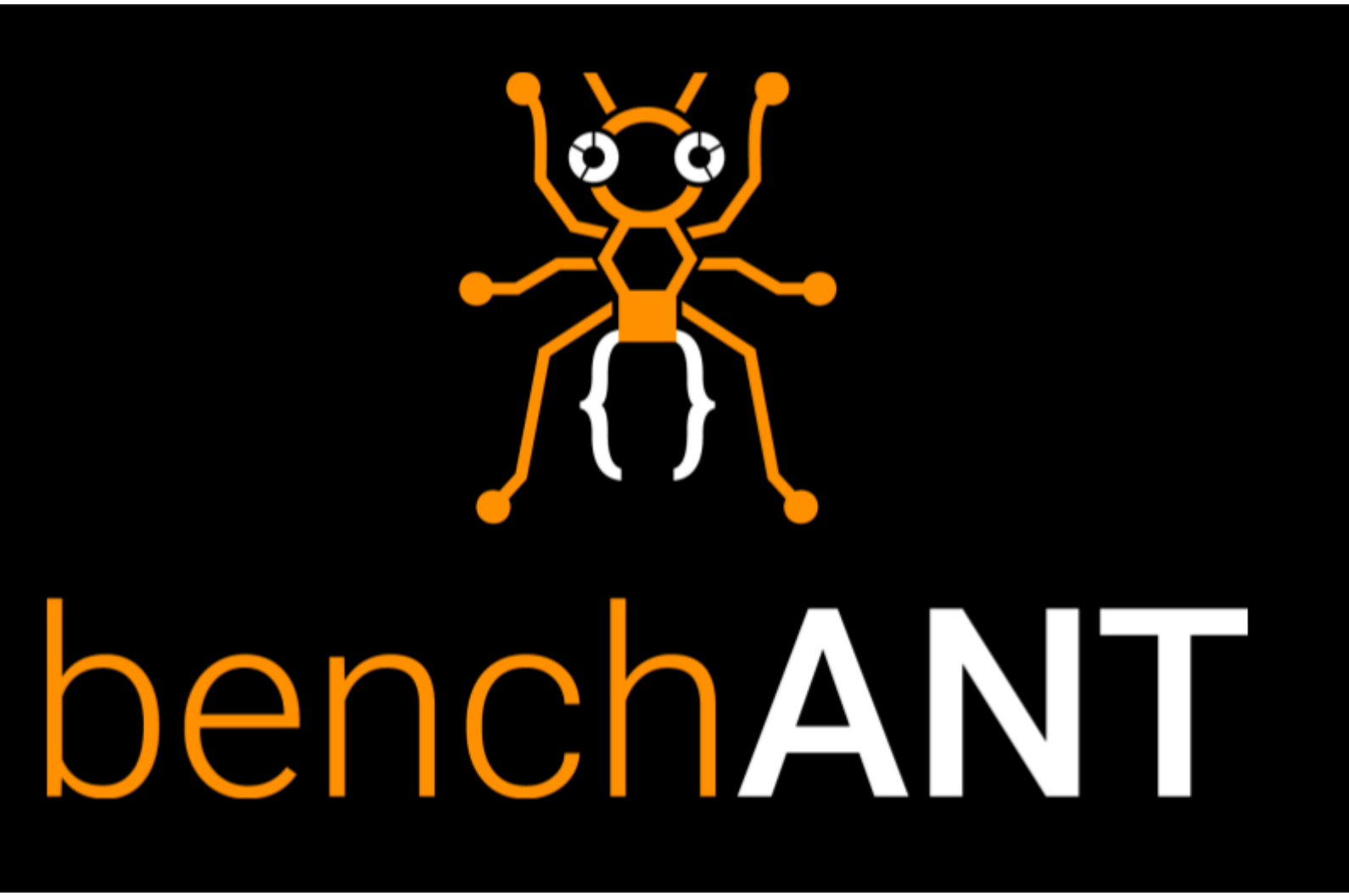


Aerospike Enterprise vs. Apache Cassandra:

Performance and resilience under mixed workload and node failure

Jörg Domaschka, Daniel Seybold, Behrad Babaee

2026

Table of contents

Executive summary **3**
Introduction **4**
Methodology and approach **4**
Databases 4
Aerospike 4
Apache Cassandra 5
Cloud environment and cluster sizing 5
Cloud environment 5
Cluster configuration 5
Benchmarking tool 6
YCSB extensions 6
Benchmarking **7**
Objective 1: Sustained performance under load 7
Workload 7
Calibration 8
Result 8
Summary 13
Objective 2 results: Performance resilience during node failure 14
Workload 14
Results 15
Latency during failure and recovery 16
Redistribution time 17
Summary 17
Disclaimer **21**
About benchANT **21**
Appendix **23**
Yahoo Cloud Serving Benchmark (YCSB) 23
Raw benchmark results 23
Aerospike configuration 23
Cassandra configuration 24

Executive summary

This paper compares Aerospike Enterprise 8.0.4 and Apache Cassandra 5.0.3 under two production-relevant scenarios: sustained mixed-workload performance at scale and performance resilience during node failure.

The benchmark is designed to measure not only peak throughput and latency, but also how each system behaves as conditions become less favorable. In production, data volumes grow, clusters move closer to capacity, workloads evolve, and failures occur. The key question is therefore not just which system performs well in ideal conditions, but which one preserves more stable and predictable behavior as conditions change.

The benchmark used an extended YCSB workload comprising 50% reads, 40% write, 6% updates, and 4% deletes, and was run on AWS EC2. Both systems were provisioned to store approximately 3.4 TB of data, requiring four nodes for Aerospike and six nodes for Cassandra.

In the sustained-load scenario, Aerospike delivered an average of 481,315 ops/s over the course of the benchmark, compared with 138,028 ops/s for Cassandra, approximately 3.5× higher throughput. It also maintained much more stable behavior as the cluster filled. Aerospike's throughput declined by less than 2% from the beginning to the end of the run, whereas Cassandra's declined by approximately 25%. Aerospike also maintained substantially lower and more tightly bounded tail latency across all operation types, with average p99 latency on the client side remaining sub-millisecond for reads, writes, and deletes, and around 1.3 ms for updates. Cassandra operated at materially higher p99 latency throughout, with significantly greater variability, especially on reads and in the extreme tail.

In the node-failure scenario, Aerospike remained operational at both 50 K ops/s and 100 K ops/s, automatically initiated redistribution, and preserved tightly bounded latency during recovery. Cassandra remained operational at 50 K ops/s, but with materially greater latency degradation and variability. At 100 K ops/s, it was unable to sustain stable operation after failure and ultimately experienced a cascading failure that terminated the benchmark.

This difference is primarily driven by their architectural approaches. Systems that defer work to background processes (such as compaction and repair), like Apache Cassandra, tend to exhibit greater variability under sustained and disrupted conditions. By contrast, systems that complete more of the required work before responding, such as Aerospike, maintain tighter and more predictable performance bounds.

Taken together, these results show that Aerospike's advantage is not only higher throughput or lower latency in isolation. It also preserves performance more consistently as the system fills, behaves more uniformly across operation types, and remains more stable during failure and recovery. In other words, Aerospike demonstrated not only stronger raw performance but also more predictable behavior under changing production conditions.

Introduction

This document presents a benchmarking study performed by benchANT comparing Aerospike Enterprise 8 and Apache Cassandra 5 under two production-relevant scenarios: sustained mixed-workload performance at scale and behavior during node failure.

The purpose of this study is not only to compare peak throughput and latency, but to understand how each system behaves as operating conditions change. In production environments, systems do not operate in steady state. Data volumes grow, clusters move closer to capacity, workloads evolve, and infrastructure disruptions such as node failures occur. These factors affect not only absolute performance, but also the consistency of that performance over time.

For this reason, the benchmark evaluates the systems not only on performance, but on how predictably that performance is maintained as conditions change. Predictability is assessed through:

- Throughput variance over time
- Tail latency bounds (p99 and p99.99)
- Performance degradation under sustained load
- Behavior during and after failure

The study examines two complementary dimensions of system behavior:

- **Sustained performance under load**, analysing how throughput and latency evolve as the system operates over time and approaches capacity
- **Performance resilience during node failure**, evaluating how each system responds to disruption, maintains service, and recovers to a stable operating state

Together, these scenarios provide a more complete view of system behavior than steady-state benchmarks alone. They allow us to assess not only how fast each system operates, but how reliably and predictably it behaves as conditions become less favorable.

The results presented in this document are intended to reflect realistic operating conditions and to highlight differences in how each system manages performance, variability, and recovery in production environments.

Methodology and approach

This benchmark is designed to compare Aerospike Enterprise 8 and Apache Cassandra 5 under realistic production-oriented conditions, while keeping infrastructure and workload assumptions as comparable as possible. The methodology focuses on two scenarios: sustained mixed-workload performance as the cluster fills, and performance resilience during node failure. In both cases, the goal is not only to measure peak throughput and latency, but to evaluate how consistently each system maintains performance as conditions change.

Databases

For the evaluations, we use Cassandra and Aerospike Enterprise in their respective latest available version. For Cassandra, this is version 5.0.3, and for Aerospike, this is Enterprise version 8.0.4.

Aerospike

Aerospike is a distributed operational database designed for low-latency, high-throughput workloads with a focus on predictable performance under sustained load. Its architecture combines in-memory indexing with data stored on SSD or NVMe devices, allowing it to deliver near in-memory latency while scaling to large datasets.

A key characteristic of Aerospike's design is that it performs most of the work required to serve a request before responding, rather than deferring significant processing to later background activity. This includes managing data layout and consistency as part of the foreground operation. As a result, performance tends to remain tightly bounded over time, even as data volumes grow or workload characteristics evolve.

This architectural approach is particularly relevant for long-running, mixed workloads where systems must maintain consistent latency and throughput as utilisation increases and operating conditions change.

Apache Cassandra

Apache Cassandra is a distributed wide-column database designed for horizontal scalability and high write throughput. It uses a log-structured storage engine and a peer-to-peer architecture, with tunable consistency levels and background processes such as compaction and repair to manage data organisation and consistency over time.

In Cassandra, a portion of the work associated with writes and updates is deferred to these background processes. This allows the system to handle high write throughput efficiently, but also introduces additional factors that can influence performance over time, particularly under sustained load or during recovery from failure. Operations such as compaction, read repair, and anti-entropy repair can affect latency and contribute to variability, especially in read paths.

These characteristics make Cassandra well-suited for workloads prioritising write scalability, but they also mean that performance behavior under changing conditions depends on how background processes interact with foreground workload and cluster state.

This distinction is particularly relevant in benchmarks that evaluate behavior over time and under disruption, where deferred work can interact with changing conditions.

Cloud environment and cluster sizing

Cloud environment

All benchmarking experiments are executed in a public cloud setting using Amazon Web Services (AWS). The following specifications are applied uniformly across both databases:

- Cloud provider: AWS EC2
- Region: eu-west-1
- Database instance type: i4g.4xlarge[1] (optimized for storage I/O performance) with one 3.75 TB local NVMe storage
- Benchmark driver instance type: c6i.8xlarge (compute-optimized)
- Operating system: Ubuntu 22.04
- Availability configuration: All instances are deployed within a single Availability Zone and private network to minimize cross-zone latency.

Cluster configuration

The cluster configurations are designed to reflect the respective best practices and requirements for each database, while ensuring comparability in terms of hardware resources.

[1] We selected I4g instances as they are optimized for workloads with small to medium sized datasets that perform a high mix of random read/write and require very low I/O latency.

Parameter	Aerospike	Apache Cassandra
Node type	i4g.4xlarge	i4g.4xlarge
Number of availability zones	1	1
Capacity	~3.5 TB	~3.5 TB
Replication factor	2 (recommended[2])	3 (recommended)
Number of nodes	4	6
Write consistency	All (default)	Quorum
Read consistency	One (default)	Quorum
Configuration	Best practice settings for Aerospike Enterprise 8 (details in appendix)	Best practice settings for Cassandra 5.0.3 (details in appendix)

Table 1: Cluster configurations used in both benchmarks

Note: To operate Cassandra in a production environment, periodic repair processes must be run on each node to maintain data consistency. These operations introduce additional load and increase latency variability.

In this benchmark, repairs were not executed, as running them continuously or introducing them at specific points during the test would have made results less consistent and harder to interpret.

Because repair activity typically degrades node performance, excluding it provides a slightly more favorable view of Cassandra's behavior than would be observed in a production deployment.

Benchmarking tool

The benchmarking was conducted using Yahoo! Cloud Serving Benchmark (YCSB), a widely used framework for evaluating NoSQL database performance under different access patterns. For this study, we use an extended version of YCSB developed and maintained by benchANT, which includes several enhancements to support long-running and more representative production workloads. The modified codebase is available as part of benchANT's public YCSB repository.

YCSB extensions

The benchANT version of YCSB introduces several improvements over the default implementation:

- **Long-running benchmark support**: Enables sustained execution over billions of operations, allowing observation of behavior beyond initial steady state.
- **Delete operation support**: Standard YCSB bindings do not consistently support deletes. This extension enables full CRUD workloads (create, read, update, delete), better reflecting real-world usage.
- **Uniform-growing distribution**: A custom request distribution that extends the traditional uniform model. Unlike the default implementation, which only targets the initial dataset, this distribution dynamically includes newly inserted records during the run phase, ensuring a uniform access pattern across both existing and newly created data.

[2] Most deployments of Aerospike employ replication factor of 2.

Due to the inclusion of delete operations, some requests will target records that no longer exist. This is expected behavior. With the configured workload mix, the proportion of “not found” operations starts near zero and gradually approaches approximately 8% during steady-state execution.

YCSB distinguishes between successful operations and failed operations. By default, “not found” responses are grouped with failures. For this benchmark, we require a more accurate representation of application-level behavior. Therefore, the YCSB extension reports combined latencies for successful operations and “not found” responses, while other error types (e.g., network failures) are still reported separately. No such errors were observed during the benchmark runs.

Benchmarking

The following sections discuss the results for the benchmarks performed for each of the two objectives.

Objective 1: Sustained performance under load

The purpose of Objective 1 is to determine the maximum throughput each system can sustain while remaining within its expected latency envelope, and to observe how that behavior evolves as the cluster fills and utilisation increases. This objective therefore evaluates not only peak throughput, but also whether performance remains stable as conditions become less favorable over the course of a long-running benchmark.

A second aim of this objective is to examine how consistently each system performs across different operation types. Production workloads are rarely dominated by a single access pattern. Systems must handle reads, writes, updates, and deletes together, and their suitability for latency-sensitive applications depends not only on aggregate throughput, but on whether behavior remains well-bounded across the full workload mix.

To identify the appropriate operating point for each technology, we first conducted a series of calibration runs. In these tests, workload intensity was gradually increased until throughput either plateaued or latency degraded beyond the expected envelope for the system. This allowed us to determine the highest stable operating point for each database without pushing it into an unstable regime characterised by throughput oscillation, rising tail latency, or other signs of instability.

Once that operating point was identified, it was used in the full-scale benchmark. In this phase, each system was evaluated against a preloaded dataset of 100 GB and then driven from a lightly populated state towards near capacity under a sustained mixed workload.

The evaluation, therefore, focuses on three related questions:

- How much throughput each system can sustain while remaining within its latency envelope?
- How stable throughput and tail latency remain as the benchmark progresses and the cluster fills?
- How consistently each system performs across reads, writes, updates, and deletes?

Workload

For Objective 1, the workload is designed to simulate a long-running, high-ingest application with a mixed operational profile. The benchmark begins with an initial dataset of 100 GB. The benchmark then executes a total of 5.8 billion operations, distributed as follows:

- 50% reads
- 40% writes
- 6% updates
- 4% deletes

Scan operations are disabled in order to focus on latency-sensitive point queries.

The workload uses the uniform-growing request distribution, which ensures that requests are spread across both the initial dataset and newly inserted records. This better reflects long-running production systems in which the working set evolves continuously over time rather than remaining fixed. Over the course of the benchmark, approximately 2.32 billion write operations are performed, resulting in substantial data growth as the system progresses from an initially light state towards near capacity.

Workload configuration

YCSB Parameter	Value
workloadClass	site.ycsb.workloads.CoreWorkload
operations	5,800,000,000
fieldCount	10
fieldLength	200
requestdistribution	uniform-growing
readProportion	0.5
updateProportion	0.06
insertProportion	0.4
scanProportion	0.0
deleteProportion	0.04

Table 2: YCSB configurations

Calibration

Each calibration run used a fresh cluster deployment, an initial dataset of 100 GB, and 2.8 billion operations. Workload intensity was varied by changing the number of concurrent YCSB client threads, allowing identification of:

- throughput saturation points
- latency degradation thresholds
- onset of unstable behavior, such as throughput oscillation or tail-latency spikes

For Aerospike, the target latency envelope was approximately 1 ms p99 for read operations. For Cassandra, the target latency envelope was approximately 10 ms p99.

Aerospike was evaluated with 200, 300, and 400 client threads. Cassandra was evaluated with 100, 200, 300, and 400 client threads. Based on these calibration results, the full benchmark was executed using 200 client threads for Aerospike and 400 client threads for Cassandra, representing the highest stable operating point for each system.

We do not systematically present calibration results, but refer to them whenever their results impact any configuration options, e.g., workload intensity.

Result

Sustained throughput

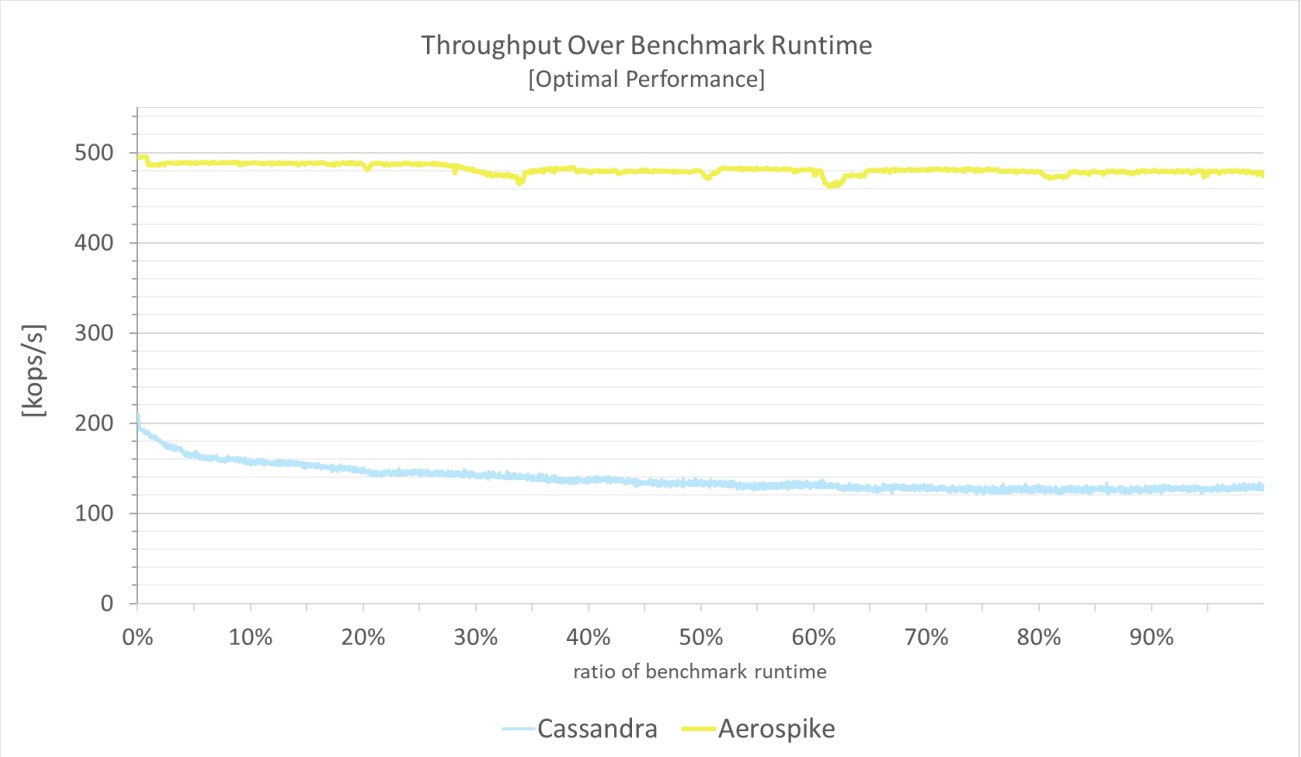
The throughput results are summarised in Table 3. The slight difference in the total number of write operations between the two runs is due to YCSB's stochastic workload generation. Since operations are selected randomly, the exact number of writes may vary slightly from run to run. In both cases, however, writes remain effectively 40% of the total workload.

	Aerospike Enterprise v8	**Cassandra v5.0.3**
Average throughput [operations/sec]	481,315	138,028
Minimum throughput	461,392	120,962
Maximum throughput	495,344	212,577
Clients threads used	200	400
Benchmark duration[3]	3h 21m 11s	11h 40m 23s
Total `write` operations *(% of all 5.8B operations)*	2,319,990,108 *(39.99983%)*	2,320,025,752 *(40.00044%)*

Table 3: Evaluation set-up and throughput results

Averaged over the full benchmark duration, Aerospike delivers approximately 3.5× higher throughput than Apache Cassandra under this workload. More importantly, Aerospike maintains significantly more stable throughput throughout the run. Its throughput remains tightly bounded, varying by approximately ±3.5% around the mean, whereas Cassandra exhibits substantially greater variability, ranging from −12% to +54% relative to its average.

Figure 1 shows throughput over the course of the benchmark. Because the two systems complete the workload in different amounts of time, the x-axis is normalised to show relative benchmark progress rather than absolute time.



[3] Duration varies, as each system requires a different amount of time to complete the same workload.

Figure 1: Throughput of Aerospike and Apache Cassandra over benchmark runtime.

As shown in Figure 1, Aerospike exhibits minimal degradation as the system moves from an almost empty state to near full capacity. Average throughput declines from 488,231 ops/s in the first 10% of the run to 478,751 ops/s in the final 10%, representing less than a 2% reduction.

Cassandra shows a much more pronounced decline. Average throughput falls from 169,382 ops/s in the first 10% of the run to 127,557 ops/s in the final 10%, a reduction of approximately 25%. Even when the initial phase is excluded, degradation remains visible: comparing the midpoint of the run (132,493 ops/s) with the final 10% still indicates a decline of approximately 4%.

Taken together, these results show that Aerospike not only operates at a substantially higher throughput level, but also preserves that throughput far more consistently as the benchmark progresses and the system fills.

Sustained latency

Table 4 summarises p99 and p99.99 latency by operation type over the full benchmark duration.

		P99				P99.99			
		Min	Avg	Max	Peak to trough	Min	Avg	Max	Peak to trough
READ	Aerospike	0.83	0.97	1.40	0.57	1.92	4.04	4.56	2.64
	Cassandra	4.81	8.41	11.07	6.26	10.08	15.84	188.03	177.95
DELETE	Aerospike	0.66	0.76	0.93	0.27	1.44	4.01	4.40	2.96
	Cassandra	3.46	4.97	5.75	2.29	8.27	12.27	86.53	78.26
UPDATE	Aerospike	1.07	1.26	1.71	0.64	2.20	4.29	6.59	4.39
	Cassandra	3.46	5.03	5.84	2.38	8.62	12.38	86.78	78.16
WRITE	Aerospike	0.68	0.80	0.98	0.30	2.14	4.29	4.57	2.43
	Cassandra	3.62	5.20	6.00	2.38	9.03	12.38	87.10	78.07

Table 4: P99 and P99.99 latency by operation type in milliseconds for both technologies.

Aerospike delivers not only substantially lower tail latency across both percentiles, but also much tighter latency bounds over time. Across all operation types, its p99 and p99.99 values remain within a relatively narrow range, indicating stable and predictable behavior throughout the benchmark.

Just as importantly, Aerospike performs well across the full workload mix. Reads, writes, and deletes remain sub-millisecond at p99 on average, while updates are only modestly slower. This indicates that Aerospike maintains a broadly consistent performance profile across different kinds of operations, rather than performing well on only a subset of them.

Cassandra, by contrast, does not show the same consistency across operation types. Its write, update, and delete paths perform relatively better than its read path, but all remain materially slower than Aerospike. Reads are especially affected, with both higher average p99 latency and significantly greater variability over time. This suggests that Cassandra's behavior is less uniformly bounded across a mixed workload, particularly as conditions become less favorable.

The difference becomes even more pronounced in the extreme tail. Aerospike's p99.99 latency remains relatively contained across all operation types, whereas Cassandra shows large excursions, especially on reads, where the maximum p99.99 reaches 188 ms, compared with 4.56 ms for Aerospike. The gap between the two systems is therefore not only one of higher average tail latency, but also of materially greater instability in the extreme tail.

This difference is consistent with the architectural behavior of the two systems. Aerospike tends to perform the work required for the operation before responding, rather than deferring clean-up or reorganisation to later background activity. That contributes to tighter latency bounds across operation types, even where one path, such as updates, may be modestly slower than another. From that perspective, the fact that Aerospike updates are slightly slower than its reads or writes is not a sign of imbalance so much as a consequence of doing more of the required work before acknowledging the operation.

Cassandra, by contrast, often postpones parts of the work associated with writes and updates, relying on later background processes such as compaction and repair to reconcile and reorganise data. That design can make some foreground operations appear relatively efficient in isolation, but it also shifts part of the cost into future work. Under sustained mixed load, this contributes to greater variability over time and less predictable read behavior, as the system must continue serving requests while background maintenance and coordination remain active.

More broadly, these results illustrate a general pattern: systems that defer work to background processes such as compaction and repair tend to exhibit greater variability over time, particularly under sustained mixed workloads. Systems that perform more of the required work before responding, by contrast, are more likely to maintain tighter latency bounds and more predictable behavior.

Figures 2 through 5 show the evolution of p99 and p99.99 latency over time for each operation type. Two patterns are clear. First, Aerospike shows only marginal degradation as the benchmark progresses and the system moves from a lightly populated state towards near capacity. Second, Cassandra exhibits both greater jitter and a clearer upward drift in latency over time, particularly on reads and in the extreme tail.

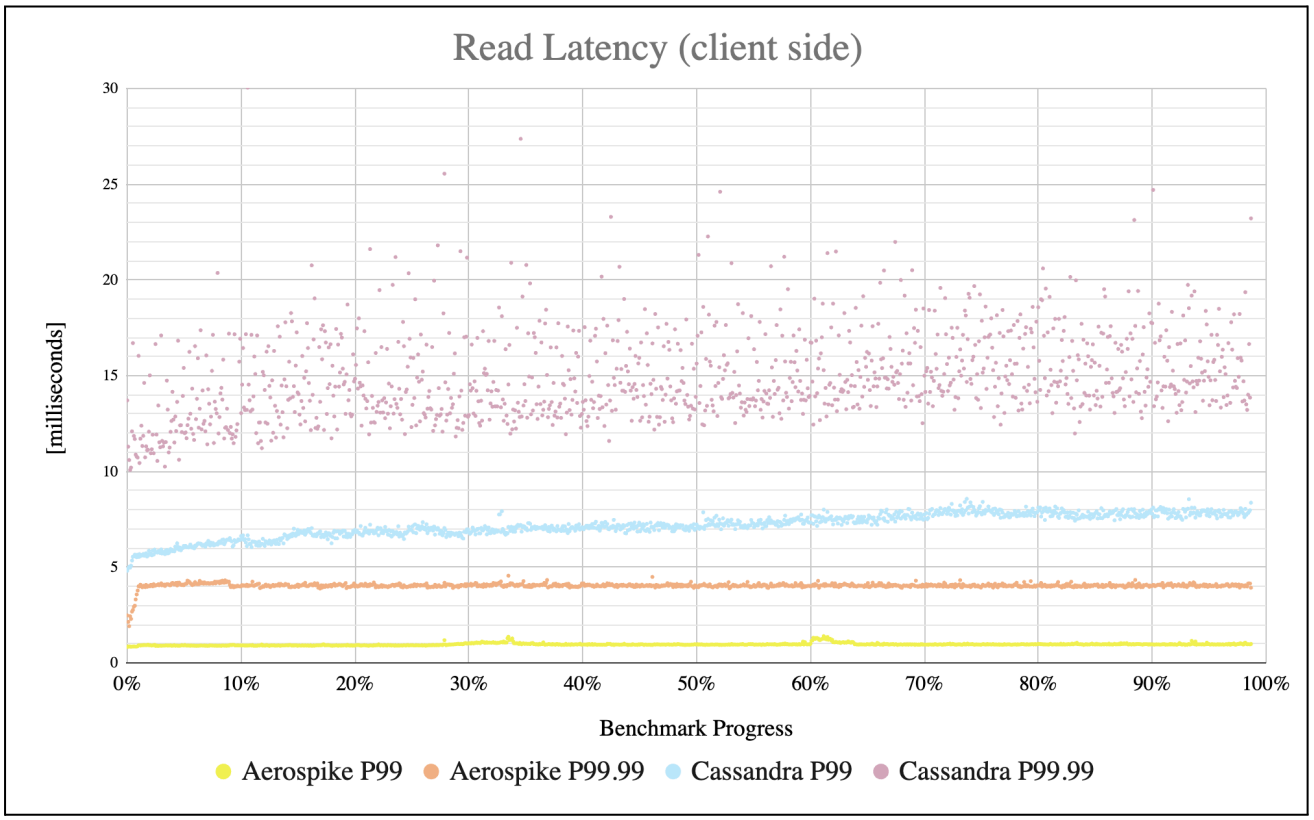


Figure 2: Aerospike and Apache Cassandra P99 and P99.99 latency.
Note: The vertical axis is capped at 30 ms, as the majority of values fall within this range.

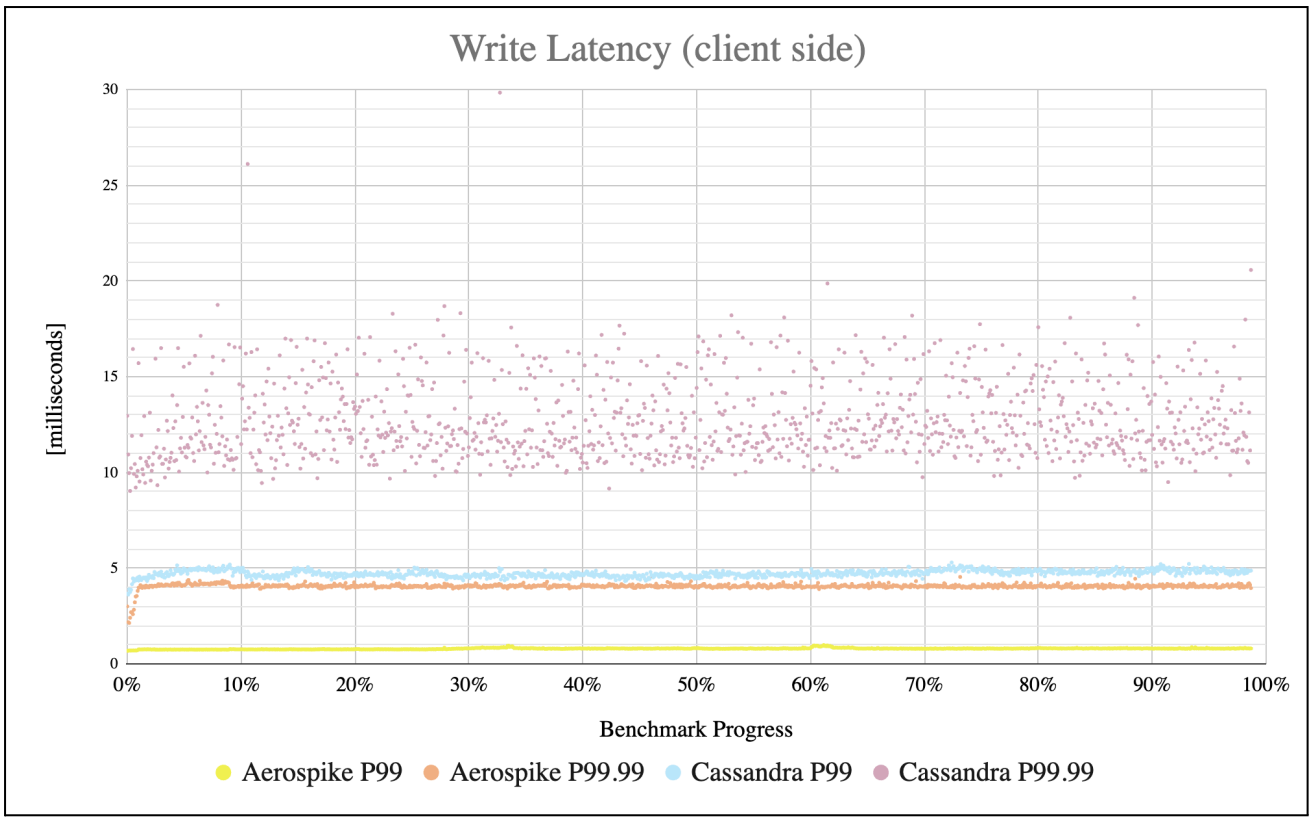


Figure 3: Aerospike and Apache Cassandra P99 and P99.99 write latency
Note: The vertical axis is capped at 30 ms, as the majority of values fall within this range.

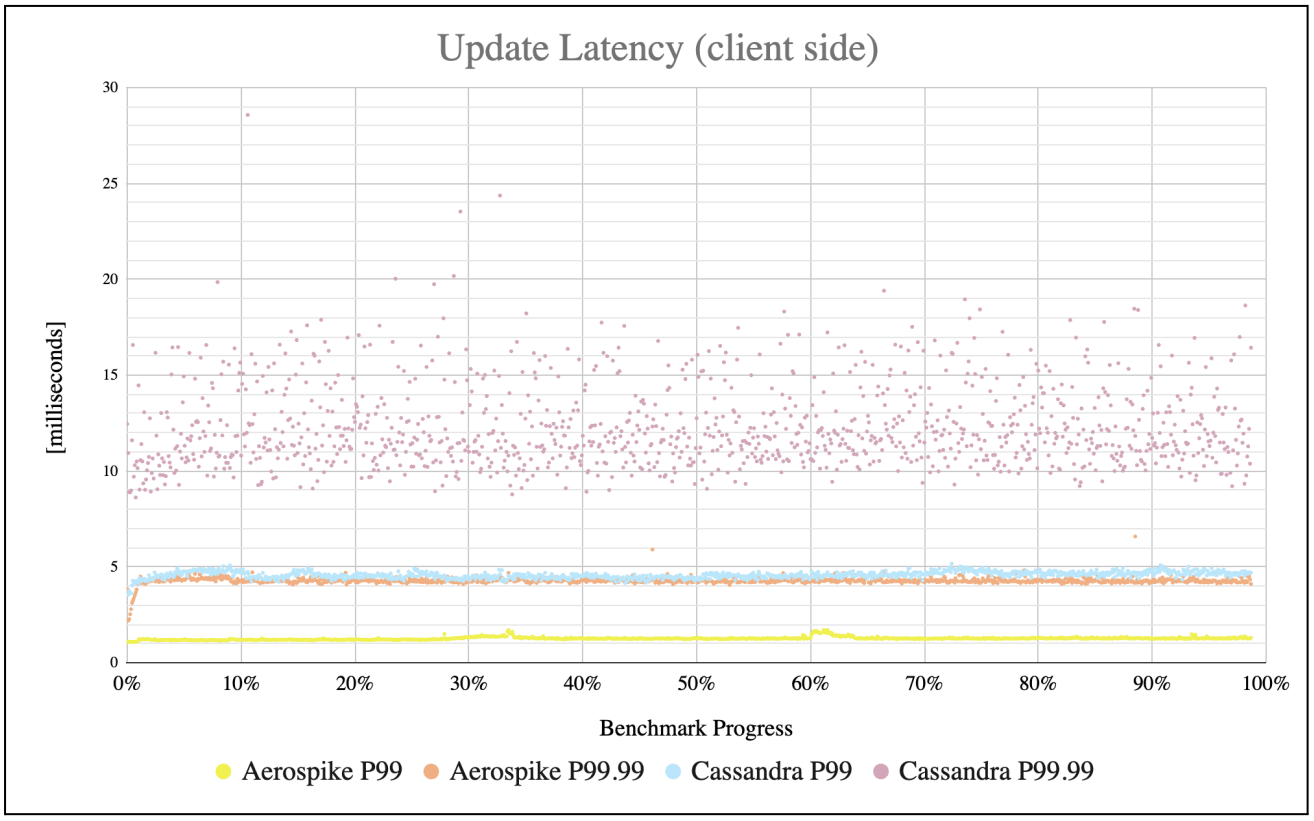


Figure 4: Aerospike and Apache Cassandra P99 and P99.99 update latency
Note: The vertical axis is capped at 30 ms, as the majority of values fall within this range.

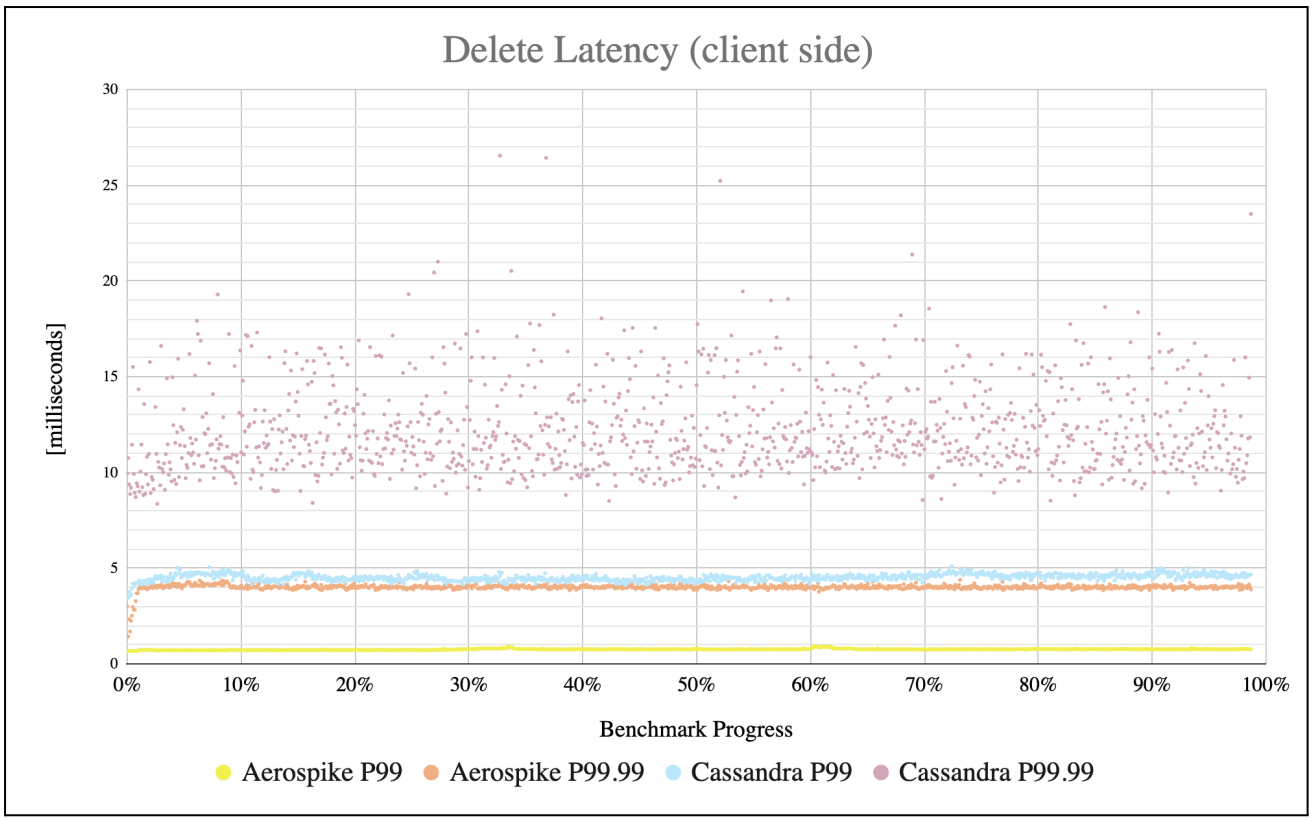


Figure 5: Aerospike and Apache Cassandra P99 and P99.99 delete latency
Note: The vertical axis is capped at 30 ms, as the majority of values fall within this range.

Summary

Under identical workload conditions, Aerospike and Apache Cassandra exhibit materially different behavior under sustained load.

Aerospike delivers approximately 3.5× higher average throughput while maintaining much tighter throughput bounds over the duration of the benchmark. As the cluster moves from lightly populated to near capacity, throughput degradation remains minimal. Cassandra, by contrast, shows much larger throughput fluctuation and substantially greater degradation over time.

The latency results reinforce the same conclusion. Across all operation types, Aerospike maintains markedly lower p99 latency, remaining sub-millisecond for reads, writes, and deletes, and around 1.3 ms for updates. Cassandra operates at materially higher latency levels throughout, with average p99 values ranging from approximately 5 ms to more than 8 ms.

The difference is even more pronounced in the extreme tail. Aerospike's p99.99 latency remains relatively bounded across all operation types, whereas Cassandra shows large excursions, especially on reads. This indicates that the gap between the two systems is not only one of lower tail latency, but also of greater stability and predictability under sustained load.

Aerospike also shows a more balanced performance profile across the workload mix. Its behavior remains tightly grouped across reads, writes, updates, and deletes, with updates only modestly slower than other operations. Cassandra does not show the same consistency, particularly on reads, where latency is both higher and more variable. This suggests that Aerospike sustains predictable behavior across mixed workloads more effectively, whereas Cassandra's performance depends more heavily on operation type and on background processes such as compaction and repair.

Taken together, these results show that Aerospike delivers not only higher throughput and lower tail latency, but also a more uniform and predictable response profile across the full workload. Cassandra, despite operating with 50% more nodes, exhibits higher latency, greater variability, and more severe extreme-tail behavior. For latency-sensitive production systems, this level of latency variability would translate into inconsistent user response times in interaction-critical systems.

Objective 2 results: Performance resilience during node failure

The purpose of Objective 2 is to evaluate how each system behaves when a node fails during an active workload. In distributed environments, node failures can occur due to hardware faults, network issues, virtual machine termination, or other infrastructure events. While distributed databases are generally designed to remain available under such conditions, the impact on throughput, latency, and recovery behavior can vary materially.

This objective, therefore, evaluates not only whether the system remains operational after a node failure, but also how well it preserves predictable performance during disruption and how it recovers to a new stable operating state. The evaluation focuses on three related questions:

- The immediate performance impact of node failure, particularly on tail latency
- Cluster behavior during data redistribution and recovery
- The time required to reach a stable post-failure operating state

Workload

Unlike Objective 1, this objective uses a fixed-throughput workload. The workload from Objective 1 is not suitable here for two reasons. First, Objective 1 deliberately pushes each system to its highest stable operating point, so removing a node under those conditions would immediately push the surviving cluster beyond its available capacity. Second, Objective 2 requires both systems to perform the same amount of work at the moment of failure, whereas Objective 1 is designed to compare the maximum throughput each technology can sustain individually.

To determine an appropriate fixed throughput, two constraints were considered. The first is that the target load must be within the capacity of Cassandra, since Cassandra showed materially lower throughput limits than Aerospike in Objective 1. The second is that, after a node failure, the remaining cluster must still retain enough headroom not only to serve the workload but also to perform redistribution and recovery work.

Cassandra sustained approximately 138 K ops/s in Objective 1 using a six-node cluster. A single-node failure removes roughly one-sixth of the available cluster capacity, reducing the implied post-failure ceiling to approximately 113 K ops/s. Allowing further headroom for redistribution and recovery, two workload levels were selected for the resilience benchmark:

- 50 K ops/s
- 100 K ops/s

These two workload levels allow evaluation under both moderate and higher post-failure stress.

The workload mix remains the same as in Objective 1:

- 50% reads
- 40% writes
- 6% updates
- 4% deletes

Each benchmark begins with an initial dataset of 100 GB. The workload runs for four hours before failure is introduced. At that point, a single database node is forcibly terminated. The failed node is

not replaced. Metrics are then collected for a further eight hours in order to capture the immediate effect of the failure, the redistribution period, and the post-recovery behavior of the system.

This setup makes it possible to evaluate system behavior under node failure at controlled and comparable workload levels while observing whether performance remains bounded during recovery.

Recovery behavior

The two systems differ materially in how recovery is initiated.

Aerospike automatically begins rebalancing after a node failure. No manual intervention is required to start redistribution. Cassandra behaves differently: after node loss, redistribution must be initiated explicitly by the operator. Because the purpose of this objective is to observe behavior during active redistribution, the redistribution process in Cassandra was started immediately after the failure was introduced.

Results

Throughput during failure and recovery

Aerospike successfully sustained the target workload in both the 50 K ops/s and 100 K ops/s scenarios. In both cases, the cluster remained operational throughout the test, automatically rebalanced the data, and continued serving the workload for the full duration of the benchmark.

Figure 6 shows the throughput results for both workload levels. At 50 K ops/s, redistribution completes in approximately 3 hours. At 100 K ops/s, redistribution takes approximately 6.5 hours. The longer redistribution time at the higher workload level is expected for two reasons. First, the cluster has ingested more data during the first four hours of the run, so more data must be redistributed after the failure. Second, more cluster resources are consumed by foreground workload processing, leaving less headroom for redistribution work.

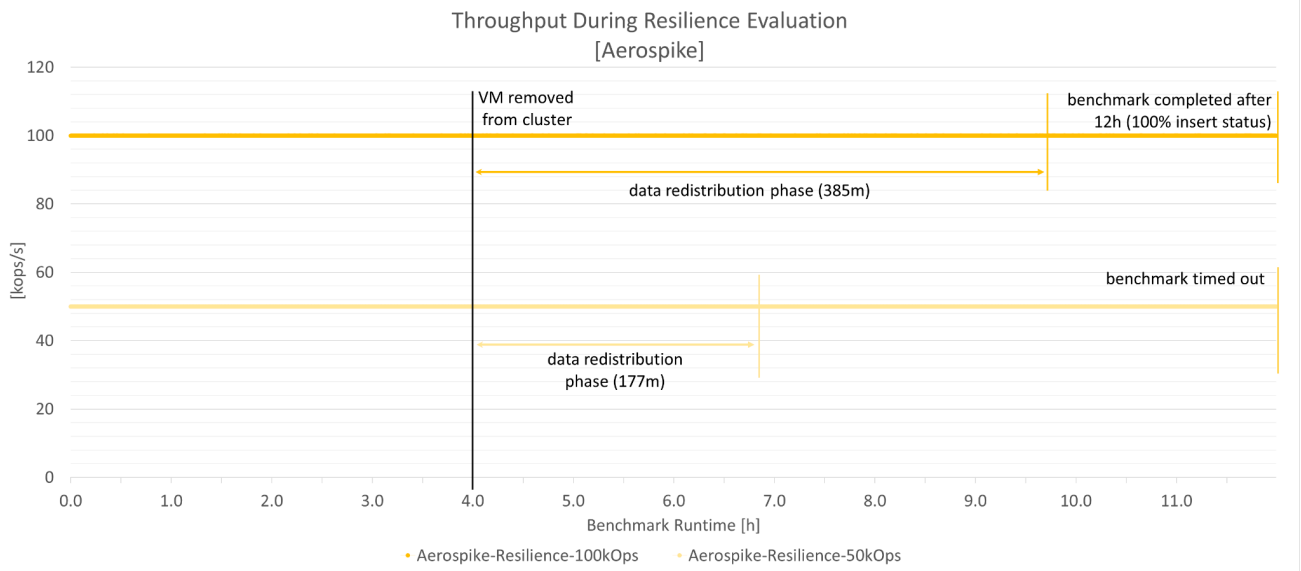


Figure 6: Aerospike throughput over benchmark runtime for resilience evaluation.

Cassandra successfully sustained the 50 K ops/s scenario, completed redistribution, and remained operational for the duration of the test. At 100 K ops/s, however, Cassandra was unable to preserve stable operation. The cluster initially survived the node failure and completed redistribution, but later became unstable. A further node failure occurred, increasing pressure on the remaining cluster and triggering a cascading failure, at which point the benchmark had to be terminated.

Figure 7 shows the throughput behavior for both Cassandra runs.

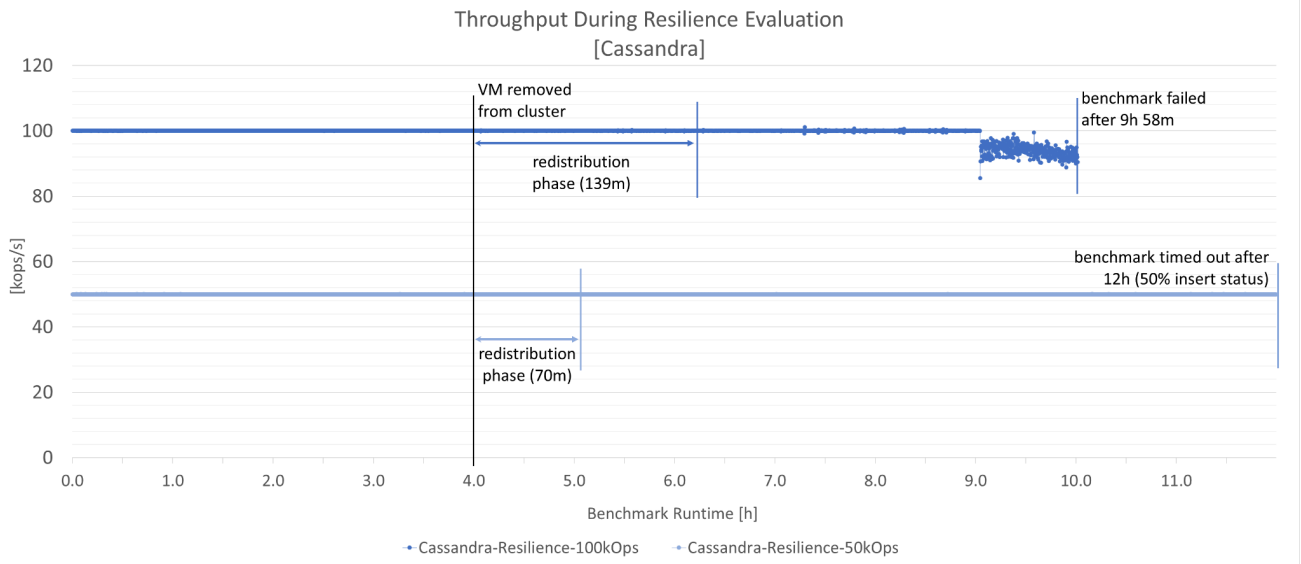


Figure 7: Cassandra throughput over benchmark runtime for resilience evaluation.

These results indicate that Aerospike remained operational and throughput-stable at both tested workload levels, whereas Cassandra remained stable only at the lower workload level.

Latency during failure and recovery

Figures 8 through 11 show p99 latency for each operation type during the resilience benchmark. Figures 12 and 13 show the corresponding p99.99 latency for both systems at both workload levels.

Aerospike's latency behavior during node failure is largely uneventful. At the point of failure and the start of rebalancing, p99 latency increases modestly by approximately 0.1 ms across all operation types. During redistribution, latency remains tightly bounded, with only occasional spikes on read and update operations. Once redistribution completes, p99 latency improves across all operations, although it does not fully return to pre-failure levels. This is expected, as the cluster continues operating with one fewer node and therefore reduced overall capacity. Just as importantly, Aerospike does not show progressive degradation over time after the failure. Once the system settles into its post-failure regime, latency remains tightly bounded and behavior remains consistent.

Cassandra shows a markedly different pattern. At 50 K ops/s, the cluster remains available and completes redistribution, but latency degradation during and after failure is much more pronounced than in Aerospike. At 100 K ops/s, latency degrades significantly after failure, and the system eventually becomes unstable enough to trigger further node loss and cluster failure.

At the point of failure, Cassandra's p99 latency for writes, updates, and deletes increases by roughly 0.5 ms, with spikes exceeding 2 ms. Read latency also increases, although that increase is less visually distinct because the read path is already considerably more variable. This elevated latency is maintained throughout the redistribution period, and write, update, and delete operations become visibly more jittery than before the failure.

At 50 K ops/s, once redistribution has completed, the p99 latency of write, update, and delete operations falls back towards the pre-failure level, and the more pronounced spikiness in those paths largely disappears. Read behavior is different: read p99 latency does not return to its previous level after redistribution and instead continues to drift upward over the remainder of the benchmark, with spikes reaching approximately 3.5 ms.

At 100 K ops/s, the pattern is more severe. During node failure and redistribution, p99 latency for all operations increases steadily. After redistribution completes, write, update, and delete latency

improve only marginally and do not return to pre-failure levels. Read latency recovers even less, continuing to rise in a spiky pattern. At roughly ten hours of runtime, additional Cassandra nodes become unavailable, and the cluster can no longer continue serving the benchmark workload.

Taken together, this is consistent with the behavior observed under sustained load: when work is deferred to background processes, disruption amplifies variability rather than containing it. As a result, performance remains less predictable during recovery.

Redistribution time

Cassandra completes redistribution more quickly than Aerospike. This is partly explained by the different redistribution conditions in the two systems.

In Cassandra, a node failure represents the loss of one out of six nodes, so approximately one-sixth of the data must be redistributed. In addition, because Cassandra uses a replication factor of three, two copies of the affected data remain in the cluster during recovery. In Aerospike, a node failure represents the loss of one out of four nodes, so approximately one-quarter of the data must be redistributed, and only one remaining copy exists during that period.

This difference in redistribution time is therefore structurally understandable. It should also be treated as secondary to the more important observation: how predictably the system behaves while redistribution is taking place. Both systems can be tuned to redistribute more or less aggressively through configuration. What matters more in this benchmark is that Aerospike maintained tightly bounded performance throughout the redistribution period, whereas Cassandra showed materially greater degradation and instability, especially at the higher workload level.

Summary

Objective 2 shows a clear difference in how the two systems behave under node failure. The results are summarised in Table 5.

Metric	Aerospike (50 K ops/s)	Cassandra (50 K ops/s)	Aerospike (100 K ops/s)	Cassandra (100 K ops/s)
Cluster remains operational	Yes	Yes	Yes	No (Cassandra failure)
Redistribution completed	Yes (2h 55m)	Yes (1h 10m)	Yes (6h 25m)	Yes (2h 19m)
Throughput maintained	Yes	Yes	Yes	No, failed under load
P99 latency during redistribution	+0.1 ms and tightly bounded	+0.5 ms and jittery	+0.1 ms and tightly bounded	+6.0 ms and extremely jittery
Post-distribution latency	No degradation and tightly bounded	Degrades over time	No degradation and tightly bounded	Degrades over time

Table 5: Objective 2 results summary

Aerospike remained operational at both 50 K ops/s and 100 K ops/s, automatically initiated redistribution, and preserved tightly bounded latency throughout the recovery period. The performance impact of node loss was visible but modest. Latency increased slightly at the point of failure, remained stable during redistribution, and settled into a new, still tightly bounded post-failure operating regime once recovery completed.

Cassandra remained operational only at the lower workload level. At 50 K ops/s, it completed redistribution and continued serving the workload, but with much greater latency degradation and

variability than Aerospike, particularly on reads. At 100 K ops/s, Cassandra was unable to sustain predictable behavior after failure and eventually suffered a cascading failure that terminated the benchmark.

The most important distinction is not simply whether redistribution completed, but how each system behaved while recovery was in progress. Aerospike preserved stable and predictable throughput and latency during disruption. Cassandra, by contrast, showed significantly more degraded and less bounded behavior during and after failure, with read performance in particular remaining impaired even after redistribution had completed.

Taken together, these results show that Aerospike provides materially stronger performance resilience during node failure. It not only remains available but also preserves predictable behavior under disruption and recovers in a controlled manner. Cassandra can recover under lower stress, but its performance during recovery is less stable, less predictable, and more sensitive to workload level.

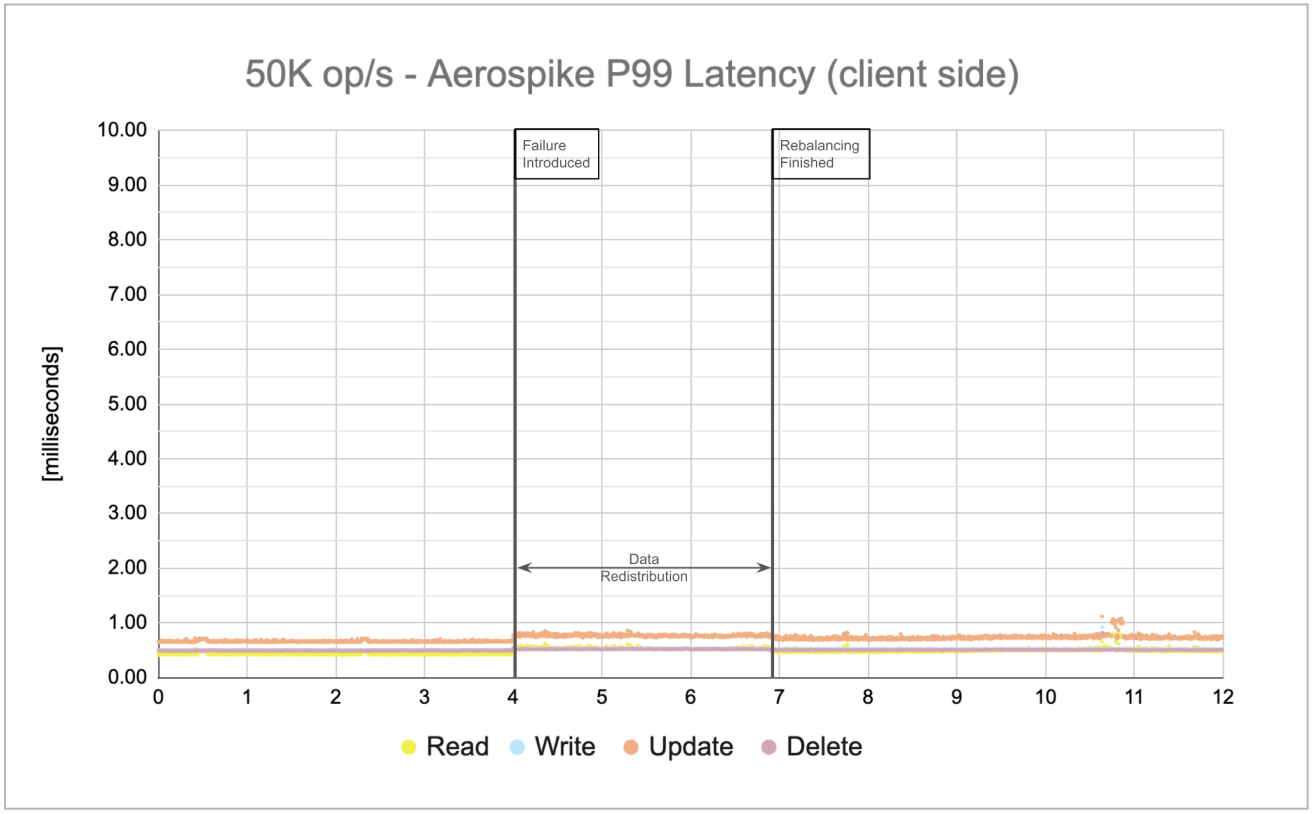


Figure 8: Aerospike P99 latency at 50K ops/s

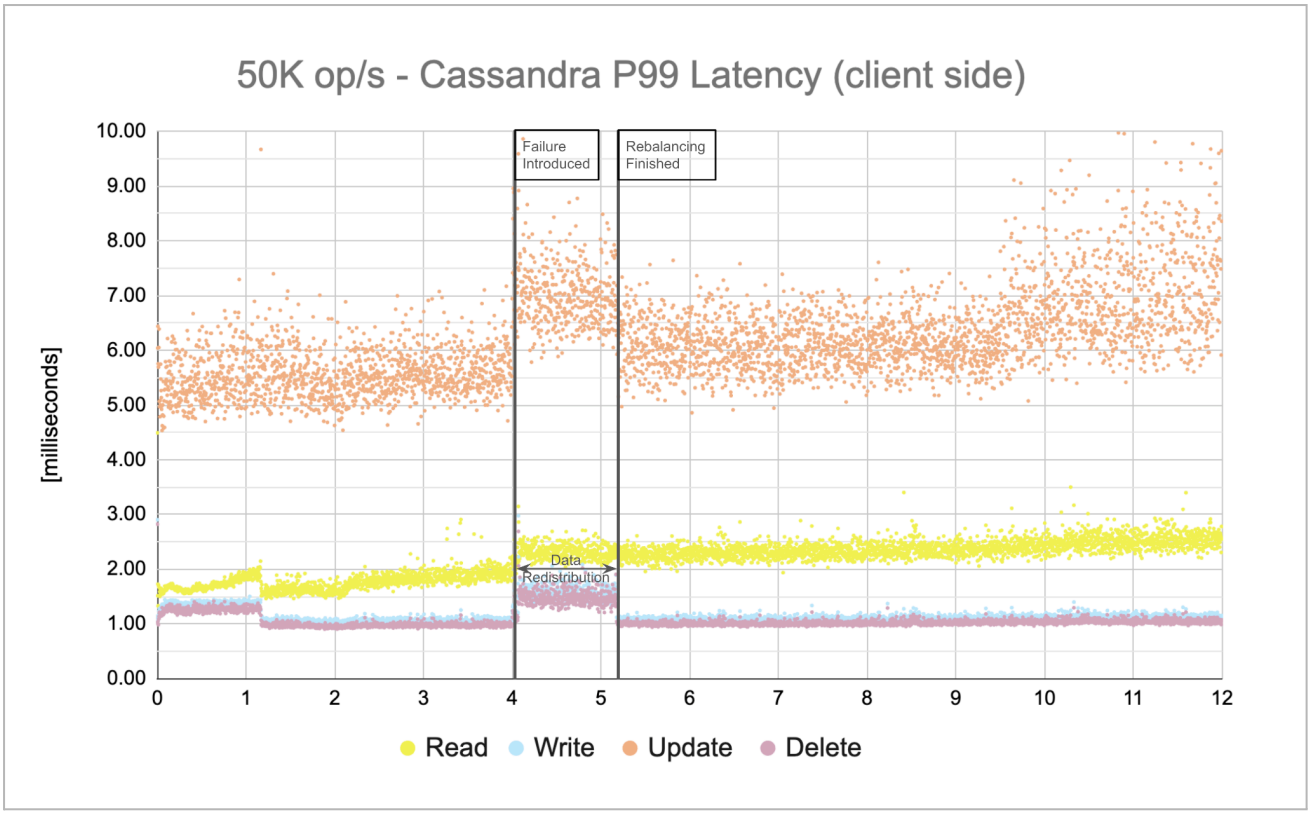


Figure 9: Cassandra P99 latency at 50K ops/s

Note: The vertical axis is capped at 10 ms, as the majority of values fall within this range.

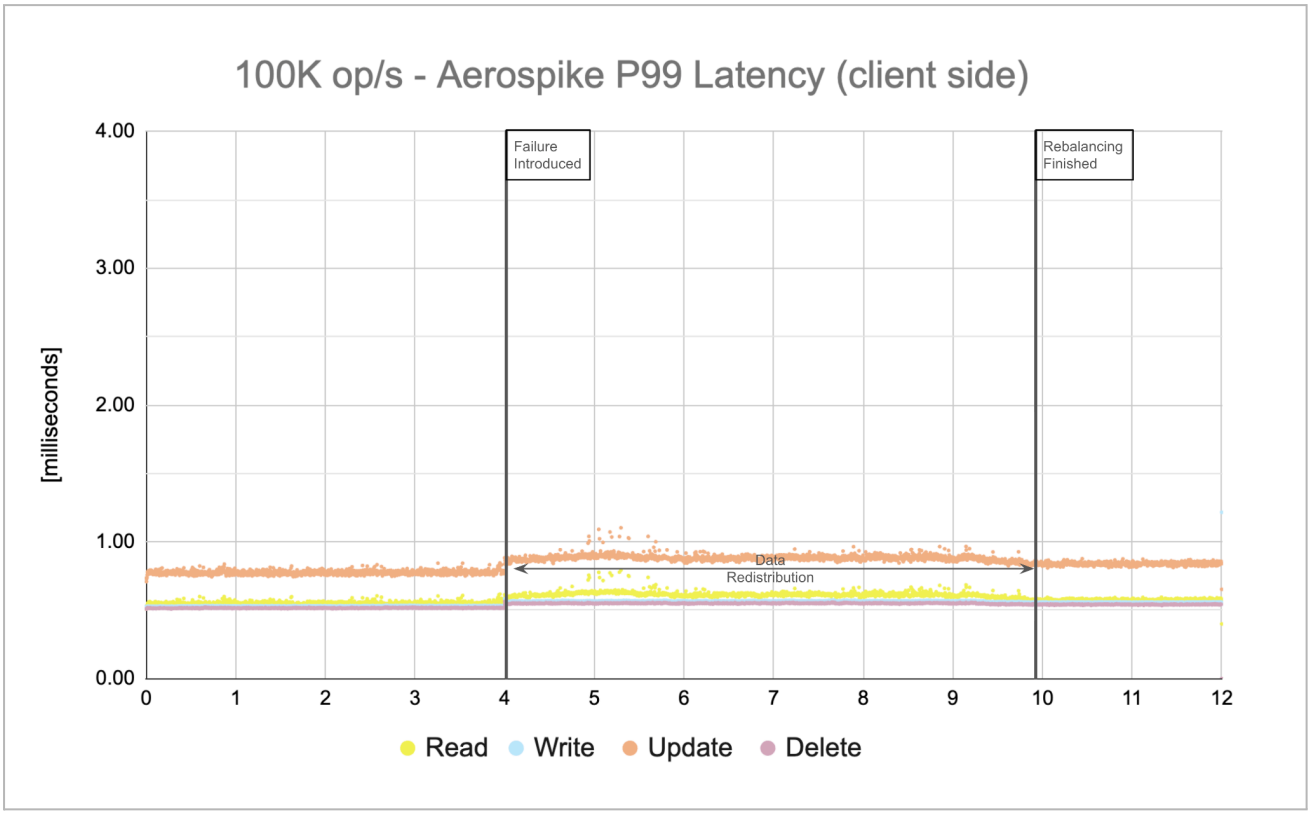


Figure 10: Aerospike P99 latency at 100K ops/s

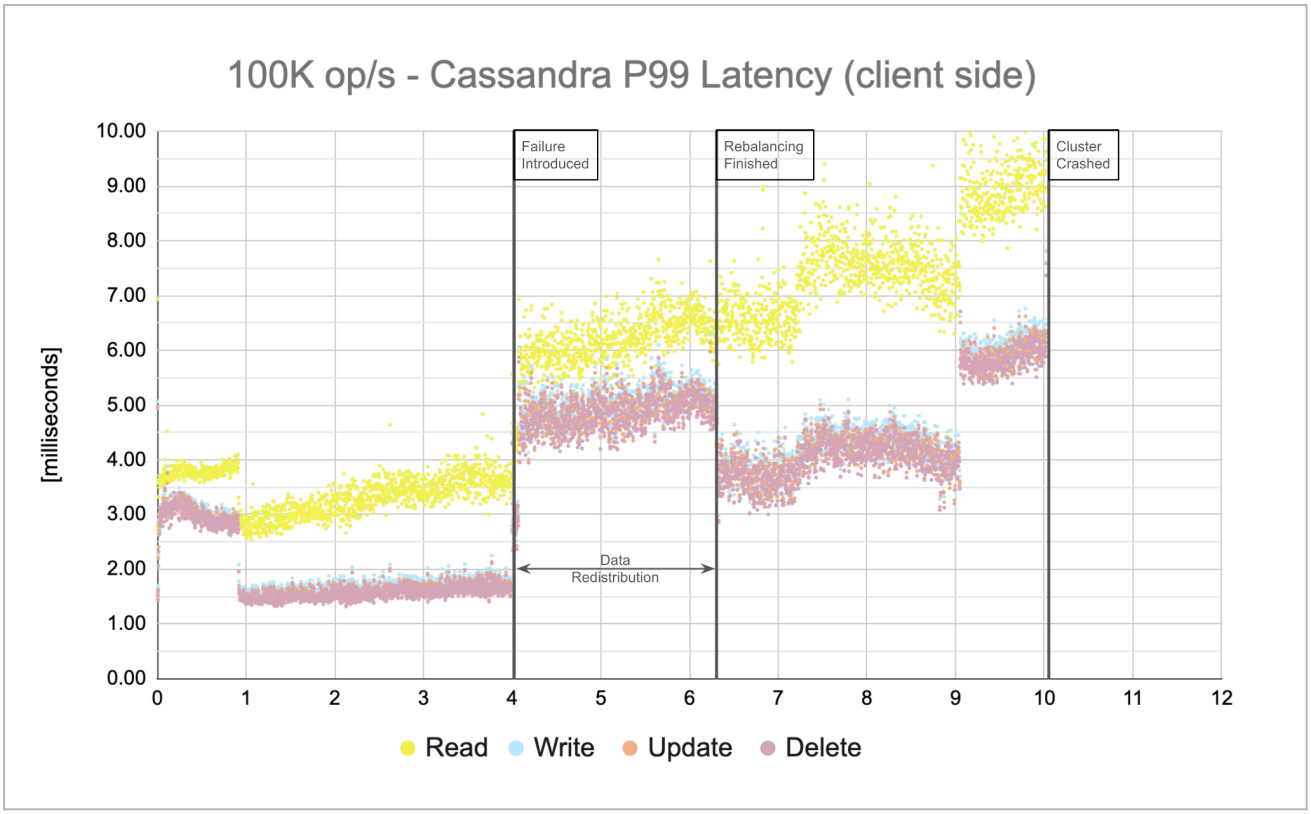


Figure 11: Cassandra P99 latency at 100K ops/s

Note: The vertical axis is capped at 10 ms, as the majority of values fall within this range.

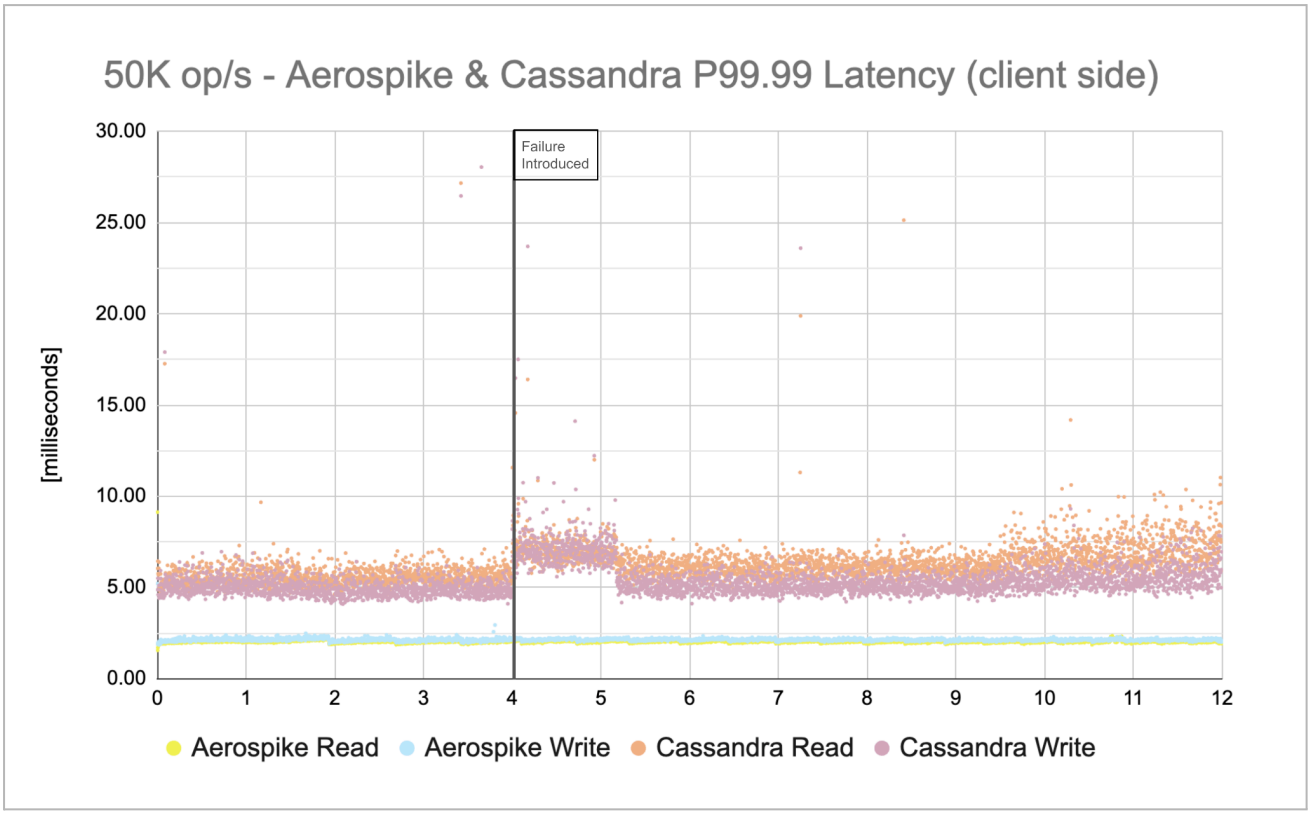


Figure 12: Aerospike and Cassandra P99.99 latency at 50K ops/s

Note: The vertical axis is capped at 30 ms, as the majority of values fall within this range.

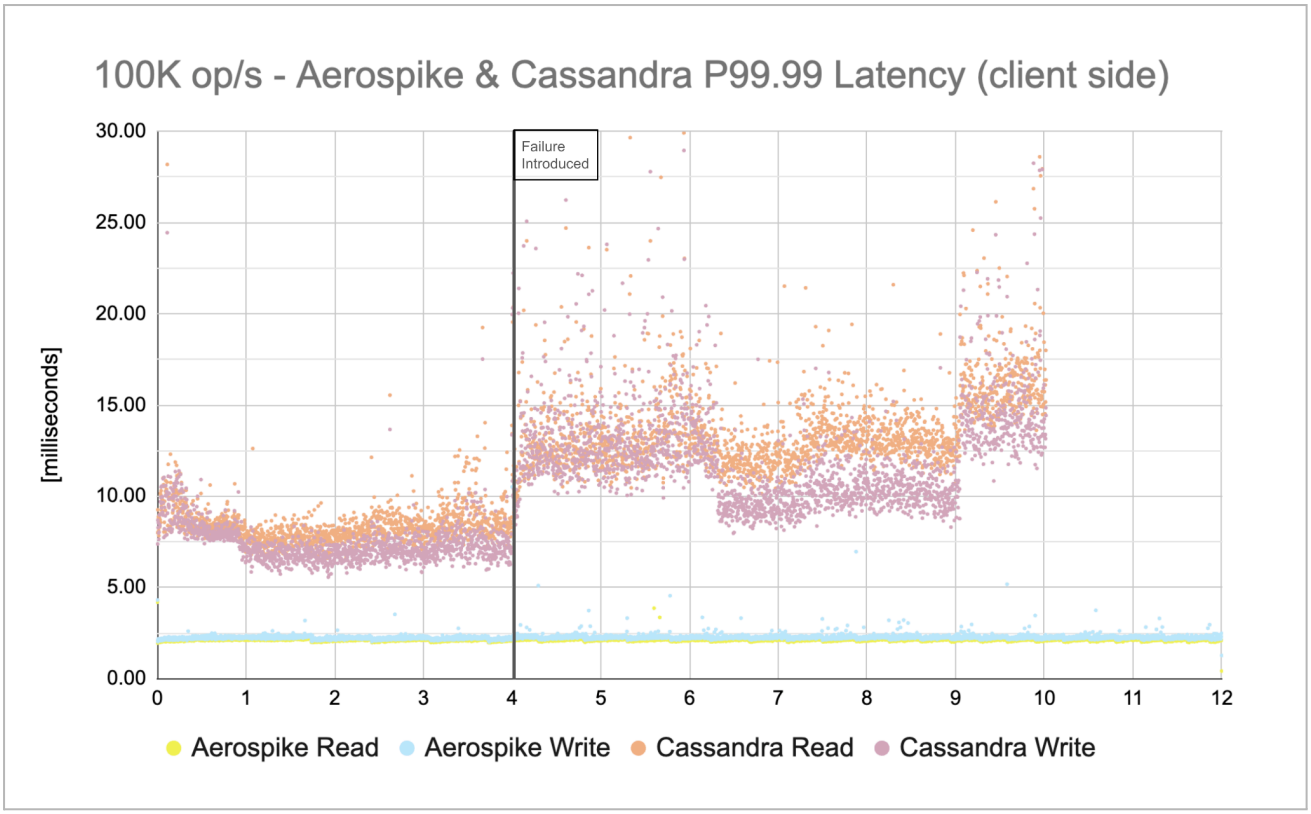


Figure 13: Aerospike and Cassandra P99.99 latency at 100K ops/s
Note: The vertical axis is capped at 30 ms, as the majority of values fall within this range.

Disclaimer

This benchmarking project, carried out by benchANT, was sponsored by Aerospike Inc. with the goal of providing a fair, transparent, and reproducible comparison of both database systems, Aerospike Enterprise and Apache Cassandra. Aerospike Inc. decided on benchmarking methodology, cluster sizes, cloud infrastructure, workload, and SLA boundaries. Further, they provided the configuration of the Aerospike cluster. benchANT updated the Yahoo Cloud Serving Benchmark (YCSB) with support for the latest versions of both databases and updated the client drivers to the respective latest version. benchANT also updated YCSB to support long-running benchmarks with billions of operations, as well as support for the workload used, which includes `delete` operations.

About benchANT

benchANT is a consulting and analytics firm specializing in comparative performance analysis of database management systems with a focus on cloud-hosted databases and database-as-a-service technologies. BenchANT provides services to database vendors, cloud providers, and end users, taking the role of an unbiased analyst, researcher, and evaluator. The experiments described in this paper have been designed, executed, and written up by two of benchANT's key employees.

Dr. Jörg Domaschka is one of benchANT's co-founders. He has been trying to understand distributed systems for more than two decades. Performance engineering and benchmarking help him with that task and allow educating others on his findings.

Dr. Daniel Seybold is a co-founder and the CTO of benchANT. Daniel started his career as a researcher with a focus on distributed systems and databases. He has extensive experience in the field of database performance testing and has been working with NoSQL databases such as MongoDB, Cassandra, and ScyllaDB for more than a decade.

Appendix

Yahoo Cloud Serving Benchmark (YCSB)

All benchmarks are based on the Yahoo Cloud Serving Benchmark (YCSB). The applied YCSB version is a fork of the initial one that updates the Aerospike and Apache Cassandra bindings to their latest version and implements the delete operation for the `site.ycsb.workloads.CoreWorkload`.

The source code is publicly available:
https://github.com/benchANT/YCSB/tree/aerospike-workload-release

Raw benchmark results

In order to ensure full transparency and reproducibility, all benchmark configurations, results, and additional metadata are made publicly available under:
https://github.com/benchANT/aerospike-apache-cassandra-benchmark

Aerospike configuration

Aerospike is basically run with its default configuration. The only noteworthy configuration that has been applied specifies the storage engine for the database as shown below:

```
namespace benchantdb {
    replication-factor 2
            storage-engine device {
                device /dev/nvme1n1p1
                device /dev/nvme1n1p2
                device /dev/nvme1n1p3
                device /dev/nvme1n1p4
                device /dev/nvme1n1p5
          }
}
```

This setting enables the Aerospike SSD storage engine and configures it to use five available partitions of the NVMe disk, which has been partitioned at VM startup.

The full configuration can be found on GitHub:
https://github.com/benchANT/aerospike-apache-cassandra-benchmark

Cassandra configuration

For Apache Cassandra, we use selected version 5 features that improve the performance over version 4 as follows:

keyspace:

```
compaction = {'class':
'org.apache.cassandra.db.compaction.UnifiedCompactionStrategy'}
```

cassandra.yml

```
sstable:
  selected_format: bti

memtable:
  configurations:
      benchant:
        class_name: TrieMemtable
```

Java

```
version: 17
garbageCollector: ZGC
```

The full configuration can be found on GitHub:
https://github.com/benchANT/aerospike-apache-cassandra-benchmark