Flat Datacenter Storage

Edmund B. Nightingale, Jeremy Elson, Jinliang Fan, Owen Hofmann, Jon Howell, Yutaka Suzue

Presented by Rashmi Vinayak
9/21/2015

(Slides sourced from Jeremy Elson’s presentation at OSDI 2012 and Alex Rasmussen’s presentation at Papers We Love SF #11 with some modifications)
Move the Computation to the Data!
Why move computation close to data?

Because remote access is slow due to oversubscription.
Locality adds complexity

• Need to be aware of where the data is
  – Non-trivial scheduling algorithm
  – Moving computations around is not easy

• Need a data-parallel programming model
  – cannot express all desired computations efficiently
What if the network is *not oversubscribed*?
Consequences

- No local vs. remote disk distinction
- Simpler work schedulers
- Simpler programming models
FDS
Object Storage
Assuming
No Oversubscription
Outline

• Introduction
• Architecture and API
• Metadata management
• Replication and Recovery
• Network
• Evaluation
• Discussion
• One-minute plug
Blob 0xbadf00d

<table>
<thead>
<tr>
<th>Tract 0</th>
<th>Tract 1</th>
<th>Tract 2</th>
<th>...</th>
<th>Tract n</th>
</tr>
</thead>
</table>

8 MB

- CreateBlob
- OpenBlob
- CloseBlob
- DeleteBlob
- GetBlobSize
- ExtendBlob
- ReadTract
- WriteTract
API Guarantees

• Tractserver writes are **atomic**
• Calls are **asynchronous**
  - Allows deep pipelining
• **Weak consistency** to clients
Outline

• Introduction
• Architecture and API
• **Metadata management**
• Replication and Recovery
• Network
• Evaluation
• Discussion
• One-minute plug
## Tract Locator Table

<table>
<thead>
<tr>
<th>Tract Locator</th>
<th>Version</th>
<th>TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>D</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>A</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>C</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>F</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Tract_Locator = TLT[(Hash(GUID) + Tract) % len(TLT)]
Tract_Locator = TLT[(Hash(GUID) + Tract) % len(TLT)]

Randomize blob’s tractserver, even if GUIDs aren’t random (uses SHA-1)
Tract_Locator = TLT[(Hash(GUID) + {\textbf{Tract}}) \mod \text{len(TLT)}] \\

Large blobs use all TLT entries uniformly
Tract_Locator = TLT[(Hash(GUID) - 1) % len(TLT)]

Blob Metadata is Distributed
# Cluster Growth

<table>
<thead>
<tr>
<th>Tract Locator</th>
<th>Version</th>
<th>TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>D</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>A</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>C</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>F</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
## Cluster Growth

<table>
<thead>
<tr>
<th>Tract Locator</th>
<th>Version</th>
<th>TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>NEW / A</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>D</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>NEW / A</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>NEW / C</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>F</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
# Cluster Growth

<table>
<thead>
<tr>
<th>Tract Locator</th>
<th>Version</th>
<th>TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>NEW</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>A</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>A</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>NEW</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>NEW</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>A</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Outline

• Introduction
• Architecture and API
• Metadata management
• Replication and Recovery
• Network
• Evaluation
• Discussion
• One-minute plug
Replication

• For both *fault-tolerance* and *availability*

• Supports variable replication factors for different blobs
  – 1-replica for intermediate computations, 3 replicas for archival data and over-replicate popular blobs
  – replication factor stored in the blob meta data
## Replication

<table>
<thead>
<tr>
<th>Tract Locator</th>
<th>Version</th>
<th>Replica 1</th>
<th>Replica 2</th>
<th>Replica 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>A</td>
<td>C</td>
<td>Z</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>A</td>
<td>D</td>
<td>H</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>A</td>
<td>E</td>
<td>M</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>A</td>
<td>F</td>
<td>G</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>A</td>
<td>G</td>
<td>P</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Replication

- Create, Delete, Extend:
  - client writes to primary
  - primary 2PC to replicas
- Write to all replicas
- Read from random replica
Recovery

- All **disk pairs** appear in the table
- $n$ disks each recover $1/n$th of the lost data in parallel
Recovery

More disks → faster recovery
Outline

• Introduction
• Architecture and API
• Metadata management
• Replication and Recovery
• Network
• Evaluation
• Discussion
• One-minute plug
How to make network not a bottleneck?

Bandwidth is (was?) scarce in datacenters due to oversubscription.

CLOS networks:
[Al-Fares 08, Greenberg 09] full bisection bandwidth at datacenter scales.
How to make network not a bottleneck?

Bandwidth is (was?) scarce in datacenters due to oversubscription.

CLOS networks: [Al-Fares 08, Greenberg 09] full bisection bandwidth at datacenter scales.

Disks: ≈ 1Gbps bandwidth each.
How to make network not a bottleneck?

Bandwidth is (was?) scarce in datacenters due to oversubscription.

CLOS networks:
[Al-Fares 08, Greenberg 09]
full bisection bandwidth at datacenter scales

FDS:
Provision the network sufficiently for every disk
1G of network per disk
Outline

• Introduction
• Architecture and API
• Metadata management
• Replication and Recovery
• Network
• Evaluation
• Discussion
• One-minute plug
Read/Write Performance

Triple-Replicated Tractbservers, 10G Clients
## Failure Recovery Results

<table>
<thead>
<tr>
<th>Disks in Cluster</th>
<th>Disks Failed</th>
<th>Data Recovered</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1</td>
<td>47 GB</td>
<td>19.2 ± 0.7s</td>
</tr>
<tr>
<td>1,000</td>
<td>1</td>
<td>47 GB</td>
<td>3.3 ± 0.6s</td>
</tr>
<tr>
<td>1,000</td>
<td>1</td>
<td>92 GB</td>
<td>6.2 ± 6.2s</td>
</tr>
<tr>
<td>1,000</td>
<td>7</td>
<td>655 GB</td>
<td>33.7 ± 1.5s</td>
</tr>
</tbody>
</table>

- We recover at about 40 MB/s/disk + detection time
- 1 TB failure in a 3,000 disk cluster: ~17s
High Application Performance: Minute Sort

<table>
<thead>
<tr>
<th>MinuteSort—Daytona class (general purpose)</th>
<th>FDS, 2012</th>
<th>Yahoo!, Hadoop, 2009 [25]</th>
</tr>
</thead>
<tbody>
<tr>
<td>256</td>
<td>1,033</td>
<td>1,401 GB</td>
</tr>
<tr>
<td>1,408</td>
<td>5,632</td>
<td>500 GB</td>
</tr>
<tr>
<td>59s</td>
<td>59s</td>
<td>46 MB/s</td>
</tr>
</tbody>
</table>
| 3 MB/s                                   |          |                           | 15x efficiency improvement!
Outline

• Introduction
• Architecture and API
• Metadata management
• Replication and Recovery
• Network
• Evaluation
• Discussion
• One-minute plug
Discussion

• Is the problem real? Why different?
  – Yes (a clean slate design when BW not a bottleneck)
  – A new combination of system assumptions (full bisection BW) + workload (blob storage)

• Influential in 10 years? Yes
  – Increasing popularity of object/blob stores and feasibility of full bisection bandwidth networks
  – SSDs will allow much finer striping
Project:  *Erasure coding for better performance*

3-Replication
Storage Overhead: 3x

(10, 4) *erasure code*
Storage Overhead: 1.4x

• Any 10 units sufficient
• Can tolerate any 4-failures
Many properties: useful beyond fault tolerance

- **Load balance** by randomly choosing 10 units
- **Straggler mitigation** by connecting to > 10 and using the first 10 to respond

Help reining in *tail latencies* or in *increasing throughput* for skewed workloads
Talk to me or send me an email if you are interested in this research project (rashmikv@eecs)

Thanks!