CIS 602-01: Scalable Data Analysis

MapReduce

Dr. David Koop
Cloud Use by Industry

How would you characterise the current presence of cloud in the following industries? % of respondents reporting a significant or pervasive presence

<table>
<thead>
<tr>
<th>Industry</th>
<th>Significant presence %</th>
<th>Pervasive presence %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Banking</td>
<td>52</td>
<td>7</td>
</tr>
<tr>
<td>Retail</td>
<td>42</td>
<td>1</td>
</tr>
<tr>
<td>Manufacturing</td>
<td>34</td>
<td>7</td>
</tr>
<tr>
<td>Education</td>
<td>31</td>
<td>8</td>
</tr>
<tr>
<td>Healthcare</td>
<td>43</td>
<td>7</td>
</tr>
</tbody>
</table>

Industry average: 31% significant, 7% pervasive


The second observation is that, as far as cloud has come, it has a long way to go. “Pervasive presence”—ready access and widespread deployment—averages out to only 7% across industries. The following industry analysis illustrates just how that rate of growth is expected to be.
Issues faced by cloud schedulers

- Machine and workload heterogeneity and variability
- Highly dynamic resource demand and availability
- Predictable, but poorly predicted, resource needs
- Resource class preferences and constraints

[Reiss et al., 2012]
Machine Configuration

<table>
<thead>
<tr>
<th>Number of machines</th>
<th>Platform</th>
<th>CPUs</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>6732</td>
<td>B</td>
<td>0.50</td>
<td>0.50</td>
</tr>
<tr>
<td>3863</td>
<td>B</td>
<td>0.50</td>
<td>0.25</td>
</tr>
<tr>
<td>1001</td>
<td>B</td>
<td>0.50</td>
<td>0.75</td>
</tr>
<tr>
<td>795</td>
<td>C</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>126</td>
<td>A</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>52</td>
<td>B</td>
<td>0.50</td>
<td>0.12</td>
</tr>
<tr>
<td>5</td>
<td>B</td>
<td>0.50</td>
<td>0.03</td>
</tr>
<tr>
<td>5</td>
<td>B</td>
<td>0.50</td>
<td>0.97</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>1.00</td>
<td>0.50</td>
</tr>
<tr>
<td>1</td>
<td>B</td>
<td>0.50</td>
<td>0.06</td>
</tr>
</tbody>
</table>

[Reiss et al., 2012]
Job Duration (Log-scale inverted CDF)

Figure 2: Log-log scale inverted CDF of job durations. Only the duration for which the job runs during the trace time period is known; thus, for example, we do not observe durations longer than around 700 hours. The thin, black line shows all jobs; the thick line shows production-priority jobs; and the dashed line shows non-production priority jobs.

3.3 Job durations

Job durations range from tens of seconds to essentially the entire duration of the trace. Over 2000 jobs (from hundreds of distinct users) run for the entire trace period, while a majority of jobs last for only minutes. We infer durations from how long tasks are active during the one month time window of the trace; jobs which are cut off by the beginning or end of the trace are a small portion (<1%) of jobs and consist mostly of jobs which are active for at least several days and so are not responsible for us observing many shorter job durations. These come from a large portion of the users, so it is not likely that the workload is skewed by one particular individual or application. Consistent with our intuition about priorities correlating with job types, production priorities have a much higher proportion of long-running jobs and the 'other' priorities have a much lower proportion. But slicing the jobs by priority or 'scheduling class' (which the trace providers say should reflect how latency-sensitive a job is) reveals a similar heavy-tailed distribution shape with a large number of short jobs.

3.4 Task shapes

Each task has a resource request, which should indicate the amount of CPU and memory space the task will require. (The requests are intended to represent the submitter's predicted "maximum" usage for the task.) Both the amount of the resources requested and the amount actually used by tasks varies by several orders of magnitude; see Figures 3 and 4, respectively. These are not just outliers. Over 2000 jobs request less than 0.0001 normalized units of memory per task, and over 8000 jobs request more than 0.1 units of memory per task. Similarly, over 70000 jobs request less than 0.0001 units of CPU per task, and over 8000 request more than 0.1 units of CPU. Both tiny and large resource requesting jobs include hundreds of distinct users, so it is not likely that the particularly large or small requests are caused by the quirky demands of a single individual or service.

We believe that this variety in task "shapes" has not been seen in prior workloads, if only because most schedulers simply do not support this range of sizes. The smallest resource requests are likely so small that it would be difficult for any VM-based scheduler to run a VM using that little memory. (0.0001 units would be around 50MB if the largest machines in the cluster had 512GB of memory.) Also, any slot-based scheduler, which includes all HPC and Grid installations we are aware of, would be unlikely to have thousands of slots per commodity machine.

The ratio between CPU and memory requests also spans a large range. The memory and CPU request sizes are correlated, but weakly (linear regression $R^2 \approx 0.14$). A large number jobs request 0 units of CPU — presumably they require so little CPU they can depend on running in the 'left-over' CPU of a machine; it makes little sense to talk about the CPU:memory ratio without adjusting these. Rounding these requests to the next largest request size, the

[Reiss et al., 2012]
5.1 Usage overview

This stability provides better predictions of resource usage than the
requests. Though there is a lot of task churn, overall resource usage is stable.

5.2 Usage stability

Figures 8 and 9 show the utilization on the cluster over the 29 day
period. We evaluated utilization both in terms of the measured resource
consumption (left side of figure) and allocations (requested resources of running tasks; right side of figure). Based on the difference between the two, we can see how much of the requested resources was actually used.

The usage statistics (green) and allocated (blue/darkest color) resources are shown in the plots. The dashed line at 1.0 represents the maximum capacity of the cluster.

[Reiss et al., 2012]
Assignment 2


• New York City Trees
  - 680,000+ trees
  - Use WebGL for visualization
  - Use a Python bridge (mapboxgl)
  - Use the fork!
  - Label subproblems and answers
  - conda install pyproj before pip
  - Due Thursday
MapReduce: Simplified Data Processing on Large Clusters

J. Dean and S. Ghemawat
Latency Numbers Every Programmer Should Know

- 1ns
- L1 cache reference: 1ns
- Branch mispredict: 3ns
- L2 cache reference: 4ns
- Mutex lock/unlock: 17ns
- Main memory reference: 100ns
- Send 2,000 bytes over commodity network: 125ns
- SSD random read: 16,000ns ≈ 16µs
- Compress 1KB with Zippy: 2,000ns ≈ 2µs
- Read 1,000,000 bytes sequentially from memory: 6,000ns ≈ 6µs
- Round trip in same datacenter: 500,000ns ≈ 500µs
- Read 1,000,000 bytes sequentially from disk: 1,000,000ns = 1ms
- Disk seek: 3,000,000ns = 3ms

[Interactive Latency (year=2017), C. Scott]
Latency Numbers Every Programmer Should Know

- Main memory reference: 100ns
  
  1,000ns ≈ 1µs

- Compress 1KB with Zippy: 2,000ns ≈ 2µs

- 10,000ns ≈ 10µs

- Send 2,000 bytes over commodity network: 125ns
  
  SSD random read: 16,000ns ≈ 16µs

- Read 1,000,000 bytes sequentially from memory: 6,000ns ≈ 6µs

- Round trip in same datacenter: 500,000ns ≈ 500µs
  
  1,000,000ns = 1ms

- Read 1,000,000 bytes sequentially from SSD: 98,000ns ≈ 98µs
  
  Disk seek: 3,000,000ns ≈ 3ms

- Read 1,000,000 bytes sequentially from disk: 1,000,000ns ≈ 1ms

- Packet roundtrip CA to Netherlands: 150,000,000ns ≈ 150ms

[Interactive Latency (year=2017), C. Scott]
MapReduce

• A simple programming model that applies to many large-scale computing problems

• **Hide messy details** in MapReduce runtime library:
  - automatic parallelization
  - load balancing
  - network and disk transfer optimizations
  - handling of machine failures
  - robustness
  - improvements to core library benefit all users of library!

[J. Dean, 2009]
Typical Problem Solved by MapReduce

• Read a lot of data
• **Map**: extract something you care about from each record
• Shuffle and Sort
• **Reduce**: aggregate, summarize, filter, or transform
• Write the results
MapReduce

P. Krzyzanowski
Key Features of MapReduce

- Fault Tolerance
  - Both Worker and Master Failure
- Locality
  - Move the computation to the data
Data Transfer Rates

(a) Normal execution  (b) No backup tasks  (c) 200 tasks killed

[Dean and Ghemawat, 2004]
Questions?
Controversy
Dewitt and Stonebraker on MapReduce (2008)

"We are amazed at the hype that the MapReduce proponents have spread about how it represents a paradigm shift in the development of scalable, data-intensive applications.

1. A giant step backward in the programming paradigm for large-scale data intensive applications

2. A sub-optimal implementation, in that it uses brute force instead of indexing

3. Not novel at all -- it represents a specific implementation of well known techniques developed nearly 25 years ago

4. Missing most of the features that are routinely included in current DBMS

5. Incompatible with all of the tools DBMS users have come to depend on"

[via Bill Howe]
## Parallel Databases vs. MapReduce

<table>
<thead>
<tr>
<th></th>
<th>Parallel DBMS</th>
<th>MapReduce</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema Support</td>
<td>✓</td>
<td>Not out of the box</td>
</tr>
<tr>
<td>Indexing</td>
<td>✓</td>
<td>Not out of the box</td>
</tr>
<tr>
<td>Programming Model</td>
<td>Declarative (SQL)</td>
<td>Imperative (C/C++, Java, …)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Extensions through Pig and Hive</td>
</tr>
<tr>
<td>Optimizations (Compression, Query Optimization)</td>
<td>✓</td>
<td>Not out of the box</td>
</tr>
<tr>
<td>Flexibility</td>
<td>Not out of the box</td>
<td>✓</td>
</tr>
<tr>
<td>Fault Tolerance</td>
<td>Coarse grained techniques</td>
<td>✓</td>
</tr>
</tbody>
</table>

[via J. Freire, from Pavlo et al., 2009 & Stonebraker et al., 2010]
Comparing Load Times

**Figure 1:** Load Times – Grep Task Data Set (535MB/node)

**Figure 2:** Load Times – Grep Task Data Set (1TB/cluster)

[Pavlo et al., 2009]
Comparing Execution Times

**Figure 4:** Grep Task Results – 535MB/node Data Set

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Vertica</th>
<th>DBMS-X</th>
<th>Hadoop</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Nodes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 Nodes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25 Nodes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50 Nodes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100 Nodes</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 5:** Grep Task Results – 1TB/cluster Data Set

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Vertica</th>
<th>DBMS-X</th>
<th>Hadoop</th>
</tr>
</thead>
<tbody>
<tr>
<td>25 Nodes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50 Nodes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100 Nodes</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[Note: Figures show execution times for Vertica, DBMS-X, and Hadoop across different node counts.]

[Source: Pavlo et al., 2009]
Dean and Ghemawat's Response

- Heterogeneous systems:
  - There is data in lots of different storage systems: files, Google Bigtable, column stores
  - Can write your own storage backend to combine data
  - Parallel databases require all data to be loaded

- Indices:
  - Database indexing techniques can be used for MapReduce
  - Push data processing to the database
  - Or, use partitioning of data

[via J. Freire, Dean and Ghemawat, 2010]
Dean and Ghemawat's Response (continued)

- Complex functions:
  - MapReduce was designed for complex tasks that do not fit well into the relational paradigm
  - RDBMS have user-defined functions but buggy/missing

- Structured data and schemas:
  - Google's MapReduce supports Protocol Buffer that allows an optimized binary representation
  - [https://developers.google.com/protocol-buffers/](https://developers.google.com/protocol-buffers/)

[ via J. Freire, Dean and Ghemawat, 2010 ]
Pig: Making Hadoop Easy

A. Gates
Test 1