nhl.com</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>queryString</th>
<th>url</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lakers</td>
<td>top</td>
<td>50</td>
</tr>
<tr>
<td>Lakers</td>
<td>side</td>
<td>20</td>
</tr>
<tr>
<td>Kings</td>
<td>top</td>
<td>30</td>
</tr>
<tr>
<td>Kings</td>
<td>side</td>
<td>10</td>
</tr>
</tbody>
</table>

COGROUP results BY queryString, revenue BY queryString

- Lakers, (Lakers, nba.com, 1)
- (Lakers, espn.com, 2)
- (Lakers, top, 50)
- (Lakers, side, 20)

- Kings, (Kings, nhl.com, 1)
- (Kings, nba.com, 2)
- (Kings, top, 30)
- (Kings, side, 10)

Pig Latin Operators: GROUP

- Special case of COGROUP, to group single data set by selected fields
- Similar to GROUP BY in SQL, but does not need to apply aggregate function to records in each group

grouped_revenue = GROUP revenue BY queryString;

Pig Latin Operators: JOIN

- Computes equi-join
  join_result = JOIN results BY queryString, revenue BY queryString;
- Just a syntactic shorthand for COGROUP followed by flattening
  temp_var = COGROUP results BY queryString, revenue BY queryString;
  join_result = FOREACH temp_var GENERATE FLATTEN(results), FLATTEN(revenue);

Other Pig Latin Operators

- UNION: union of two or more bags
- CROSS: cross product of two or more bags
- ORDER: orders a bag by the specified field(s)
- DISTINCT: eliminates duplicate records in bag
- STORE: saves results to a file
- Nested bags within records can be processed by nesting operators within a FOREACH operator

MapReduce in Pig Latin

map_result = FOREACH input GENERATE FLATTEN(map(*));
key_groups = GROUP map_result BY $0;
output = FOREACH key_groups GENERATE reduce(*);

- Map() is a UDF, where * indicates that the entire input record is passed to map()
- $0 refers to first field, i.e., the intermediate key here
- Reduce() is another UDF
Implementation

- SQL
- automatic rewrite + optimize
- Pig
- Hadoop Map-Reduce
- user

Compilation into Map-Reduce

- Every group or join operation forms a map-reduce boundary
- Other operations pipelined into map and reduce phases

Pig System

- Parser
- parsed program
- Pig Compiler
- cross-job optimizer
- execution plan
- map-red. jobs
- Map-Reduce
- output
- X
- Y
- filter
- join
- %1

Is Pig a DBMS?

<table>
<thead>
<tr>
<th>DBMS</th>
<th>Pig</th>
</tr>
</thead>
<tbody>
<tr>
<td>workload</td>
<td>Bulk and random reads &amp; writes; indexes, transactions</td>
</tr>
<tr>
<td></td>
<td>Bulk reads &amp; writes only</td>
</tr>
<tr>
<td>data representation</td>
<td>System controls data format</td>
</tr>
<tr>
<td></td>
<td>Must pre-declare schema</td>
</tr>
<tr>
<td>programming style</td>
<td>System of constraints</td>
</tr>
<tr>
<td></td>
<td>Sequence of steps</td>
</tr>
<tr>
<td>customizable processing</td>
<td>Custom functions second-class to logic expressions</td>
</tr>
<tr>
<td></td>
<td>Easy to incorporate custom functions</td>
</tr>
</tbody>
</table>