Object Storage on CRAQ

High throughput chain replication for read-mostly workloads

Jeff Terrace
Michael J. Freedman
Data Storage Revolution

• Relational Databases
  - Microsoft SQL Server 2008
  - Oracle
  - MySQL
  - PostgreSQL

• Object Storage (put/get)
  - Dynamo
  - PNUTS
  - CouchDB
  - MemcacheDB
  - Cassandra

- Speed
- Scalability
- Availability
- Throughput
- No Complexity
Eventual Consistency
Eventual Consistency

• Writes ordered after commit
• Reads can be out-of-order or stale
• Easy to scale, high throughput
• Difficult application programming model
Traditional Solution to Consistency

Two-Phase Commit:
1. Prepare
2. Vote: Yes
3. Commit
4. Ack
Strong Consistency

- Reads and Writes strictly ordered
- Easy programming
- Expensive implementation
- Doesn’t scale well
Our Goal

• Easy programming

• Easy to scale, high throughput
van Renesse & Schneider (OSDI 2004)
Chain Replication

- Strong consistency
- Simple replication
- Increases write throughput
- Low read throughput

- Can we increase throughput?
- Insight:
  - Most applications are read-heavy (100:1)
CRAQ

- Two states – **clean** and **dirty**
CRAQ

- Two states per object – **clean** and **dirty**
- If latest version is **clean**, return value
- If **dirty**, contact **tail** for latest version number
Multicast Optimizations

- Each chain forms group
- Tail multicasts ACKs
Multicast Optimizations

- Each chain forms group
- Tail multicasts ACKs
- Head multicasts write data
CRAQ Benefits

• **From Chain Replication**
  – Strong consistency
  – Simple replication
  – Increases write throughput

• **Additional Contributions**
  – Read throughput scales :
    • Chain Replication with *Apportioned* Queries
  – Supports Eventual Consistency
High Diversity

- Many data storage systems assume locality
  - Well connected, low latency

- Real large applications are geo-replicated
  - To provide low latency
  - Fault tolerance

(source: Data Center Knowledge)
Multi-Datacenter CRAQ

DC1

HEAD

Replica

Replica

Replica

Replica

DC2

Replica

Replica

Replica

DC3

TAIL

Replica

Replica

Replica
Multi-Datacenter CRAQ
## Chain Configuration

<table>
<thead>
<tr>
<th>Motivation</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Popular vs. scarce objects</td>
<td>1. Specify chain size</td>
</tr>
<tr>
<td>2. Subset relevance</td>
<td>2. List datacenters</td>
</tr>
<tr>
<td></td>
<td>– dc₁, dc₂, ..., dcᴺ</td>
</tr>
<tr>
<td>3. Datacenter diversity</td>
<td>3. Separate sizes</td>
</tr>
<tr>
<td></td>
<td>– dc₁, chain_size₁, ...</td>
</tr>
<tr>
<td>4. Write locality</td>
<td>4. Specify master</td>
</tr>
</tbody>
</table>
Master Datacenter

DC1

DC2

DC3
Implementation

• Approximately 3,000 lines of C++

• Uses Tame extensions to SFS asynchronous I/O and RPC libraries

• Network operations use Sun RPC interfaces

• Uses Yahoo’s ZooKeeper for coordination
Coordination Using ZooKeeper

- Stores chain metadata
- Monitors/notifies about node membership
Evaluation

- Does CRAQ *scale* vs. CR?
- How does *write rate* impact performance?
- Can CRAQ recover from *failures*?
- How does *WAN* effect CRAQ?

- Tests use Emulab network emulation testbed
Read Throughput as Writes Increase

- CRAQ–7
- CRAQ–3
- CR–3

Reads/s vs. Writes/s graph showing data points and error bars.
Failure Recovery (Read Throughput)
Geo-replicated Read Latency

![Graph showing Geo-replicated Read Latency](image-url)
If Single Object Put/Get Insufficient

- **Test-and-Set, Append, Increment**
  - Trivial to implement
  - Head alone can evaluate

- **Multiple object transaction in same chain**
  - Can still be performed easily
  - Head alone can evaluate

- **Multiple chains**
  - An agreement protocol (2PC) can be used
  - Only heads of chains need to participate
  - Although degrades performance (use carefully!)
Summary

• CRAQ Contributions?
  – Challenges trade-off of consistency vs. throughput
• Provides strong consistency
• Throughput scales linearly for read-mostly
• Support for wide-area deployments of chains
• Provides atomic operations and transactions

Thank You

Questions?