BDAS and Spark

University of California, Berkeley
School of Information
*IS 257: Database Management*
Announcement

• Sign up for final project presentations
  – http://doodle.com/poll/46ivgbr5b73yzagr
Outline

• Review
  – Mapreduce
  – Pig
  – Hive

• BDAS and Spark
Map/Reduce ala Google

• **map(key, val)** is run on each item in set
  – emits new-key / new-val pairs

• **reduce(key, vals)** is run for each unique key emitted by **map()**
  – emits final output

From “MapReduce…” by Dan Weld
Programming model

- Input & Output: each a set of key/value pairs
- Programmer specifies two functions:
  - `map (in_key, in_value) -> list(out_key, intermediate_value)`
    - Processes input key/value pair
    - Produces set of intermediate pairs
  - `reduce (out_key, list(intermediate_value)) -> list(out_value)`
    - Combines all intermediate values for a particular key
    - Produces a set of merged output values (usually just one)

From “MapReduce: Simplified data Processing…”, Jeffrey Dean and Sanjay Ghemawat
count words in docs

– Input consists of (url, contents) pairs

– map(key=url, val=contents):
  • For each word $w$ in contents, emit $(w, "1")$

– reduce(key=word, values=uniq_counts):
  • Sum all “1”s in values list
  • Emit result “(word, sum)”

From “MapReduce…” by Dan Weld
Count, Illustrated

map(key=url, val=contents):
   For each word w in contents, emit (w, “1”)
reduce(key=word, values=uniq_counts):
   Sum all “1”s in values list
   Emit result “(word, sum)”

```
see bob throw
see spot run
```

```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>see</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>bob</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>run</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>see</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>spot</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>throw</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
```

From “MapReduce…” by Dan Weld
Example

- Page 1: the weather is good
- Page 2: today is good
- Page 3: good weather is good.
Map output

• Worker 1:
  – (the 1), (weather 1), (is 1), (good 1).

• Worker 2:
  – (today 1), (is 1), (good 1).

• Worker 3:
  – (good 1), (weather 1), (is 1), (good 1).

From “MapReduce: Simplified data Processing…”, Jeffrey Dean and Sanjay Ghemawat
Reduce Input

• Worker 1:
  – (the 1)

• Worker 2:
  – (is 1), (is 1), (is 1)

• Worker 3:
  – (weather 1), (weather 1)

• Worker 4:
  – (today 1)

• Worker 5:
  – (good 1), (good 1), (good 1), (good 1)

From “MapReduce: Simplified data Processing…”, Jeffrey Dean and Sanjay Ghemawat
Reduce Output

- Worker 1:
  - (the 1)
- Worker 2:
  - (is 3)
- Worker 3:
  - (weather 2)
- Worker 4:
  - (today 1)
- Worker 5:
  - (good 4)

From “MapReduce: Simplified data Processing…”, Jeffrey Dean and Sanjay Ghemawat
Data Flow in a MapReduce Program in Hadoop

- InputFormat
- Map function
- Partitioner
- Sorting & Merging
- Combiner
- Shuffling
- Merging
- Reduce function
- OutputFormat
Fault tolerance

• On worker failure:
  – Detect failure via periodic heartbeats
  – Re-execute completed and in-progress map tasks
  – Re-execute in progress reduce tasks
  – Task completion committed through master

• Master failure:
  – Could handle, but don't yet (master failure unlikely)

From “MapReduce: Simplified data Processing…”, Jeffrey Dean and Sanjay Ghemawat
Refinement

- Different partitioning functions.
- Combiner function.
- Different input/output types.
- Skipping bad records.
- Local execution.
- Status info.
- Counters.

From “MapReduce: Simplified data Processing…”, Jeffrey Dean and Sanjay Ghemawat
Performance

• Scan \(10^{10}\) 100-byte records to extract records matching a rare pattern (92K matching records) : 150 seconds.

• Sort \(10^{10}\) 100-byte records (modeled after TeraSort benchmark) : normal 839 seconds.

From “MapReduce: Simplified data Processing… “, Jeffrey Dean and Sanjay Ghemawat
More and more mapreduce

From “MapReduce: Simplified data Processing…”, Jeffrey Dean and Sanjay Ghemawat
But – Raw Hadoop means code

• Most people don’t want to write code if they don’t have to
• Various tools layered on top of Hadoop give different, and more familiar, interfaces
• Hbase – intended to be a NoSQL database abstraction for Hadoop
• Hive and it’s SQL-like language
PIG – A data-flow language for MapReduce
MapReduce too complex?

• Restrict programming model
  – Only two phases
  – Job chain for long data flow

• Put the logic at the right phase
  – In MR programmers are responsible for this

• Too many lines of code even for simple logic
  – How many lines do you have for word count?
Pig...

• High level dataflow language (Pig Latin)
  – Much simpler than Java
  – Simplify the data processing
• Put the operations at the appropriate phases (map, shuffle, etc.)
• Chains multiple MapReduce jobs
• Similar to relational algebra, but on files instead of relations
Pig Latin

• Data flow language
  – User specifies a sequence of operations to process data
  – More control on the processing, compared with declarative language

• Various data types are supported
• "Schema"s are supported
• User-defined functions are supported
Motivation by Example

• Suppose we have user data in one file, website data in another file.
• We need to find the top 5 most visited pages by users aged 18-25
In Pig Latin

Users = load 'users' as (name, age);
Fltrd = filter Users by
    age >= 18 and age <= 25;
Pages = load 'pages' as (user, url);
Jnd = joinFltrd(by name, Pages by user;
Grpd = groupJnd by url;
Smmdb = foreachGrpd generate group,
COUNT(Jnd) as clicks;
Srtdd = orderBySmmdb by clicks desc;
Top5 = limitSrtdd 5;
store Top5 into 'top5sites' ;
Pig runs over Hadoop

Job executes on cluster

Pig resides on user machine

User machine

Hadoop Cluster

No need to install anything extra on your Hadoop cluster.
How Pig is used in Industry

• At Yahoo!, 70% MapReduce jobs are written in Pig

• Used to
  – Process web log
  – Build user behavior models
  – Process images
  – Data mining

• Also used by Twitter, LinkedIn, Ebay, AOL, etc.
Hive - SQL on top of Hadoop
Map-Reduce and SQL

• **Map-Reduce is scalable**
  – SQL has a huge user base
  – SQL is easy to code

• **Solution: Combine SQL and Map-Reduce**
  – Hive on top of Hadoop (open source)
  – Aster Data (proprietary)
  – Green Plum (proprietary)
Hive

• **A database/data warehouse on top of Hadoop**
  – Rich data types (structs, lists and maps)
  – Efficient implementations of SQL filters, joins and group-by’s on top of mapreduce

• **Allow users to access Hadoop data without using Hadoop**

• **Link:**
Hive QL – Join

- **SQL:**
  
  ```sql
  INSERT INTO TABLE pv_users
  SELECT pv.pageid, u.age
  FROM page_view pv JOIN user u ON (pv.userid = u.userid);
  ```

<table>
<thead>
<tr>
<th>pageid</th>
<th>userid</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>111</td>
<td>9:08:01</td>
</tr>
<tr>
<td>2</td>
<td>111</td>
<td>9:08:13</td>
</tr>
<tr>
<td>1</td>
<td>222</td>
<td>9:08:14</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>userid</th>
<th>age</th>
<th>gender</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>25</td>
<td>female</td>
</tr>
<tr>
<td>222</td>
<td>32</td>
<td>male</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pageid</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25</td>
</tr>
<tr>
<td>2</td>
<td>25</td>
</tr>
<tr>
<td>1</td>
<td>32</td>
</tr>
</tbody>
</table>
Hive QL – Join in Map Reduce

page_view

<table>
<thead>
<tr>
<th>page_id</th>
<th>user_id</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>111</td>
<td>9:08:01</td>
</tr>
<tr>
<td>2</td>
<td>111</td>
<td>9:08:13</td>
</tr>
<tr>
<td>1</td>
<td>222</td>
<td>9:08:14</td>
</tr>
</tbody>
</table>

user

<table>
<thead>
<tr>
<th>user_id</th>
<th>age</th>
<th>gender</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>25</td>
<td>female</td>
</tr>
<tr>
<td>222</td>
<td>32</td>
<td>male</td>
</tr>
</tbody>
</table>

map

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>&lt;1,1&gt;</td>
</tr>
<tr>
<td>111</td>
<td>&lt;1,2&gt;</td>
</tr>
<tr>
<td>222</td>
<td>&lt;1,1&gt;</td>
</tr>
</tbody>
</table>

shuffle

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>&lt;1,1&gt;</td>
</tr>
<tr>
<td>111</td>
<td>&lt;1,2&gt;</td>
</tr>
<tr>
<td>222</td>
<td>&lt;1,1&gt;</td>
</tr>
</tbody>
</table>

sort

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>&lt;2,25&gt;</td>
</tr>
<tr>
<td>222</td>
<td>&lt;2,32&gt;</td>
</tr>
</tbody>
</table>

reduce

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>222</td>
<td>&lt;1,1&gt;</td>
</tr>
<tr>
<td>222</td>
<td>&lt;2,32&gt;</td>
</tr>
</tbody>
</table>
Hive QL – Group By

• **SQL:**

  ```sql
  INSERT INTO TABLE pageid_age_sum
  SELECT pageid, age, count(1)
  FROM pv_users
  GROUP BY pageid, age;
  ```

<table>
<thead>
<tr>
<th>pageid</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25</td>
</tr>
<tr>
<td>2</td>
<td>25</td>
</tr>
<tr>
<td>1</td>
<td>32</td>
</tr>
<tr>
<td>2</td>
<td>25</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pageid</th>
<th>age</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>25</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>32</td>
<td>1</td>
</tr>
</tbody>
</table>
Hive QL – Group By in Map Reduce

**Map**

<table>
<thead>
<tr>
<th>pageid</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25</td>
</tr>
<tr>
<td>2</td>
<td>25</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;1,25 &gt;</td>
<td>1</td>
</tr>
<tr>
<td>&lt;2,25 &gt;</td>
<td>1</td>
</tr>
</tbody>
</table>

**Shuffle**

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;1,25 &gt;</td>
<td>1</td>
</tr>
<tr>
<td>&lt;1,32 &gt;</td>
<td>1</td>
</tr>
<tr>
<td>&lt;2,25 &gt;</td>
<td>1</td>
</tr>
</tbody>
</table>

**Sort**

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;1,25 &gt;</td>
<td>1</td>
</tr>
<tr>
<td>&lt;2,25 &gt;</td>
<td>1</td>
</tr>
</tbody>
</table>

**Reduce**

<table>
<thead>
<tr>
<th>pageid</th>
<th>age</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>32</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>25</td>
<td>2</td>
</tr>
</tbody>
</table>
Beyond Hadoop – Spark
Spark

• One problem with Hadoop/MapReduce is that it is fundamental batch oriented, and everything goes through a read/write on HDFS for every step in a dataflow

• Spark was developed to leverage the main memory of distributed clusters and to, whenever possible, use only memory-to-memory data movement (with other optimizations)

• Can give up to 100fold speedup over MR
Spark

- Developed at the AMP lab here at Berkeley
- Open source version available from Apache
- DataBrick was founded to commercialize Spark
- Related software includes a very-high-speed Database – SparkDB
- Next time we will hear a talk (recorded) from Michael Franklin about BDAS & Spark