Simplified Data Processing on Large Clusters
(Without the Agonizing Pain)

Presented by Aaron Nathan
The Problem

• Massive amounts of data
  – >100TB (the internet)
  – Needs simple processing
• Computers aren’t perfect
  – Slow
  – Unreliable
  – Misconfigured
• Requires complex (i.e. bug prone) code
MapReduce to the Rescue!

• Common Functional Programming Model
  – Map Step
    \[
    \text{map (in\_key, in\_value) } \rightarrow \text{ list(out\_key, intermediate\_value)}
    \]
    • Split a problem into a lot of smaller subproblems
  – Reduce Step
    \[
    \text{reduce (out\_key, list(intermediate\_value)) } \rightarrow \text{ list(out\_value)}
    \]
    • Combine the outputs of the subproblems to give the original problem’s answer

• Each “function” is independent

• Highly Parallelizable
map(String input_key, String input_value):
// input_key: document name
// input_value: document contents
for each word w in input_value:
    EmitIntermediate(w, "1");

reduce(String output_key, Iterator intermediate_values):
// output_key: a word
// output_values: a list of counts int result = 0;
for each v in intermediate_values:
    result += ParseInt(v);
Emit(AsString(result));
Some Example Applications

- Distributed Grep
- URL Access Frequency Counter
- Reverse Web Link Graph
- Term-Vector per Host
- Distributed Sort
- Inverted Index
The Implementation

• Google Clusters
  – 100s-1000s Dual Core x86 Commodity Machines
  – Commodity Networking (100mbps/1Gbps)
  – GFS
• Google Job Scheduler
• Library linked in c++
Execution

User Program

Master

(1) fork
(1) fork
(1) fork

(2) assign map
(2) assign reduce.

worker

(3) read
(4) local write

worker

(5) remote read
worker

(6) write

output file 0

output file 1

split 0
split 1
split 2
split 3
split 4

Input files
Map phase
Intermediate files (on local disks)
Reduce phase
Output files
The Master

• Maintains the state and identify of all workers
• Manages intermediate values
• Receives signals from Map workers upon completion
• Broadcasts signals to Reduce workers as they work
• Can retask completed Map workers to Reduce workers.
In Case of Failure

• Periodic Pings from Master->Workers
  – On failure resets state of assigned task of dead worker

• Simple system proves resilient
  – Works in case of a 80 simultaneous machine failures!

• Master failure is unhandled.

• Worker Failure doesn’t effect output
  (output identical whether failure occurs or not)
  – Each map writes to local disk only
  – If a mapper is lost, the data is just reprocessed
  – Non-deterministic map functions aren’t guaranteed
Preserving Bandwidth

• Machines are in racks with small interconnects
  – Use location information from GFS
  – Attempts to put tasks for workers and input slices on the same rack
  – Usually results in LOCAL reads!
Backup Execution Tasks

• What if one machine is slow?
• Can delay the completion of the entire MR Operation!

• Answer: Backup (Redundant) Executions
  – Whoever finishes first completes the task!
  – Enabled towards the end of processing
Partitioning

- $M = \text{number of Map Tasks}$
  (the number of input splits)
- $R = \text{number of Reduce Tasks}$
  (the number of intermediate key splits)
- $W = \text{number of worker computers}$
- In General:
  - $M = \frac{\text{sizeof(Input)}}{64 \text{ MB}}$
  - $R = W \times n$ (where $n$ is a small number)
- Typical Scenario:
  - InputSize = 12 TB,
  - $M = 200,000$,
  - $R = 5000$,
  - $W = 2000$
Custom Partitioning

- Default Partitioned on intermediate key
  - $\text{Hash}(\text{intermediate\_key}) \mod R$

- What if user has apriori knowledge about the key?
  - Allow for user-defined hashing function
  - Ex. $\text{Hash}(\text{Hostname(url\_key)})$
The Combiner

• If reducer is associative and commutative
  – \((2 + 5) + 4 = 11\) or \(2 + (5 + 4) = 11\)
  – \((15+x) + 2 = 2 + (15+x)\)

• Repeated intermediate keys can be merged
  – Saves network bandwidth
  – Essentially like a local reduce task
I/O Abstractions

• How to get initial key value pairs to map?
  – Define an input “format”
  – Make sure splits occur in reasonable places
  – Ex: Text
    • Each line is a key/pair
    • Can come from GFS, bigTable, or anywhere really!
  – Output works analogously
Skipping Bad Records

• What if a user makes a mistake in map/reduce
• And only apparent on few jobs..

• Worker sends message to Master
• Skip record on >1 worker failure and tell others to ignore this record
Removing Unnecessary Development Pain

• Local MapReduce Implementation that runs on development machine

• Master has HTTP page with status of entire operation
  – Shows “bad records”

• Provide a Counter Facility
  – Master aggregates “counts” and displayed on Master HTTP page
A look at the UI (in 1994)

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 15 min 31 sec
1707 workers, 1 deaths

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>8841</td>
<td>1707</td>
<td>878934.6</td>
<td>621608.5</td>
<td>369459.8</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>0</td>
<td>500</td>
<td>369459.8</td>
<td>326986.8</td>
<td>326986.8</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>0</td>
<td>0</td>
<td>326986.8</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Counters

- Mapped (MB/s): 706.5
- Shuffle (MB/s): 419.2
- Output (MB/s): 0.0
- doc-index-hits: 4982870667
- docs-indexed: 17229926
- dups-in-index-merge: 0
- mr-operator-calls: 17272056
- mr-operator-outputs: 17229926

Performance Benchmarks

Sorting AND Searching
Search (Grep)

- Scan through $10^{10}$ 10C byte records (1TB)
- $M = 15000$, $R = 1$
- Startup time
  - GFS Localization
  - Program Propagation
- Peak -> 30 GB/sec!
Sort

- 50 lines of code
- Map->key + textline
- Reduce-> Identity
- \( M = 15000, R = 4000 \)
  - Partition on init bytes of intermediate key
- Sorts in 891 sec!
What about Backup Tasks?

(b) No backup tasks

(c) 200 tasks killed
And wait...it’s useful!

NB: 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>
Open Source Implementation

• Hadoop
  – Relies on HDFS
  – All interfaces look almost exactly like MapReduce paper
• There is even a talk about it today!
  – 4:15 B17 CS Colloquium:
  Mike Cafarella (Uwash)
Active Disks for Large-Scale Data Processing
The Concept

• Use aggregate processing power
  – Networked disks allow for higher throughput

• Why not move part of the application onto the disk device?
  – Reduce data traffic
  – Increase parallelism further
Shrinking Support Hardware...
Example Applications

• Media Database
  – Find similar media data by “fingerprint”

• Real Time Applications
  – Collect multiple sensor data quickly

• Data Mining
  – POS Analysis required adhoc database queries
Approach

• Leverage the parallelism available in systems with many disks

• Operate with a small amount of state, processing data as it streams off the disk

• Execute relatively few instructions per byte of data
Results- Nearest Neighbor Search

- Problem: Determine k items closest to a particular item in a database
  - Perform comparisons on the drive
  - Returns the disks closest matches
  - Server does final merge
Media Mining Example

• Perform low level image tasks on the disk!

• Edge Detection performed on disk
  – Sent to server as edge image
  – Server does higher level processing
Why not just use a bunch of PC’s?

• The performance increase is similar
• In fact, the paper essentially used this setup to actually benchmark their results!
• Supposedly this could be cheaper
• The paper doesn’t really give a good argument for this...
  – Possibly reduced bandwidth on disk IO channel
  – But who cares?
Some Questions

• What could a disk possibly do better than the host processor?
• What added cost is associated with this mediocre processor on the HDD?
• Are new dependencies are introduced on hardware and software?
• Perhaps other (better) places to do this type of local parallel processing?
• Maybe in 2001 this made more sense?