Aerospike

WHITE PAPER BENCHMARK

A document use case comparison: Aerospike vs. MongoDB Atlas

Written by Phil Allsopp
Principal Performance and Reliability Engineer
Aerospike

Contents

Executive summary

A benchmark comparison was executed, employing 10 terabytes of JSON data on Google Cloud, between Aerospike 7.0, a document store, and MongoDB Atlas v7.08, a native document database. Comparisons were made in several areas, including latency, throughput, and resilience for a mixed read/write/update workload, along with operational considerations.

Since no standard benchmark dataset exists for document data, this benchmark employs document data from a real-world gaming simulation (protecting players from exceeding limits as they play) in JSON format.

Each vendor's provided database driver was used with the driver's default settings.

The results of our tests show:

- **Throughput-to-VM ratio:** Aerospike exhibits nine times more throughput than MongoDB, while MongoDB needs more than three times the VMs to run the same number of horizontal partitions or shards[1] as Aerospike.
- **Predictable performance**
  - MongoDB latencies triple, and throughput drops by 38.9% while the database size grows from 0-to-10TB, whereas Aerospike latencies hold steady, while throughput drops only 5% (See Table 1).
  - Aerospike write latencies for both p95 and p99 are one millisecond, compared to 15 and 50 for MongoDB, respectively (See Table 1).
- **Node failure handling:** MongoDB performance drops 80% for 40 seconds during its server failure (see Figure 16). Aerospike performance drops 15.4% for a similar time period (See Figures 15.1 and 15.2). In addition, MongoDB loses connectivity when the same failure scenarios are tested and only allows for failovers across a single replica set.
- **Operations while scaling:** While database size elasticity is relatively easy to manage with both systems, there are key differences operationally:
  - Aerospike can be resized up or down at any time with no loss of connectivity.
  - MongoDB Atlas loses connectivity when resizing.
  - Scaling from a MongoDB Atlas replica set to a sharded cluster is one-way; you cannot go back, and thus, you are prone to overprovisioning.
- **Total cost of ownership (TCO):** Aerospike has up to 5.4 times better TCO, depending on the ultimate Aerospike licensing costs factored in for the same workload and the performance and resilience advantages cited herein (see Table 4).

[1] As Aerospike partitions data automatically and distributes evenly across nodes, for the purpose of this paper, one shard in Aerospike is equivalent to an instance or server. To learn more on Aerospike horizontal scaling, see: https://aerospike.com/compare/mongodb-vs-aerospike/#scalability_options.

	MongoDB Atlas	Aerospike	Aerospike advantage
Throughput, mean (OPS)	35,968	321,000	9x
Throughput of decay as % of mean (max - last) / mean	38.9% (46.1k - 32.2k) / 35.8k	5% (332k - 316k) / 321k	7.8x
P95 latency over 10TB ingestion period	5ms – 15ms	1ms	5x to 15x
P99 latency over 10TB ingestion period	12ms – 50ms	1ms	12x to 50x
VMs required	18 VMs	5 VMs	3.6x
Servers needed per shard added[1]	3 servers	1 server	3x

Table 1: Aerospike vs. MongoDB Atlas results summary from 0 – 10TB as data is generated.

Considerations

This benchmark report tests a number of factors that impact a customer's real-world use of the database. We asked the following questions, which we materialized through a set of tests and statistical analyses:

1. **How many write and read transactions per second ('TPS') can this database handle?**
   When compared to another vendor's database. This is a straight throughput comparison for an application that models a real-world use case.
2. **How quickly does my database respond ('latency')?**
3. **Can my business continue to run smoothly when my database encounters a significant problem?**
   This is always materially important to a customer (albeit a rare thing to consider when benchmarking).
4. **What happens when I need to temporarily bring a node out of service to patch the O/S?**
   How well does the database perform when running an external maintenance task?
5. **How well does the database handle re-configuration while under load?**
   If you elect to add another node as your workload has increased, how well does this work? Conversely, if you just had a busy holiday season and now want to scale back down, how well does that work, and will this impact the business while the scale-down progresses?

Testing approach

There are numerous ways to run tests against databases, such as stress tests, soak tests, load tests, etc.

There are many different benchmark tests that can be run, such as asbench TPC-C, YCSB, and many more—the list is long. It is important to remember that different workloads will materialize different outcomes in comparative testing. This applies equally to standard benchmark testing and application-specific workload testing.

There are singular benefits you can glean from these synthetic benchmarks as they allow direct comparisons of a particular test to be done. Some of these tests try to simulate a 'typical application' with a 'typical database access pattern.' In contrast, other tests allow you to simulate specific scenarios, e.g., "What happens if we use ten threads to run 100 database operations versus 100 threads to run ten operations each?" and so on. This may help inform you what works well with a particular database system so that you can develop or tune your application code appropriately.

However, companies want to test against their own real-world applications to determine the suitability of a specific database.

With this in mind, we examined the performance of a preexisting application that was well-suited to any NoSQL database to provide an objectively comparative test with conclusions that provide insights beyond the specificity of benchmarking.

To ensure that each test started from a clean slate, Ansible scripts instantiated a new cluster for each test using either the MongoDB Atlas API or Aerospike's AeroLab tool.

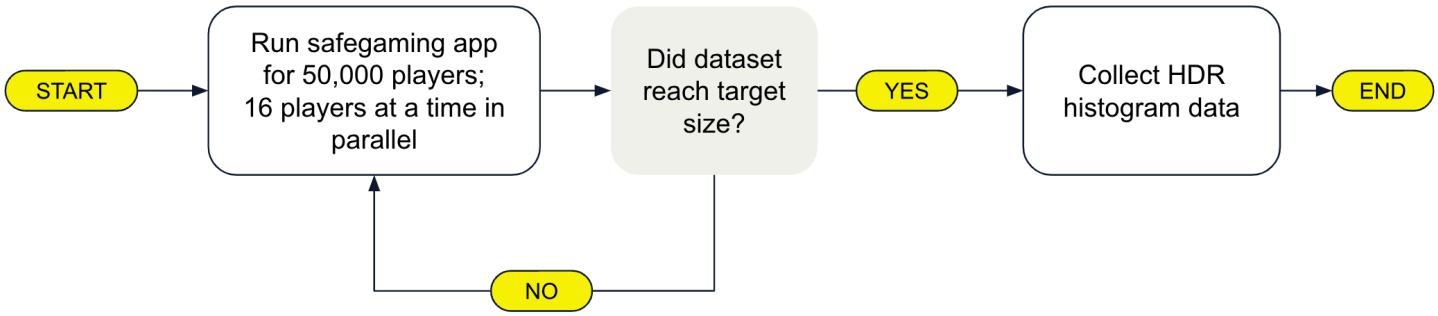


Figure 1: Load generator system run rules.

Figure 2: Main evaluation runs.

Aerospike leveraged two applications:

1. Player Protection Game Simulator, as it was easily possible to customize this application's access to the database (while maintaining consistency of application logic) to get the best out of both Aerospike and MongoDB from a complex real-world application.
2. Player Protection Reader application simulates a consistent amount of reads per second and adjusts in real-time as necessary to achieve the desired number of read transactions per second.

The Game Simulator application is write-heavy, consisting of a combination of inserts (two-thirds) and updates (one-third). It also incorporates some atomic operations using the Aerospike Operate[2] and MongoDB Transaction[3] handling methods. The Game Simulator generates no reads

The Reader application was used to generate a fixed 2000 transactions worth of user data per second against the databases. (See Appendix B for more details.)

Between the two applications, the resultant Read/Write ratio was targeted to approximately 20/80 R/W.

This Game Simulator application was developed as a showcase application with one of our partners. It is meant to demonstrate a real-world application of the Aerospike client and the database access patterns and data documents that suit a NoSQL JSON-style document database.

We stressed each database using multiple Load Generation Servers (LGSs) running the Player Protection Game Simulator and the Player Protection Reader in parallel.

We also evaluated the outcome of a partial database failure.

Infrastructure

Shards (Horizontal partitions)

The first question is, "Why did we choose five shards for both MongoDB and Aerospike?" to run a 10 TB test. We made our decision because we wanted to be fair to and work well with MongoDB. Aerospike works well with 3, 5, 8, or however many shards you may want to use.

We looked around for solid advice from MongoDB staff members' blogs and their technical posts for guidance. One such post[4] cited that MongoDB Atlas allows up to 4TB per shard, and their guidance indicated that 1TB was a bit easier to manage but also notes that, "Well, we could go higher here."

Given this and many other similar MongoDB blogs, we realized that 10 1TB MongoDB shards would need 30 VMs just for the basic replica sets. We felt this was a little over the top for a fairly small 10TB test and chose the midpoint between 0 and the Atlas Maximum of 4TB, thus 2TB, which meant five shards.

We then used five Aerospike shards to match the five MongoDB Atlas shards.

[2] https://aerospike.com/docs/server/guide/transactions
[3] https://www.mongodb.com/docs/manual/core/write-operations-atomicity/
[4] https://www.linkedin.com/pulse/mongodb-sizing-guide-sepp-renfer

Test clusters

MongoDB Atlas is a DBaaS, and the test cluster was set up and configured using the Atlas API.

Aerospike was set up using IaaS using the publicly available AeroLab tool with a general purpose production configuration.

Aerospike typically uses NVMe drives as part of its standard deployment. However, given that MongoDB Atlas is unable to use NVMe drives on the GCP platform, this would have been an unfair advantage to Aerospike in the performance evaluations.

Therefore, Aerospike was set up as a self-managed install on GCP and MongoDB Atlas instances running on GCP were selected, with neither using NVMe devices.

Thus, SSD persistent disks for both Aerospike and MongoDB storage were used.

(Note: Aerospike's standard practice is to use locally attached NVMe which is significantly quicker than the network-attached SSD used in these tests.)

Aerospike was set up on GCP n2 instance types that used Ice Lake CPUs. It did not use the significantly faster CPU options that were available (e.g., C3 or C3D Instances, Intel Sapphire Rapids, AMD Genoa, etc.). GCP describes the selected Aerospike instance types as "balanced price/performance."

Data consistency and guarantees

The selected consistency and guarantees were left at the default setting for each database. This should offer a good balance of durability vs performance for each system.

The explanatory text shown below describes the default settings and is duplicated from the respective vendors' websites.

Aerospike[5]

By default, Aerospike applies writes immediately to replicas. When the network is operating correctly, this will result in data consistency by involving all replicas of a record during each transaction, and not creating situations where there are stale or dirty reads. In the case where network partitions are occurring, Aerospike will prioritize availability over consistency --- Aerospike will allow reads and writes in every sub-cluster.

The default Aerospike server-client behavior provides these behaviors:

- Write transactions (including deletes and user defined function (UDF) applications) will write locally, then write all replicas synchronously before successfully returning from the transaction.
- Read transactions only consult a single replica (usually the master) even during cluster reconfiguration.

MongoDB[6]

The default write concern is { w: "majority" }

Write concern for replica sets describe the number of data-bearing members (i.e. the primary and secondaries, but not arbiters) that must acknowledge a write operation before the operation returns as successful. A member can only acknowledge a write operation after it has received and applied the write successfully.

[5] https://docs.aerospike.com/server/architecture/consistency#high-availability-mode
[6] https://www.mongodb.com/docs/manual/core/replica-set-write-concern/

For replica sets, a write concern of w: "majority" requires acknowledgment that the write operations have been durably committed to a calculated majority of the data-bearing voting members. For most replica set configurations, w: "majority" is the default write concern.

Performance results

The Player Protection Game Simulator was run on multiple Load Generation Servers (LGSs).

The Aerospike cluster consisted of five nodes (VMs), and the MongoDB cluster consisted of an R200 five-shard cluster, which equates to a total of 18 VMs. Each Aerospike and MongoDB VM had 32 vCPUs and 256GB of memory. Each evaluation run started with a newly created set of database nodes, fresh disks, and so on. (Please see Appendix A for a full description of hardware.)

The Player Protection Game Simulator works on a small subset of the dataset at any one time.

This means that a small portion of the database is very hot, a larger part of the database is warm and the largest proportion of the database is reasonably cold at any point.

This manifests itself as a fairly consistent TPS throughput over time which explains the mean performance being very close to peak performance on the following throughput summary graph. This should also clarify why the peak and mean TPS are very similar at different dataset sizes on the same database engine.

The results show Aerospike to be the clear throughput winner:

	MongoDB Atlas	Aerospike	Aerospike advantage
Throughput, mean (OPS)	35,968	321,000	8.9x
Throughput of decay as % of mean (max - last) / mean	38.9% (46.1k - 32.2k) / 35.8k	5% (332k - 316k) / 321k	7.8x
P95 latency over 10TB ingestion period	5ms – 15ms	1ms	5x to 15x
P99 latency over 10TB ingestion period	12ms – 50ms	1ms	12x to 50x
VMs required	18 VMs	5 VMs	3.6x
Servers needed per shard added	3 servers	1 server	3x

Table 2: Performance, VMs, and Servers summary for five shards of Aerospike versus MongoDB Atlas on a R200 cluster.

Throughput graphs

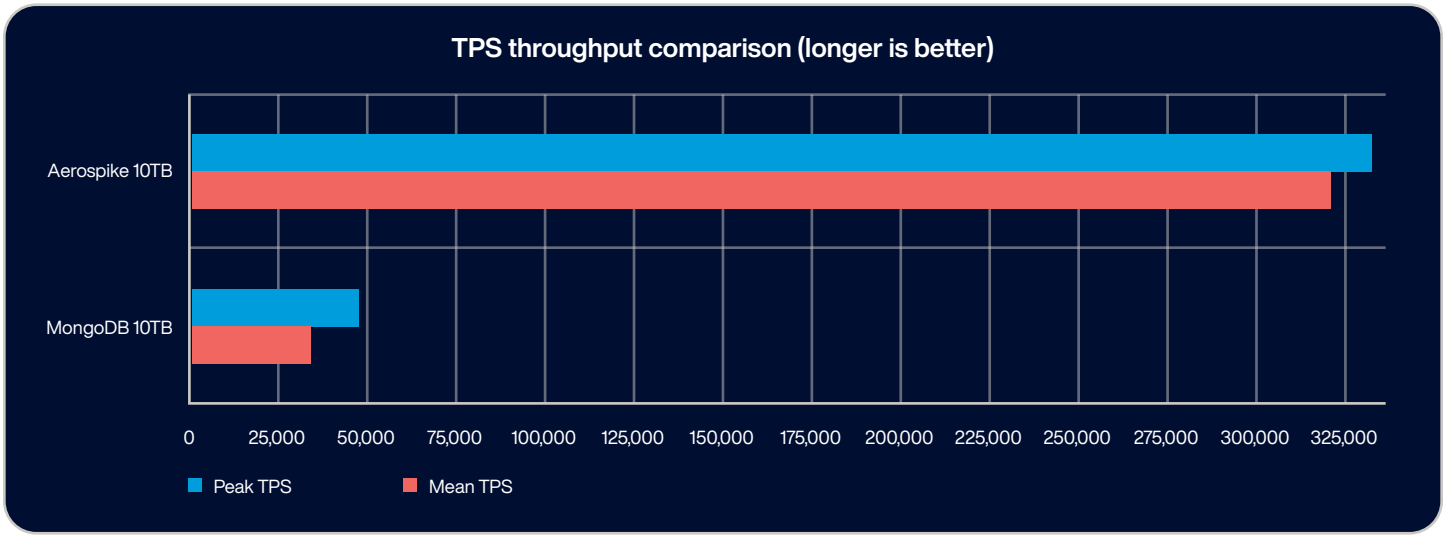


Figure 3: Peak and Mean throughput comparison as reported by the databases.

Aerospike's maximum and mean differ by just over 3% on the 10TB dataset test showing that Aerospike performance is solid, reliable and consistent as the dataset size increases.

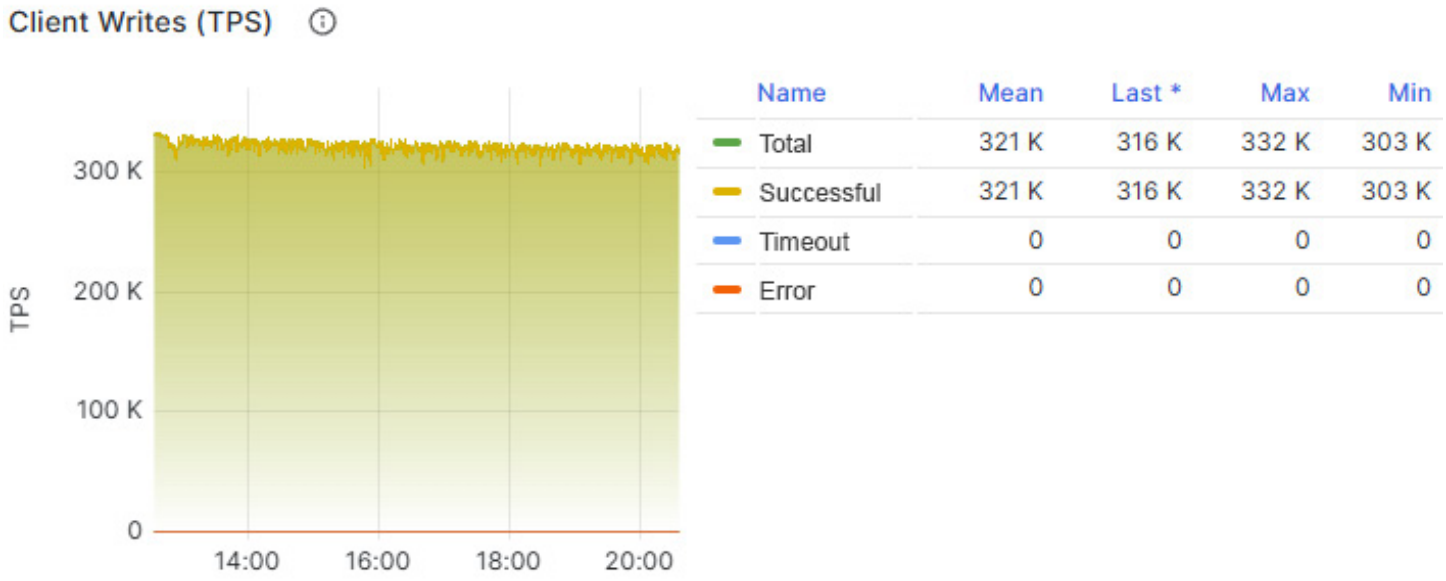


Figure 4: Aerospike throughput as presented in Grafana over 7 hours.

MongoDB's Maximum (when the database size was just getting initiated) and Mean TPS results are quite different, as is its Last (when the database hit 10 terabytes) showing that the performance is not consistent. It's clear from the graphs that MongoDB performance decays as the dataset size increases.

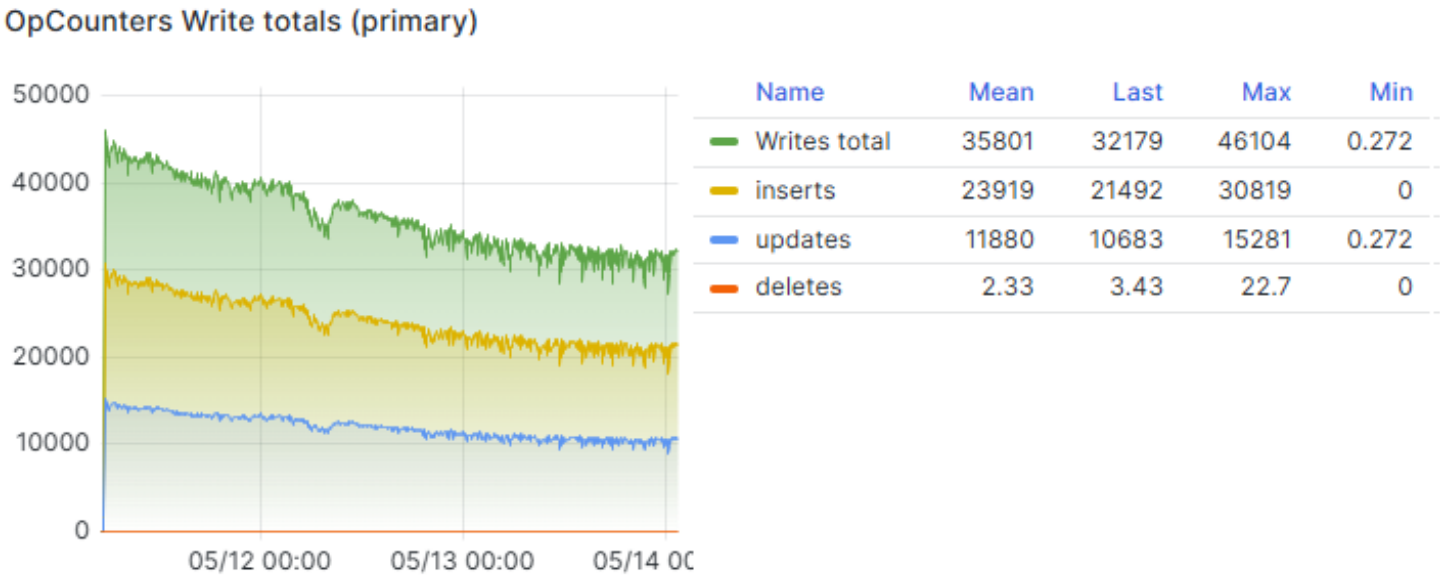


Figure 5: MongoDB Atlas throughput over the whole 10TB ingestion period, as presented in Grafana, shows performance decay when scaling.

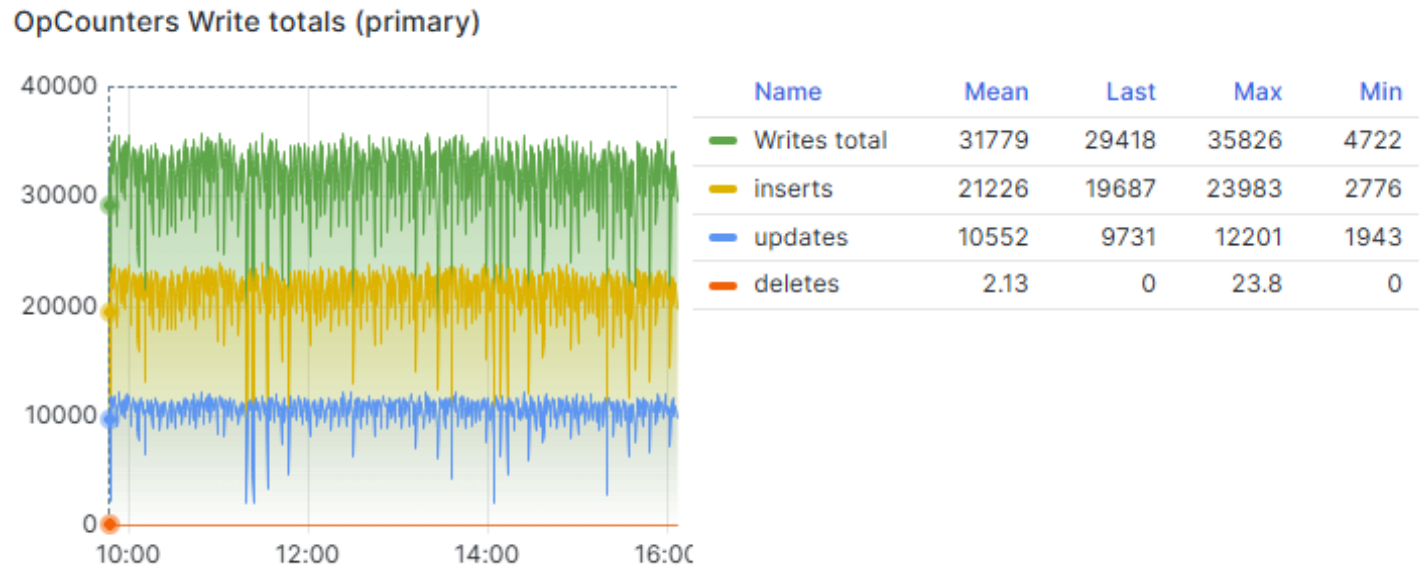


Figure 6: MongoDB Atlas throughput as presented in Grafana, zoomed in to display a 6-hour period exhibiting how varied the throughput is.

	MongoDB Atlas	Aerospike	Aerospike advantage
Mean (OPS)	35,081	321,000	9x
Last (OPS)	32,179	316,000	9.8x
Max (OPS)	46,104	332,000	7.2x
Variance	38.9%	5%	7.8x

Table 3: Throughput comparison as database builds from 0 to 10 terabytes.

Latency graphs

Latency, p95 write comparison

All Aerospike latency graphs are reported by Prometheus/Grafana console/dashboard.

All MongoDB latency graphs are reported by MongoDB Atlas console/dashboard, all shards.

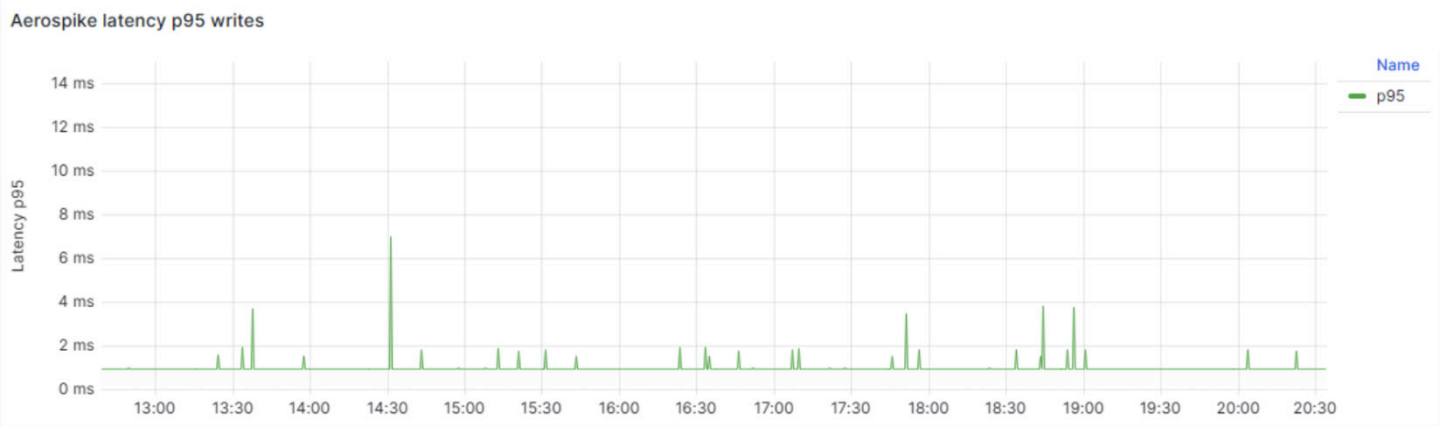


Figure 7. Aerospike consistently has low write latency at 95% across the database build from 0 to 10 TBs.

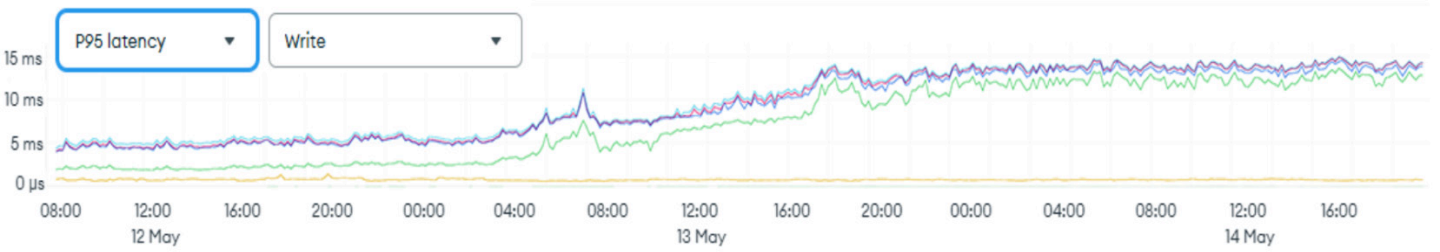


Figure 8: MongoDB Atlas p95 write latency: MongoDB p95 write latency for the five primary shards worsens from either ~1ms or 5ms (depending on the shard) to 15ms as the database builds from 0TB to 10TB over a three-day period.

Latency, p99 write comparison

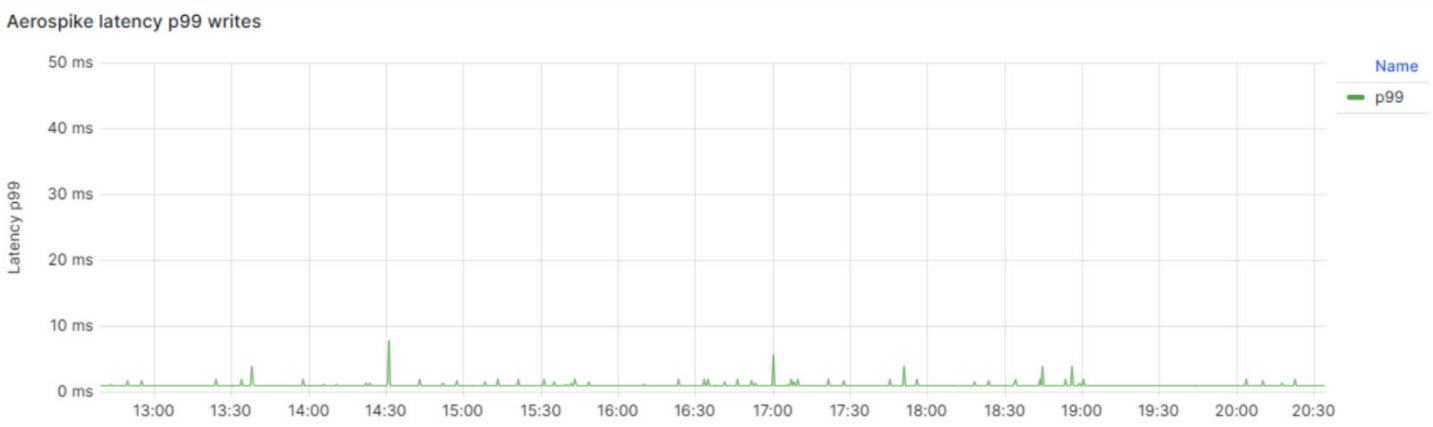


Figure 9: Aerospike consistently low write latency at p99 while the database builds from 0 to 10 TBs.

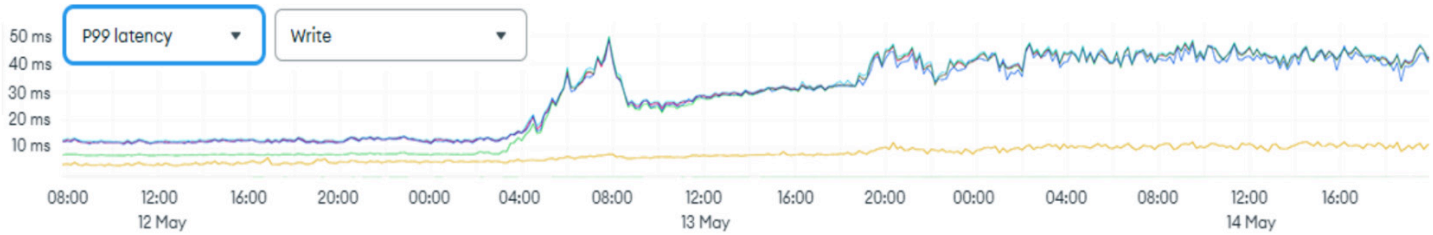


Figure 10: MongoDB Atlas p99 write latency: MongoDB p99 write latency worsens for the five primary shards from ~5ms or 12ms (depending on the shard) to 50ms as the database builds from 0TB to 10TB over a three-day period.

Latency, p95 read comparison

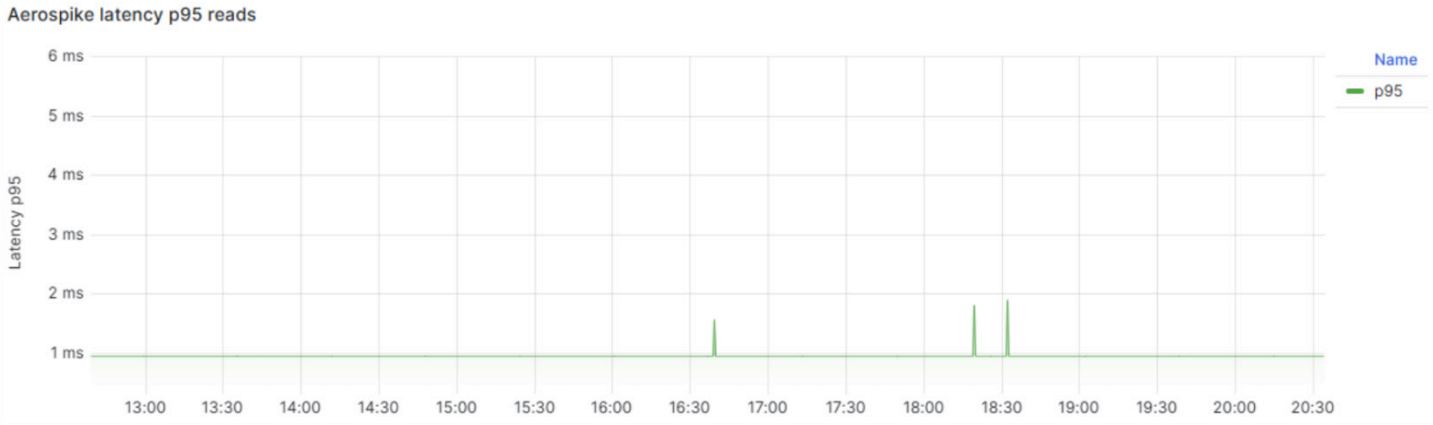


Figure 11: Aerospike consistently low read latency at p95.

Figure 12: MongoDB Atlas read latency p95, displaying the 5 primary shards.

Latency, p99 read comparison

Figure 13: Aerospike consistently low read latency p99.

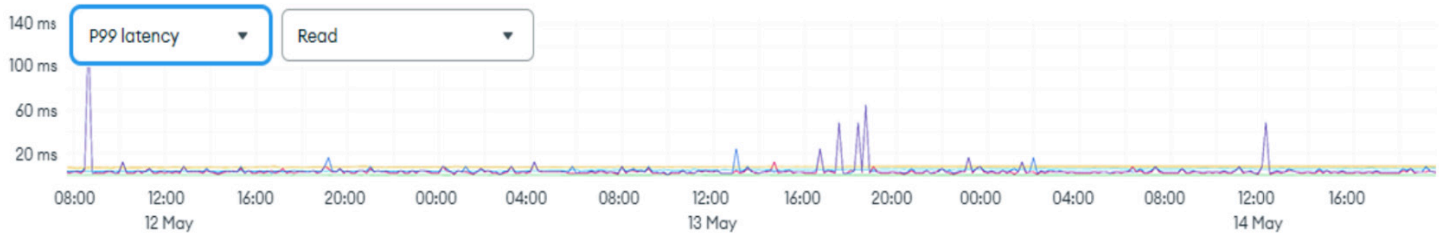


Figure 14: MongoDB Atlas read latency p99.

Resilience: Failure of a node, process, and diagnosis

We wanted to test node failures while a system was under load.

For the avoidance of any doubt, we need to be clear that the failures we are simulating are not identical. The Aerospike failure test is significantly more severe than the MongoDB Atlas failure test.

We credit MongoDB for making this failure test option available and we would like to see this functionality available in more DBaaS offerings.

Aerospike test of a node failure
No preparation was done. A random Aerospike node was stopped, and all database data on that node was deleted before the node was restarted, all while the database was under load.

This was a more difficult failure to handle than MongoDB's straightforward restart of a primary node.

MongoDB test of a primary node restart
A primary Mongo node was restarted while under load using the test primary failover[7] option in the Atlas dashboard. The data on that node was not deleted.

Note: The reason these tests differ is that MongoDB Atlas presents a DBaaS interface and consequently the failure simulation options were limited to what the DBaaS permits, whereas with Aerospike we had access to the hardware and O/S which allowed us to test a more serious failure scenario.

The MongoDB documentation states:

The following statements describe Atlas behavior during rollovers and when testing failover in sharded clusters:

- *If the original primary accepted write operations that had not been successfully replicated to the secondaries when the primary stepped down, the primary rolls back those write operations when it re-joins the replica set and begins synchronizing[8].*
- *Only the mongos processes that are on the same instances as the primaries of the replica sets in the sharded cluster are restarted.*
- *The primaries of the replica sets in the sharded cluster are restarted in parallel.*

Aerospike failure testing

An Aerospike cluster was set up.

This was a standard five (5) node cluster as described in environment 1 of Appendix A that a competent Aerospike practitioner would normally configure for production use.

The Load Generation Servers (LGS) were started. Writes and reads were started as per the main test with half as many LGS running, twenty (20) LGS.

For a standard commercial installation, Aerospike sizes clusters for peak throughput plus some overhead to accommodate failures, and it is in this spirit that we conduct this test, not with a fully saturated cluster.

[7] https://www.mongodb.com/docs/atlas/tutorial/test-resilience/test-primary-failover/
[8] https://www.mongodb.com/docs/atlas/tutorial/test-resilience/test-primary-failover/

At the point that 1TB of data had been inserted into the database the following steps were taken:

1. **Node stopped:** A random node was stopped cleanly (using `systemctl stop aerospike`). Any node can be chosen as all nodes take up an equal amount of the load. E.g. in a five-node system, each Aerospike node takes primary responsibility for one-fifth of the data.
2. **Data wipe:** We decided to make it harder for Aerospike by removing all the data on the stopped node. The first 8MB of each data storage device was zeroed out using the Linux `blkdiscard` command, this makes sure that when this Aerospike node joins the cluster, Aerospike considers the node to be a brand new empty node.
3. **Reboot:** The node was then rebooted, using the Linux command `reboot` .
4. **Start Aerospike:** Once the node restarted, the `systemctl start aerospike` command was issued.
5. **Data migration:** The node then starts the partition migration process (the Aerospike node recovery process) and when ready the node will take over primary duties for a portion of the dataset, in this case one-fifth of the dataset.

While steps one to five occur, there is never a situation where the cluster is unavailable, even though some open transactions that are connected to the node will fail as soon as the database is stopped, those transactions simply need to be retried (just once), these transactions will be simply redirected to another Aerospike node that took over the duties of the failed node.

This kind of retry-upon-failure is standard well-written application behavior and the application will simply see a small drop in performance for around a few seconds while Aerospike reconfigures itself.

Figure 15.1 - Aerospike during a node failure: no loss of service.

Figure 15.2 - "Zoom in" of Aerospike during a node failure (as it's hard to notice): no loss of service with Min dropping from 149K to 126K.

Even upon close examination, it's hard to see where the node dropped out and later rejoined. It happened at 02:26 on the graph above. There is **never** a moment where there is a total database connectivity loss - also known as an outage.

Aerospike takeaways

Aerospike smoothly handles the failure of a node and continues to deliver connectivity to an application even with this example where 20% of the Aerospike cluster was just stopped dead on the spot followed by fully deleting the database data on that node.

When you are dealing with financial services, fraud detection, payment systems, AdTech, e-commerce, and any other use case, this kind of resilience and continued responsiveness is critical to facilitate a good service provider experience and a good end-user experience.

MongoDB Atlas failure testing

A MongoDB Atlas database cluster was set up.

This was a standard five shard R200 configuration that a competent MongoDB Atlas practitioner would normally configure for production use - i.e. a cluster with enough headroom/contingency to ideally the same amount of client throughput to be achieved before and after the loss of a single primary database node.

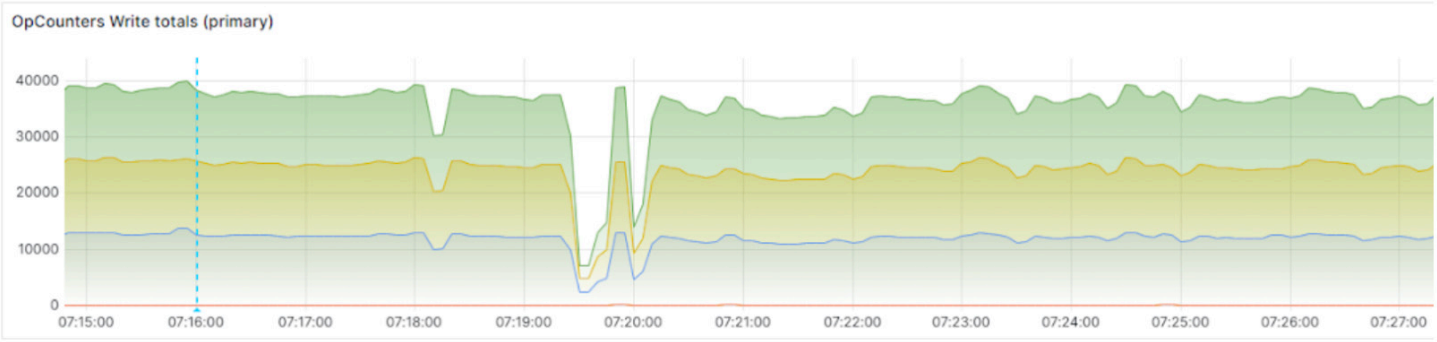
The LGS were started and writes and reads were started as per the main test, with half as many LGS running i.e. twenty (20) LGS.

At the point that 1TB of data had been inserted into the database the following steps were taken:

1. The MongoDB Atlas console allows you to test primary failover. This action was taken at the six minute point, at 07:16:00
2. When you submit a request to test primary failover, Atlas simulates a failover event[9]. During this process:
    - Atlas shuts down the current primary.
    - The members of the replica set hold an election to choose which of the secondaries will become the new primary.
    - Atlas brings the original primary back to the replica set as a secondary. When the old primary rejoins the replica set, it will sync with the new primary to catch up any writes that occurred during its downtime.

    MongoDB Atlas DB delayed restarting the primary for three minutes and twenty nine seconds (until 07:19:29) after the UI element was clicked. We can assume that MongoDB Atlas was preparing for the primary failover test for just over three minutes, whereas in a real world failure, there is no prep time.
3. A significant performance hit was seen where the write TPS dropped to 7192 TPS from ~36,000 before recovering after around 40 seconds.

[9] https://www.mongodb.com/docs/atlas/tutorial/test-resilience/test-primary-failover/ (retrieved 2024-01-24).



Name	Mean	Last	Max	Min
Writes total	36078	37694	39962	7192
inserts	24124	25267	26421	4830
updates	11944	12427	13824	2361
deletes	10.7	0.700	302	0

Figure 16: Zoom-in of MongoDB Atlas showing failure when issuing a node restart to test reliance, a large performance drop is seen.

MongoDB takeaways

When a MongoDB primary server fails, an election takes place between the remaining secondaries, one of those secondaries is elected and takes over the role of primary and then the cluster continues.

The key takeaway is that the performance takes a significant 80% hit for the period of the server failure.

For some organizations a 40 second period where there is wobbly performance before recovering is considered to be a fairly fast failover and can be rather undesirable yet begrudgingly acceptable.

When you are dealing with financial services, fraud detection, payment systems, AdTech, e-commerce, and any other use case that affects a public-facing service and thus can impact a user experience, a 40-second period is a significant amount of time during which MongoDB has a performance hit under what Aerospike feels should ideally be an instantly recoverable failure scenario.

Total cost of ownership

One area that should not be overlooked when comparing Aerospike to MongoDB Atlas is the cost to run each. For this benchmark with a 10TB workload, Aerospike has up to 5.4 times the cost advantage depending on Aerospike Database 7 license charges. However, given the sizable cost differential listed in Table 4, below, it would not be much of a stretch to envision a cost advantage for Aerospike. Furthermore, when considering the performance advantages, any cost advantage is meaningful when considering price-performance. As Aerospike does not publish its software licensing costs, one would have to contact sales@aerospike.com for more information.

	Instance[10] (per instance details)	Instance cost per hour	Instance cost per month[11]	Number of instances	Cost, annual[12]
Aerospike	**GCP** **n2-highmem-32** 256 GB memory 32 vCPU Intel Cascade Lake CPU 4x1500 GB storage (PD SSD, Zonal) us-central1-a	-	$1,917.35[13]	5	$115,041
MongoDB Atlas	GCP R200 Cluster 32 vCPU 256 GB memory 4096 GB storage 5 shards	$71.54[14]	$52,224	-	$626,690
Aerospike advantage					Up to 5.4x[11]

Table 4: Total cost of ownership (TCO) for Aerospike vs. MongoDB Atlas for 10 TB document workload.

[10] See Appendix A, Environment specifications for more detail.

[11] 1 year committed for GCP instances for Aerospike.

[12] Costs for Aerospike do not include 10TB of Aerospike Database 7 software licensing costs. Contact sales@aerospike.com for more information.

[3] GCP instance costs that Aerospike used: https://cloud.google.com/products/.

[14] MongoDB Atlas instance costs with storage and shards as calculated at https://cloud.mongodb.com and shown in Appendix A, Environment 2.

Conclusions

Overall, it is worth considering Aerospike when you have NoSQL database needs; it runs faster, scales better, is more resilient, is easier to maintain, and will have significantly lower TCO than MongoDB Atlas:

- Aerospike exhibits nine times the throughput compared to MongoDB Atlas using the same number of shards on 3.6 times fewer vCPUs.
- Aerospike scales linearly, whereas MongoDB does not.
- Aerospike handles node failures smoothly and is available at all times; MongoDB performance drops drastically when a failure occurs and takes some time to fully recover when the same failure scenarios are tested.
- Aerospike is self-healing across the whole cluster, and MongoDB allows for failovers across a single replica set.
    - Database size elasticity is relatively easy to manage with both systems;
    - Aerospike allows resizing up or down at any time with no connectivity loss.
    - MongoDB Atlas loses connectivity when resizing.
- However, if you scale from a MongoDB Atlas replica set to a sharded cluster, you cannot go back.
- Aerospike has a significantly lower TCO than MongoDB Atlas with significantly better performance levels.

Afterword

Objection handling

It's easy to offer criticism (whether justified or not) to any comparison exercise so we'll quickly list a few of the expected criticisms along with our thoughts.

1. **"The test wasn't fair. It's write heavy, not 50/50 read/write."**

   It's fairer to say the Player Protection Game Simulator application is write and update-heavy, not just write-heavy. This write/update profile is in the nature of the use case. The application accepts large quantities of real-time data at high velocity from clients, and that data is either stored directly as a recorded data point or used to update counters, sums, and aggregates, which are then used as part of the intervention calculations.

   The game reader app introduces a read load in parallel to the write/update profile of the Game Simulator. As a result, the read latencies for each vendor were similar and the resultant workload approximated 20/80 read/write.

2. **"MongoDB Atlas is a DBaaS and Aerospike is a cloud VM-based application."**

   We expect that MongoDB has optimized its application as well as it can (the right server vCPUs, O/S settings, drives, network access, etc.).

   Aerospike typically uses NVMe drives (but used persistent SSD storage for this test). MongoDB would have called out any NVMe usage as they cannot use those same fast drives on GCP (for unspecified reasons - perhaps cost).

   Aerospike did almost no optimization of the Aerospike database; it was simply a selection of three midrange SSD persistent drives per node.

3. **“GCP was chosen deliberately over other platforms for some benefit, maybe because MongoDB Atlas does not support NVMe on GCP.”**

   GCP was simply a choice amongst one of the main hyperscaler’s platforms. With any platform, you could argue that one has some benefit over another.

   With respect to the NVMe question, (as mentioned above in #2), Aerospike actually avoided their own best practice guidance which is to use local NVMe drives and instead used Network Attached SSD Persistent drives (a midrange choice of drive) to play more fairly against MongoDB.

4. **“You are keeping the benchmark code to yourselves so we can’t see what you did.”**

   The Player Protection Game Simulator application code has been released to https://github.com/aerospike-community/safegaming, along with the details of the test parameters so that you can test it yourself.

5. **“You didn’t follow your own good benchmarking guidelines, specifically the 48-hour minimum soak period for Aerospike.”**

   We’d like to agree... however:

   With respect to the 48-hour minimum test period, Aerospike would have ended up around the 80TB user data size, which would have skewed the instance and storage sizes and would no longer be a fair test as Aerospike would have a dataset some eight times larger than MongoDB.

6. **“Aerospike tests used RF of 2 whereas MongoDB has an effective RF of 3, that suggests Aerospike is only doing two-thirds of the work that MongoDB is doing.”**

   Many Aerospike customers use an RF of 2 because Aerospike can automatically migrate and rebalance the cluster in the event of a node failure, starting 1.5 seconds after a failure is identified by Aerospike (the default).

   MongoDB Atlas requires a minimum of three servers per replica set. This is because of the architectural choices made by MongoDB, meaning an odd number of servers is required.

   Even calculating two-thirds of the performance of Aerospike still shows Aerospike to be significantly faster than MongoDB on this workload.

7. **“How can you cite the Total Cost of Ownership? Aerospike isn’t disclosing its licensing costs!”**

   This is easily remedied by contacting an Aerospike sales representative to fill in the blanks, so to speak, as instance costs are clearly cited. Any total cost advantage is notable, especially given the performance advantages are multiples better, so price-performance overall would be significant.

Business impact of database reconfiguration

Once you have been working with a cluster of either Aerospike or MongoDB for a while, there will come a time when you will need to expand or shrink the cluster.

How do Aerospike and MongoDB handle this capacity elasticity?

Aerospike

With the Aerospike database, it’s really easy. When you need to scale up, you simply add one or more nodes to the cluster. These nodes can be larger, or you could just add more of the same, you might want to replace all the nodes with nodes that have newer or more storage, or you may just have a corporate rule that servers get replaced on e.g. a three-year schedule. With Aerospike, you add a new node by installing the Aerospike software, setting up the config file to match the existing nodes, and

starting the software. The Aerospike database will work the rest out, your data will be migrated automatically without drama.

When you need to upgrade to handle a peak in traffic, you can just add capacity. The Aerospike database will reconfigure and rebalance itself, and when the peak has finished, you can just remove that node to scale back down.

Aerospike does not experience any service interruption while you scale up or down. These scale-down and scale-up effects were witnessed in the above failure test in removing, zeroing out, and then re-adding that node.

MongoDB

MongoDB Atlas allows you to upgrade from one tier to another tier; it allows you to increase the storage size, size of a replica set and more. It also allows you to scale down a replica set to a smaller replica set. This appears easy to execute.

With MongoDB, you are advised to scale vertically (also known as "get a bigger box") but there will be a limit to how far this can take you.

Once you get beyond a certain vertical scaling limit, MongoDB Atlas allows you to upgrade a single replica set to a sharded configuration. Once you have a sharded configuration, you cannot go back to a replica set. This is a strictly one-way operation.

MongoDB has service interruptions while you scale up or down or adjust tiers, as this requires a replica set election as the primary node in a replica set is removed and updated.

External maintenance tasks

Here we considered the impact on performance of running an external maintenance task, either administratively controlled by yourself or forced upon you by the vendor.

For both databases, we have configured them in the way that a competent practitioner would normally configure them for production use—i.e., clusters with enough headroom/contingency to allow the same amount of client throughput to be achieved before and after the maintenance tasks were executed on a single primary database node.

For a workload, a single Load Generator Service (LGS) was used with the Player Protection application, and the application was executed using 16 concurrent players.

MongoDB Atlas

It feels like there shouldn't be much maintenance work to do on a MongoDB Atlas cluster as it is a DBaaS; however, there are situations where you may wish to change the cluster size, tier, or some other similar operation. These operations can incur a short-lived yet significant hit where you have no connectivity to your MongoDB Cluster.

With Atlas, you can select maintenance windows for most things, but in this day and age of 24-hour businesses (for which you would likely need a product like MongoDB or Aerospike), you should not have to accept a situation where you experience an outage because of maintenance requirements.

Quoted directly from MongoDB's site:

> *"Atlas performs urgent maintenance activities such as security patches as soon as they are needed without regard to scheduled maintenance windows."*
>
> *"Once you schedule a maintenance window for your cluster, you cannot change it until any ongoing maintenance operations have been completed."*
>
> *"This procedure requires at least one replica set election during the maintenance window per replica set."*

The last paragraph seems fairly innocuous until you appreciate that each shard (each shard is a distinct replica set) can have its own small outage.

However, there will still be some effect/"glitch" akin to the behavior where you lose connectivity akin to what was seen in the above section, MongoDB Atlas failure testing.

Aerospike

With Aerospike, you can choose to quiesce a node. The quiesced node can undergo any necessary maintenance (e.g., an O/S upgrade, patch day, etc.). You can then simply reverse the quiesce operation and carry on smoothly.

This is particularly powerful as resizing nodes, updating nodes, and any other standard maintenance task against an Aerospike cluster or node can be done with almost no impact on the database.

Shown below is the impact of running maintenance on a node after quiescing the node first; there is hardly any impact.

The sequence of events is as follows: apply a load, wait six minutes, quiesce a node, issue a recluster command, stop Aerospike on that node, remove that node from the cluster, wait a minute to simulate some maintenance work, restart that node (reboot), and then restart Aerospike.

Figure 17: Quiescing an Aerospike node at 03:40 has no impact on throughput.

For comparison, here is a performance graph showing an Aerospike DBA making a mistake and forgetting to quiesce the node before the maintenance was done. Even with a mistake like this one, Aerospike only sees a small performance impact when under heavy load, and the database continues to accept and handle queries successfully at all times.

Figure 18: The impact of a DBA mistake showing only a small throughput impact against Aerospike.

An operations perspective

For this section, we'll assume the role of an operations team lead, making sure things are running smoothly and properly on the selected platform and database.

What we want is to finish work at a decent hour, have no interruptions over the weekend, and not be concerned with needing to change personal plans. We don't want to be worried that something will break and cause panic, requiring a significant amount of time on calls while trying to stabilize the chosen database.

There is also a budget to work with—we can't just hire lots of staff or have ten redundancies per node.

So can Aerospike and MongoDB give us what we want?

Aerospike

To be overly safe for this use case with Aerospike, we would use the Rack Aware Functionality (which ensures the master and replicas are kept separate in different racks across availability zones) and set the Replication Factor (RF) to 3.
That's it—we're done.

When using the default Aerospike high availability (AP in CAP theorem-speak) mode, the RF can be changed on-the-fly[15], so if one has a long weekend planned, holidays coming up, or has staff illnesses, etc., one can adjust the system to RF=current RF+1 to offer more resilience. This way one can be more comfortable for that period - although this will use more storage for the duration, it will not cost any more in license fees as Aerospike charges by unique data, not the number of servers.

If there is a node failure or a rack failure, one would be covered. The moment a node failure is recognized - the default amount of time this would take is 1.5 seconds (10 consecutive connection failures of 150ms each is the default trigger) - Aerospike will smoothly migrate partitions from one node to another to spread the load around the available nodes to make sure there are three full copies of the data (the RF=3 desired state as I requested) spread out over the remaining nodes.

This can take a few minutes, but it's all built-in and automatic, and you have database availability throughout the migration. Assuming the initial sizing was executed correctly as per Aerospike guidance, you will still have the base performance that you need to achieve your performance SLAs, and there will be no outage of the database whatsoever.

In the event of a permanent failure of the node or rack, one can simply bring up a fresh node or a rack, and connect it to the existing cluster. The Aerospike database will automatically migrate the data partitions and rebalance things quickly, smoothly and without any further effort on an operator's part.

Note, as per the External Maintenance Tasks section, that you can quiesce a node before any maintenance is undertaken. This is if you want to avoid migration/rebalancing while doing work on a node. When there is a petabyte of user data (not uncommon across Aerospike customers), the migration can be a network-heavy task—there is no reason to impact it with simple maintenance work.

Aerospike operational downtime

Aerospike doesn't have any operational downtime for maintenance work.

If sized correctly, Aerospike won't have any downtime and will continue to offer the same SLA guarantees, handling the same target TPS regardless of whether a rack or a node has failed. It will offer the same SLA guarantees and uptime guarantees while you do your maintenance tasks. Aerospike smoothly handles - all automatically - the partition migrations of the data over the remaining nodes when required with zero operator intervention.

[15] https://docs.aerospike.com/server/reference/configuration#replication-factor

If you want to expand the cluster, simply instantiate a new node, install Aerospike, add the IP or DNS name of an existing node to that node's configuration file, and start Aerospike. If you need to shrink your cluster, then simply decommission a node. Again, Aerospike handles the contraction automatically.

Seasoned operations team members are aware that firewalls and networks etc. will also need to be configured. However, this is the standard basic operational detail we all go through; nothing out of the ordinary is needed for the Aerospike database.

MongoDB Atlas

It's more difficult to determine the exact internal processes used with a MongoDB Atlas cluster as Atlas is a DBaaS, but in testing, it is possible to simulate a primary node restart - note this is not a failure but a restart. You can click on the 'restart primary' button in the console/dashboard, and around two to three minutes later, you see performance hit for around 40 seconds, followed by a recovery of the cluster.

MongoDB operational downtime

When a primary node fails or cluster maintenance is done, i.e. any time a replica set election takes place, an outage is observed, this can be as low as 16 seconds (in our testing).

Clearly, an undesirable operations environment.

Glossary

Aerospike AP mode	An Aerospike operating mode is Available and Partition-tolerant (AP) as defined in the CAP theorem.
Aerospike SC mode	An Aerospike operating mode is Consistent and Partition-tolerant (CP) as it pertains to the CAP theorem. Aerospike refers to this as Strong Consistency mode (SC)
Aerospike migration	Aerospike balances data in the cluster by migrating it between cluster nodes. Data moves as a part of a partition and is monitored on a per-namespace basis. For each namespace, every record is mapped to one of 4096 partitions
Aerospike partition	Each namespace is divided into 4096 logical partitions, which are evenly distributed between the cluster nodes. This means that if there are *n* nodes in the cluster, each node stores ~1/n of the data
Aerospike quiesce	This allows an Aerospike node to become temporarily dormant. This is beneficial when upgrading/downgrading and maintaining clusters.
Aerospike shard	As Aerospike partitions data automatically and distributes evenly across nodes, for the purpose of this paper, one shard or horizontal partition in Aerospike is equivalent to an instance or server. Learn more about Aerospike horizontal scaling.
LGS	Load Generation Server, sometimes described as a VM running the client application but not the database. See also SUT
SUT	System Under Test. The database that is being tested. See also LGS

Appendix A: Test environments

The environment numbers relate to specific test configurations, in terms of hardware and software.

Load Generation Server (LGS) specification

Platform	Google Cloud Platform (GCP)
Zone	us-central1-a
LGS Instance type	n2-highcpu-8
vCPU type	Intel Cascade Lake CPU
vCPU count	8 per instance
Memory	8 GB per instance
Number of LGS servers used	Forty (40)

Environment 1 specification: Aerospike

Aerospike's best practice is to use NVMe locally attached storage, not network-attached mid-range drives that were used here.

The network-attached persistent SSD disks were selected to mimic MongoDB Atlas's lack of NVMe capability on the GCP platform.

Platform	Google Cloud Platform (GCP)
Zone	us-central1-a
Database	Aerospike 7.0 Enterprise
Cluster size	5 nodes
Database instance type	n2-highmem-32
vCPU type	Intel Cascade Lake CPU
vCPU count per node	32
Memory per node	256 GB
Root volume per node	pd-ssd:20
Data volume per node	pd-ssd:1500 pd-ssd:1500 pd-ssd:1500 pd-ssd:1500

Aerospike configuration	`namespace safegaming {`
	`  default-ttl 0`
	`  index-stage-size 128M`
	`  replication-factor 2`
	`  sindex-stage-size 128M`
	`  stop-writes-sys-memory-pct 90`
	`  storage-engine device {`
	`  device /dev/sdb`
	`  device /dev/sdc`
	`  device /dev/sdd`
	`  device /dev/sde`
	`  stop-writes-avail-pct 5`
	`  stop-writes-used-pct 70`
	`  write-block-size 1024K`
	`  }`
	`}`

Note: "safegaming" is the internal name for the Player Protection simulation.

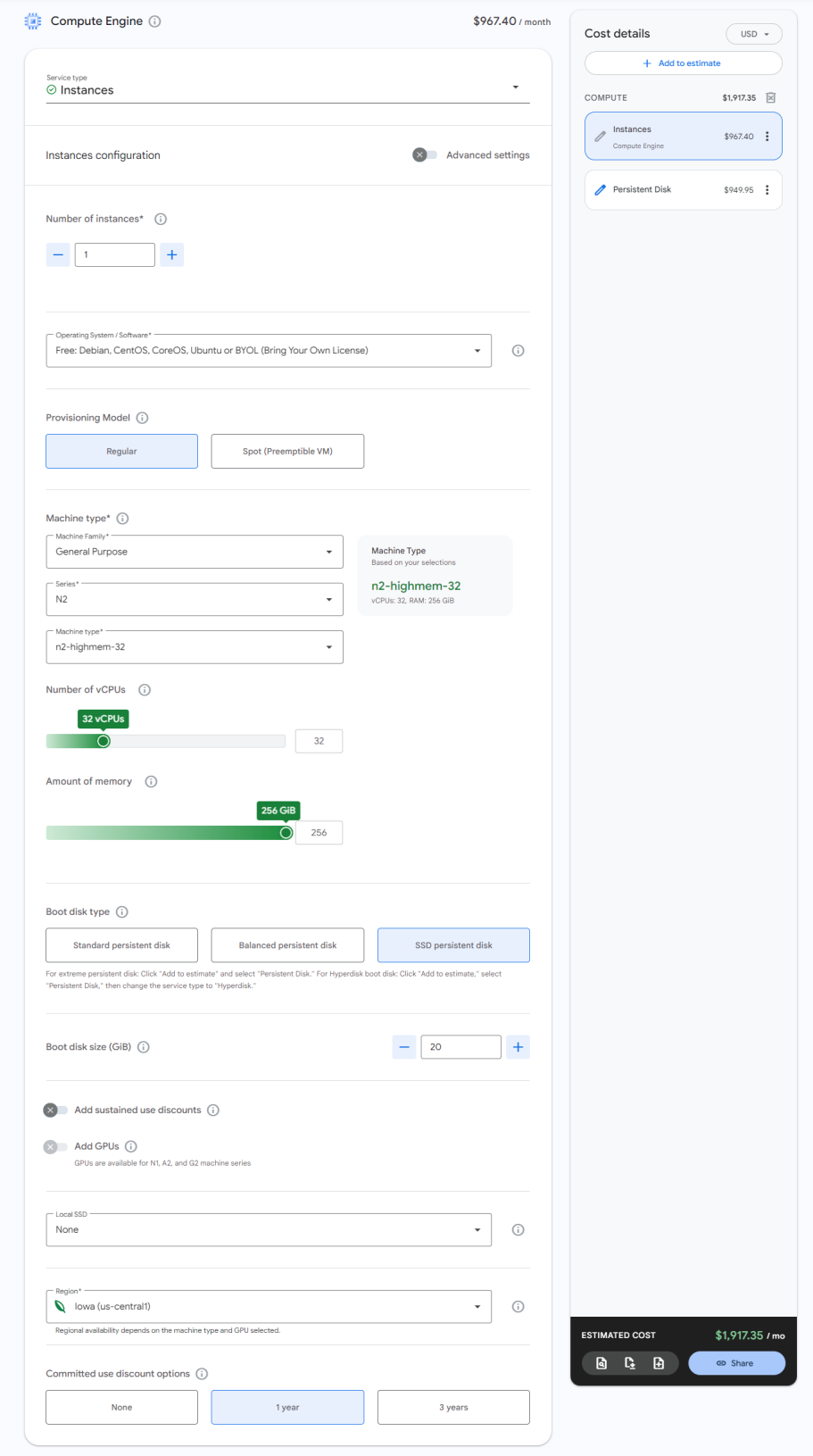


Figure 19: GCP Instance and Storage pricing for Aerospike.

Environment 2 specification: MongoDB Atlas sharded

Platform	Google Cloud Platform (GCP)
Zone	Iowa (us-central1)
Database	MongoDB Atlas 7.08
Cluster size	18 servers total - made up as follows: R200 cluster with five shards. [16] Each shard consists of one primary and two secondary servers, which is a total of 15 servers. Plus 1 * M30 config server cluster consisting of 3 servers total, made up of one primary and two secondary servers
Database instance type	R200
vCPU type	Not known
vCPU count	32 per cluster VM
Memory	256 GB per cluster VM
Storage	4096 GB per cluster VM
Other information	60,000 Total IOPS - up to 60,000 Read IOPS and 60,000 Write IOPS Additional Info 128000 max connections Extremely High network performance

[16] Shard sizing per Testing Approach, Shards section.

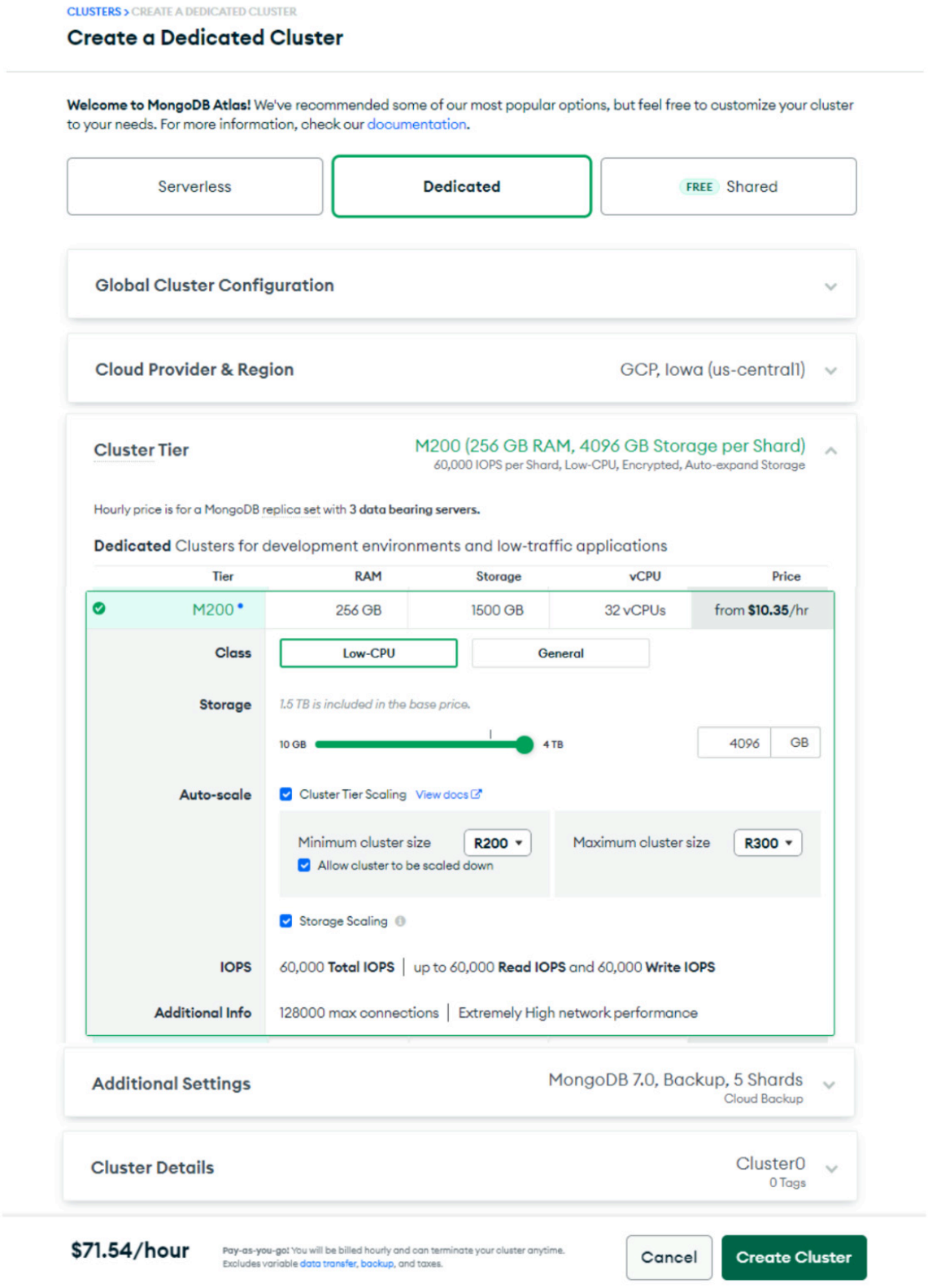


Figure 20: MongoDB Atlas pricing for five shards.

Appendix B: What is the Player Protection Simulator?

The Player Protection Simulator was developed with one of Aerospike's partners, Intuita, to showcase how Aerospike can be used as a document data platform for real time intervention of risky online betting behaviors. Online gaming is experiencing significant growth, and there is an increased burden on operators to demonstrate that they are capable of intervening when players demonstrate unhealthy gambling behaviors. Failure to do so results in large fines for the operators of these online casinos[17].

The Player Protection Simulator creates a large number of simulated players from different FIPS 6-4 areas (USA counties or county equivalents), simulates appropriate access patterns to casinos or similar establishments, and for each FIPS code uses statistical models of income, spending habits, propensity to gamble, how much is spent per gamble (based on historical data for that FIPS code) and uses the standard statistical rules and laws for percentage wins and losses per gamble per game (roulette, blackjack, slots etc).

Note: Here is a video demo of the simulator itself.

Many player parameters were based on census data per FIPS area (age, profile, status, family size and makeup, loan and credit status and value, etc.), and player behaviors were based on long-known empirical study observations. Some parameters were randomized between appropriate minimum and maximum limits (how much initial ready cash a player has to gamble with for example).

The tests were conducted using two applications, the Player Protection Simulator and the Player Protection Reader. These two applications were run in parallel and were used to generate Read/Write/Update workloads with a breakdown as follows:

- Using the Simulator, as much throughput as possible with a split of 66% writes and 33% updates
- Using the Reader, a fixed 2000 transactions worth of user data per second.

Note: Between the two applications, the resultant Read/Write ratio was 20/80 R/W.

There are three main phases and three main termination situations.

Main phases:

- Create simulated players that match observed reality as closely as possible based on US census data.
- Each player has a portfolio based on their selected county. This portfolio contains demographic, economic, and social data required for realistic play and interventions.
- Simulated game play can be one or more spins, pulls, wager, etc. on multiple games. The odds of winning/losing are the actual odds for that game including payout amounts. Simulated play also takes into account the amount of time a play takes (e.g., one spin on a slot machine) plus player breaks (e.g., getting a drink, restroom, etc.), player session time lengths, etc.

[17] CA$ 150,000 penalty: https://www.agco.ca/blog/lottery-and-gaming/nov-2023/agco-issues-150000-penalties-pointsbet-violations-internet-gaming.
European Union Gambling Commision Enforcement penalties, news thereof: https://www.gamblingcommission.gov.uk/news/enforcement-action.
CA$ 100,000 penalty: https://www.agco.ca/blog/lottery-and-gaming/aug-2023/agco-issues-100000-penalties-apollo-entertainment-violations.

Play termination:

- Player decided to stop playing. This is a simulation of reality; in many cases, players were casual and just gambled a little money, some players won and decided to take their winnings, others were a little more serious and spent more, and some stopped playing when they faced significant losses but were not yet in difficult financial positions and some players continue to try to play to the detriment of their means, i.e. until they ran out of all money.
- In the last two scenarios, the simulator provides appropriate intervention triggers. The logic behind the intervention triggers is more complicated than it may appear, as it is the combination of the pattern of behavior when considered together with their financial position that is what matters.
- The key point here is that intervention triggers are provided in real time, so interventions can be made when it matters.

In summary, the simulation is as close to real-world behavior as possible, and the goal is to identify and intervene when a gambling disorder is identified.

This all constitutes a significant amount of processing as the simulation is complex.

The simulation presents a high-velocity, complex series of data points that all have to be correctly logged so that the provenance of each intervention decision can be fully traced.

The Player Protection Game Simulator code base and game logic were unchanged, i.e., identical when tested against Aerospike and MongoDB.

The Player Protection Database

The data stored by the Game Simulator is complex and does not easily conform to a standard RDBMS style n normal form style of formal schema.

The data is however well suited to any database that can store document type schemaless data such as MongoDB or Aerospike.

With MongoDB, the data is stored in the native BSON form, a compressed binary JSON-like serialization format.

With Aerospike, the data is stored in the Aerospike Map Collection Data Type (“Map CDT”).

Each database was used to its strengths; no conversion of the data was required in the provided driver, nor were any conversions needed at the database level.

Player Protection Parameter selection

The Player Protection Game Simulator allows for many parameters to be selected - most of these are related to histograms, starting player id numbers (so that the load can be applied from multiple servers), debug flags, and so on.

The important parameters for our benchmark runs are :

-d, --MaxDegreeOfParallelism

-s, --start-key ; The value used to start counting the player id generation

-k, --keys [Default Value “100”] The number of players generated (NbrPlayers)

Each simulation was run with a setting of a value of approximately -k 50000 and -d 16 on each LGS.

Approximate means that each run was a different number of keys to make sure there was no coordinated completion and start of a new run at the same point in time as this could present itself as a noticeable drop in performance for a second or so while a new simulation run was initiated. Both Aerospike and MongoDB used 40 LGS.

This meant that each LGS allowed up to 16 players to play at any one time, in parallel.

As each player completed their gaming session, another player took their place until a total of ~50,000 players had completed their gaming sessions.

Each LGS ran the above multiple times with a different -s value (to avoid player ID conflicts) until the overall dataset reached the target size of 10TB.

Each player has a session and they play a game or different games within that session. A session ends with the time limit for that player has exceeded, if they hit a financial threshold (e.g., ran out of money, hit their limit, etc.) or had an intervention.

A player can have multiple sessions running while executing the application. Once the session has ended, a new player can begin or a prior player may begin if they meet the time between sessions and have the ability to still play (good standing).

As each player is generated and plays games, a large number of database touches are generated as each game the player plays is logged, along with the wager amount, win or loss amounts, time of game, type of game, and so on.

It should also be noted that the average time between gambles, roulette spins, bathroom breaks, time to get a drink or food, and more are adjustable to match real-world gambling intervals. For the test, these intervals were made smaller so that the number of database touches would be condensed in time.

For this evaluation and using the setting listed above, the Game Simulator issues writes/updates at an approximate split of 66% inserts and 33% updates. The average object size reported by MongoDB is 1324 bytes (which will be a similar size on Aerospike). This size will vary a little over different runs. Aerospike uses 1 namespace containing 6 sets, MongoDB uses 1 database containing 6 collections, and all indexes are primary key indexes, no secondary indexes are used.

Reading records

When we introduce a new player, we set up a player record: name, address, socio-economic demographic information, etc.

Each player then plays a series of games according to some random chances and associations depending on their status and various other factors (time of day, likelihood of that player with specific demographics and location would play specific games, etc.). Also, after a win - which is determined using standard well-known gambling odds - a player gets more plays because their risk is lowered at that point, meaning more records then get associated with that user.

So, the player and their associated records grow over time until they are deemed to be risky players (decided according to the law of the region), at which point the Player Protection component of the simulation steps in to stop playing (with a warning first).

A read randomly picks a player and reads all their data, 2000 users per second, so the precise number of actual database records read varies and is meant more of a load as a mixed workload rather than a load to be measured in precise throughput and latency. In general, the read load was targeted at approximately 20% of the total load, with 80% being writes for each database tested in this benchmark.

Appendix C: Typical document in a collection/set

The following is a GitHub link to an example of a "Current Player" document that would be stored in Aerospike as a record of collection data types or in MongoDB as a BSON document. Six collections/sets are used in Player Protection, and both use identical document data models.

The "Current Player" document encompasses a player's current state, which consists of multi-level nested documents. Below is the player document model. (Note: As the player data structure is quite sizable, please feel free to access sample customer data via the intended Github link)

The takeaway is that this is a reasonably complex document of a reasonable size. It is not a one-line non-nested simplistic JSON document just to get as many of those through to a database as possible in as short a time as possible.

In this example document, all names, characters, and gambling-related information as portrayed are fictitious. No identification with actual persons (living or deceased), places, buildings, products, amounts, or games is intended, nor should it be inferred.

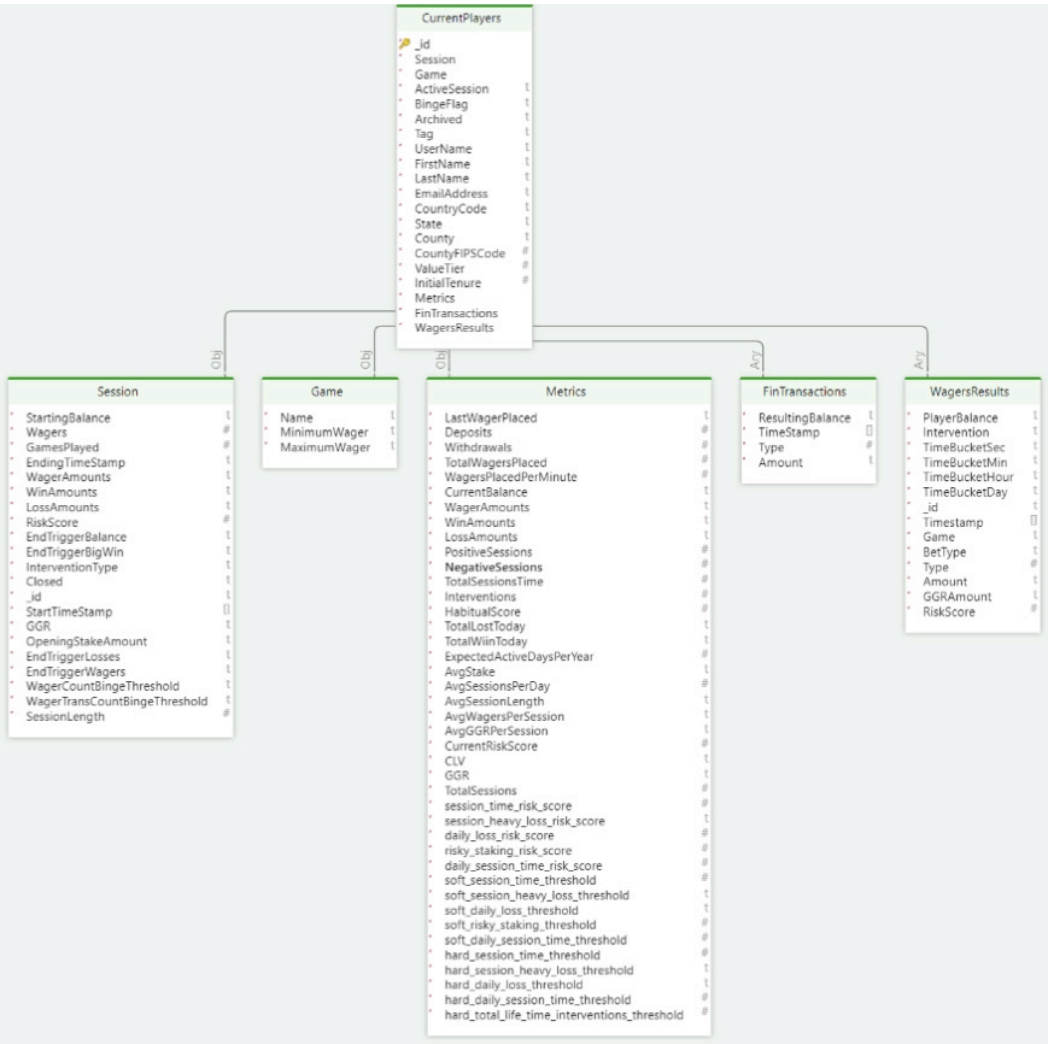


Figure 21: Example document.

About the author

Phil Allsopp

Phil is a Principal Performance & Reliability Engineer at Aerospike. He has also worked as a Principal Global Solutions Architect for one of the largest Postgres vendors. Prior to this, Phil had designed and developed a number of flight simulators, and numerous e-commerce systems for a large proportion of the worldwide music industry, as well as for some of the world's largest charities. All of these different projects/products brought theoretical and real-world performance and reliability challenges.

2440 W. El Camino Real, Suite 100, Mountain View, CA 94040 | (408) 462-2376 | aerospike.com