LoLKV: The Logless, Linearizable, RDMA-based Key-Value Storage System

Ahmed Alquraan, Sreeharsha Udayashankar, Virendra Marathe, Bernard Wong, Samer Al-Kiswany

University of Waterloo

Oracle Labs
Production-quality Key-Value Stores

• Leader-based consensus protocols → Strong consistency

Raft
Paxos

ZAB
Mu

APUS
DARE

Viewstamped Replication
Shortcomings of Current Systems

- Leader
  - Operation Log
    - W(k1) W(k5) W(k3)
  - Key-value Store
    - k1 X
    - k5 Y
    - k3 Z

- Follower 1
  - Operation Log
    - W(k1) W(k5) W(k3)
  - Key-value Store
    - k1 X
    - k5 Y
    - k3 Z

- Follower 2
  - Operation Log
    - W(k1) W(k5) W(k3)
  - Key-value Store
    - k1 X
    - k5 Y
    - k3 Z

Write (k3, Z) → Leader → Follower 1 → Follower 2

Ack → Leader

Ack → Follower 1

Ack → Follower 2
Shortcomings of Current Systems

- Log-based replication
  - Log is a serialization point
  - Unnecessary data copying
Shortcomings of Current Systems

- Log-based replication
  - Log is a serialization point
  - Unnecessary data copying

- All replicas apply committed operations
  - Work repetition on all replicas
Shortcomings of Current Systems

- Log-based replication
  - Log is a serialization point
  - Unnecessary data copying

- All replicas apply committed operations
  - Work repetition on all replicas

- Multiple single-threaded shards
  - Inefficient for skewed workloads
  - Resource fragmentation: separate memory regions per shard
Shortcomings of Current Systems

- Log-based replication
  - Log is a serialization point
  - Unnecessary data copying

- All replicas apply committed operations
  - Work repetition on all replicas

- Multiple single-threaded shards
  - Inefficient for skewed workloads
  - Resource fragmentation: separate memory regions per shard

LoLKV

A new linearizable RDMA-based KV Store

- A novel logless replication
- Combines replication and apply

- Passive followers

- A novel multi-threaded shard design
Design

• Leader-based system
  • Two main components
    • Storage
    • Worker threads
  • RDMA-based system
    • UD for communication with clients
    • RC for communication between replicas
Worker Threads

• Design Goals
  • Highly-concurrent design
  • Avoid sharding the key space among threads

• Employs multiple worker threads
• Each thread has its own RDMA resources
• Each thread serves requests for any key
  • Run the consensus protocol
  • Update the storage
Storage Design

• Design Goals
  • Minimize RDMA communication
  • Minimize contention between threads

• Storage
  • Memory divided into segments
  • Each segment stores a set of objects
  • A segment is owned by one thread at a time
  • One RDMA Write to commit an operation

• Hash Table
  • Stores pointers to objects in the storage
  • A lock-free linear probing hash table
  • Shared between all threads
  • One RDMA Write to apply an operation
LoLKV Write Request Path

- Put operations processing
  - Replication phase ①②
  - Local apply phase ③
  - Remote apply phase ④

One RDMA Write for replication
One async RDMA Write for apply
Replication Phase

Replica 1
Segment 1
Metadata
a, 1
b, 5

Replica 2
Segment 2
Metadata
:...:

Replica 3
Segment 1
Metadata
a, 1
b, 5

Put (c) at time $t_1$
Replication Phase

Leader

Segment 1
Metadata
a, 1

b, 5

Thread 1 Metadata
key
c
value
val
seq_num
6

used
Free

Put (c)
t1

Replica 2
Segment 1
Metadata
a, 1

Replica 3
Segment 1
Metadata
a, 1

b, 5
Replication Phase

- **Put (c)**
  - Leader
    - Segment 1
      - Metadata
        - a, 1
        - b, 5
        - c, 6
  - Replica 2
    - Segment 1
      - Metadata
        - a, 1
        - b, 5
        - c, 6
  - Replica 3
    - Segment 1
      - Metadata
        - a, 1
        - b, 5
        - c, 6

**t_1**
Replication Phase

- Replication Phase
- Metadata
  - a, 1
  - b, 5
  - c, 6

- Leader
  - Segment 1
    - Metadata
    - a, 1
    - ...
    - b, 5
    - c, 6

- Replica 2
  - Segment 1
    - Metadata
    - a, 1
    - ...
    - b, 5
    - c, 6

- Replica 3
  - Segment 1
    - Metadata
    - a, 1
    - ...
    - b, 5
    - c, 6

- Put (c)
Local Apply Phase

- The leader applies the operation to its hash table

- Hashes the key to find the hash table entry

- Terminates probing if
  - Finds an empty entry
  - Finds an entry pointing to the same key
Asynchronous Remote Apply Phase

- The leader updates followers hash tables lazily
  - Using RDMA Write
LoLKV is a Complete System

- Concurrent writes
- Fault tolerance
  - Follower failure
  - Leader failure
  - Torn writes
- Leader election protocol
- Garbage collection protocol
- Proof of correctness
  - Proof sketch
  - TLA+ model checking
Concurrent Writes to Different Keys

- Objects are committed in parallel
- Objects are applied in parallel
Concurrent Writes to Different Keys

- Objects are committed in parallel
- Objects are applied in parallel
- Hash table is updated using CAS
  - Handles concurrent access
Concurrent Writes to Different Keys

- Objects are committed in parallel
- Objects are applied in parallel
- Hash table is updated using CAS
  - Handles concurrent access
- If CAS fails, repeat linear probing
Concurrent Writes to the Same Key

- Incarnation Array
  - Array of atomic counters

- Each Put has an incarnation number

<table>
<thead>
<tr>
<th>Key range</th>
<th>[0, 4096)</th>
<th>[4096, 8192)</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incarnation Number</td>
<td>100</td>
<td>115</td>
<td>...</td>
</tr>
</tbody>
</table>
Concurrent Writes to the Same Key

- Incarnation Array
  - Array of atomic counters
- Each Put has an incarnation number
Concurrent Writes to the Same Key

- Incarnation Array
  - Array of atomic counters

- Each Put has an incarnation number
Concurrent Writes to the Same Key

- Incarnation Array
  - Array of atomic counters
- Each Put has an incarnation number
- Orders Puts for the same key

<table>
<thead>
<tr>
<th>Key range</th>
<th>[0, 4096)</th>
<th>[4096, 8192)</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incarnation Number</td>
<td>102</td>
<td>115</td>
<td>...</td>
</tr>
</tbody>
</table>
Leader Election

- Any replica can become a leader

- The new leader might be stale for some threads
  - Different threads replicate operations on different majorities

- State synchronization brings the new leader up-to-date
Evaluation

Alternatives (best configuration per system)
• DARE (8 shards)
• APUS (7 shards)
• Mu (4 shards)
• uKharon (4 shards)

Workloads
• YCSB benchmark
• Different workload skewness
• Different read-to-write ratios

Metrics
• Throughput
• Latency
• Scalability

Testbed
12 machines in CloudLab
• 8-core CPU (2.1 Ghz)
• 16 GB of RAM
• Infiniband network (56 Gbps)
• Mellanox CX3
LoLKv outperforms other systems in terms of throughput and latency.
Skewed Workload

- Uniform write-only workload
- One popular shard
- Control the percentage of operations served by that shard

Other systems performance decreases with skewness
- Popular shard is overwhelmed

LoLKV efficiently handles skewed workloads
Conclusion

• LoLKV is a low-latency, highly-concurrent, and linearizable object store

• LoLKV adopts a novel logless design
  • Eliminates the serialization point
  • Eliminates unnecessary memory copy operations

• LoLKV adopts a novel multi-threaded shard design
  • Efficient for both uniform and skewed workloads
  • Eliminates resource fragmentation

• LoLKV outperforms state-of-the-art systems
  • At least 1.7× higher throughput
  • At least 20% lower latency
  • Better scalability