ZHT: Const – Eventual Consistency Support For ZHT

Group Member: Shukun Xie
Ran Xin
Outline

- Problem Description
- Project Overview
- Solution
  - Maintains Replica List for Each Server
  - Operation without Primary Server Failure
- Working-on
  - Operation with Primary Server Failure
- Performance Evaluation
Outline

- Problem Description
- Project Overview
- Solution
  - Maintains Replica List for Each Server
  - Operation without Primary Server Failure
- Working-on
  - Operation with Primary Server Failure
- Performance Evaluation
Problem Description

- ZHT aims to provide High Availability, Good Fault Tolerance, High Throughput, and Low Latency
- ZHT applies Replication-based Fault Tolerance
- Consistency issue exists among data copies
Outline

- Problem Description
- Project Overview
- Solution
  - Maintains Replica List for Each Server
  - Operation without Primary Server Failure
- Working-on
  - Operation with Primary Server Failure
- Performance Evaluation
Project Overview

• Replication-based Fault Tolerance

• Consistency

<table>
<thead>
<tr>
<th></th>
<th>Eventual Consistency</th>
<th>Strong Consistency</th>
</tr>
</thead>
</table>
| Design                  | • Write Ack return to Client after Primary updates first Replica  
                          | • Version                                                   | Write Ack return to Client after Primary updates all Replica servers |
| Benefits                | Low latency on write tasks                      | Consistency Guaranteed                                   |
|                         | Low latency for requests to Primary             |                                                         |
| Drawbacks               | Latency on Lookup may increase                  | High Latency on write tasks                              |

• Both of Primary and Replica servers can serve Lookup requests.
Outline

- Problem Description
- Project Overview
- Solution
  - Maintains Replica List for Each Server
  - Operation without Primary Server Failure
- Working-on
  - Operation with Primary Server Failure
- Performance Evaluation
Outline

- Problem Description
- Project Overview
- Solution
  - Maintains Replica List for Each Server
  - Operation without Primary Server Failure
- Working-on
  - Operation with Primary Server Failure
- Performance Evaluation
1. Client sends Lookup request to Primary Server
2. Primary sends Lookup result to Client
1. Client sends Lookup request to Replica Server
2. Replica sends Version Compare request to Primary Server
3. Primary sends Version Compare result to Replica Server
4. Replica server sends Lookup result to Client
Version Compare – On Replica

1. Received Lookup Request from Client
2. Execute local lookup and get local version
3. Send Version Compare request to the working Primary Server in Replica List
4. Check the received Version Compare result from Primary Server
   - Data Removed: Execute local Remove and return "ZSC_REC_NONEXISTKEY" to Client
   - Same Version: The request key-value pair is updated on Replica. Return this key-value pair to Client
   - Version Conflict: Execute local Insert/Append and then return the updated key-value pair to Client
1. Client sends Insert/Append/Remove request to Primary Server
2. Primary Server synchronizes I/A/R request to first Replica
3. First Replica sends I/A/R acknowledgement to Primary
4. Primary Server sends Insert/Append/Remove acknowledgement to Client
Outline

- Problem Description
- Project Overview
- Solution
  - Maintains Replica List for Each Server
  - Operation without Primary Server Failure
- Working-on
  - Operation with Primary Server Failure
- Performance Evaluation
Lookup with Primary Server Failure

1. Client sends Lookup request to Primary Server
2. Client sends Lookup request to random Replica Server (Replica 2)
3. Replica 2 sends Version Compare request to Primary Server
4. Replica 2 sends Version Compare request to Replica Server 1
5. Replica 1 sends Version Compare result to Replica Server 2
6. Replica 2 sends Lookup result to Client
1. Client sends I/A/R request to Primary Server
2. Client sends I/A/R request to next reachable Replica (Replica 1)
3. Replica 1 synchronizes I/A/R request to next reachable Replica (Replica 2)
4. Replica 2 sends I/A/R acknowledgement to Replica 1
5. Replica 1 sends I/A/R acknowledgement to Client
Outline

- Problem Description
- Project Overview
- Solution
  - Maintains Replica List for Each Server
  - Operation without Primary Server Failure
- Working-on
  - Operation with Primary Server Failure
- Performance Evaluation
Experiment Environment

- Testbeds
  - HEC Cluster

- Workload
  - Same number of Clients and Server nodes (4, 6, 8)
  - 1000 key-value pairs for each operation

- Metrics
  - Latency
  - Throughput
Performance – Latency

Insert

Lookup

Append

Remove

Scale (# of nodes)

Latency (ms)
Performance – Throughput

**Insert**

- Throughput (tasks/sec)
- Scale (# of nodes)

**Lookup**

- Throughput (tasks/sec)
- Scale (# of nodes)

**Append**

- Throughput (tasks/sec)
- Scale (# of nodes)

**Remove**

- Throughput (tasks/sec)
- Scales (# of nodes)

Legend:
- Blue: Eventual Consistency
- Red: Strong Consistency
- Green: Laziness Enventual Consistency
- Purple: No Replica
Conclusion

- Compare with Strong Consistency
  - Achieve lower latency on write tasks

- Compare with Laziness Eventual Consistency
  - Achieve lower latency on Lookup
  - More reliable due to active inconsistency repair between Primary and Replica servers
Questions?