Implementing Linearizability at Large Scale and Low Latency

Speaker: Yuze Lou

Authors: Collion Lee, Seo Jin Park, Ankita Kejriwal, Satoshi Matsusshita, and John Ousterhout
Motivation

• Consistency is important
  ✷ Linearizability

• Impact
  ✷ throughput, latency
  ✷ Scalability

• Mechanisms like automatic retry, two PC are not enough
  ✷ At-least-once semantics

• Linearizability without huge impact
Background

• Linearizable Operations
  ✧ Occur instantaneously and exactly once
  ✧ Between its invocation and completion
  ✧ Reflects all requests that have finished

<table>
<thead>
<tr>
<th>Time Line</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>A:</td>
<td>W(0)</td>
</tr>
<tr>
<td></td>
<td>R(1)</td>
</tr>
<tr>
<td></td>
<td>W(0)</td>
</tr>
<tr>
<td>B:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W(1)</td>
</tr>
<tr>
<td></td>
<td>R(0)</td>
</tr>
<tr>
<td>A:</td>
<td>W(0)</td>
</tr>
<tr>
<td></td>
<td>R(1)</td>
</tr>
<tr>
<td></td>
<td>W(0)</td>
</tr>
<tr>
<td>B:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W(1)</td>
</tr>
<tr>
<td></td>
<td>R(1)</td>
</tr>
</tbody>
</table>
Background

- Linearizable Operations
  - Few of large-scale storage systems have linearizability
  - "At-least-once" semantics
  - "Exactly-once" semantics
    - Determine whether to retry
    - Return the same result
Proposed Method

- **RIFL (Reusable Infrastructure for Linearizability)**
  - Reusable mechanism
  - Reconfiguration tolerance
  - Low latency and scalable

- **Assumptions**
  - Remote Procedure Calls (RPC)
    - Issued to whichever server stores the relevant data
  - A request-response protocol (not asynchronous)
  - Automatic retries
RIFL Architecture

• Uniquely identify each RPC
  ✧ ID of a client
  ✧ A sequence number

• Retries use the same ID
RIFL Architecture

- Completion record
  - Stored durably
  - Including PRC ID + result
RIFL Architecture

- Completion record
  ✧ Data migration?
  ✧ Retries must find CR
  ✧ Object identifier
RIFL Architecture

• Each operation is related to a particular object(s),
  ✪ e.g. \( W(A) = 3 \)
  ▪ Store the CR with that object
  ▪ All retries necessarily involve the same object
  ▪ CR is discovered
  ▪ More than one object?

Completion Record

<table>
<thead>
<tr>
<th>Client ID</th>
<th>Sequence Number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RPC result for client</td>
</tr>
<tr>
<td></td>
<td>Object Identifier</td>
</tr>
</tbody>
</table>
RIFL Architecture

• Lease
  ✷ Allocate unique identifiers for clients
  ✷ Detect crashes

• Lease server
  ✷ Lease expiration time in memory
  ✷ The existence of leases are stored durably
RIFL Architecture

- Garbage collection
- When to delete the completion record for a RPC?
  - Client has crashed
  - Client has moved on
    - Client: include the sequence # of the first incomplete request in each RPC
    - Server: Process above info in each received RPC and delete CR accordingly
RIFL Implementation on RamCloud

• RamCloud
  ✷ General purpose distributed in memory key-value storage system
  ✷ Keep all data in DRAM all time
  ✷ Record all data on secondary storage durably

• Transaction
  ✷ Must success or fail as a complete unit
  ✷ Take effect as a complete unit.
  ✷ E.g. $T = [W(A)=3; W(B)=4, \text{commit()}$]
RIFL Implementation for Transactions

- E.g. $T = [W(A)=3; W(B)=4; \text{commit()}$
- Defefer the updates until commit
- 2PC
  ✷ The client issued the transaction is the coordinator
  ✷ The servers has related objects are the participants
RIFL Implementation for Transactions

• $T = [W(A)=3; W(B)=4; \text{commit()}]

• 1. Client send \textbf{PREPARE} to (1) the server who has object A, (2) the server who has object B

• 2. The servers do some check and work around the required objects, and returns \textbf{ABORT} or \textbf{PREPARED}

• 3. The client send \textbf{DECISION} to the participants

• Each message is a separate linearizable operation
RIFL Implementation for Transactions

• What exactly do servers do in step 2?
  ✷ Try locking the object
  ✷ Check version number
  ✷ Create a durable record of the lock in the log
  ✷ Return PREPARED

• When does commit() return?
  ✷ After sending all DECISION?
Experiments

• What is RIFL’s impact on latency and throughput?
• Does RIFL limit scalability?
• How does the performance of transactions implemented with RIFL?
Latency Impact of RIFL

- 100B writes were issued sequentially by a single client to a single server
- A point \((x, y) = y\) of the measured writes took at least \(x\) us to finish
Throughput Impact of RIFL

- The throughput for one server serving 100B writes with and without RIFL
Scalability Impact of RIFL

• The median write latency and a number of tail latencies (99% refers to 99th-percentile latency)
Evaluating RIFL-based Transactions

- Reverse cumulative distribution of transaction commit latency
- A point \((x, y) = y\) of the measured commit took at least \(x\) μs to finish
Conclusions. Thanks!

• A mechanism for achieving exactly-once RPC semantics for large-scale distributed systems

• General-purpose, independent of any operation or system

• The authors have implemented RIFL in RAMCloud

• Basically able to make any operation linearizable