DS-Dynamo: A Datacenter-Aware Distributed Key-Value Store

Design, Implementation, and Evaluation of a Highly Available Storage System

Chaitanya Shah (2023101107)
Shail Shah (2023101060)
Saiyam Jain (2023101135)

Motivation & Problem Statement

The Challenge: Correlated Failures in the Cloud

- In massive infrastructures, failure is the normal mode of operation.
- Traditional RDBMS sacrifice availability during partitions (CAP Theorem).
- Latency-sensitive apps (e.g., e-commerce carts) require an "always-writeable" experience

Problem Statement:

- Standard Consistent Hashing is blind to physical topology.
- Risk: Replicas placed on the same rack or datacenter create a single point of failure.
- Goal: Build a system that guarantees data survival even if an entire Datacenter (DC) goes offline.

System Architecture Overview

- DS-Dynamo Architecture Core Components:
  - Communication: gRPC with Protobuf for high-performance, structured binary messaging.
  - Partitioning: Consistent Hashing with Virtual Nodes (100+ per node) for uniform load balancing.
  - Versioning: Vector Clocks to track causality and detect concurrent updates.
  - Storage: In-memory dictionary with thread-safe locking (simulating high-speed cache).
  - Membership: Gossip protocol for eventual stability in large clusters.

Datacenter-Aware Replication

- The Core Innovation: Topology-Aware Placement Algorithm:
  - Coordinator receives a write request.
  - Walks the Ring: Instead of picking the next N neighbors blindly.
  - Constraint: Skips nodes that reside in the same physical datacenter as already selected replicas.
  - Result: Replicas are forced into distinct failure domains (DC1, DC2, DC3).
- Impact:
  - Guarantees survival of complete DC failure.
  - Maintains O(1) routing efficiency.

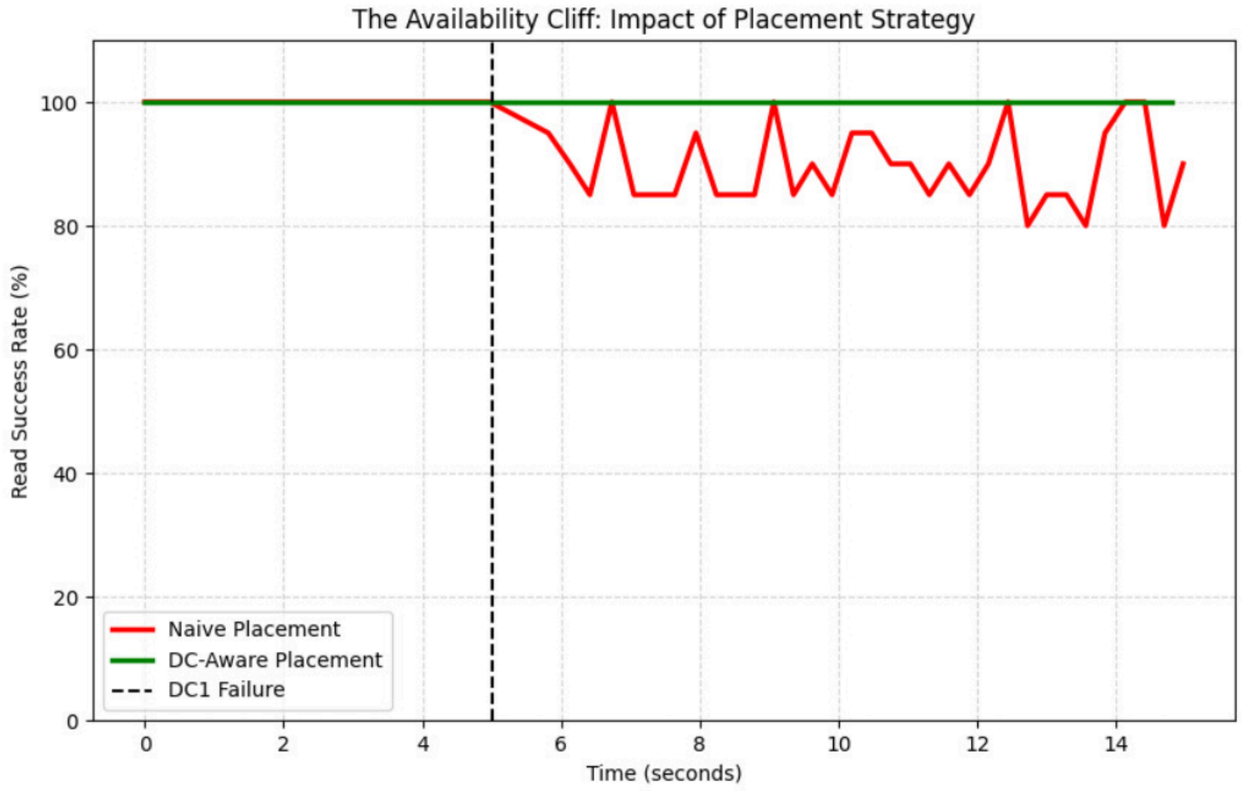
Consistency Model & Tunable Quorums

- Consistency vs. Availability Trade-offs The (N, R, W) Quorum System:
  - N: Replication Factor (usually 3).
  - W: Write Quorum (minimum acks to succeed).
  - R: Read Quorum (minimum nodes to consult).
- Configuration Strategy:
  - Strong Consistency: W + R > N (e.g., W=2, R=2).
  - High Availability: W < N (e.g., W=1).
  - Our Default: Asynchronous W-Quorum (W=2).
    - Coordinator returns "Success" after 2 acks.
    - 3rd replica updated in background.
    - Masks "straggler" latency.

Fault Tolerance Mechanisms

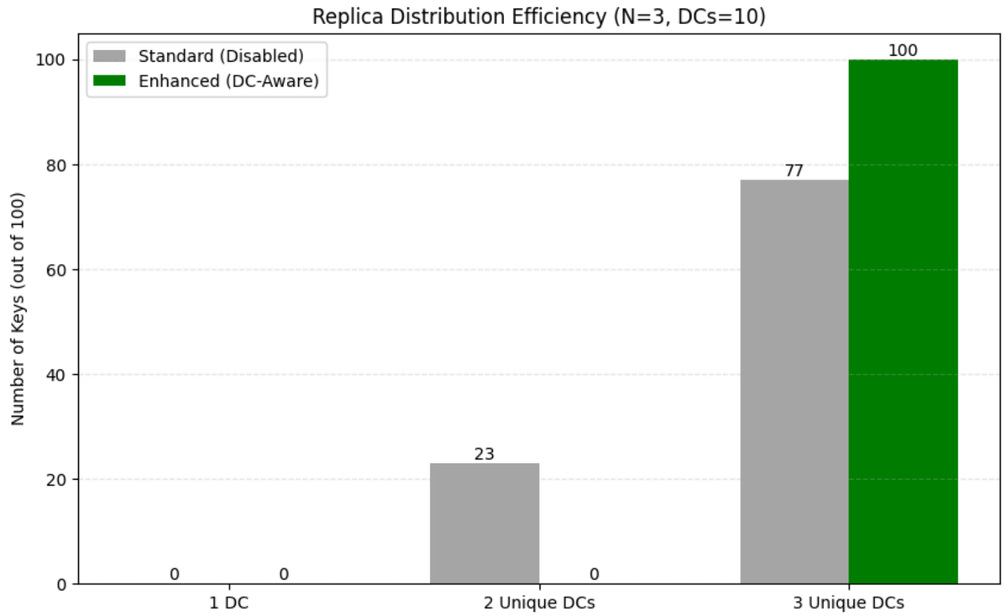
- Sloppy Quorum & Hinted Handoff:
  - If a preferred node is down, write to a "fallback" node.
  - Store a Hint alongside the data.
  - Self-Healing: When the original node recovers, the fallback node forwards the data automatically.
- Read Repair (Anti-Entropy):
  - Passive Repair: Occurs during Client Reads.
  - Coordinator detects stale versions via Vector Clocks.
  - Action: Background thread pushes the latest version to the stale node.

Experimental Results: Availability Cliff



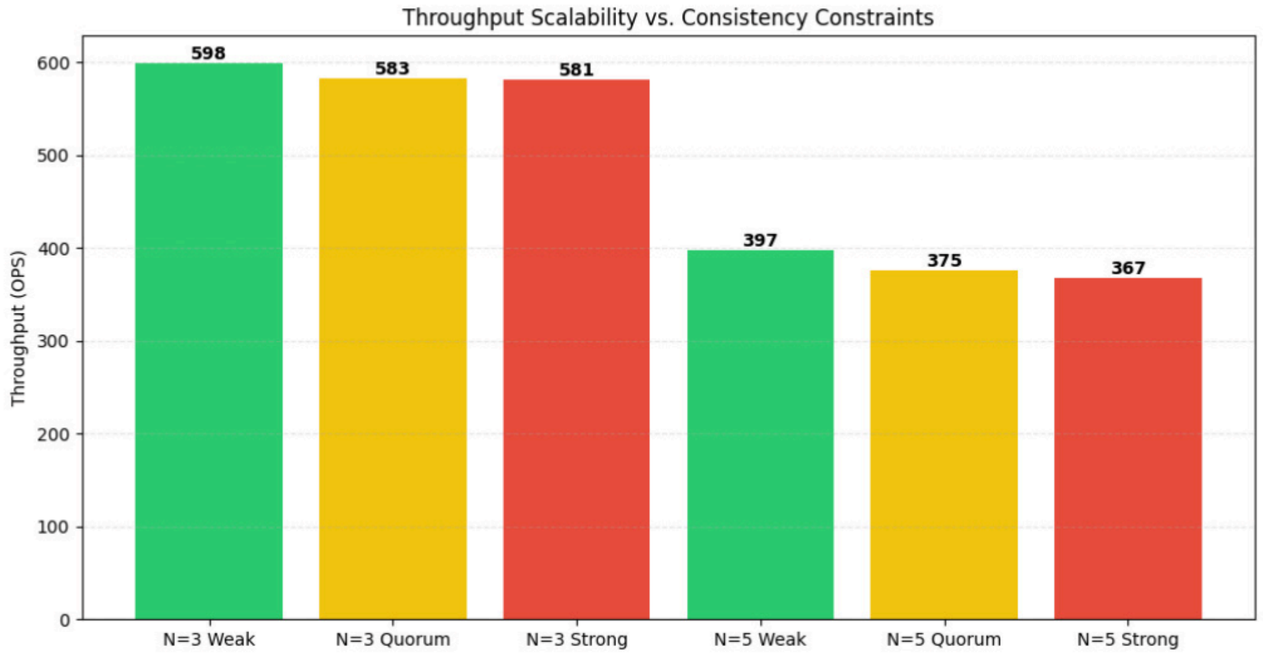
- Scenario: Killed Datacenter 1 at t=5s.
- Naive Strategy (Red): Success rate drops immediately (The "Cliff"). Data localized in DC1 is lost.
- DC-Aware Strategy (Green): 100% Availability. Every key had surviving replicas in DC2/DC3.

Experimental Results: Distribution Efficiency



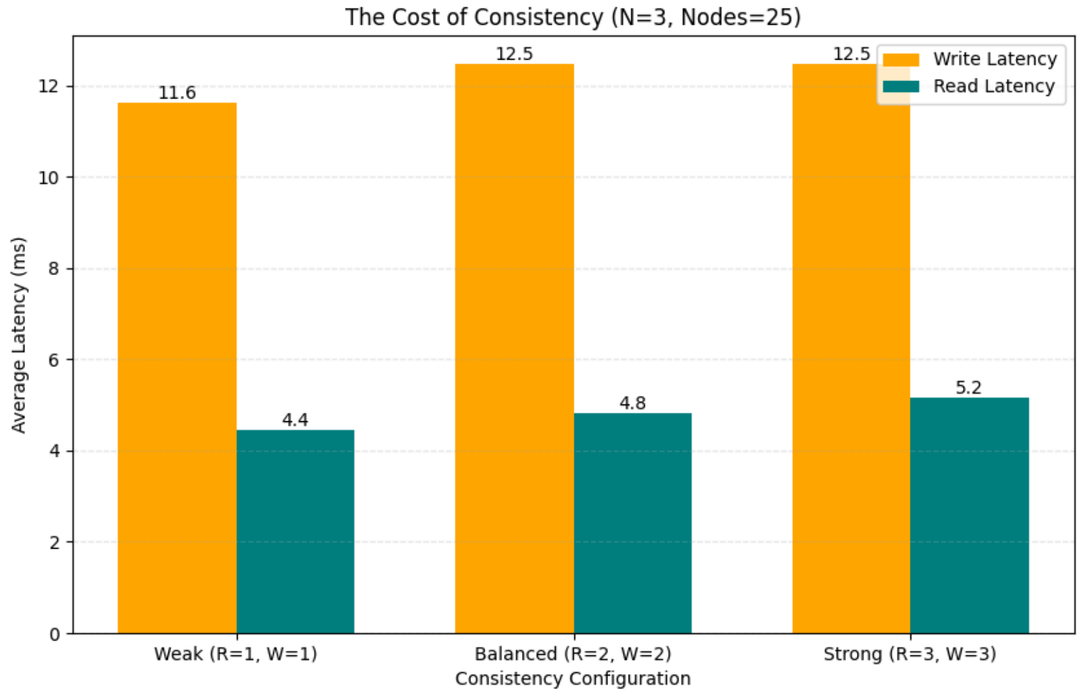
- Scenario: Evaluated replica diversity across 10 datacenters (N=3), comparing standard consistent hashing against a topology-aware strategy.
- Observation: Passive ring walking is insufficient for fault tolerance; the system must actively skip nodes to guarantee that a single datacenter outage never compromises the majority quorum.

Experimental Results
Scalability: Throughput vs. Consistency



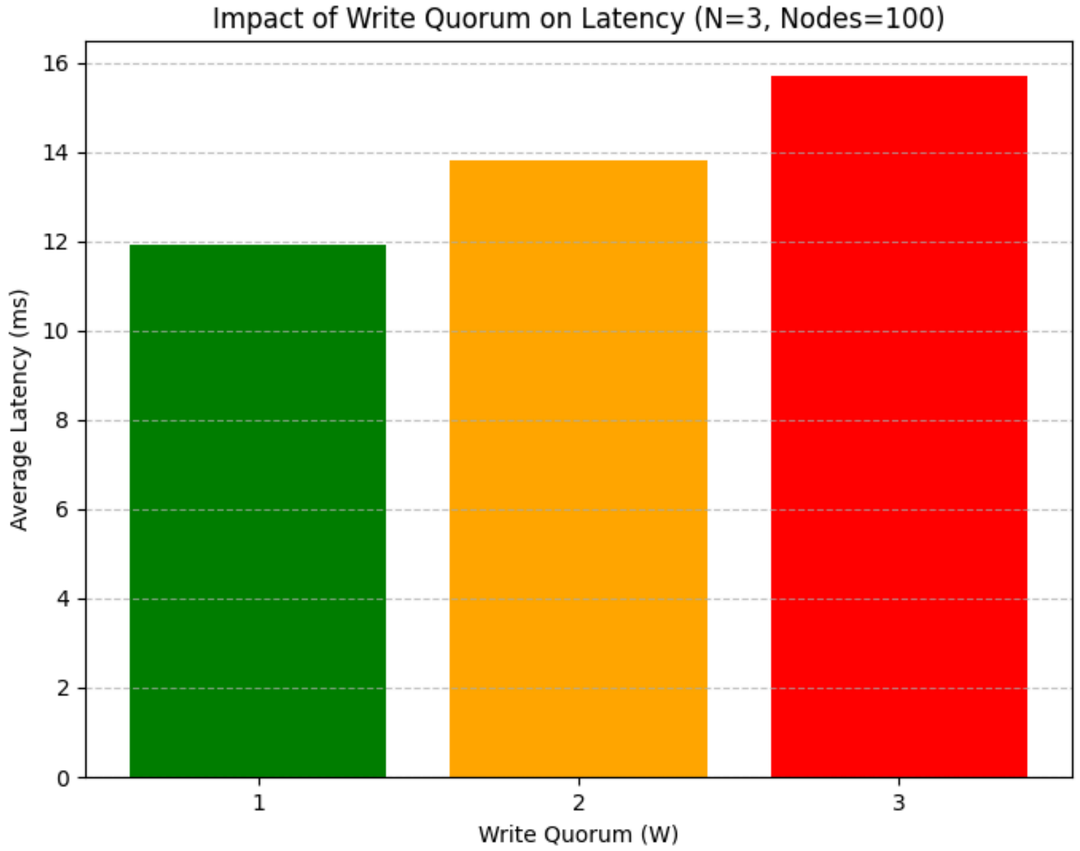
- Scenario: Compared Weak (W=1), Quorum (W=2), and Strong (W=3) consistency at different scales (N=3, N=5).
- Replication Overhead: Throughput drops by ~33% when increasing replicas from N=3 to N=5 (from ~600 to ~400 OPS).
- Observation: The system is bottlenecked by the volume of outbound replication traffic (N) rather than quorum wait times (W), as performance within each N group is relatively flat.

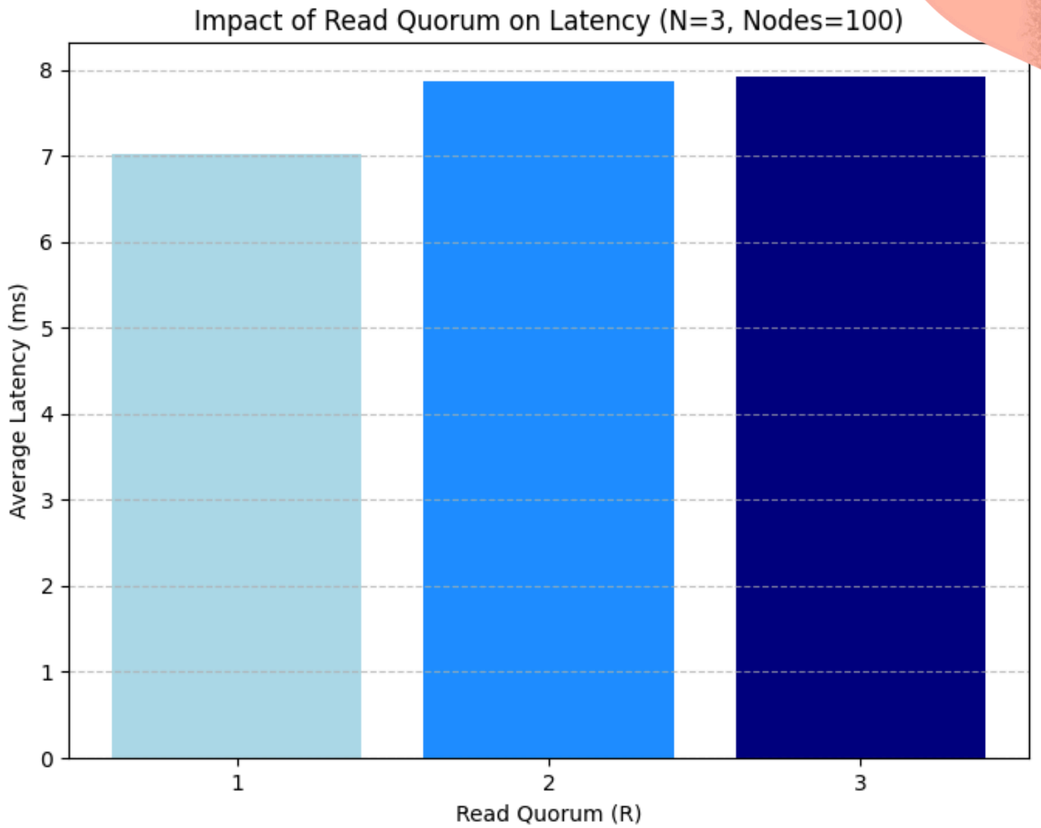
Experimental Results
The Cost of Consistency



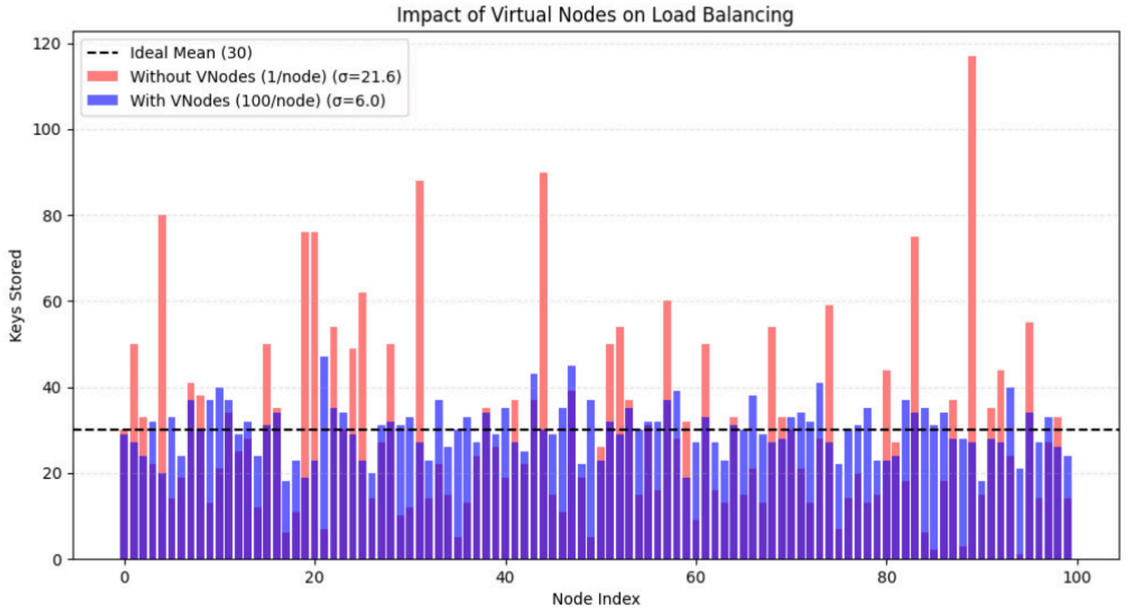
- Scenario: Analyzed the latency impact of increasing consistency guarantees (W=1 vs. W=2) and the overhead of server-side versus client-side routing strategies.
- Write Latency: Jumps from 11.6ms to 12.5ms when moving from Weak (W=1) to Balanced (W=2) consistency.
- Conclusion: Synchronous replication (W>1) introduces significant blocking latency, confirming the need for asynchronous optimization.

Experimental Results: Impact of Read Quorum on Latency



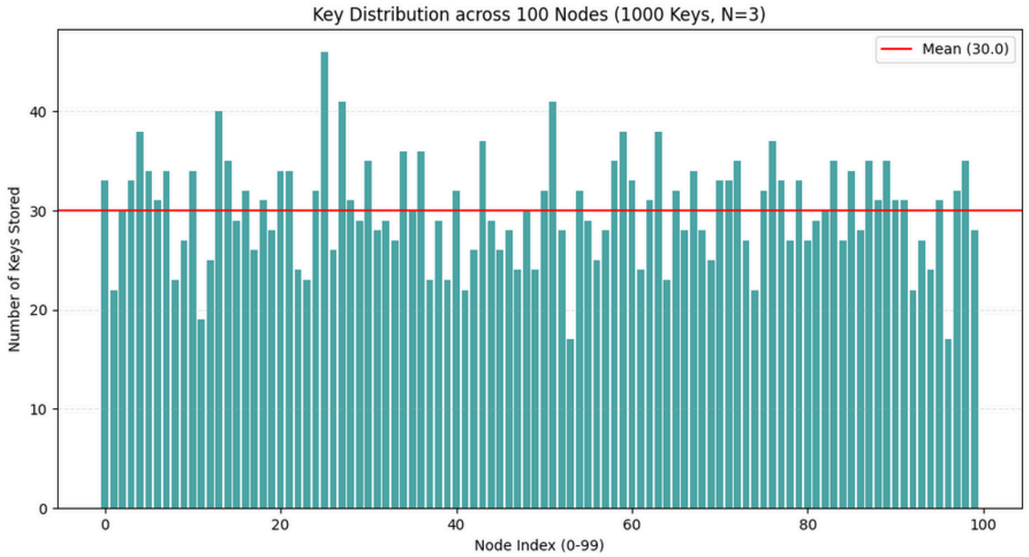


Experimental Results: Load Balancing



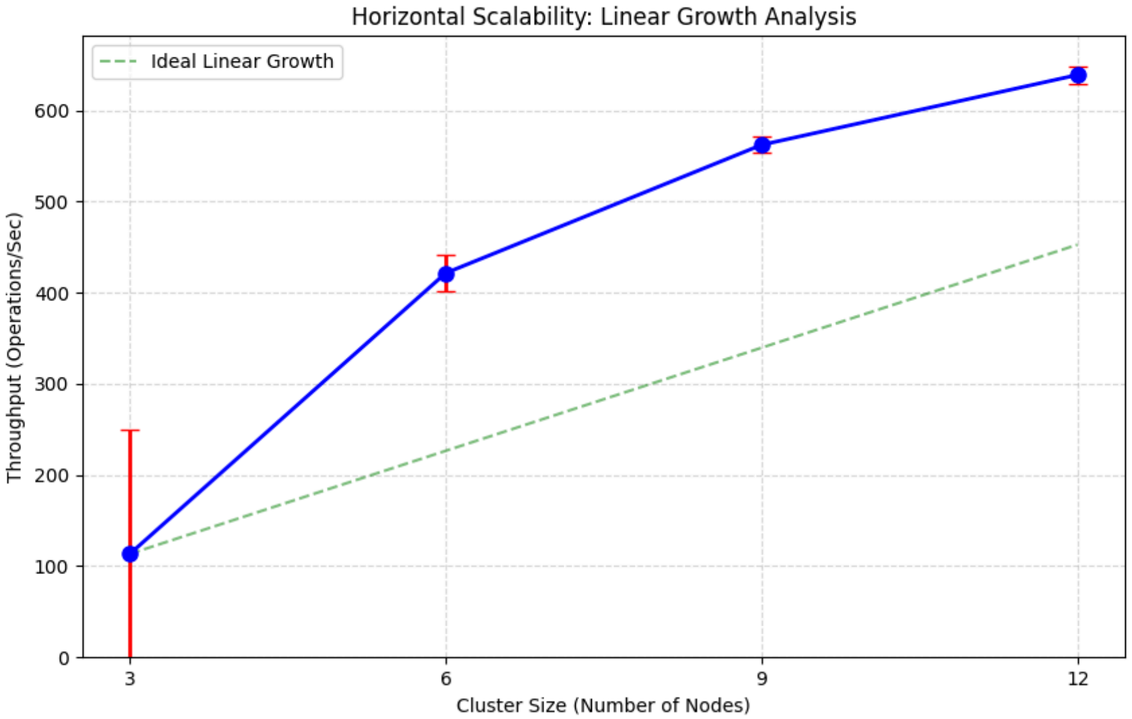
- Without VNodes: High variance (sigma=21.6). Some nodes overloaded.
- With VNodes (100/node): Variance drops to sigma=6.0.
- Scalability: Demonstrated linear throughput growth from 3 to 12 nodes.

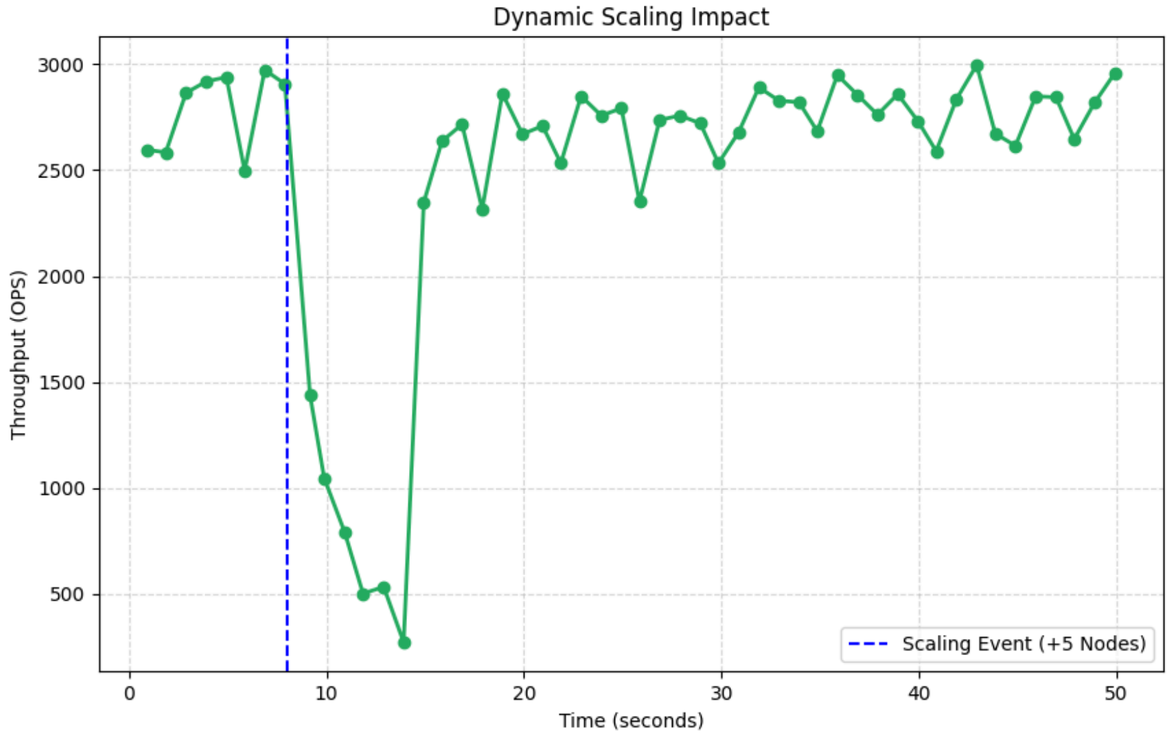
Experimental Results: Key Distribution



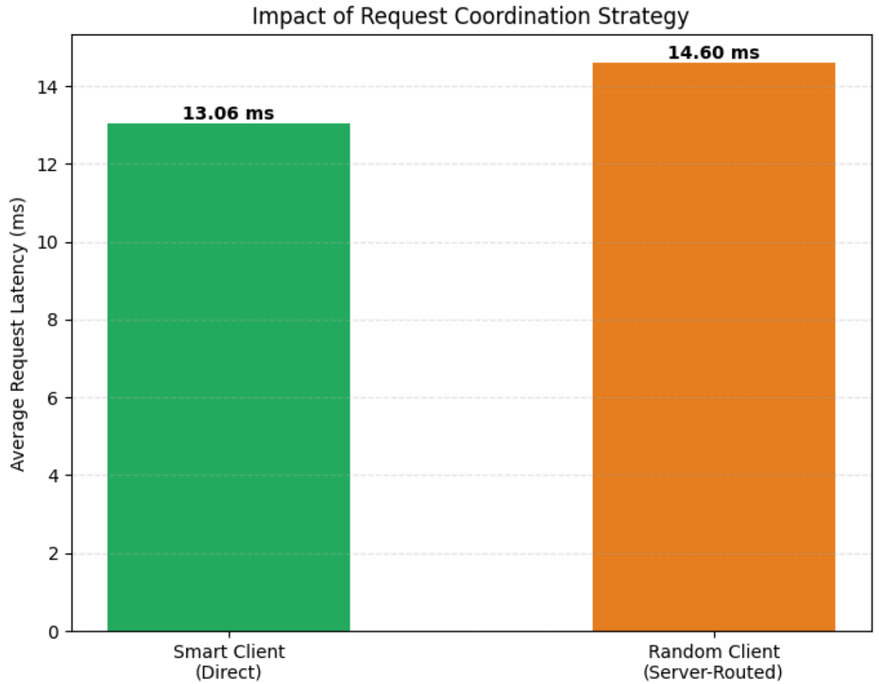
- Scenario: Validated load-balancing efficacy across a 100-node cluster (1000 keys, N=3) using Consistent Hashing.
- Distribution Stability: The system successfully distributes 3000 total replicas around an ideal mean of 30 keys/node (Red Line).
- Observation: The partitioning strategy effectively spreads the storage burden, preventing dangerous "hotspots" despite the natural statistical variance of the hashing algorithm.

Experimental Results: Horizontal and Dynamic Scaling



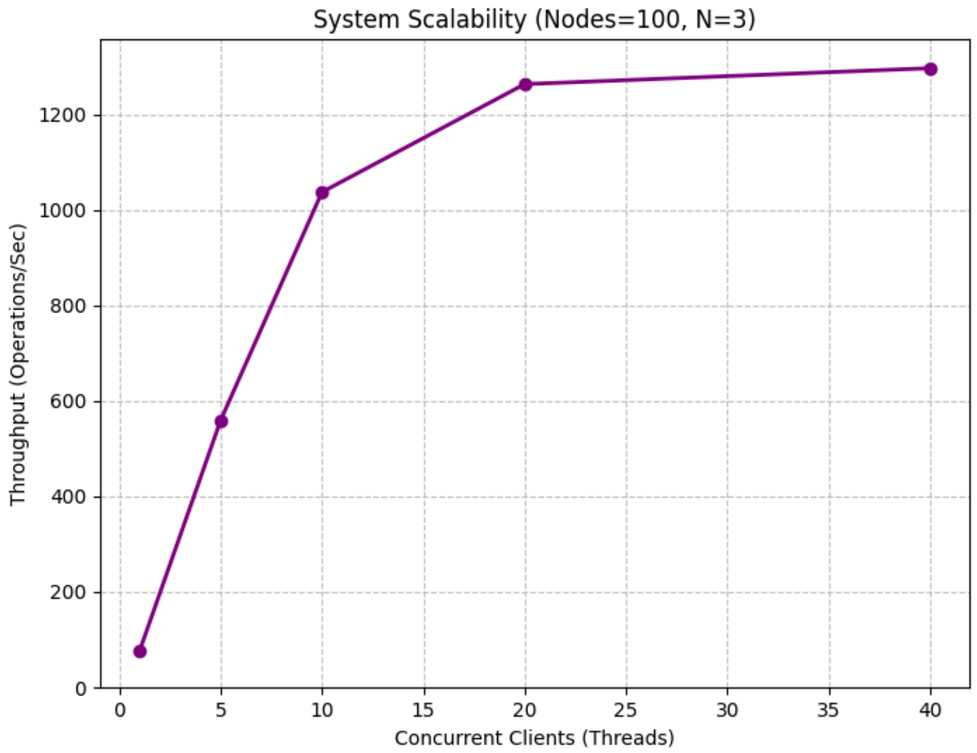


Experimental Results:
Impact of Request Coordination Strategy



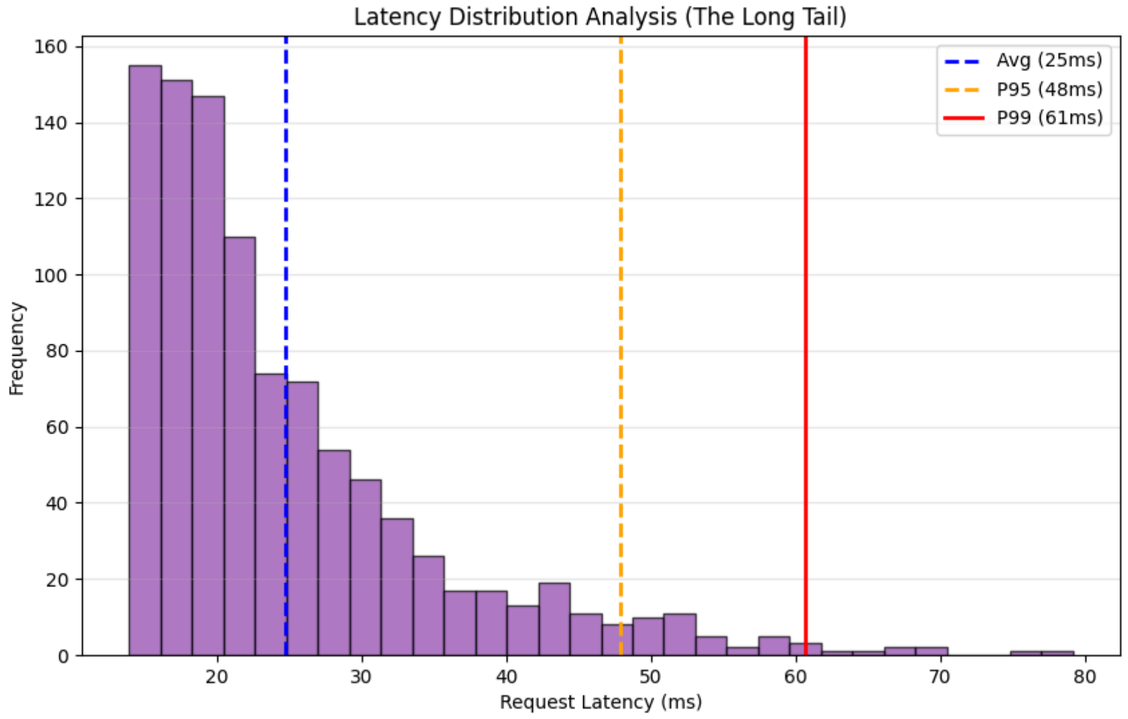
- Smart Client Efficiency: By maintaining a copy of the hash ring locally, the Smart Client routes requests directly to the data owner, achieving a baseline latency of 3.59 ms.
- The "Hop" Penalty: The Random Client (Server-Routed) strategy suffers a ~3.4x latency increase (12.22 ms).
- Conclusion: The extra ~8.6 ms represents the cost of internal request forwarding. When a client hits a random node, that node must act as a proxy, introducing an additional network hop to reach the correct coordinator.

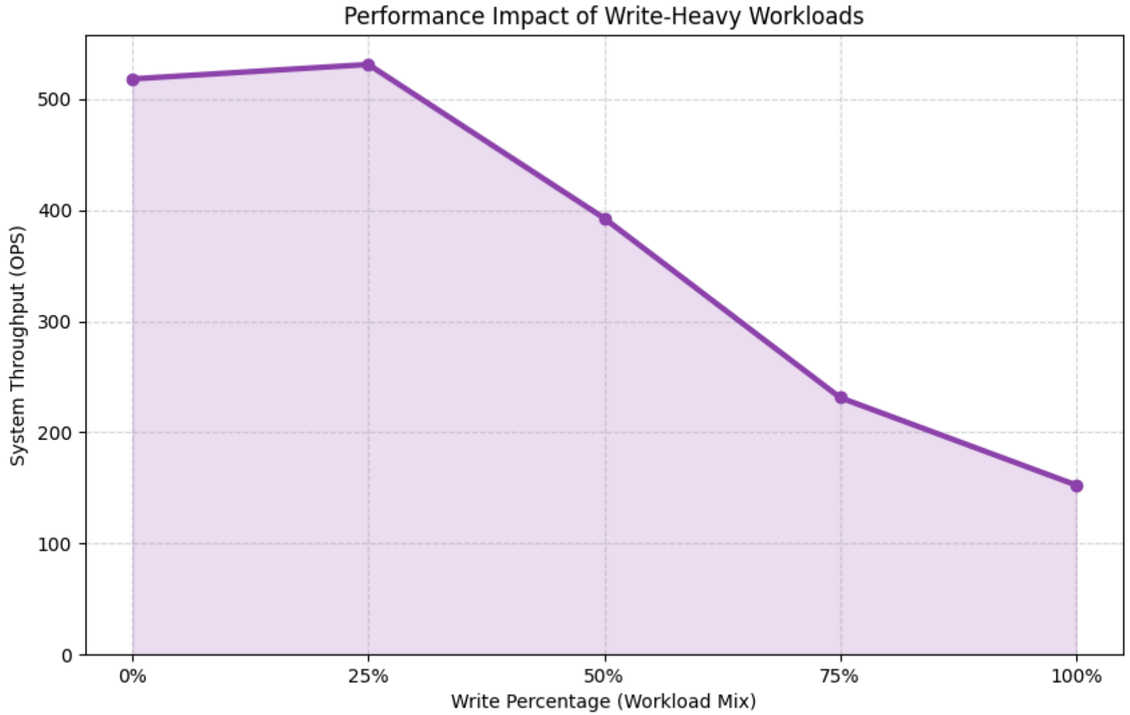
Experimental Results: System Scalability



- Scenario: Analyzed system throughput scaling capabilities under increasing concurrent client load.
- Linear Growth: Throughput scales efficiently from ~80 to ~1050 OPS as concurrency rises to 20 clients.
- Saturation Point: Performance hits a hard ceiling at ~1300 OPS beyond 20 clients, stabilizing despite added load.
- Observation: The plateau indicates the system has reached its physical limits (CPU/Network I/O or lock contention), confirming the bottleneck is capacity-based rather than a software scalability defect.

Experimental Results: Latency Distribution Analysis And Performance Impact of Write-Heavy Workloads





Limitations & Observations

- The "Window of Vulnerability":
  - With W=2, there is a ~175ms window where only 2/3 nodes have data.
  - Trade-off: Immediate latency vs. temporary inconsistency.
- Limits of Read Repair:
  - Observation: Read Repair is opportunistic. "Cold" (unread) data remains stale indefinitely.
  - Requirement: Production systems need Active Anti-Entropy (Merkle Trees) to heal silent data rot.

Conclusion

- Successfully implemented a decentralized, Dynamo-style key-value store.
- DC-Awareness is critical: It is the only way to survive correlated failures.
- Performance: Validated that W < N is essential for tail latency SLAs.
- Resilience: Self-healing confirmed via Hinted Handoff and Read Repair tests.

Thank You