Announcements

• Today: MapReduce & flavor of Pig
• Next class: Cloud platforms and Quiz #6
• HW #4 is out and will be due 04/27

• Grading questions:
  – Class participation
  – Homeworks
  – Quizzes
  – Class project
“Data Systems” Landscape

Source: Lim et al, “How to Fit when No One Size Fits”, CIDR 2013.
Data Systems Design Space

<table>
<thead>
<tr>
<th>Internet</th>
<th>Shared memory</th>
<th>Latency</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private data center</td>
<td>RDBMS</td>
<td>Data Stores</td>
<td>HPC</td>
</tr>
</tbody>
</table>

Source: Adapted from Michael Isard, Microsoft Research.
MapReduce

- MapReduce = high-level programming model and implementation for large-scale parallel data processing
- Inspired by primitives from Lisp and other functional programming languages
- History:
  - 2003: built at Google
  - 2004: published in OSDI (Dean & Ghemawat)
  - 2005: open-source version Hadoop
  - 2005 - 2014: very influential in DB community
MapReduce Literature

Source: David Maier and Bill Howe, "Big Data Middleware", CIDR 2015.
Data Model

MapReduce knows files!
A file = a bag of (key, value) pairs

A MapReduce program:
• Input: a bag of (inputkey, value) pairs
• Output: a bag of (outputkey, values) pairs
Step 1: Map Phase

- User provides the map function:
  - Input: one \text{(input key, value)} pair
  - Output: bag of \text{(intermediate key, value)} pairs

- MapReduce system applies the map function in parallel to all \text{(input key, value)} pairs in the input file

- Results from the Map phase are stored to disk and redistributed by the intermediate key during the Shuffle phase
Step 2: Reduce Phase

- MapReduce system groups all pairs with the same intermediate key, and passes the bag of values to the Reduce function.

- User provides the Reduce function:
  - Input: (intermediate key, bag of values)
  - Output: bag of output values

- Results from Reduce phase stored to disk
Canonical Example

Pseudocode for counting the number of occurrences of each word in a large collection of documents

```java
map(String key, String input_value):
    // key: document name
    // input_value: document contents
    for each word in input_value:
        EmitIntermediate(word, "1");

reduce(String inter_key, Iterator inter_values):
    // inter_key: a word
    // inter_values: a list of counts
    int sum = 0;
    for each value in inter_values:
        sum += ParseInt(value);
    EmitFinal(inter_key, sum);
```

Source: Adapted from “MapReduce: Simplified Data Processing on Large Clusters” (original MapReduce paper).
MapReduce Illustrated

Source: Yahoo! Pig Team
Romeo, Romeo, wherefore art thou Romeo?

What, art thou hurt?

Source: Yahoo! Pig Team
MapReduce Illustrated

Romeo, Romeo, wherefore art thou Romeo?  
What, art thou hurt?

Romeo, 1  
Romeo, 1  
wherefore, 1  
art, 1  
thou, 1  
Romeo, 1

map  
map

reduce  
reduce

Source: Yahoo! Pig Team
Romeo, Romeo, wherefore art thou Romeo?

What, art thou hurt?

Source: Yahoo! Pig Team
MapReduce Illustrated

Romeo, Romeo, wherefore art thou Romeo?

What, art thou hurt?

Source: Yahoo! Pig Team
Rewritten as SQL

Documents(document_id, word)

```
SELECT word, COUNT(*)
FROM Documents
GROUP BY word
```

Observe:
Map + Shuffle Phases = Group By
Reduce Phase = Aggregate

More generally, each of the SQL operators that we have studied can be implemented in MapReduce
Relational Join

Employees(emp_id, last_name, first_name, dept_id)
Departments(dept_id, dept_name)

```
SELECT *
FROM Employees e, Departments d
WHERE e.dept_id = d.dept_id
```
**Relational Join**

Employees(`emp_id`, `emp_name`, `dept_id`)

<table>
<thead>
<tr>
<th><code>emp_id</code></th>
<th><code>emp_name</code></th>
<th><code>dept_id</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Alice</td>
<td>100</td>
</tr>
<tr>
<td>21</td>
<td>Bob</td>
<td>100</td>
</tr>
<tr>
<td>25</td>
<td>Carol</td>
<td>150</td>
</tr>
</tbody>
</table>

Departments(`dept_id`, `dept_name`)

<table>
<thead>
<tr>
<th><code>dept_id</code></th>
<th><code>dept_name</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Product</td>
</tr>
<tr>
<td>150</td>
<td>Support</td>
</tr>
<tr>
<td>200</td>
<td>Sales</td>
</tr>
</tbody>
</table>

**SELECT**

```
SELECT e.emp_id, e.emp_name, d.dept_id, d.dept_name
FROM Employees e, Departments d
WHERE e.dept_id = d.dept_id
```
Relational Join

Employees(emp_id, emp_name, dept_id)

<table>
<thead>
<tr>
<th>emp_id</th>
<th>emp_name</th>
<th>dept_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Alice</td>
<td>100</td>
</tr>
<tr>
<td>21</td>
<td>Bob</td>
<td>100</td>
</tr>
<tr>
<td>25</td>
<td>Carol</td>
<td>150</td>
</tr>
</tbody>
</table>

Departments(dept_id, dept_name)

<table>
<thead>
<tr>
<th>dept_id</th>
<th>dept_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Product</td>
</tr>
<tr>
<td>150</td>
<td>Support</td>
</tr>
<tr>
<td>200</td>
<td>Sales</td>
</tr>
</tbody>
</table>

Input:
Employee, 20, Alice, 100
Employee, 21, Bob, 100
Employee, 25, Carol, 150
Departments, 100, Product
Departments, 150, Support
Departments, 200, Sales

Output:
k=100, v=(Employee, 20, Alice, 100)
k=100, v=(Employee, 21, Bob, 100)
k=150, v=(Employee, 25, Carol, 150)
k=100, v=(Departments, 100, Product)
k=150, v=(Departments, 150, Support)
k=200, v=(Departments, 200, Sales)
Relational Join

Employees(emp_id, emp_name, dept_id)

<table>
<thead>
<tr>
<th>emp_id</th>
<th>emp_name</th>
<th>dept_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Alice</td>
<td>100</td>
</tr>
<tr>
<td>21</td>
<td>Bob</td>
<td>100</td>
</tr>
<tr>
<td>25</td>
<td>Carol</td>
<td>150</td>
</tr>
</tbody>
</table>

Departments(dept_id, dept_name)

<table>
<thead>
<tr>
<th>dept_id</th>
<th>dept_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Product</td>
</tr>
<tr>
<td>150</td>
<td>Support</td>
</tr>
<tr>
<td>200</td>
<td>Sales</td>
</tr>
</tbody>
</table>

Input:
- k=100, v=[(Employee, 20, Alice, 100), (Employee, 21, Bob, 100), (Departments, 100, Product)]
- k=150, v=[(Employee, 25, Carol, 150), (Departments, 150, Support)]
- k=200, v=[(Departments, 200, Sales)]

Output:
- 20, Alice, 100, Product
- 21, Bob, 100, Product
- 25, Carol, 150, Support
Hadoop on One Slide

Source: Huy Vo, NYU Poly
MapReduce Internals

- Single master node
- Master partitions input file by key into \( M \) splits (> servers)
- Master assigns workers (=servers) to the \( M \) map tasks, keeping track of their progress
- Workers write their output to local disk, partition into \( R \) regions (> servers)
- Master assigns workers to the \( R \) reduce tasks
- Reduce workers read regions from the map workers’ local disks
Key Implementation Details

• Worker failures:
  – Master pings workers periodically, looking for stragglers
  – When straggle is found, master reassigns splits to other workers
  – Stragglers are a main reason for slowdown
  – Solution: pre-emptive backup execution of last few remaining in-progress tasks

• Choice of M and R:
  – Larger than servers is better for load balancing
MapReduce Summary

• Hides scheduling and parallelization details
• Not most efficient implementation, but has great fault tolerance
• However, limited queries:
  – Difficult to write more complex tasks
  – Need multiple MapReduce operations
• Solution:
  – Use high-level language (e.g. Pig, Hive, Sawzall, Dremel, Tenzing) to express complex queries
  – Need optimizer to compile queries into MR tasks
MapReduce Summary

• Hides scheduling and parallelization details
• Not most efficient implementation, but has great fault tolerance
• However, limited queries:
  – Difficult to write more complex tasks
  – Need multiple MapReduce operations
• Solution:
  – Use high-level language (e.g. Pig, Hive, Sawzall, Dremel, Tenzing) to express complex queries
  – Need optimizer to compile queries into MR tasks
Pig & Pig Latin

- An engine and language for executing programs on top of Hadoop
- Logical plan → sequence of MapReduce ops
- Free and open-sourced (unlike some others) http://hadoop.apache.org/pig/
- ~70% of Hadoop jobs are Pig jobs at Yahoo!
- Being used at Twitter, LinkedIn, and other companies
- Available as part of Amazon, Hortonworks and Cloudera Hadoop distributions
Why use Pig?

Find the top 5 most visited sites by users aged 18 - 25. Assume: user data stored in one file and website data in another file.

Source: Yahoo! Pig Team
In MapReduce

```java
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
...
```

170 lines of code, 4 hours to write

Source: Yahoo! Pig Team
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 = join Fltrd by name, Pages by user;
Grpd = group Jnd by url;
Smmd = foreach Grpd generate group,
    COUNT(Jnd) as clicks;
Srtd = order Smmd by clicks desc;
Top5 = limit Srtd 5;
store Top5 into ‘top5sites’;

9 lines of code, 15 minutes to write

Source: Yahoo! Pig Team
Emerging Analytics Pipeline

New data sources

MapReduce

DBMS

Legacy databases

Operational databases

BI tools

Portals
Optional References

MapReduce: Simplified Data Processing on Large Clusters [Dean & Ghemawarat OSDI ‘04]

Pig Latin: A Not-So-Foreign Language for Data Processing [Olston et. al. SIGMOD ‘08]

Hive – A Petabyte Scale Data Warehouse Using Hadoop [Thusoo VLDB ‘09]

Designs, Lessons and Advice from Building Large Distributed Systems [Dean LADIS ‘09]

Tenzing: A SQL Implementation On The MapReduce Framework [Chattopadhyay et. al. VLDB ‘11]
Next Class

• Cloud platforms (guest speaker Jacob Walcik)
• Quiz #6