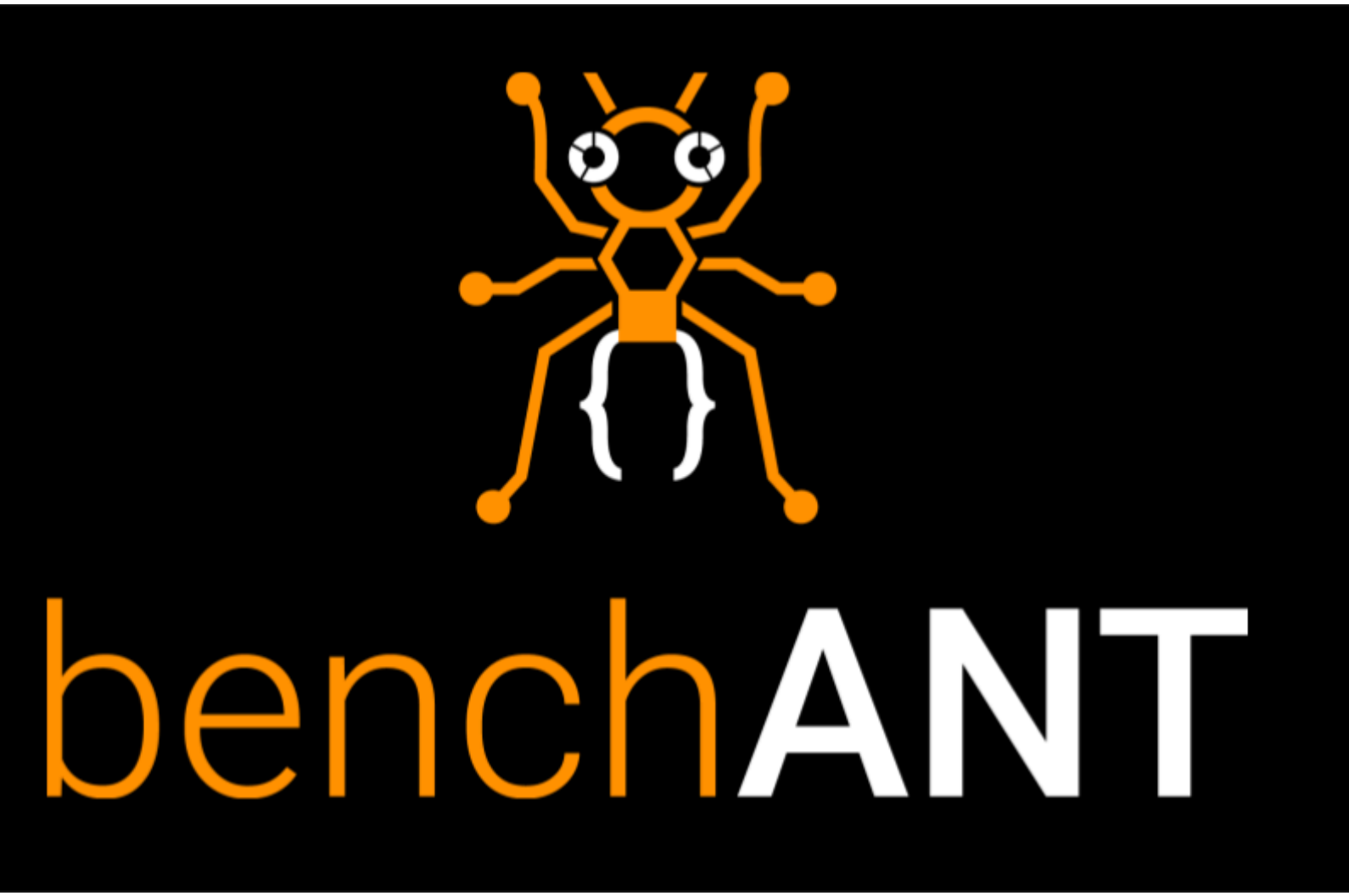

Aerospike Enterprise vs. Apache Cassandra:

Benchmark Comparison for Performance, Resilience, and Cost

Jörg Domaschka, Daniel Seybold

2025-04-23

Table of Contents

Executive Summary

This paper compares Aerospike Enterprise 8.0.4 with Apache Cassandra 5.0.3—the latest respective version of each database system—in three different scenarios. All three scenarios used an extended YCSB workload with 50% reads (by primary key), 40% inserts, 6% updates, and 4% deletes. Further, all scenarios use a 4-node Aerospike cluster with a replication factor of 2 and a 6-node Cassandra cluster with a replication factor of 3. The replication factors conform to best practices for each respective database, and the differing cluster sizes are an immediate consequence of this. All evaluations are run on AWS EC2 using Virtual Machines of type i4g.4xlarge for the database cluster and Virtual Machines of type c6i.8xlarge for the workload drivers.

Scenario 1 aims at finding the maximum throughput for this workload that can be achieved under a fixed client-side P99 latency threshold for read operations. For Aerospike, this threshold is set to 1ms, while for Cassandra it is set to 10ms following best practices for each. This leads to 200 parallel clients for Aerospike and 400 parallel clients for Cassandra. For running the workload, each database is initialized with 100GB of data; then, 4.4B operations of the above workload are executed. Hence, at the end of the run, around 3.4TB of data have been added. During this time, we monitor throughput and various types of latency metrics including average, P95, P99, and P99.99 in 10s intervals, but also for the entire benchmark runtime. Table 1 summarizes the results.

	Aerospike				Cassandra			
Latencies [µs]	**Avg**	**P95**	**P99**	**P99.99**	**Avg**	**P95**	**P99**	**P99.99**
Reads	383	650	965	4,047	3,733	6,571	8,615	15,799
Updates	593	892	1262	4,291	1,981	3,445	5,093	12,303
Deletes	387	555	756	4,023	1,943	3,389	4,979	12,167
Inserts	426	586	795	4,067	2,082	3,589	5,203	12,743
Throughput [ops/sec]	480,467				138,018			

Table 1: Latency comparison between Aerospike and Cassandra across operation type and SLA plus average throughput comparison.

Table 1 shows that Aerospike is superior not only for all four latency metrics, but also with respect to throughput. The following table shows how much higher Cassandra's latency percentiles are compared to Aerospike and how much lower their throughput is.

What % advantage does Aerospike exhibit?				
	Avg	**P95**	**P99**	**P99.99**
Reads	875%	911%	793%	290%
Updates	234%	286%	304%	187%
Deletes	402%	511%	559%	202%
Inserts	389%	512%	554%	213%
Throughput	248%			

Table 2: Throughput and Latency across operation type comparisons.
Note: higher is better for throughput; lower is better for latency (percentages take this into account).

Aerospike is much better for read workloads: Cassandra's P95 latency is 10x higher than Aerospike's. Yet, even in Aerospike's weakest discipline, update operations, requests are still processed 3x faster than with Cassandra. One may argue that these numbers are not surprising as the read threshold was 1ms for Aerospike and 10ms for Cassandra. Yet, despite this much more demanding threshold, the throughput for Aerospike is more than 3x the one for Cassandra. Putting it differently: while it is possible to get P99 latencies for insert, update, delete to 1ms and read P99 latency to less than 2ms for Cassandra, this means to sacrifice another 66% of throughput, as such latencies are reached at a throughput of around 50,000 ops/s. This is a side insight we gained from Scenario 2 and Scenario 3 (see below).

Cost-wise, the major difference between both clusters lies in the number of nodes: 4 vs 6 nodes. As we use the same Virtual Machine type for both databases (AWS' i4g.4xlarge), the cost per VM is identical ($1.236 per hour), so that overall Aerospike ends up with a 33% cheaper hardware set-up ($43,293 per year vs. $64,939 per year). Taking throughput into account, Aerospike achieves 11.1 ops/s for each $ spent per year. For Cassandra, it is 2.1 ops/s per $ spent per year.

	Aerospike	Cassandra	% Aerospike advantage
Nodes [i4g.4xlarge]	4	6	33%
Cost per hour per instance	$1.23552	$1.23552	-
Cost per year	$43,293	$64,939	33%
Throughput [ops/sec])	480,467	138,018	248%
Ops/sec per $ for one year	**11.1**	**2.1**	**422%**

Table 3: Throughput cost advantage

Scenario 2 and Scenario 3 are concerned with system behaviour while scaling out (adding one node to the database cluster) and node failure (forcibly removing one node from the cluster). In both scenarios results are less in favour of one or another competitor. For our evaluation we used a workload intensity of 25%-30% of the above workload per competitor and evaluated the behaviour of a database system when one node is forcefully removed from and added to the cluster respectively. For both node failure and scale-out evaluation the failure / start-up event was triggered when 20% of the operations from the regular workload were executed, i.e. when about 800GB of data resided in the database.

	Scale out		Resilience	
	Aerospike	Cassandra	Aerospike	Cassandra
Data re-distribution time [h]	3.5	0.9	4.9	1.75
Throughput [kops/s]	130	44	130	44
P99 Latency during change: read [ms]	0.5-1.2	1.5-1.9	0.55	1.6-2.3
P99 Latency during change: insert [ms]	0.3	0.9	0.4	1.0-1.3
P99 Latency during change: delete [ms]	0.3	0.8	0.4	1.0-1.3
P99 Latency during change: update [ms]	0.7-1.4	0.8	0.7-0.8	1.0-1.3

Table 4: Scale out and resilience performance comparison

Both databases can deal with both scenarios without causing any unavailability. Table 3 summarizes the results for both resilience and scale-out evaluation.

While scaling out Aerospike shows stable latencies of around 0.3ms for insert and delete operations while Apache Cassandra is up to 3x slower, but still sub-ms. For update operations Aerospike has some partially spiky behaviour with P99 latencies between 0.5ms and 1.2ms while Apache Cassandra has more stable P99 latencies at around 0.8ms. For read operations both systems show spiky behaviour. While Aerospike's P99 latencies are in a range between 0.5ms and 1.2ms, Apache Cassandra's are between 1.5ms and 1.9ms.

For the resilience case, the P99 latencies for Aerospike are consistently better than for Apache Cassandra. For insert operations and delete operations P99 latencies for Apache Cassandra are 2.5x-3.25x higher than for Aerospike. For read operations the difference grows to 2.9x-4.55x. In contrast, for update operations, it is only 1.25x-1.86x.

Introduction

This document presents an extensive benchmarking study performed by benchANT. It compares Aerospike Enterprise 8 and Apache Cassandra 5—the latest respective version of each database system—in three different settings each of which targets a specific objective: optimal performance under high load, performance under scale-out conditions, and performance under failure conditions.

Objective 1: Optimal Performance

This objective aims at determining the optimal performance characteristics of both database systems under test. Specifically, the goal is to identify the maximum sustainable throughput that each system can deliver while keeping the P99 of client-side latencies below a database-specific threshold.

For achieving this goal we run a set of calibration benchmarks for both database systems in which we gradually increase the workload intensity until either the system's throughput plateaus or latency starts to degrade significantly. A system is considered to operate at optimal performance when it can consistently handle a high volume of operations per second (ops/s) without entering unstable regimes such as throughput fluctuation, tail latency spikes, or increased operation failure rates. Calibration benchmarks use less data and run on a shorter time-span. Only once we find the workload performing best in calibrations, we use it to run a full-sized benchmark. During any benchmark, we measure throughput, mean latency and tail latency metrics (e.g., P99).

Objective 2: Scale-out

The second objective focuses on evaluating the scale-out capabilities of each database system, a critical aspect in scenarios required for elastic capacity handling. Scale-out operations are especially relevant in environments with rapid and continuous data growth, seasonal workload spikes, or multi-tenant architectures where infrastructure must adapt dynamically to changing demand. Common use cases include online retail during peak seasons, IoT applications with increasing sensor data, or real-time analytics platforms expanding their ingest capacity.

To assess this capability, we perform a live scale-out operation by adding an additional node to the database cluster while it is actively serving a benchmark workload. This allows us to measure the system’s ability to maintain service quality and availability during reconfiguration events.

Doing so, we can gain insights on the following characteristics:

- Throughput Impact: Monitoring how the system’s throughput evolves during the redistribution of data, including any temporary drops or instability.
- Latency Behavior: Evaluating changes in operation latency, particularly tail latencies (e.g., P95, P99), to identify potential degradation due to rebalancing or increased coordination overhead.
- Operational Readiness: Measuring the time required for the newly added node to become fully integrated and contribute to the workload. This encompasses data redistribution, index rebuilding, and balancing of traffic.

This objective aims to provide insight into the elasticity and operational robustness of each system, which is essential for real-world deployments requiring continuous availability and rapid scaling.

Objective 3: Resilience

The third objective addresses the resilience of the evaluated database systems—specifically, their ability to maintain operational integrity and to recover gracefully in the face of node-level failures. In distributed environments, such failures can arise from various causes including hardware faults, network partitions, VM crashes, or infrastructure-level maintenance events. Robust failure handling is essential for maintaining service-level agreements in production systems.

To assess resilience, we inject a controlled node failure into a running cluster during an active workload. The failure is introduced by forcibly shutting down the virtual machine hosting one of the database nodes. The node is not replaced during the evaluation, allowing us to observe how well the remaining infrastructure absorbs the impact.

The evaluation focuses on the following aspects:

- Performance Impact: Quantifying the immediate effects of node failure on throughput and latency, with particular attention to tail latencies during the failover and reconfiguration period.
- Recovery Dynamics: Analyzing how the system behavior evolves until a new equilibrium is reached.
- Recovery Time: Measuring the duration from failure injection to full cluster stabilization, defined as the point when the cluster resumes steady-state performance and completes all internal rebalancing operations.

This objective provides a clear view of each system's fault tolerance mechanisms, including data replication, request routing under node loss, and efficiency of background recovery processes. Understanding these behaviors is critical for evaluating the real-world operational risk associated with each system under non-ideal conditions.

Methodology and Approach

The methodology captures four core domains: (i) which databases were used in which version; (ii) which cloud environment is used for running the database as well as workload drivers; (iii) what cluster sizes and configurations will be used for the databases. (ii) what workload should be run at which intensity.

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 high-performance, distributed NoSQL database optimized for low-latency operations and predictable throughput at scale. It follows a hybrid memory architecture, storing indexes in RAM while persisting data on SSDs or NVMe devices, enabling sub-millisecond read and write latencies. Aerospike supports strong consistency configurations and provides mechanisms for cross-datacenter replication. The architecture is particularly suited for workloads that demand real-time analytics, high throughput, and low latency.

Apache Cassandra

Apache Cassandra is a wide-column store designed for scalability, fault tolerance, and eventual consistency. It follows a masterless, peer-to-peer architecture and employs a log-structured storage engine with tunable consistency levels. Version 5.0.3 introduces several improvements, including the new storage engine "Stargate", enhanced compaction strategies, and improved garbage collection behavior. Cassandra is commonly used in scenarios requiring high write throughput and horizontal scalability, such as messaging systems, financial trading, and recommendation engines.

Cloud Environment and Cluster Sizing

Cloud Environment

All benchmarking experiments were executed in a public cloud setting using Amazon Web Services (AWS). The following specifications were 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 deployed within a single Availability Zone and private network to minimize cross-zone latency.

Cluster Sizing

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

Aerospike

- **Number of Nodes:** 4
- **Replication Factor:** 2[2]
- **Configuration:** Custom settings aligned with Aerospike Enterprise best practices (details in appendix)
- **Monitoring:** Combined usage of the Aerospike monitoring stack and benchANT telemetry tools

Apache Cassandra

- **Number of Nodes:** 6
- **Replication Factor:** 3 (Cassandra's default)
- **Read Consistency Level:** Quorum
- **Write Consistency Level:** Quorum
- **Configuration:** Custom settings optimized for Cassandra 5.0.3 (details in appendix)
- **Monitoring:** Performed using the benchANT monitoring stack

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

Workload

The benchmarking workload is based on the Yahoo! Cloud Serving Benchmark (YCSB) framework, a widely adopted tool for evaluating the performance of NoSQL databases under various access patterns. For the purpose of this evaluation, we utilized an extended version of YCSB developed and maintained by benchANT, which introduces important enhancements for contemporary benchmarking needs. The modified codebase is publicly available under an open-source license as a branch of benchANT's YCSB repository.

benchANT's repository already contains several extensions to the standard YCSB. Particularly, it has support for long running benchmarks. To support a broader and more realistic range of operations, the following extensions were introduced in the branch:

- DELETE Operation Support: The default YCSB implementation does not include a DELETE operation. We extended the core workload to include delete semantics, enabling a more complete representation of typical CRUD (Create, Read, Update, Delete) usage patterns.
- Uniform-Growing Distribution: A novel request distribution model, `uniform-growing`, was added to overcome a limitation in the traditional uniform distribution. The new model dynamically adjusts to consider newly inserted records during the benchmark's RUN phase, ensuring a uniform access pattern across both initial and growing datasets.

When running a benchmark, the database was initially loaded with 100 GB prior to the start of the benchmark. The operational phase comprised a total of 4.4 billion operations, distributed across four different types of operations. The following table summarizes the most relevant YCSB workload configuration parameters:

YCSB Parameter	Value
`workloadClass`	site.ycsb.workloads.CoreWorkload
`operations`	4,400,000,000
`fieldCount`	10
`fieldLength`	200
`requestdistribution`	uniform-growing
`readProportion`	0.5
`updateProportion`	0.06
`insertProportion`	0.4
`scanProportion`	0.0
`deleteProportion`	0.04

Table 5: The most relevant YCSB workload configuration parameters

This workload configuration emphasizes read (50%) and insert (40%) operations, with smaller proportions of update (6%) and delete (4%) operations. Scans are explicitly disabled to maintain focus on latency-critical point operations. The uniform-growing distribution ensures even load distribution over a dynamically expanding key space, better reflecting the behavior of long-running, high-ingest real-world applications. With this workload and distribution of queries, we expect around

1.76B insert operations amongst the 4.4 billion total operations leading to around 3.5TB of data added to the databases.

It should be noted that because of the inner workings of YCSB and the way the benchmark is set-up, items will be deleted which will later not be found by read, update, and other delete operations. This is an intended behaviour. With the data ratio specified for the evaluation, one can expect that the ratio of operations which do not find their data item will be very low at the beginning to then asymptotically approach 8% during steady state operation.

The YCSB in its default version reports latencies of successful operations and of failed operations. In its default methodology, an item that was not found, because it was deleted earlier on, is summarized under failed queries. It is possible to configure YCSB to report latencies per failure type. This means that one obtains the latencies for queries that were “not found” separated from those that failed because of a different reason, e.g. network problems.

Considering our workload and methodology, we require the combined latencies of successful queries and queries that returned with “not found”. With YCSB, this is not possible out of the box. For this reason, we introduced a further metric into our YCSB extension that reports these two results combined. Queries that return with a different error are still considered differently. We did not experience any such error in this or any further benchmark series. The following table summarizes the results for both competitors.

Calibrations

To fulfill the first objective—identifying the optimal performance point for each system—a series of calibration benchmarks was conducted prior to the main performance evaluation. The calibration phase serves to determine a suitable workload intensity that maximizes throughput while maintaining stable system behavior and acceptable latency.

Each calibration benchmark adhered to the following procedure:

- A fresh deployment of the database cluster was instantiated for each individual run to eliminate any side effects from prior workloads, ensuring the comparability and reproducibility of results.
- The workload was executed against an initial dataset of 100 GB, consistent with the production-scale data volume used in the main benchmarking runs.
- Each run processed a total of 2.8 billion operations, allowing the system to transition through both warm-up and steady-state phases, and enabling observation of potential long-term performance artifacts such as memory pressure, garbage collection, or compaction behavior.

Workload intensity was varied systematically across benchmark runs by adjusting the number of concurrent clients (threads in YCSB terminology), enabling identification of performance saturation points and latency tipping points:

- Aerospike was evaluated using 200, 300, and 400 client threads, providing a focused range around expected throughput saturation levels, given the system’s typically lower latency profile and higher concurrency handling. The envisaged P99 latency threshold for Aerospike is ~1ms for READ operations.

- Apache Cassandra was evaluated using 100, 200, 300, and 400 client threads, accounting for its more variable latency behavior under load and the impact of internal mechanisms such as compaction and consistency coordination. The envisaged P99 latency threshold for Cassandra is < 10 ms for any type operations.

The insights gained from these calibration runs guided the selection of thread counts for the subsequent benchmarking experiments. The chosen workload intensities strike a balance between maximizing system utilization and maintaining performance predictability under load.

Configurations for Resilience and Scale-out Cases

While above Calibrations lead to configurations to use for the benchmarks for objective 1, other configurations are needed for the resilience case, as a smaller cluster will not be able to store as much data. This is particularly the case for Aerospike where the cluster size will decrease from 4 nodes to 3 nodes. Hence, we set the total number of operations to 75% of the maximum number used for the objective 1 benchmarks.

For the resilience benchmark, we are mostly interested in the time span between node failure and system stabilization. Therefore, we set a benchmarking time limit of 12h. If the workload has not completed until then, but system stabilization has completed, we terminate the run. Finally, we use a workload intensity that is the equivalent of 25-30% of the maximum possible workload intensity found for Objective 1.

For the sake of symmetry and better comparability of failure and scale-out case, we copy the resilience configuration for the scale-out case.

In Cassandra, a repair process should be run periodically on every node of the cluster. This step is not executed during benchmarking. It is also left out for the scale-out case and the resilience case. As the performance of nodes that undergo repair is likely to degrade, this decision in the benchmarking methodology will likely produce slightly better results for Apache Cassandra compared to a production set-up.

Benchmarking Results

The following sections discuss the results for the benchmarks performed for each of the three objectives. We do not systematically present calibration results, but refer to them whenever their results impact any configuration options, e.g. workload intensity.

Objective 1 Results: Optimal Performance

This section presents the results of the benchmarks executed to achieve Objective 1. They evaluate the sustained throughput of Aerospike and Apache Cassandra under a mixed workload. The workload consists of 5.8 billion total operations, including 50% reads, 40% inserts, 6% updates, and 4% deletes. The benchmark is executed against a preloaded dataset of 100 GB. Based on the calibration results, a total of 200 client threads were used for evaluating Aerospike and a total 400 threads were used for evaluating Cassandra.

Throughput Results

The observed results are summarized in the following table and the following charts. Please note that the difference in “Total insert operations” stems from the fact that YCSB chooses operations randomly which may lead to slight deviations in how many of the total 5.8B operations ended up being inserts. Yet, for both cases, the total inserted operations can be considered to be 40% of all operations.

	Aerospike Enterprise v8	Cassandra v5.0.3
Average Throughput [operations/sec]	480,467	138,018
Clients Used	200	400
Benchmark runtime	3h 21m 11s	11h 40m 23s
Total insert operations *(% of all 5.8B operations)*	2,319,990,108 *(39.99983%)*	2,320,025,752 *(40.00044%)*

Table 6: Evaluation Set-up and Throughput Results

Averaged over the entire benchmark runtime, Aerospike delivers around 3.5x higher throughput than Cassandra under this workload despite using only half the number of clients.

Figure 1 illustrates the throughput over the runtime of the benchmarks. As both competitors required a different amount of time, the x-axis of the plot represents the relative progress each competitor has made at the respective point in time, i.e. at 50% the benchmark for Aerospike has run for around 1h 40m while the one for Cassandra has already run for 4h 50m.

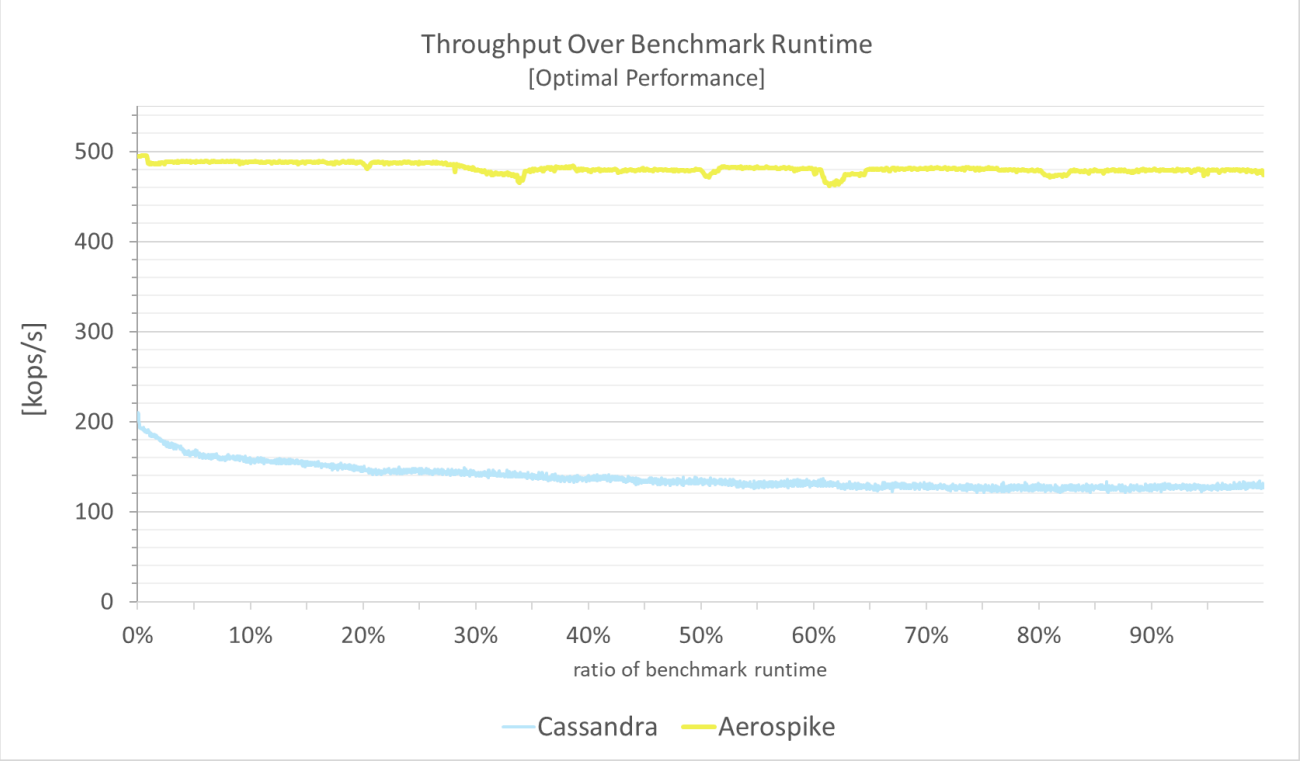

Figure 1: Throughput over benchmark runtime.
Note: for better comparability of both systems, the x-axis is normalized by benchmark runtime.

Aerospike exhibits high and consistent throughput throughout the benchmark run. The throughput starts off at around 490 kOps/s and until 35% slightly decreases to 480 kOps/s. This throughput is then kept until the end of the run. Further, the curve shows four dips two of which are minor while with the two others the throughput drops to 460 kOps/s for a couple of minutes. Overall, the system maintains a steady state even during long-duration testing and with increasing data volume.

In contrast, Cassandra shows significantly lower throughput, despite using twice the number of clients and 50% more hardware. At the beginning of the benchmark Cassandra sustains around 190 kOps/s, but not for too long. As with Aerospike a gradual degradation is observed over time. Yet, in contrast to Aerospike, this degradation does not stop at some point, but continues to the very end of the benchmark run. At the end, Cassandra achieves around 125 kOps/s.

Latency Results

This section presents the detailed P99 latency profile of both competitors under the same benchmark conditions used in the throughput evaluation: a mixed workload of 5.8 billion operations with 100 GB of initial data and 200/400 client threads. Our analysis focuses on the P99 latency metrics (99th percentile) for each operation type—insert, read, update, and delete—captured over the entire duration of the benchmark.

Latency metrics (P99) [ms]	**Aerospike**	**Cassandra**
INSERT	0.795	5.203
READ	0.965	8.615
UPDATE	1.262	5.039
DELETE	0.756	4.979

Table 7: Latency results for P99 by operation.

The P99 latencies over the benchmark runtime are captured in Figure 2 for Aerospike and in Figure 3 for Apache Cassandra. The following sections discuss both systems individually.

Aerospike Analysis

Figure 2 shows the 99th percentile of latencies per operation type over the full runtime of the Aerospike Benchmark. It shows that across all operation types, Aerospike maintains sub-millisecond to low single-digit millisecond P99 latencies, even under a sustained high-throughput workload (~480,000 ops/s). More precisely, insert and delete operations are performed fastest with latency P99 between .7 and .8 ms. This result highlights the efficiency of Aerospike's write-optimized storage architecture and the minimal overhead in processing mutations under high concurrency.

Read operations require more time and their P99 value most of the time resides between .9 and 1ms. Finally, update operations are slowest. Yet, they only experience a P99 latency between 1.2ms and 1.4ms for the majority of samples. The values for both reads and updates remain within acceptable boundaries for real-time applications. Overall, the latencies reflect the system's ability to provide predictable performance at scale, which is critical for latency-sensitive applications.

What is notable are the two peaks before 1.25h and right after 2.00h runtime. Here, all four operations experience an increase in the P99 latency for a limited amount of time of less than 10 minutes. These peaks match with the drops we saw in throughput in Figure 1. Overall, Aerospike demonstrates low and stable latency characteristics across all operation types.

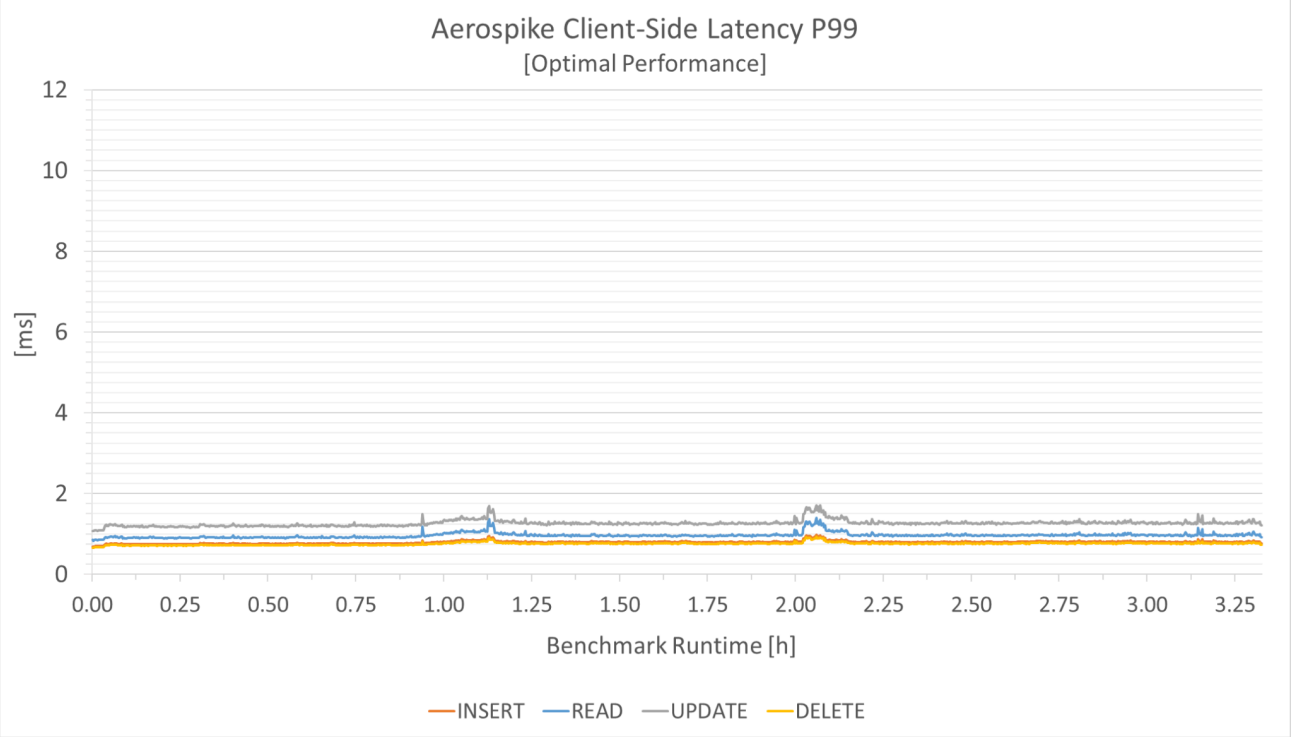

Figure 2: Aerospike Client-side Latency P99 per Operation Type over Benchmark Runtime

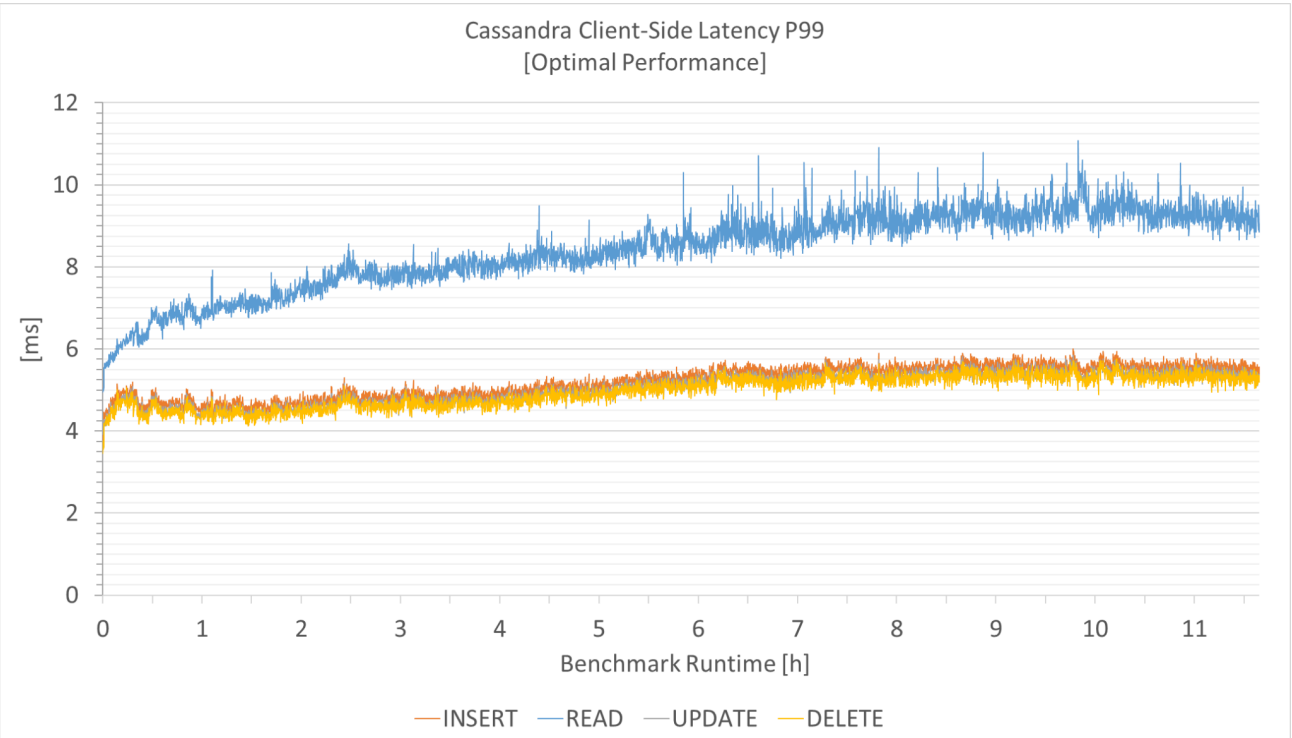

Figure 3: Cassandra Client-side Latency P99 per Operation Type over Benchmark Runtime

Cassandra Analysis

Figure 3 shows the 99th percentile of latencies per operation type over the full runtime of the Cassandra benchmark. At first glance, the plots may give the impression that the Cassandra results are more stable than the Aerospike results. Yet, this impression is caused by the different scales of the y-axis of the plots.

It shows that Cassandra exhibits a significantly higher P99 latency for read operations than for the three other operations. This behavior is consistent with Cassandra's architecture, where reads often require coordination across multiple replicas and can be affected by compaction processes, SSTable lookup amplification, and disk I/O contention—especially under large data volumes.

What is more, the P99 latencies for all four types of operations increase during the runtime of the benchmark. While P99 latencies for read operations start below 6ms and increase to around 8ms after three hours, the curve reaches a maximum around 9 ms with frequent peaks to and even beyond 10ms. The P99 latencies for all other three types of operations start around 4.5ms and slowly increase to 5.5ms with some small peaks reaching close to 6ms. Overall the write operations are more stable and less spiky than read.

Summary

A direct comparison of P99 latency metrics under identical workload conditions reveals substantial differences in the operational latency behavior of Aerospike and Apache Cassandra.

Across all operation types, Aerospike demonstrates significantly lower P99 latency, maintaining sub-millisecond performance for inserts, reads, and deletes, and remaining below 1.3 ms for updates. In contrast, Cassandra shows elevated latency across the board, with P99 values ranging from approximately 5 ms to over 8 ms.

The most striking contrast is observed for read operations, where Cassandra's P99 latency is nearly 9× higher than Aerospike's. Even for inserts—typically Cassandra's strongest path—latency exceeds 5 ms, compared to 0.8 ms for Aerospike.

These results underline Aerospike's architectural efficiency in delivering low and predictable tail latency under high-throughput, mixed-operation workloads. Cassandra, while scalable, despite running with 50% more resources demonstrates greater latency variability and coordination overhead, which can be critical for latency-sensitive use cases.

Objective 2: Scale-out

To assess the scale-out capabilities of Aerospike and Apache Cassandra, a live cluster expansion was performed during an active workload execution. One additional node was added to each system mid-benchmark, triggering data redistribution while maintaining full workload availability. The objective was to observe how each system absorbs new capacity and rebalances data without service degradation.

For the throughput during this evaluation we targeted a load between 25% and 30% of the maximum throughput achieved in the Optimal Performance evaluation. A further calibration step suggested that 32 clients (threads) are sufficient for both databases to achieve this load level.

	Aerospike	Cassandra
Clients	32	32
Targeted average throughput [operations/second]	~120,000	~34,500
Failure Trigger Time [seconds]	9,000 (~2.5h)	25,800 (7h 10m)
Cluster Expansion	+ 1 node	+ 1 node

Table 8: Scale-out test parameters

The methodology of the evaluation is different to the one used for the Optimal Performance evaluation. Instead of targeting a maximum of 2.3B insert operations (and 5.8B operations in total), we limit the maximum amount of insert operations to 1.76B and the total number of operations to 4.4B, around ~75% of the data from the Optimal Performance evaluation. This was done in order to cater for the resilience case below which ends up with one node less, 75% less in the case of Cassandra. Yet, to avoid very long running benchmarks, we also add a timeout after 12 hours. As we expect that the new node will have been added at that time, a longer runtime only provides very little new insight.

Scale-out Throughput

Prior to node addition, Aerospike maintains consistent throughput at around 127 kOps/s. When the scale-out is triggered at 9,000 seconds, a modest dip below 125 kOps/s occurs. During the entire 3.5h long data redistribution period the throughput slightly further decreases down to 122 kOps/s. Some larger fluctuations are visible towards the end of the rebalancing phase where throughput punctually drops to 115 kOps/s. Once the data data redistribution has completed, the throughput reaches its 125 kOps/s again which is almost its old levels. The system continues processing data and completes the benchmark within the time threshold.

The throughput pattern of Cassandra before the scale-out event is very similar to the one observed in the Optimal Performance evaluation: the throughput slightly decreases over time due to more data being added to the database and due to holes being created in the data set by the delete operations. At the time the scale-out is triggered, Cassandra exposes a throughput of slightly more than 42 kOps/s. Once the data redistribution is running, there is a very minor drop by around 1 kOps/s (not visible in the plot). During data redistribution, the trend of decreasing throughput continues. Once data distribution has been completed after 52 minutes, the throughput increases to almost 45 kOps — higher than before the scale-out was triggered. Because of the very low throughput, Cassandra does not manage to perform all operations. Instead, the benchmark times out after a runtime of 12h.

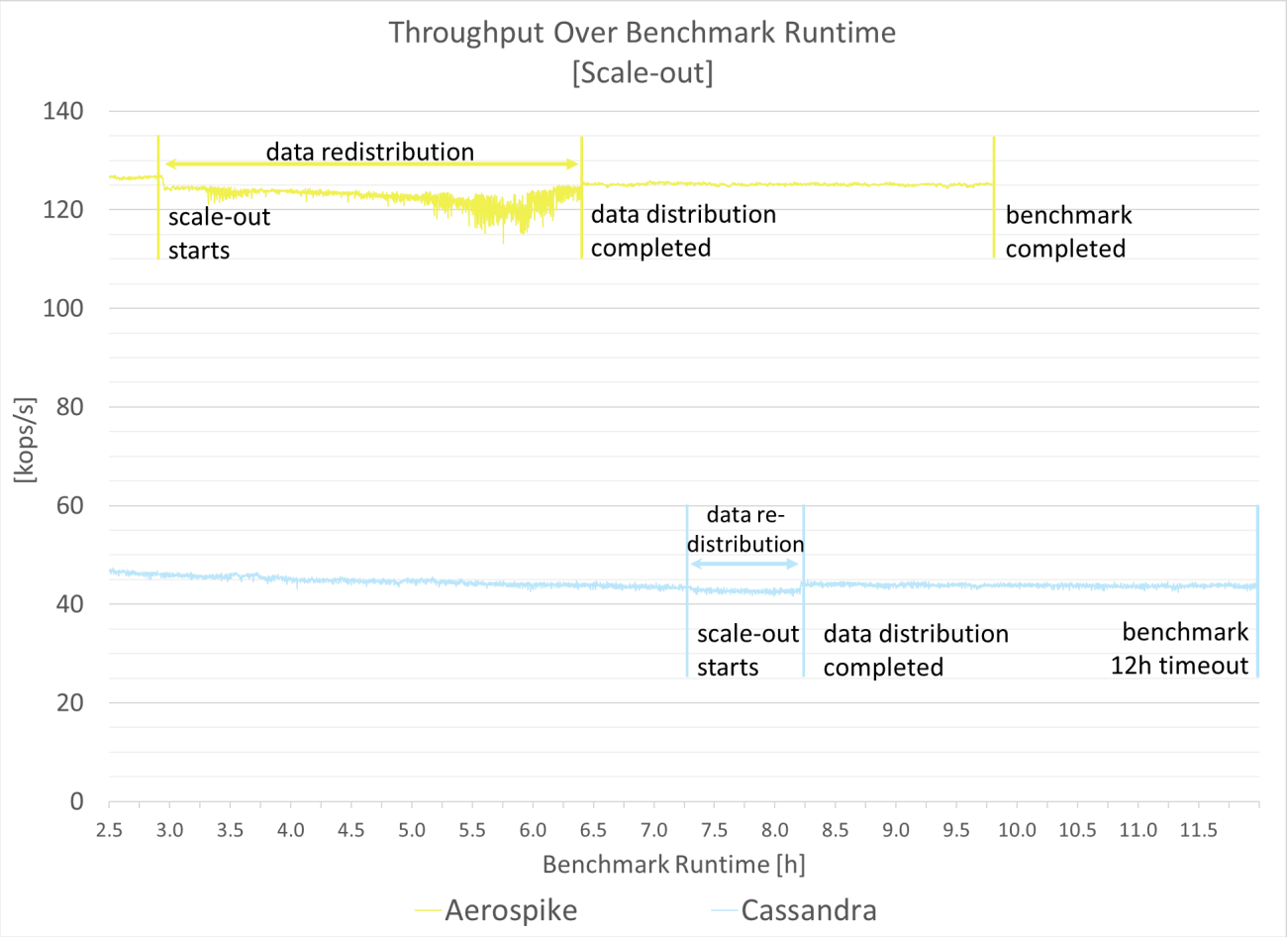

Figure 4: Throughput during scale-out

Scale-out Latency Results

This section presents the P99 latency profile of both competitors under the same benchmark conditions used in the throughput evaluation: a mixed workload of 4.4 billion operations with 100 GB of initial data and 32 clients putting stress on the system. After around 1.16 billion operations a scale-out is triggered, i.e. a new database node is added to the cluster.

As before, our analysis focuses on the P99 latency metrics (99th percentile) for each operation type—insert, read, update, and delete. Yet, our discussion only considers the time shortly before the new node is added as well as the entire data redistribution phase. The individual results for Aerspike and Apache Cassandra over the entire benchmark runtime are shown in Figure 5 and Figure 6 respectively.

Aerospike Analysis

Figure 5 shows the 99th percentile of latencies per operation type over the relevant part of the Aerospike benchmark. Prior to scale-out, Aerospike maintains consistently low P99 latencies, with all operations typically below .7ms (update), .5ms (read), and slightly above 0.3ms (update and delete). At the moment of scale-out, an increase in latencies is observed, consistent across reads, updates, and deletes.

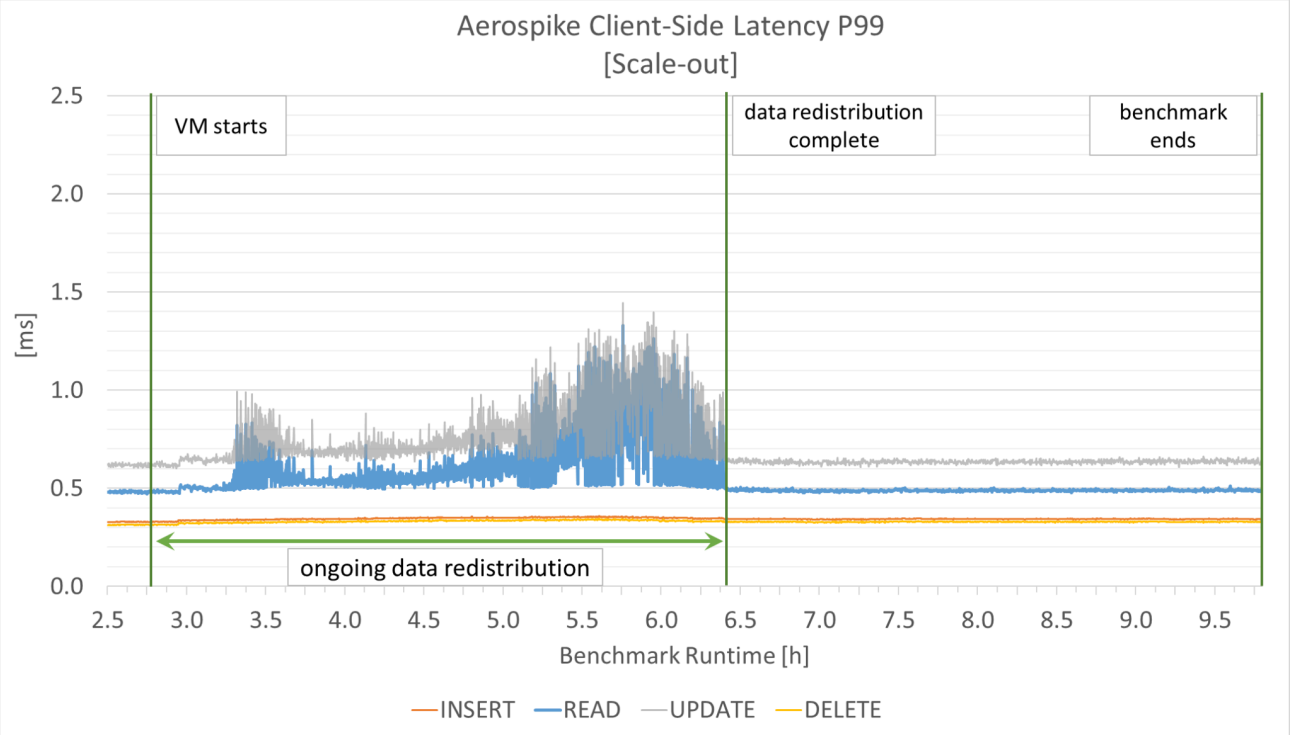

Figure 5: Per operation type P99 latency for Aerospike at a workload with 32 clients during a scale-out case

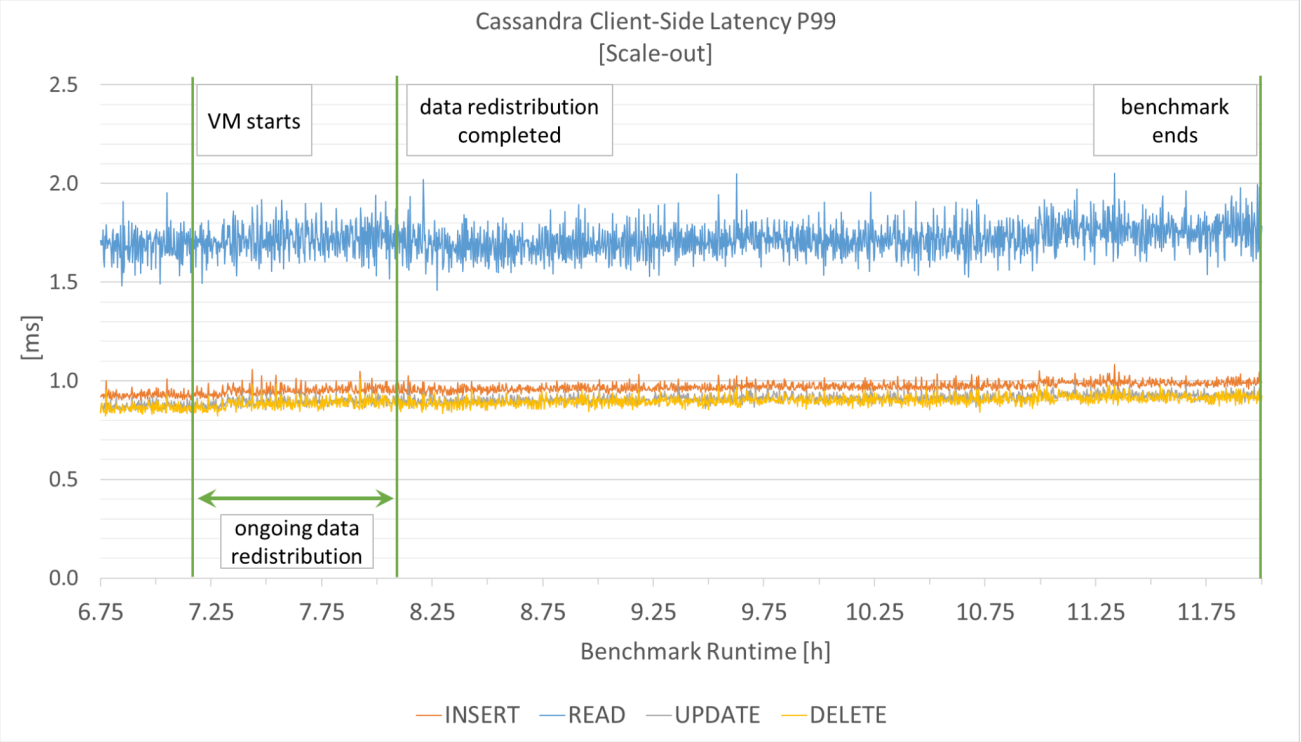

Figure 6: Per operation P99 latency for Apache Cassandra at a workload with 32 clients during a scale-out case

As shown in Figure 5 the increase is only minor for delete and insert operations. In contrast it is more turbulent for reads and updates. Yet, the magnitude of latency fluctuation remains low: the P99s remain below 1 ms for both operations almost all the time. While the P99 latency for read crosses the 1ms boundary a couple of times, the P99 latency for update operations occasionally reaches up to 1.4ms.

Once the scale-out has completed, latencies return to pre-scale-out levels within a short window. The system continues to handle the incoming workload with minimal tail latencies.

Cassandra Analysis

Figure 6 shows the 99th percentile of latencies per operation type over the relevant part of the Cassandra benchmark. Prior to scale out, Cassandra is able to achieve P99 latencies around 1ms for all operations except reads. For read operations, we measure an increasing latency starting from 1.5ms and going up to ~1.8ms until the scale out is triggered at 25,800 seconds.

Upon node addition, there is barely any increase in P99 latency at all. Also while data distribution is running, barely any changes are visible. Data distribution only takes ~52 minutes.

Objective 3 Results: Resilience

This case is the inversion to the scale-out case. To achieve best comparability between scale-out and resilience case, both cases use the same configuration and same trigger times. In this resilience case, a node failure was intentionally injected into each system during an ongoing production workload. The virtual machine hosting a database node was forcibly shut down, simulating a realistic infrastructure fault. The failed node is not replaced, allowing observation of how well each system absorbs the impact, rebalances data, and returns to operational stability.

During the benchmark all operations were successfully completed. None of the clients experienced any erroneous request.

	Aerospike	**Cassandra**
Clients	32	32
Failure Trigger Time [sec]	9,000 (2.5h)	25,800 (7h 10m)
Failure Type	VM Shutdown (1 node)	VM Shutdown (1 node)
Node Replacement	No	No

Table 9: Resilience test parameters

Resilience Throughput

Prior to node removal, Aerospike maintains consistent throughput of slightly more than 125 kOps/s. When the scale-out is triggered after 9,000 seconds in the benchmark, a dip in throughput occurs that lasts during the entire data redistribution period (5h) and beyond. Yet, despite the throughput drop of around 15%, the throughput remains stable during the entire data redistribution phase. Once the data data redistribution has completed, the throughput increases slightly, but never reaches its old level again. This is expected as 25% of the previous processing capabilities are missing. The benchmark completes before the 12h timeout.

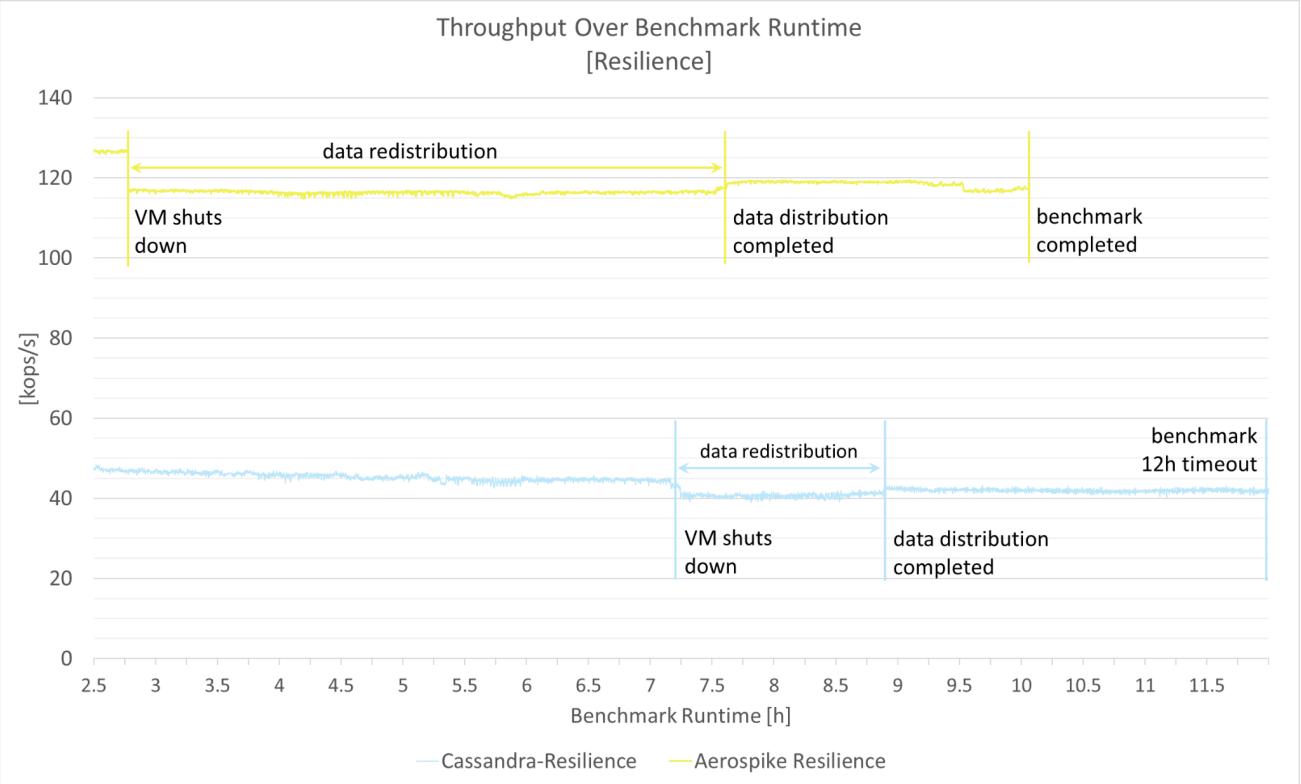

Figure 7: Throughput over benchmark runtime for resilience evaluation.

For Cassandra prior to node removal, the same behaviour is visible as before, a slight decrease in throughput over time. At node removal time, there is a dip from ~45 kOps/s to ~40 kOps/s. Data redistribution is triggered and completed after around 1h 40m. During that period, Cassandra exposes stable throughput that is slightly increasing towards the end of the data redistribution period. After data redistribution has completed, there is a small, further increase of throughput to almost 42 kOps/s. For the remainder of the benchmark, this throughput remains mostly stable. Because of Cassandra's slower processing speed, the benchmark times out after 12h. It would have required 17+h to run to completion.

Summarizing, with respect to throughput both Cassandra and Aerospike behave stable under error conditions despite the demanding workload that was used. Both systems even expose a similar behaviour. Cassandra is able to return to a redundant state quicker than Aerospike, but Aerospike is able to maintain a much higher workload for the entire time. The additional rebalancing time required by Aerospike can be explained to some extent by the fact that the Aerospike cluster lost 25% of its capacity, while the Cassandra cluster only lost 16%.

Resilience Latency Results

This section presents the P99 latency profile of both competitors under the same benchmark conditions used in the scale-out evaluation. As before, our analysis focuses on the P99 latency metrics (99th percentile) for each operation type—insert, read, update, and delete. As before, our discussion starts right before the node failure is triggered. The individual results for Aerospike and Apache Cassandra over the entire benchmark runtime are shown in Figure 8 and Figure 9 respectively.

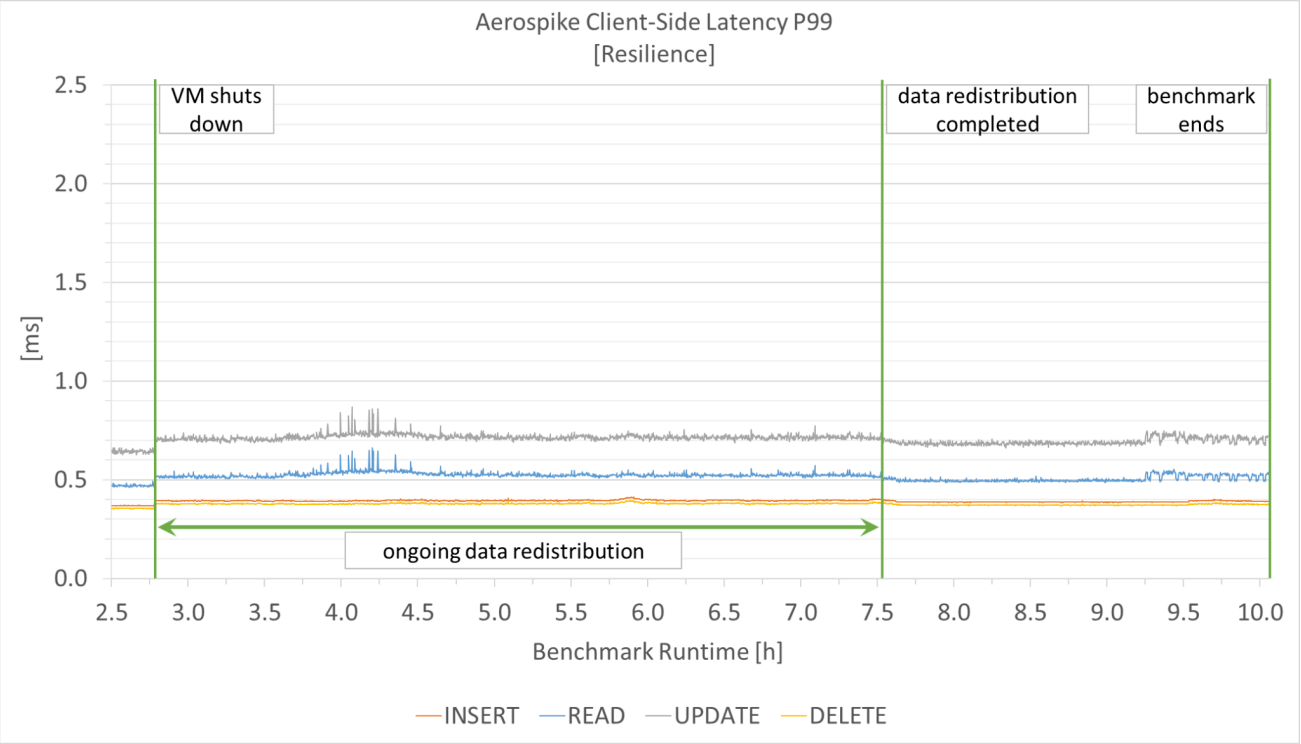

Figure 8: Per operation P99 latency for Aerospike at a workload with 32 clients during a node failure case

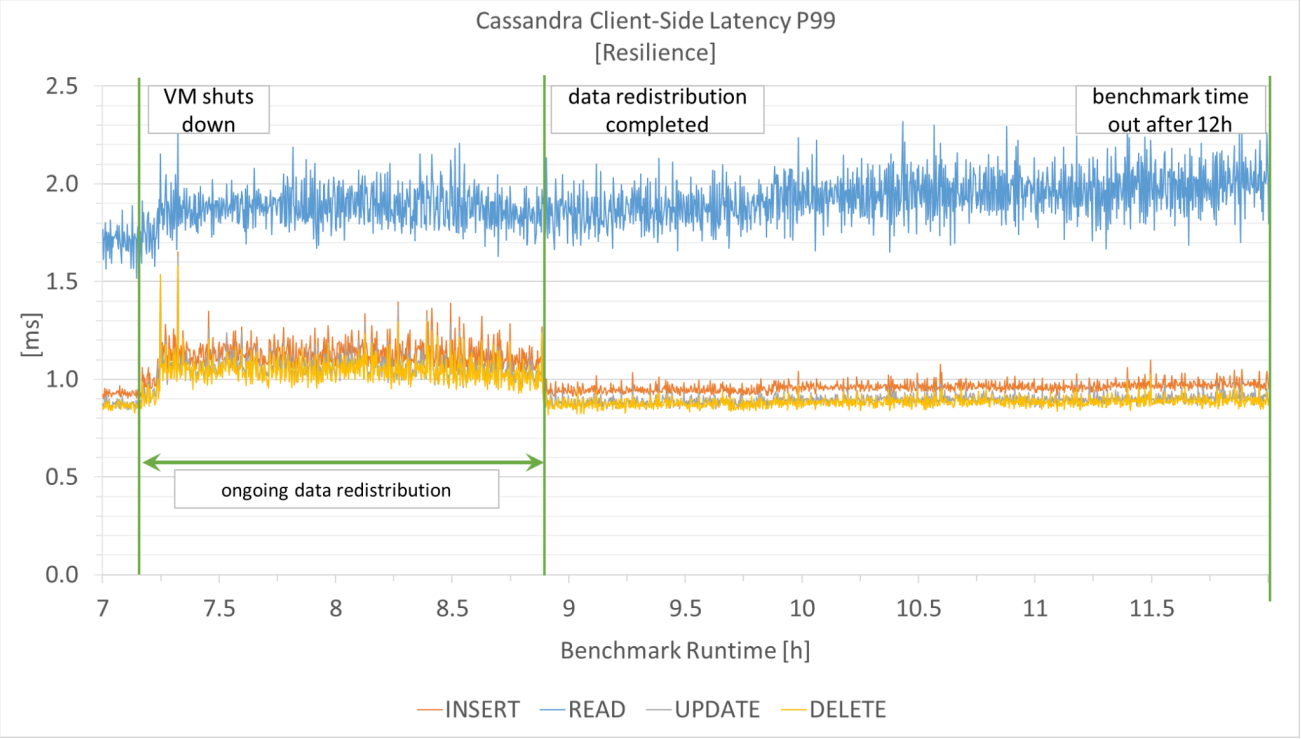

Figure 9: Per operation P99 latency for Apache Cassandra at a workload with 32 clients during a node failure case

Aerospike Analysis

Figure 8 shows the 99th percentile of latencies per operation type over the relevant part of the Aerospike benchmark. The virtual machine hosting one Aerospike node was shut down at 9,000 seconds into the benchmark. At the time of failure, a moderate increase of ~0.05ms in P99 latencies is observed across all operation types. During the data redistribution phase, this level of latencies is mostly kept while occasional spikes for read operations reach up to 0.65ms and spikes for updates reach up to almost to .9ms.

After data redistribution has completed, the P99 of latencies for all operation types decrease slightly. Yet, they do not fully go back to the same level as before the node was shut down.

Cassandra Analysis

Figure 9 shows the 99th percentile of latencies per operation type over the relevant part of the Cassandra benchmark. The node failure was triggered at approximately 25,800 seconds (7.1h) into the benchmark. Before that failure, Cassandra P99 latencies exposed the expected characteristics: stable sub-ms values for inserts, updates, and deletes and slowly increasing values for reads. The latencies for reads are also spiky. Yet, barely any of the spikes exceed 2.0ms, most of them remain below 1.8ms.

At the time of failure deletes, inserts, and updates experience a raise in P99 latency of around .2ms with few spikes reaching beyond 1.5ms. Read operations experience around the same raise. Yet in their case it is less noticeable due to the spikiness of the pattern. This increased level of latencies is kept throughout the entire data redistribution phase. During that time, deletes, inserts, and updates also expose a more spiky pattern than before.

After data redistribution has completed, this spikiness ceases and the P99 latencies drop by ~0.3ms. While this drop is consistent and stable for inserts, deletes, and updates, the impact for read operations vanishes as read P99 latencies reach 2.0ms and beyond. Because of Cassandra's slower processing speed, the benchmark times out after 12h. At this point in time 33% of the targeted data have been ingested.

Summary

This paper compared Aerospike Enterprise 8.0.4 with Apache Cassandra 5.0.3—the latest respective version of each database system—in three different scenarios. All three scenarios used an extended YCSB workload with 50% reads (by primary key), 40% inserts, 6% updates, and 4% deletes. Further, all scenarios used a 4-node Aerospike cluster with a replication factor of 2 and a 6-node Cassandra cluster with a replication factor of 3.

Scenario 1 aims at finding the maximum throughput for above workload that can be achieved under a fixed client-side P99 latency threshold for read operations. For Aerospike, this threshold is set to 1ms, while for Cassandra it is set to 10ms. For this scenario Aerospike was able to reach a throughput of more than 480 kops/s while Apache Cassandra only reached 138 kops/s despite Aerospike using less hardware resources. The P99 latencies for both of the database systems heavily depends on the operation types. For Aerospike delete and insert operations perform best with a latency P99 of just below 0.8ms; read operations perform second best with just below 1ms; update operations are around 1.3ms. For all operations (and all percentiles we considered) Aerospike is multiple times better than Apache Cassandra. Regarding the P99 latencies, Apache Cassandra performs best for deletes and updates (both around 5ms); with 5.2ms inserts are second best; reads are worst with around 6.6ms.
Scenario 2 and Scenario 3 are concerned with system behaviour while scaling out (adding one node to the database cluster) and node failure (forcibly removing one node from the cluster). Yet, under a 70% reduced load compared to Scenario 1. In both scenarios results are less in favour of one or another competitor. Both databases can deal with both scenarios without causing any unavailability.

Cost-wise, the major difference between both clusters lies in the number of nodes: 4 vs 6. As we use the same Virtual Machine type for both databases (AWS' i4g.4xlarge), the cost per VM is identical ($1.236 per hour), so that overall Aerospike ends up with a 33% cheaper hardware set-up ($43,293 per year vs. $64,939 per year). Taking throughput into account, Aerospike achieves 11 ops/s for each $ spent per year. For Cassandra, it is merely 2 ops / s per $ spent per year.

Yet, it needs to be considered that Aerospike Enterprise 8 that was used for the evaluation requires acquisition of a software license. This license, in turn, includes 24/7 technical support which is not the case for the OpenSource Cassandra version. Here, additional support needs to be acquired if needed.

Disclaimer

This benchmarking project carried out by benchANT was sponsored by Aerospike Inc. with the goal to provide 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 is 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