Paper List


Outline

- MapReduce Overview
- MapReduce Acceleration
- Energy issues of MapReduce
Outline

• MapReduce Overview

• MapReduce Acceleration

• Energy issues of MapReduce
Map and Reduce in Functional Programming

• **Map:** applies a given function element-wise to a list of elements and returns a list of results.

• **A Simple Example:**
  
  – L = (1, 2, 3, 4, 5);
  
  – f : Multiply an element by two;
  
  – Map(f, L) returns (2, 4, 6, 8, 10).
Map and Reduce in Functional Programming

- **Reduce**: deals with a combining function and a list of elements of some data structure. The Reduce then proceeds to combine elements of the data structure using the function in some systematic way.

- **A Simple Example**:
  - \( L = (1, 2, 3, 4, 5) \);
  - \( f : \) add two elements;
  - \( \text{Reduce}(f, L) \) apply \( f \) to \( L \) recursively, returns 15.
Motivation: Large Scale Data Processing

- Want to process lots of data
- Want to parallelize across hundreds/thousands of CPUs
- Want to make it easy
Basic Idea

• **Split** input file into pieces. (Generate a list)

• Instead of applying a function to the original data, apply it to each piece (or **split**). (Map a function to a list)

• **Collect** all the output from each piece to calculate the final result. (Reduce a list based on a function)

• Users can customize map and reduce functions.
Implementation

- **Master node:**
  - Split input data into pieces;
  - Assign tasks to map workers;
  - Inform reduce workers where the outputs of map phase are.

- **Worker node:**
  - Conduct mapping or Reducing;
  - Mappers run map function on *splits*;
  - Reducers run reduce function on *partitions*. 
Implementation

• Job
  – Defined on input file;
  – To finish one job, master node will invoke multiple map Tasks and reduce Tasks.

• Task
  – Defined on a split;
  – Assigned by master node to worker node.
Outline

• MapReduce Overview

• MapReduce Acceleration

• Energy issues of MapReduce
Problem: Too much data to pass from Map worker to Reduce worker

- Solution: Map nodes apply combiner functions to their local output.

Cited from [2]
Acceleration

• Problem: No reduce can start until map is complete.
  – Solution: Master redundantly executes “slow-moving” map tasks; uses result of first copy to finish.
  – Solution: Intermediate data is pipelined between mappers and reducers. Thus, reducers begin processing data as soon as it is produced by mappers.
Evaluation

Online Aggregation

Map - Reduce - Top 5 - Top 10 - Top 20

Progress

0% 20% 40% 60% 80% 100%

Time (seconds)

0 10 20 30 40 50 60 70 80 90 100 110 120 130 140

Cited from [2]
Outline

• MapReduce Overview
• MapReduce Acceleration
• Energy issues of MapReduce
Knobs

- Configuration [3][4]
- Machine On/Off [5][6][7]
- DVFS
- Temperature
- Workload [4][6]
Cluster Configuration

• General cost metric \[3\]
  – All cost over all benefits
  – Minimize the metric

\[
\frac{f_1(\text{energy}) f_2(\text{latency}) f_3(\text{number of machines})}{f_4(\text{workload size}) f_5(\text{degree of replication})} = \frac{f_1(E) f_2(t) f_3(M)}{f_4(W) f_5(R)}
\]
Cluster Configuration

• Their model [3]

\[
\frac{E}{WR} = \frac{Pt}{WR}
\]

E: Energy  
t: Delay  
M: Number of machines  
W: Workload  
R: Replication
Arguments

\[ \frac{E}{WR} = \frac{Pt}{WR} \quad \text{vs} \quad \frac{Et}{WR} = \frac{Pt^2}{WR} \]

- Too heavily prioritize the latency.
- A system could achieve better performance just by decreasing the workload.

E: Energy

E: Energy

t: Delay

M: Number of machines

W: Workload

R: Replication
Arguments

- Too heavily prioritize the number of machines.
- A system could achieve better performance just by using fewer machines.

\[
\frac{E}{WR} = \frac{Pt}{WR} \quad \text{vs} \quad \frac{EM}{WR} = \frac{E_{\text{per machine}}M^2}{WR}
\]

E: Energy
\(t\): Delay
M: Number of machines
W: Workload
R: Replication
<table>
<thead>
<tr>
<th>Map task</th>
<th>Reduce task</th>
<th>Component stressed</th>
<th>Workloads</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute $\pi$ w/ random samples</td>
<td>--</td>
<td>Map compute</td>
<td>20, 80, 160 hundreds of millions of samples</td>
</tr>
<tr>
<td>Read from input</td>
<td>--</td>
<td>Map read</td>
<td>40, 160, 320 millions of pairs</td>
</tr>
<tr>
<td>Write interm. pairs</td>
<td>--</td>
<td>Map write</td>
<td>40, 160, 320 millions of pairs</td>
</tr>
<tr>
<td>Write interm. pairs</td>
<td>Sort</td>
<td>Reduce sort</td>
<td>40, 160, 320 millions of pairs</td>
</tr>
<tr>
<td>--</td>
<td>Write to output</td>
<td>Reduce write</td>
<td>40, 160, 320 millions of pairs</td>
</tr>
<tr>
<td>--</td>
<td>Compute $\pi$ w/ random samples</td>
<td>Reduce compute</td>
<td>20, 80, 160 hundreds of millions of samples</td>
</tr>
<tr>
<td>Number of machines</td>
<td>Replication</td>
<td>Number of map jobs</td>
<td>Number of reduce jobs</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
<td>--------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>13</td>
<td>3</td>
</tr>
</tbody>
</table>
Result

Map Compute
1. Launch jobs as they arrive vs. queue up the job and launch them in **batches**;

2. For batched execution, launch all jobs on the queue at the same time vs. in a **staggered** fashion;

3. Use the standard HDFS block size of 64MB vs. **larger block sizes**;

4. Assign the default 4 task trackers per node vs. **more task trackers** per node.
### Result

<table>
<thead>
<tr>
<th>Configuration number</th>
<th>Batch?</th>
<th>Staggered launch?</th>
<th>HDFS block size</th>
<th>Task trackers per node</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>No</td>
<td>N/A</td>
<td>64MB</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>Yes</td>
<td>No</td>
<td>64MB</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>Yes</td>
<td>No</td>
<td>64MB</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>Yes</td>
<td>No</td>
<td>80MB</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>Yes</td>
<td>No</td>
<td>80MB</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>Yes</td>
<td>No</td>
<td>80MB</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>Yes</td>
<td>Yes</td>
<td>80MB</td>
<td>8</td>
</tr>
</tbody>
</table>

#### Normalized performance

<table>
<thead>
<tr>
<th>Config. #</th>
<th>Duration (s)</th>
<th>Energy (MJ)</th>
<th>Avg. power (W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>86524 ± 13</td>
<td>15.6 ± 0.14</td>
<td>180 ± 1.5</td>
</tr>
<tr>
<td>2</td>
<td>6830 ± 971</td>
<td>1.31 ± 0.19</td>
<td>192 ± 0.2</td>
</tr>
<tr>
<td>3</td>
<td>19756 ± 6930</td>
<td>3.71 ± 1.2</td>
<td>188 ± 3.8</td>
</tr>
<tr>
<td>4</td>
<td>8496 ± 2662</td>
<td>1.58 ± 0.47</td>
<td>186 ± 3.3</td>
</tr>
<tr>
<td>5</td>
<td>8180 ± 3664</td>
<td>1.54 ± 0.69</td>
<td>188 ± 2.0</td>
</tr>
<tr>
<td>6</td>
<td>7981 ± 2735</td>
<td>1.49 ± 0.55</td>
<td>186 ± 4.1</td>
</tr>
<tr>
<td>7</td>
<td>6266 ± 399</td>
<td>1.18 ± 0.09</td>
<td>188 ± 3.6</td>
</tr>
</tbody>
</table>
Knobs

- Cluster Configuration [3][4]
- Machine On/Off [5][6][7]
- DVFS
- Temperature
- Workload [4][6]
Main Challenge

- DFS: Files are stored in Distributed File Systems, each machine holds a subset of whole data. Turning off a set of machine can make some data unavailable.

- SLA: The cluster should satisfy Service Level Agreement even after turning off some machines.
• Covering Set (CS) [5] “On the Energy (In)efficiency of Hadoop Clusters”

• Hot Zone/Cold Zone (HC) [6] “GreenHDFS: Towards An Energy-Conserving, Storage-Efficient”

• All-In Strategy (AIS) [7] “Energy Management for MapReduce Clusters”
At least one *replica* of a data-block must be stored in a subset of nodes referred as covering set. Large numbers of nodes can be gracefully turned off without affecting the availability of data.
“On the Energy (In)efficiency of Hadoop Clusters”
Hot Zone/Cold Zone\[6\]

- Classify data by its “temperature”
  - They use age of file, as defined by the last access to the file, as the measure of temperature of the file.
  - Hot files stored on high performance hot zone servers.
  - Cold files stored on cold zone servers with large storage space.

“GreenHDFS: Towards An Energy-Conserving, Storage-Efficient”
Hot Zone/Cold Zone[6]

- 26% energy savings within a three-month trace simulation.

“GreenHDFS: Towards An Energy-Conserving, Storage-Efficient”
• In cases where there is a consistent low utilization period, AIS would **batch** the MR jobs in a queue.

• **Periodically** power up the entire system and run the entire batch of jobs as fast as they can and power off again.
“Energy Management for MapReduce Clusters”
Workload Distribution

• Explicitly
  – Modify task assignment algorithm.

• Inexplicitley
  – Change frequency;
  – Modify data storage strategy;
• HC stores “hot” data on running and powerful machines.
• MapReduce will try to run a task locally first.
• Most task will be run on hot servers.
Hot Zone/Cold Zone

- HC stores “hot” data on running and powerful machines.
- MapReduce will try to run a task locally first.
- Most task will be run on hot servers.
Workload Generation

• Motivation
  – Company Competitive concerns
  – Better evaluation
Trace Analysis
Trace Analysis
• Simple non-parametric statistics are average and standard deviation.
  – Problem: Distributions are irregular, skewed, and asymmetric. Hence, averages and standard deviations are insufficient.

• Solution: use percentiles.
  – The authors choose percentiles based on Gaussian model;
  – Five-number summary: 1\textsuperscript{st}, 25\textsuperscript{th}, 50\textsuperscript{th}, 75\textsuperscript{th}, 99\textsuperscript{th},
  – Seven-number summary and so on.
Thank you!