Parallelism in the Cloud: 

MapReduce

Kyle Jamieson
UCL Computer Science

COMPM038/COMPGZ06
31st January 2012
MapReduce programming model

- Simple abstraction that hides messy details of distributed computing:
  - **map**: \((k1, v1) \rightarrow \text{list}(k2, v2)\)
  - **reduce**: \((k2, \text{list}(v2)) \rightarrow \text{list}(v2)\)

- **Word count example**
  - Count the number of occurrences of each word in a large collection of documents

**Word count example:**

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

reduce(String key, Iterator values):
    // key: a word
    // values: a list of counts
    int result = 0;
    for each v in values:
        result += ParseInt(v);
    Emit(AsString(result));
```
In this section, we discuss the implementation of MapReduce across commodity machines, which is essential for the PC cluster.

The MapReduce framework is a parallel computation model designed to process large datasets efficiently. It consists of two main phases: Map and Reduce. Each of these phases runs on distributed machines.

### Map Phase
The map phase takes input files and produces intermediate files. These files are created on local disks, not moved across the network.

### Reduce Phase
The reduce phase takes the intermediate files from the map phase and reduces them to a smaller output. This phase is executed on the same distributed machines as the map phase.

### Distributed Machines
In a typical MapReduce setup, there is a Master node and multiple worker nodes. The Master node coordinates the overall execution, while worker nodes execute the map and reduce tasks.

### Example
Consider a simple MapReduce job that processes a file containing key-value pairs. Each worker node reads a partition of the input file, applies the map function to produce intermediate key-value pairs, and then sends these pairs to their corresponding Reduce worker nodes.

### Key-Value Pairs
In the context of MapReduce, key-value pairs are processed and reduced. Each key corresponds to a value, and theReduce function combines these values according to a specific operation defined by the user.

### Availability
Collectively, these distributed settings and the ability to execute MapReduce on commodity machines are what make it a versatile and widely used framework.
Data transfer rates for *sort* program

(a) Normal execution

Compare: *grep* program
**sort: effect of disabling backup tasks**

(a) Normal execution  
(b) No backup tasks
**sort: effect of machine failures**

(a) Normal execution

(b) No backup tasks

(c) 200 tasks killed
2003–2004: MapReduce catches on

MapReduce jobs in August 2004

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of jobs</td>
<td>29,423</td>
</tr>
<tr>
<td>Average job completion time</td>
<td>634 secs</td>
</tr>
<tr>
<td>Machine days used</td>
<td>79,186 days</td>
</tr>
<tr>
<td>Input data read</td>
<td>3,288 TB</td>
</tr>
<tr>
<td>Intermediate data produced</td>
<td>758 TB</td>
</tr>
<tr>
<td>Output data written</td>
<td>193 TB</td>
</tr>
<tr>
<td>Average worker machines per job</td>
<td>157</td>
</tr>
<tr>
<td>Average worker deaths per job</td>
<td>1.2</td>
</tr>
<tr>
<td>Average map tasks per job</td>
<td>3,351</td>
</tr>
<tr>
<td>Average reduce tasks per job</td>
<td>55</td>
</tr>
<tr>
<td>Unique map implementations</td>
<td>395</td>
</tr>
<tr>
<td>Unique reduce implementations</td>
<td>269</td>
</tr>
<tr>
<td>Unique map/reduce combinations</td>
<td>426</td>
</tr>
</tbody>
</table>
3rd February
Adapting to the Wireless Channel I
SampleRate (Bicket)

NEXT TIME