Large-Scale Sorting: Breaking World Records

Mike Conley

CSE 124 Guest Lecture

12 March 2015
Sorting

• Given an array of items, put them in order

<table>
<thead>
<tr>
<th>5</th>
<th>2</th>
<th>8</th>
<th>0</th>
<th>2</th>
<th>5</th>
<th>4</th>
<th>9</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

• Many algorithms exist
  – Mergesort, Bubblesort, Quicksort, etc
  – Efficient $O(n \log n)$ algorithms
Sorting

int[] data = loadData();
Arrays.sort(data);

• Java documentation says this uses quicksort
• What’s wrong with this?
  – What if ‘data’ is 100TB?
Sorting Huge Data Sets

```java
int[] data = loadData();
Arrays.sort(data);
```

• Suppose ‘data’ is 100TB in size
• What happens?
  – Crashes? Swapping/thrashing?
• Do you even have 100TB of disk space?
Sorting Constraints

• Single server doesn’t have enough disk space
  – Use many servers
  – Requires high speed network

• Cluster doesn’t have enough RAM
  – This laptop has 8GB, need more than 11,000
  – Requires high speed storage

• Enough CPU to actually sort the data
  – Almost an afterthought
Measuring Sorting Efficiency: GraySort

- Sorting contest [Jim Gray et al., 1985]
  - Importance of the IO subsystem
  - 1985: Sort 100MB
  - 1999: Sort 1TB
  - 2009: Sort 100 TB

Sorting as a Benchmark

• Canonical problem
  – Underlies many applications
  – MapReduce for example

• Broadly applicable
  – Business transaction databases
  – Internet search
  – Facebook

• “Easy” (problem, solution, theory)
Inefficiency of Sorting Systems

• Analysis of GraySort contest results*
  – On average: 94% disk IO idle; 33% of CPU idle

• Case study: 2009 Yahoo! Hadoop Cluster
  – Sorted 100TB with 3,452 nodes in ≈3 hours
  – 1% disk efficiency

* Anderson and Tucek, “Efficiency matters!” SIGOPS OSR 44, 1 (March 2010)
3452 nodes at 1% efficiency
35 nodes at 100% efficiency
Daytona 100TB GraySort

• 2009 – Hadoop by Yahoo!
  – 578 GB/min with 3452 servers

• 2011 – TritonSort
  – 725 GB/min with 52 servers
  – 25% faster
  – >8000% more efficient
## Top 2014

<table>
<thead>
<tr>
<th>Daytona</th>
<th>2-way tie:</th>
<th>2014, 4.35 TB/min</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TritonSort</strong></td>
<td>100 TB in 1,378 seconds</td>
<td>(32 vCores - 2.50Ghz Intel Xeon E5-2670 v2, 244GB memory, 8x800 GB SSD)</td>
</tr>
<tr>
<td></td>
<td>186 Amazon EC2 i2.8xlarge nodes x</td>
<td>Michael Conley, Amin Vahdat, George Porter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>University of California, San Diego</td>
</tr>
<tr>
<td>Gray</td>
<td>2014, 4.27 TB/min</td>
<td><strong>Apache Spark</strong></td>
</tr>
<tr>
<td></td>
<td>100 TB in 1,406 seconds</td>
<td>(32 vCores - 2.5Ghz Intel Xeon E5-2670 v2, 244GB memory, 8x800 GB SSD)</td>
</tr>
<tr>
<td></td>
<td>207 Amazon EC2 i2.8xlarge nodes x</td>
<td>Reynold Xin, Parviz Deyhim, Xiangrui Meng, Ali Ghodsi, Matei Zaharia</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Databricks</strong></td>
</tr>
</tbody>
</table>
• 2010 – 2 world records
  – 100TB, 60 second
• 2011 – 5 world records
  – 100TB (2), 60 second, energy efficiency (2)
• 2014 – 3 new world records, and 2 from 2011
  – 100TB, public cloud (2)
Live Demo
How do you build a sorting system?

• Needs to be distributed
  – Size constraints we saw earlier

• Distribution sort
  – Class of algorithms
  – Distribute – intelligently split data
  – Process – usually local sorting
  – Combine – build the final sorted output
Example: Distributed Mergesort

Phase One

Disks → Read → Sort → Distribute → Network → Heap-Sort → Write → Disks

Phase Two

Disks → Read → Merge-Sort → Write → Disks
Mergesort

5 2 8 0 2 5 4 9 0 1 0 0

5 2 8 0 2 5

4 9 0 1 0 0

0 2 2 5 5 8

0 0 0 1 4 9

0 0 0 0 1 2 2 4 5 5 8 9
Distributed Mergesort

5 2 8 0 2 5

2 0 2 4 0 1 0 0

4 9 0 1 0 0

5 8 5 9

2 0 2 4

0 1 0 0

5 8 5 9

0 2 2 4

0 0 0 1

5 5 8 9

0 0 0 0 1 2 2 4

5 5 8 9
Problems?

• Choice of distribution function
Distributed Mergesort
Problems?

• Choice of distribution function
  – Random sampling?
    • What if the data is sorted already?

• Network transfer is expensive
Distributed Mergesort
Problems?

• Choice of distribution function
  – Random sampling?
    • What if the data is sorted already?

• Network transfer is expensive
  – Also challenging to implement
    • Getting 40 Gb/s out of TCP/IP sockets

• What about all this merging?
Distributed Mergesort

<table>
<thead>
<tr>
<th>5</th>
<th>2</th>
<th>8</th>
<th>0</th>
<th>2</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Random I/O (seeks)

<table>
<thead>
<tr>
<th>0</th>
<th>2</th>
<th>2</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>2</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4</th>
<th>9</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>8</td>
<td>5</td>
<td>9</td>
<td>5</td>
<td>8</td>
<td>5</td>
</tr>
</tbody>
</table>
TritonSort

Phase One

Disks \rightarrow \text{Read} \rightarrow \text{Distribute} \rightarrow \text{Cluster Nodes} \rightarrow \text{Network} \rightarrow \text{Write} \rightarrow \text{Disks}

Phase Two

Disks \rightarrow \text{Read} \rightarrow \text{Sort} \rightarrow \text{Write} \rightarrow \text{Disks}
TritonSort

```
5 2 8 0 2 5
0 0 0 0 2 1 2
4 9 0 1 0 0 0
4 5 9 8 5
0 0 0 0
4 5 5 8 9
1 2 2
```
TritonSort

Random I/O
But you can batch writes!

Sequential I/O
One partition at a time
The 2-IO Property

• Fundamental Result (★)
  – If: Data set >> memory
  – Then: At least 2 I/Os per record

• 2-IO Property
  – Exactly 2 I/Os per record

• Counterexample: Hadoop writes
  – Local mapper output (fault tolerance)
  – Shuffle output (merge sort)
  – Reducer output (stable storage)

Why is 2-IO Important?

• I/O (storage, network) is incredibly expensive
  – Examples in CSE124
    • TCP performance
    • Data center network topologies
    • Load balancing + fault tolerance

• The fewer I/O’s the better
Performance Debugging

• Typically debugging = correctness
  – gdb, stack traces, printf
• Performance bugs
  – Hardware capable of X
  – Software performing at Y < X
  – Typically harder to fix
    • Run profiler on a single server
    • What about a distributed setting?
Distributed Performance Debugging

• Measure system to get data (logfiles)
• Analyze data with tools (scripts)
• Display data with more tools (GUI)
• Identify a problem (bottleneck)
• Fix the problem (write some code)
• Repeat
### Job 'batch_17' Phase 'phase_two' Epoch 0

(click for per-host info, click headers to sort)

<table>
<thead>
<tr>
<th>Worker Type</th>
<th>Total Input Size (MB)</th>
<th>Mean Stage Runtime (s)</th>
<th>Max Stage Runtime (s)</th>
<th>Worker Runtime Span (s)</th>
<th>Mean Pipeline Saturation Time (us)</th>
<th>% Total Time Waiting for Memory</th>
<th>% Total Time Waiting to Enqueue Work</th>
<th>% Total Time Waiting for Work</th>
</tr>
</thead>
<tbody>
<tr>
<td>reader_converter</td>
<td>2440000.0</td>
<td>1173.488799</td>
<td>1196.450483</td>
<td>1196.65691</td>
<td>14403</td>
<td>0.00%</td>
<td>77.04%</td>
<td>14.01%</td>
</tr>
<tr>
<td>reader</td>
<td>2440000.0</td>
<td>1159.086038</td>
<td>1184.736004</td>
<td>1185.197019</td>
<td>6</td>
<td>0.38%</td>
<td>67.33%</td>
<td>0.00%</td>
</tr>
<tr>
<td>sorter</td>
<td>2635200.0</td>
<td>1210.8217</td>
<td>1234.491881</td>
<td>1232.165139</td>
<td>6675187</td>
<td>0.06%</td>
<td>79.47%</td>
<td>1.05%</td>
</tr>
<tr>
<td>reducer</td>
<td>2635200.0</td>
<td>1236.119065</td>
<td>1259.616896</td>
<td>1253.947651</td>
<td>10307289</td>
<td>0.05%</td>
<td>89.24%</td>
<td>0.69%</td>
</tr>
<tr>
<td>writer</td>
<td>2440000.0</td>
<td>1270.14538</td>
<td>1307.050998</td>
<td>1301.376881</td>
<td>14329177</td>
<td>0.00%</td>
<td>0.00%</td>
<td>13.33%</td>
</tr>
</tbody>
</table>
Distributed Logging

- Need logs from all servers to generate plots
- Process a whole for system-wide results
- Log processing is a distributed system!
  - Does your logging system need a logging system?
NFS Saves the Day

• Sysnet group has a NetApp filer
  – More than 100TB of shared storage
  – Accessible via NFS mount

• NFS allows all servers log to shared file system
  – Eliminates need for complex logging mechanism
  – Use standard read(), write() interface
Fault Tolerance

• Fault tolerance is another important topic covered in this course
• TritonSort has no fault tolerance
  – It’s a feature not a bug 😊
Coarse Grained Fault Tolerance

• Typically fault tolerance is fine grained
  – Small piece of work is redone
    • Read a block from a mirrored disk in RAID1
    • Repeat a map task in Hadoop MapReduce

• Coarse grained fault tolerance
  – Large piece of work is redone (eg. the whole job)
  – Hope MTBF >> job runtime
  – Can be true for small, efficient clusters
Coarse Grained Fault Tolerance

% Improvement vs. Task-Level FT

Cluster Size

Job-level FT 1x faster
Job-level FT 2x faster
Job-level FT 4x faster
Job-level FT 8x faster
Fighting the Disks

- Consumer-grade 500GB drives suffer from non-negligible rate of Latent Sector Errors
  - Extremely likely to happen at least once during 100TB sort run
  - MTBF < job runtime!

- Solution: buy better, enterprise-grade disks!
<table>
<thead>
<tr>
<th>Time</th>
<th>Event Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>November 2009</td>
<td>Project starts</td>
</tr>
<tr>
<td>December 2009</td>
<td>Initial design (Heap/Merge) complete</td>
</tr>
<tr>
<td>January – March 2010</td>
<td>Optimizing Heap/Merge</td>
</tr>
<tr>
<td>March 2010</td>
<td>Switched from quicksort to radix sort</td>
</tr>
<tr>
<td>April 23 2010</td>
<td>Abandoned Heap/Merge architecture</td>
</tr>
<tr>
<td>April 26 2010</td>
<td>Logical Disk architecture v1 complete</td>
</tr>
<tr>
<td>May 12 2010</td>
<td>MinuteSort implementation begins</td>
</tr>
<tr>
<td></td>
<td>GraySort attempts begin</td>
</tr>
<tr>
<td>May 14 2010</td>
<td>MinuteSort implementation complete</td>
</tr>
<tr>
<td>May 15 2010</td>
<td>9:30 PM – GraySort succeeds</td>
</tr>
<tr>
<td></td>
<td>10:23 PM – MinuteSort run finalized</td>
</tr>
</tbody>
</table>
Systems Building Blocks

• Building high-performance, large-scale systems is hard

• Many technologies to choose from
  – Language choice: Java? C/C++? Something else?
  – Communication: RPC? Message passing? Sockets?
  – Storage management: RAID? JBOD?
  – File system: XFS? Ext3/4?

• Stuff you learn in classes matters!
Moving to the Cloud

• Amazon Elastic Compute Cloud
  – Rent virtual machines (VMs) for ¢/$ per hour
  – Highly configurable – 42 VM types
  – Access to high speed flash devices and 10 Gb/s networking
  – “Unlimited” resources
Moving to the Cloud

• Previously, storage was the main bottleneck
  – HDDs are slow!

• In the cloud, network is usually the bottleneck
  – High speed flash devices > 10 Gb/s
  – Network oversubscription
Network Topologies

- 2010/2011 records used star topology
  - 52 servers on a single switch
- Full bisection bandwidth
Oversubscription in the Cloud

• Full bisection bandwidth too expensive
• Underutilized links are fast (nearby)
• Heavily utilized links are slow (far away)
Network Placement

• Amazon offers Placement Groups
  – VMs launched together are “closer” in the network
  – Better performance, but restricts cluster size
  – Launching 100 VMs can fail...

• Placement groups are critical
  – SSDs capable of 13-14 Gb/s read/write
  – 8-9 Gb/s network bandwidth is good
Local vs. Network Storage

• TritonSort shines on locally attached storage
  – HDDs, SSDs
• If these devices “fail”, your data is lost
  – Hardware failure
  – Virtualization failure (hypervisor)
  – Power loss
  – VM migration
Network Attached Storage

• Cloud providers offer stable network attached storage services
  – Amazon S3, EBS
  – Google Cloud Storage

• Storage devices are virtualized
  – Appears to be a normal block device
  – Persist across VM shutdown, movable, etc
Network Attached Storage

• Network storage service is a huge bottleneck!

• Example: Amazon EBS
  – Best case scenario – 500 MB/s
  – Network interface – 1100 MB/s
  – Local SSDs – 1700 MB/s

• Bottom line: network is a major problem
  – We need you guys to go fix this!
Applicability of Sorting

Phase One

Disks → Read → Distribute → Cluster Nodes

Phase Two

Network → Write → Disks

Disks → Read → Sort → Write → Disks
Applicability of Sorting

Phase One

Disks → Read → Map → Distribute

Cluster Nodes

Phase Two

Disks → Network → Write → Disks

Disks → Read → Sort → Reduce → Write → Disks
• Themis MapReduce
  – TritonSort + map() + reduce()
  – 2-IO property still holds! 😊
  – Coarse grained fault tolerance 😞

• Compare to Hadoop MapReduce
  – Many more I/Os 😞
  – Fine grained fault tolerance 😊
Performance

![Bar chart showing performance throughput (MB/s/disk) for various benchmarks. The chart compares Phase One and Phase Two for each benchmark.](chart.png)
Call to Action

• Lots of cool networking problems to solve
  – Example: high performance sorting

• Building real systems is hard!
  – But use many of the topics in this course

• State of the art is really bad
  – You can make a huge difference in the world!

• Thanks! Questions?