
TECHNICAL REPORT TR-2025-003

Distributed Key–Value Store with Consensus-Based Replication

Design, Implementation, and Performance Analysis

Michael Torres, Ananya Krishnan, Erik Johansson

Infrastructure Engineering Team
Distributed Systems Division

Document ID:	TR-2025-003
Version:	2.1
Date:	February 10, 2025
Classification:	Internal – Engineering
Status:	Final

Acme Cloud Systems, Inc.
1200 Technology Drive, San Jose, CA 95134

Revision History

Version	Date	Author	Changes
1.0	2024-11-15	M.	
Torres	Initial draft		
1.5	2024-12-20	A.	
Krishnan	Added benchmarks and security analysis		
2.0	2025-01-28	E.	
Johansson	Performance tuning and final benchmarks		
2.1	2025-02-10	M.	
Torres	Minor corrections, final review		

Executive Summary

This report presents the design, implementation, and performance evaluation of **AcmeKV**, a distributed key-value store with strong consistency guarantees based on the Raft consensus protocol. AcmeKV is designed to serve as the foundational storage layer for Acme Cloud Systems' next-generation microservices platform, replacing the existing Cassandra-based infrastructure which has proven difficult to maintain and reason about under our strong consistency requirements.

Key findings of this report include:

- AcmeKV achieves **285,000 reads/sec** and **142,000 writes/sec** on a 5-node cluster with p99 latencies under 5ms for reads and 12ms for writes.
- The system correctly maintains linearizable consistency under all tested failure scenarios, including leader failure, network partitions, and disk failures.
- Compared to our existing Cassandra deployment, AcmeKV provides a **2.3×** improvement in write latency and a **40%** reduction in operational complexity as measured by incident response time.
- The system is ready for production deployment with the caveats noted in Section 5.3.

Contents

Revision History	1
Executive Summary	2
1 Introduction	5
1.1 Motivation	5
1.2 Requirements	5
1.3 Scope	6
2 System Design	7
2.1 Architecture Overview	7
2.2 Consensus Protocol	7
2.3 Data Model	8
3 Performance Evaluation	9
3.1 Test Environment	9
3.2 Throughput Results	9
3.3 Latency Results	10
4 Conclusion and Recommendations	11
4.1 Summary	11
4.2 Recommendations	11
4.3 Known Limitations	11

List of Tables

1.1	System requirements for AcmeKV.	6
3.1	Hardware configuration for benchmark cluster.	9
3.2	Throughput (operations/second) under different workload mixes.	9
3.3	Latency percentiles (milliseconds) for read and write operations.	10

Chapter 1

Introduction

1.1 Motivation

Acme Cloud Systems’ microservices platform currently relies on Apache Cassandra as its primary key-value store. While Cassandra provides excellent horizontal scalability and availability, its eventual consistency model has been a persistent source of bugs and operational complexity. Over the past 18 months, our incident reports show that 34% of P1 incidents were attributable to stale reads or write conflicts arising from Cassandra’s eventual consistency guarantees.

As our platform has evolved, we have identified a growing set of use cases that require strong consistency:

1. **Configuration management:** Service configurations must be read with linearizable guarantees to prevent inconsistent behavior across replicas.
2. **Distributed locking:** Leader election and distributed locks require consensus to function correctly.
3. **Financial transactions:** Payment processing requires serializable isolation to prevent double-spending.
4. **Metadata catalogs:** Dataset and schema registries must reflect the latest state to avoid data corruption.

1.2 Requirements

Based on stakeholder interviews and workload analysis, we established the following requirements for AcmeKV:

Table 1.1: System requirements for AcmeKV.

ID	Requirement	Priority
R1	Linearizable read/write consistency	Must
R2	$\geq 100\text{K}$ writes/sec (5-node cluster)	Must
R3	p99 read latency $< 10\text{ms}$	Must
R4	Automatic leader failover < 5 seconds	Must
R5	Online cluster membership changes	Should
R6	Point-in-time snapshots for backup	Should
R7	TLS encryption for all communication	Must
R8	Role-based access control (RBAC)	Should
R9	Observability (metrics, tracing, logging)	Must
R10	Cross-datacenter replication	Could

1.3 Scope

This report covers the design and implementation of AcmeKV version 1.0, addressing requirements R1–R9. Cross-datacenter replication (R10) is deferred to version 2.0 and will be covered in a subsequent report.

Chapter

2

System Design

2.1 Architecture Overview

AcmeKV follows a replicated state machine architecture built on the Raft consensus protocol. The system consists of the following core components:

Raft Consensus Module

Implements leader election, log replication, and safety guarantees per the Raft specification. Our implementation extends the basic protocol with pre-vote, leadership transfer, and joint consensus for membership changes.

Storage Engine

A log-structured merge-tree (LSM-tree) storage engine optimized for write-heavy workloads. The engine uses a two-level compaction strategy with bloom filters for efficient point lookups.

API Gateway

gRPC-based API layer that routes client requests to the appropriate cluster node. Read requests can be served by any node (with a consistency check), while writes are forwarded to the leader.

Snapshot Manager

Handles periodic snapshots of the state machine for log compaction and backup/restore operations.

2.2 Consensus Protocol

We implement the Raft consensus protocol with several optimizations for production use:

1. **Batched log replication:** Multiple client requests are batched into a single `AppendEntries` RPC, amortizing the cost of consensus across multiple operations.
2. **Pipeline replication:** The leader sends the next batch of entries before receiving acknowledgment for the previous batch, improving throughput on high-latency networks.
3. **Lease-based reads:** The leader maintains a time-based lease that allows it to serve linearizable reads without requiring a round of consensus, reducing read latency significantly.

The correctness of our implementation is verified through a combination of property-based testing (using Jepsen) and TLA+ model checking of the core consensus logic.

2.3 Data Model

AcmeKV supports a simple key-value data model with the following operations:

Listing 2.1: AcmeKV core API definition.

```
1 service AcmeKV {  
2   // Put stores a key-value pair.  
3   rpc Put(PutRequest) returns (PutResponse);  
4  
5   // Get retrieves the value for a key.  
6   rpc Get(GetRequest) returns (GetResponse);  
7  
8   // Delete removes a key-value pair.  
9   rpc Delete(DeleteRequest) returns (DeleteResponse);  
10  
11  // Scan returns key-value pairs in a range.  
12  rpc Scan(ScanRequest) returns (stream ScanResponse);  
13  
14  // Txn executes a multi-key transaction.  
15  rpc Txn(TxnRequest) returns (TxnResponse);  
16 }
```

Keys are limited to 256 bytes and values to 1MB. The system supports optional TTL (time-to-live) on keys and atomic multi-key transactions with serializable isolation using optimistic concurrency control.

Chapter 3

Performance Evaluation

3.1 Test Environment

All benchmarks were conducted on a 5-node cluster with the following hardware configuration:

Table 3.1: Hardware configuration for benchmark cluster.

Component	Specification
CPU	Intel Xeon Gold 6348 (28 cores, 2.6 GHz)
Memory	256 GB DDR4-3200 ECC
Storage	2× Intel Optane P5800X 1.6TB (NVMe)
Network	25 Gbps Ethernet (Mellanox ConnectX-6)
OS	Ubuntu 22.04 LTS (kernel 5.15)

3.2 Throughput Results

Table 3.2 summarizes the throughput results under various workload configurations.

Table 3.2: Throughput (operations/second) under different workload mixes.

Workload	AcmeKV	etcd v3.5	Cassandra (QUORUM)
100% Read	285,000	142,000	310,000
100% Write	142,000	48,000	165,000
95% Read / 5% Write	271,000	128,000	295,000
50% Read / 50% Write	198,000	85,000	225,000

AcmeKV achieves approximately 2× the throughput of etcd while providing the same linearizable consistency guarantees. Cassandra achieves slightly higher raw throughput, but this comes at the cost of weaker consistency (quorum reads/writes provide only regular register semantics, not linearizability).

3.3 Latency Results

Table 3.3: Latency percentiles (milliseconds) for read and write operations.

Operation	p50	p90	p99	p99.9
Read (lease-based)	0.8	1.5	3.2	8.1
Read (consensus)	2.1	3.8	7.4	15.3
Write	3.5	6.2	11.8	28.4
Transaction (2-key)	5.2	9.1	18.5	42.7

Chapter

4

Conclusion and Recommendations

4.1 Summary

AcmeKV successfully meets all “Must” and “Should” priority requirements established in Section 1.2. The system provides linearizable consistency with production-grade performance, achieving 285K reads/sec and 142K writes/sec with sub-5ms p99 read latency on lease-based reads. Jepsen testing confirms correctness under all tested failure scenarios.

4.2 Recommendations

Based on our evaluation, we recommend the following deployment plan:

1. **Phase 1 (Q1 2025):** Deploy AcmeKV for configuration management and distributed locking workloads, which are currently the largest sources of consistency-related incidents.
2. **Phase 2 (Q2 2025):** Migrate the metadata catalog service to AcmeKV after completing integration testing with the data platform team.
3. **Phase 3 (Q3 2025):** Evaluate AcmeKV for financial transaction workloads, pending completion of the compliance audit and penetration testing.

4.3 Known Limitations

- **Value size:** The 1MB value size limit may be insufficient for some use cases. We are investigating chunked storage for larger values.
- **Cross-DC replication:** Not yet implemented. The current system is limited to a single datacenter.
- **Range queries:** Scan performance degrades for ranges exceeding 10,000 keys due to LSM compaction overhead. We plan to implement a read cache to address this.