MochiDB: A Byzantine Fault Tolerant Datastore

Tigran Tsaturyan
Saravanan Dhakshinamurthy
1. BFT Key-Value datastore (read(k), write(k,v), delete(k))
2. Consistent
3. Supports transactions
4. In-built sharding
5. Optimized for reads and writes over WAN
Database to store configurations for infrastructure.

- Most infrastructure as *key -> value*
- Need to update multiple props together
- Infrastructure needs to be consistent
- Located in different part of the world (next slide)
Source: Amazon AWS + https://wondernetwork.com/pings
1. Quorum Based BFT
   Client is a coordinator for transaction
2. Transactions can be two types - READ and WRITE
3. Min server requirement - $3f + 1$
objectX
1. Value
2. WriteCertificate
3. Timestamp (TS)
4. …..

objectY
1. Value
2. WriteCertificate
3. Timestamp (TS)
4. …..

“How that object happens to be that way” (Signed confirmations from the servers)
BFT Write: Protocol view

objectX
1. Value
2. WriteCertificate
3. Timestamp (TS)
4. ..... 

objectY
1. Value
2. WriteCertificate
3. Timestamp (TS)
4. ..... 

Collection of grants (object, timestamp, trHash)

Transaction + Random seed (0-1000)
Server grants client to write object at some TS
WriteCertificate - collection of grants from 2f+1 servers

client

Acks that transaction was performed

server1

server2

server3

server4
**BFT Write: Server processing**

Transaction 1

- WRITE("ObjectX", "12")
- RAND_seed = 315

Current object TS = 5334

Transaction 2

- WRITE("ObjectX", "48")
- RAND_seed = 467

Current object TS = 6315

**Order**

<table>
<thead>
<tr>
<th>TR1</th>
<th>TR2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write1</td>
<td>Write1</td>
</tr>
<tr>
<td>Write2</td>
<td>Write2</td>
</tr>
</tbody>
</table>

**Epochs**

- Old epochs
-Epoch 5000
- Epoch 6000

**Write1 grants**

- Write1 grant for TR1
- Write1 grant for TR2

**Current object states**

- COMMITTED

**Time**

- Current object TS = 5334
- Current object TS = 6315
- Current object TS = 6467

Write1 grant for TR2
Features

- **Sharding:**
  1024 tokens equally spread across the ring and assign to servers. Data is replicated \((\text{replicationFactor})\) on the Nth subsequent servers

- **GC:**
  Need to cleanup old write grants that are never fulfilled. Server initiates GC, get agreement on object TS, prune non needed data

- **Permissions:**
  Client have READ, WRITE, ADMIN permissions embedded into its certificate

- **Configuration changes:**
  Similar to 2PC

- more....
Implementation

- Java/Netty/ProtoBufs/Spring
- In-memory object store (for now)

Lessons learned

- Async IO, AWS fees
- Full cluster within JVM and testing framework
- Releasing resources
- Concurrent operations
- Do not make presentation in google docs :)

Testing

- See paper
- Local: 6ms -50%, 20 ms - 99% - READS; 16 ms - 50%, 60 ms - 99% WRITES
THANK YOU!

Ready to run images
https://hub.docker.com/r/mochidb/mochi-db/

Source code (48,310 lines of code):
https://github.com/saravan2/mochi-db

CONTRIBUTIONS APPRECIATED!
Mochi