ALTOROS™

2022 Performance Evaluation of NoSQL Databases as a Service:

Couchbase Capella and MongoDB Atlas

Using Yahoo! Cloud Serving Benchmark, this 21-page report compares the throughput and latency of the two databases across four business scenarios and three cluster configurations.

By **Ivan Shryma**, Data Engineer, Altoros

Q3 2022

Table of Contents

1. Executive Summary

NoSQL encompasses a wide variety of database technologies that were developed in response to a rise in the volume of data and the frequency with which information is stored, accessed, and updated. In contrast, relational databases were not designed to cope with scalability and agility challenges that modern applications require. Furthermore, on-premises databases cannot take advantage of the affordable storage and processing power available in today's cloud environments. Meanwhile, new-generation NoSQL solutions help to achieve the highest levels of performance and uptime for modern application workloads. Finally, teams are more regularly seeking Database-as-a-Service (DBaaS) options to avoid having to invest increasing amounts of time and money on cluster support, deployment, and maintenance.

This report compares the performance results of two modern NoSQL DBaaS offerings: Couchbase Capella™ (the release as of March 2022) and MongoDB™ Atlas v5.0. The goal of this report is to measure the relative performance in terms of latency and throughput that each database can achieve. The evaluation was conducted on three different cluster configurations—6, 9, and 18 nodes—as well as under four different workloads.

The first workload performs *update-heavy* activity, invoking 50% reads and 50% updates of the data. The second workload performs a *short-range scan* that involves 95% scans and 5% updates, where short ranges of records are queried instead of individual ones. The third workload represents a query with a *single filtering* option to which an offset and a limit are applied. Finally, the fourth workload is a `JOIN` query with grouping and ordering applied.

As a default tool for evaluation consistency, we utilized the Yahoo! Cloud Serving Benchmark (YCSB)—an open source specification and program suite for evaluating retrieval and maintenance capabilities of computer programs.

The results of these test activities illustrate that Couchbase Capella outperforms MongoDB Atlas when running the same workloads on the same AWS cloud resources. Furthermore, the operating costs of Capella were over 40% less than MongoDB Atlas.

2. A Testing Environment

2.1 YCSB instance configuration

To provide verifiable results, the benchmark was performed on easily obtained Amazon Elastic Compute Cloud (EC2) instances. The YCSB client was deployed to four compute-optimized large instances, except for Workload A, which was configured using 10 instances. Each client instance of YCSB produces threads from 25 to 175 in 25-thread increments, just for Workload A it has been used from 10 to 70 in 10-thread increments. This means the total load on the database ranged from 100 to 700 threads with increments of 100 during each test. For this report, the median number of threads (generally, 400) was used to determine the throughput and latency.

Table 2.1 *A detailed description of the Amazon EC2 instance deployed to the YCSB client*

Family	Compute-optimized
Type	c4.2xlarge
vCPUs	8
Memory (GiB)	15
EBS-optimized available	Yes
Network performance	High
Platform	64-bit
Operating system	Ubuntu 18.04 LTS
AWS region	us-east-1

2.2 Couchbase Capella cluster configuration

Couchbase Capella is a fully managed database as a service. It combines features of a key-value store allowing operations on single documents and acts as a schemaless document store to access the documents by querying through SQL++ (SQL for JSON documents).

The Capella Control Panel includes a cluster sizing page, offering customers multiple options to choose from—such as instance sizes, configurations, and quantities. Capella can also be fine-tuned to deploy specific services to a single node or more in the cluster. The vendor calls this feature “multidimensional scaling.”

Clusters were configured to run one or more Data, Index, and Query services. The Data service is the most fundamental of all Couchbase services, providing access to data in memory and on disk. The Index service supports the creation of primary and global secondary indexes on items stored within Couchbase Server. The Query service supports the querying of data by means of SQL++ and relies on both the Index and Data services. Figure 2.2 shows the architecture of an example Capella cluster.

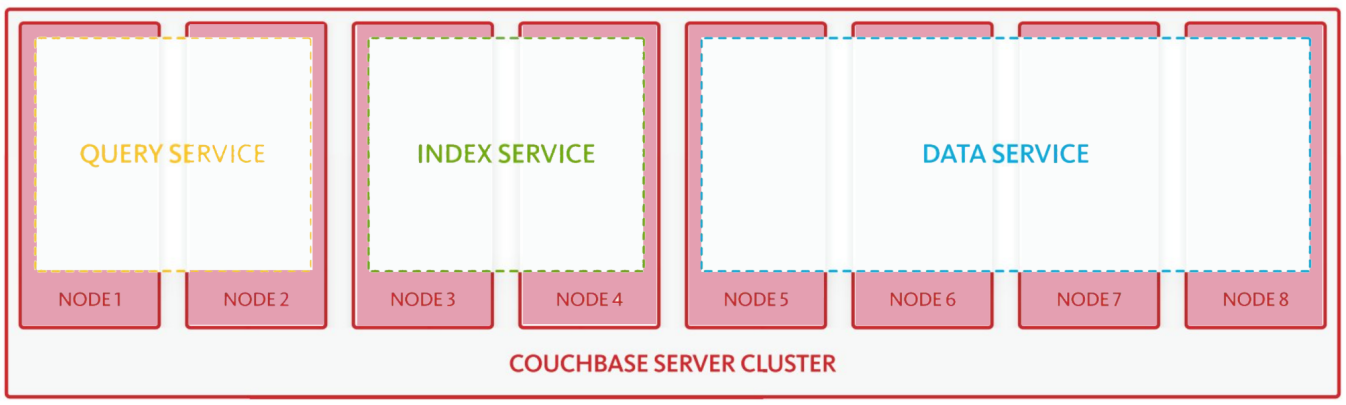


Figure 2.2 The architecture of a Couchbase Capella cluster

The table below shows the specification of each of the Couchbase Capella instances used for this benchmarking.

Table 2.2 *Specification of a Couchbase Capella instance*

vCPUs	8
Memory (GB)	64
EBS storage (GB)	200
IOPS	5,700
AWS region	us-east-1

2.3 MongoDB Atlas cluster configuration

MongoDB Atlas is a document-oriented NoSQL database. It has extensive support for a variety of secondary indexes and API-based ad hoc queries, as well as strong features for manipulating JSON documents. The database puts forward a separate and incremental approach to data replication and partitioning that happen as completely independent processes.

In this evaluation, we utilized MongoDB Atlas v5.0. Though v6.0 has been recently released, the newly introduced features are unlikely to have a significant impact on the performance of the database. It employs a hierarchical cluster topology that combines router processes, configuration servers, and data shards. For each cluster size (6, 9, and 18 nodes), the following production-grade configurations were used for deployment.

- A config server was deployed as a three-member replica set (a separate machine, not counted in a cluster).
- Each shard was deployed as a three-member replica set (one primary, two secondaries).
- MongoDB Atlas's routers were deployed on each node for each shard.

Automatic installation and configuration for a MongoDB Atlas sharded cluster is a simple procedure. Users can choose their preferred cloud provider, region and type of nodes, count of shards, as well as the size of a replica set. The configurational server was a three-member replica set deployed automatically. Figure 2.3 shows the typical architecture of a MongoDB Atlas cluster.

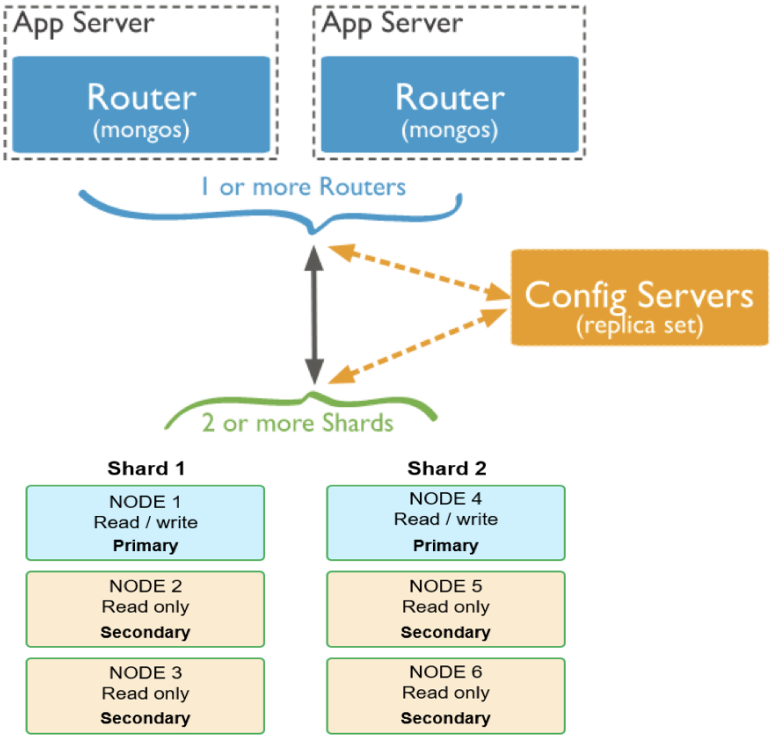


Figure 2.3 The architecture of a MongoDB cluster

MongoDB Atlas distributes shards at the collection level. MongoDB Atlas's sharding feature partitions the collections' data using a shard key. Hash-based partitioning was used for all the models. To support hash-based sharding, MongoDB Atlas provides a hashed index type that indexes the hash of a field value. With hash-based partitioning, two documents with "close" shard key values are unlikely to be part of the same chunk. This ensures more random distribution of collections in the cluster.

***Table 2.3** A detailed description of a MongoDB Atlas instance*

Type	M60
vCPUs	8
Memory (GB)	64
SSD storage (GB)	200
IOPS	5,700
AWS region	us-east-1

2.4 Pricing

2.4.1 Couchbase Capella

The monthly billing report for running Couchbase Capella includes per instance–hour costs billed by Capella.

Approximate monthly total for supporting a Capella cluster of specified configuration:

- 6 nodes amounted to around $5,284
- 9 nodes amounted to around $7,926
- 18 nodes amounted to around $15,851

Note that charges in Couchbase Capella are billed in Couchbase Capella Credits.

2.4.2 MongoDB Atlas

The pricing for the MongoDB Atlas database is calculated based on the services that are used for cluster configuration. For this report, the following services were used:

- Atlas Instance—$1 per server per hour
- Atlas Data Storage—$0.000182 per GB per hour
- Atlas Data Transfer—$0.01 per GB

Approximate monthly total for supporting a cluster of specified configuration:

- 6 nodes amounted to around $9,756
- 9 nodes amounted to around $14,292
- 18 nodes amounted to around $28,050

Note that costs for Couchbase Capella and MongoDB Atlas for this particular testing environment are different. The costs for Couchbase Cappella are more than 40% less than that of MongoDB Atlas. The reasons for that are different architectures and customization of a MongoDB Atlas installation to create fair comparison conditions under this benchmark.

3. Workloads and Tools

Database performance is defined by the speed at which a database processes basic operations. A basic operation is an action performed by a workload executor that drives multiple client threads. Each thread executes a sequential series of operations by making calls to a database interface layer both to load a database (the load phase) and to execute a workload (the transaction phase). The threads throttle the rate at which they generate requests, making it possible to directly control the load against the database. In addition, the threads measure latency, as well as the achieved throughput of their operations, and then report these measurements to the statistics module.

3.1 Workloads

The performance of each database was evaluated under the following workloads.

1. **Workload A.** Update heavily: 50% read and 50% update, request distribution is Zipfian.
2. **Workload E.** Scan short ranges: 95% scan and 5% update, request distribution is Uniform.
3. **Pagination Workload.** Filter with offset and limit.
4. **JOIN Workload.** JOIN operations with grouping and aggregation. (In the case of Couchbase, ANSI JOIN was evaluated, as well.)

3.2 Tools

The YCSB client was used as a worker, consisting of the following components:

- a workload executor
- YCSB client threads
- extensions
- a statistics module
- database connectors

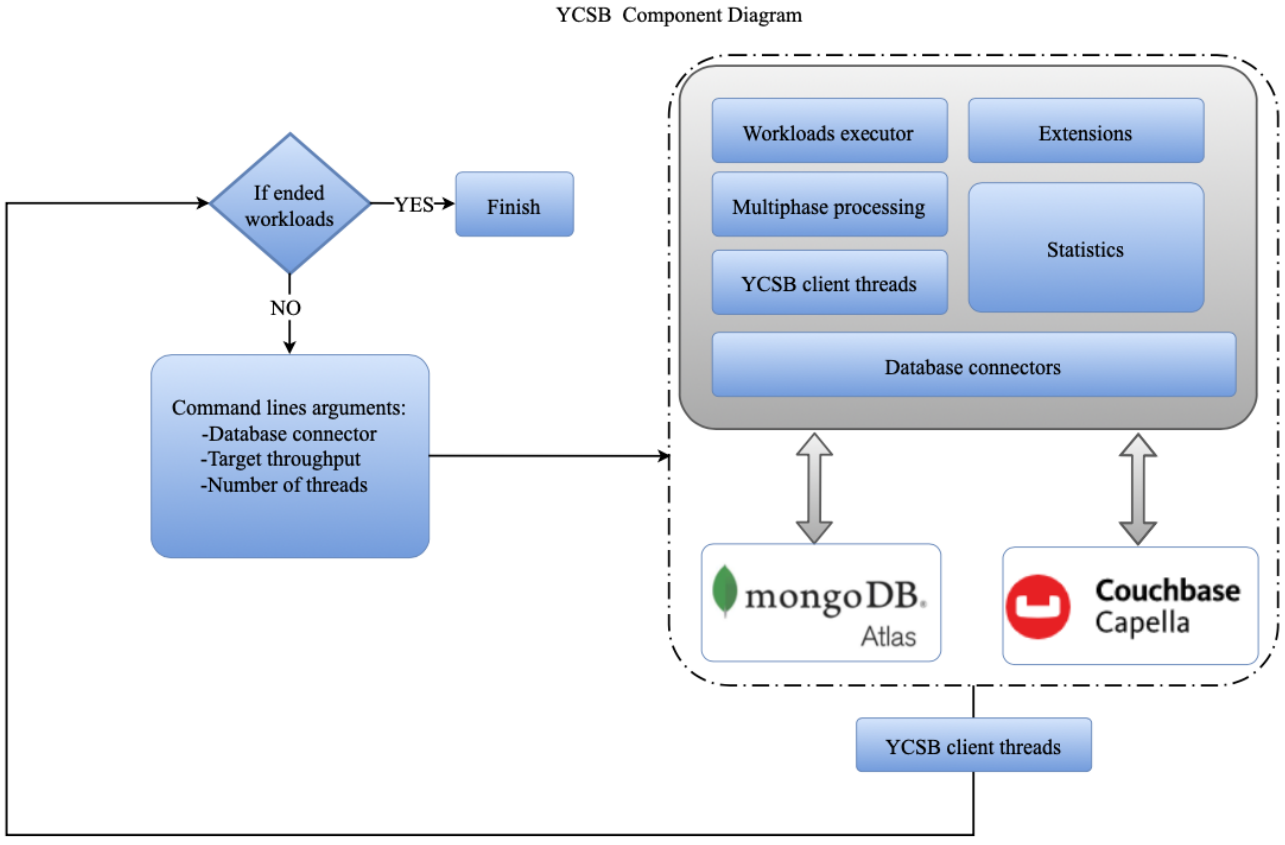


Figure 3.2.1 The components of the YCSB client

The workloads were tested under the following conditions:

- Data fits memory.
- Durability is false.
- Replication is set to “1,” signifying that just a single replica is available for each data set.

Workloads A and E are standard workloads provided by YCSB. Default data models were used for these workloads. Pagination Workload and JOIN Workload represent scenarios from real-life domains: finance (server-side pagination for listing filtered transactions) and e-commerce (a series of reports on various products and services utilized by customers).

To emulate these scenarios on a domain level, a customer order model was introduced for these workloads.

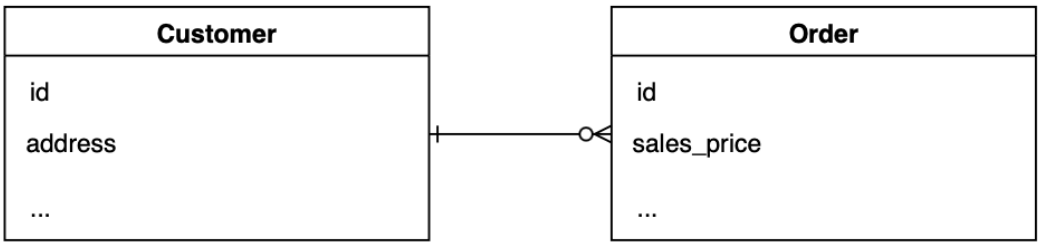


Figure 3.2.2 A graphic representation of the customer order model

4. YCSB Benchmark Results

4.1 Workload A: The update-heavy mode

4.1.1 Workload definition and model details

Workload A is an update-heavy workload that simulates typical actions of an e-commerce application. This is a basic key-value workload. The scenario was executed with the following settings:

- The read/update ratio was 50%–50%.
- The Zipfian request distribution was used.
- The size of a data set was scaled in accordance with the cluster size: 50 million records (each 1 KB in size, consisting of 10 fields and a key) on a 6-node cluster, 100 million records on a 9-node cluster, and 200 million records on a 18-node cluster.

Couchbase Capella stores data in buckets and collections, which are the logical groups of items—key-value pairs. vBuckets are physical partitions of the bucket data. By default, Capella automatically creates a number of master vBuckets per bucket to store bucket data and evenly distribute vBuckets across all cluster nodes.

Querying with document keys is the most efficient method, as a query request is sent directly to a proper vBucket holding target documents. This approach does not require any index creation and is the fastest way to retrieve a document due to the key-value storage.

4.1.2 Query

The following queries were used to perform Workload A.

Table 4.1.2 Evaluated queries

Query name	Couchbase SQL++	MongoDB Query
READ	`collection.get(id, getOptions().timeout(kvTimeout))`	`db.ycsb.find({_id: $1})`
UPDATE	`collection.upsert(id, content, upsertOptions().timeout(kvTimeout).expiry(documentExpiry).durability(persistTo, replicateTo))`	`db.ycsb.update( { _id: $1 }, { $set: { fieldN: $2 } })`

4.1.3 Evaluation results

On a 6-node cluster, Couchbase Capella outperformed MongoDB Atlas. MongoDB Atlas produced 31,030 ops/sec—meanwhile, Couchbase Capella had a throughput of 355,520 ops/sec.

Couchbase Capella significantly outperformed MongoDB Atlas on 9-node and 18-node clusters, as well. The database was able to process up to 417,980 ops/sec on a 9-node cluster and 523,020 ops/sec on an 18-node cluster. MongoDB Atlas had the highest throughput on an 18-node cluster with 55,870 ops/sec. This was an improvement over the 38,340 ops/sec throughput on a 9-node cluster.

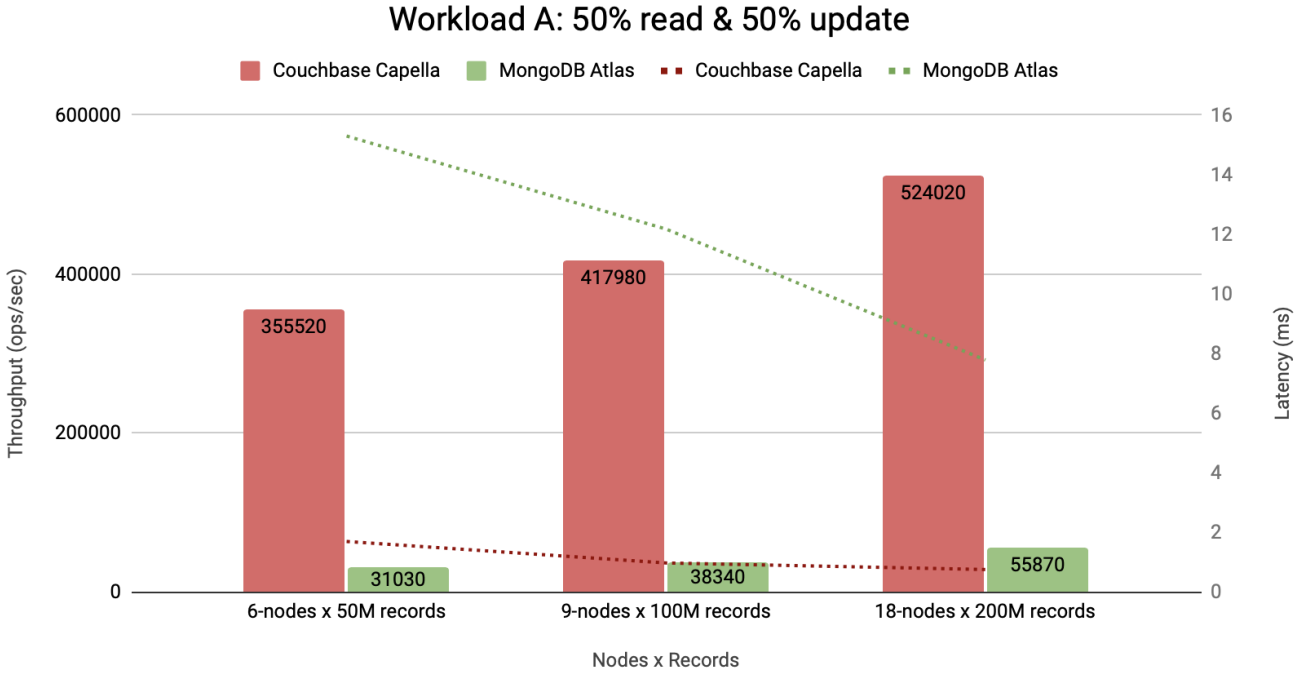


Figure 4.1.3 Performance results under Workload A on 6-, 9-, and 18-node clusters

4.1.4 Summary

The throughput of each database grew constantly, depending on the type of a cluster. Both databases achieved the throughput limit for each cluster type. Couchbase Capella showed high throughput growth and clearly outperformed MongoDB Atlas on each type of cluster.

Couchbase Capella stood out during each test with a latency of a millisecond—1.5 ms on a 6-node cluster and less than 1 ms on 9-node and 18-node clusters. The latency of MongoDB Atlas was halved from 15 ms on a 6-node cluster to 7 ms on an 18-nodes cluster. MongoDB Atlas and Couchbase Capella showed stable results without failed operations. Though MongoDB Atlas did not perform as well as Couchbase Capella, the results were predictable.

4.2 Workload E: Scanning short ranges

4.2.1 Workload definition and model details

Workload E is a short-range scan workload in which short ranges of records are queried instead of individual ones. This workload simulates threaded conversations, where each scan goes through the posts in a given thread (assuming the entries are clustered by ID). The scenario was executed under the following settings.

- The scan/update ratio was 95%–5%.
- The Zipfian request distribution was used.
- The size of a data set was scaled in accordance with the cluster size: 50 million records (each 1 KB in size, consisting of 10 fields and a key) on a 6-node cluster, 100 million records on a 9-node cluster, and 200 million records on a 18-node cluster.
- The maximum scan length reached 100 records.
- Uniform was used as a scan length distribution.

MongoDB Atlas distributes data using a shard key. There are two types of shard keys supported by this database: range- and hash-based. The range-based partitioning supports more efficient range queries. Given a range query on a shard key, a query router can easily determine which chunks overlap this range and route the query to only those shards that contain such chunks. However, the range-based partitioning can result in an uneven data distribution, which may negate some of the benefits of sharding.

The hash-based partitioning ensures an even distribution of data at the expense of efficient range queries. Hashed key-value results in random distribution of data across chunks and, therefore, shards. However, random distribution makes it more likely that a range query on a shard key will not be able to target a few shards, but would more likely query every shard in order to return a result. The hash-based partitioning was used for all partitioning, so some performance degradation is expected here.

4.2.2 Query

The following queries were used to evaluate Workload E.

Table 4.2.2 Evaluated queries

Query name	Couchbase SQL++	MongoDB Query
SCAN	`SELECT meta().id FROM `bucket` WHERE meta().id >= $1 ORDER BY meta().id LIMIT $2`	`db.ycsb.find({ _id: { $gte: $1 }, { _id: 1 }).sort({ _id: 1 }).limit($2)`
UPDATE	`collection.replace(id, content, replaceOptions().timeout(kvTimeout).expiry(documentExpiry).durability(persistTo, replicateTo))`	`db.ycsb.update( { _id: $1 }, { $set: { fieldN: $2 } })`

4.2.3 Evaluation results

On 6-node clusters, Couchbase Capella had a throughput of 30,388 ops/sec, while MongoDB Atlas had the lower rate of 20,532 ops/sec. MongoDB Atlas managed nearly the same amount of operations on 9-node clusters as on 6-node clusters with 21,220 ops/sec, while Couchbase Capella rose to 53,092 ops/sec. Furthermore, Couchbase Capella continued to increase the throughput up to 98,808 ops/sec on 18-node clusters—unlike MongoDB Atlas, which had similar results on all cluster sizes.

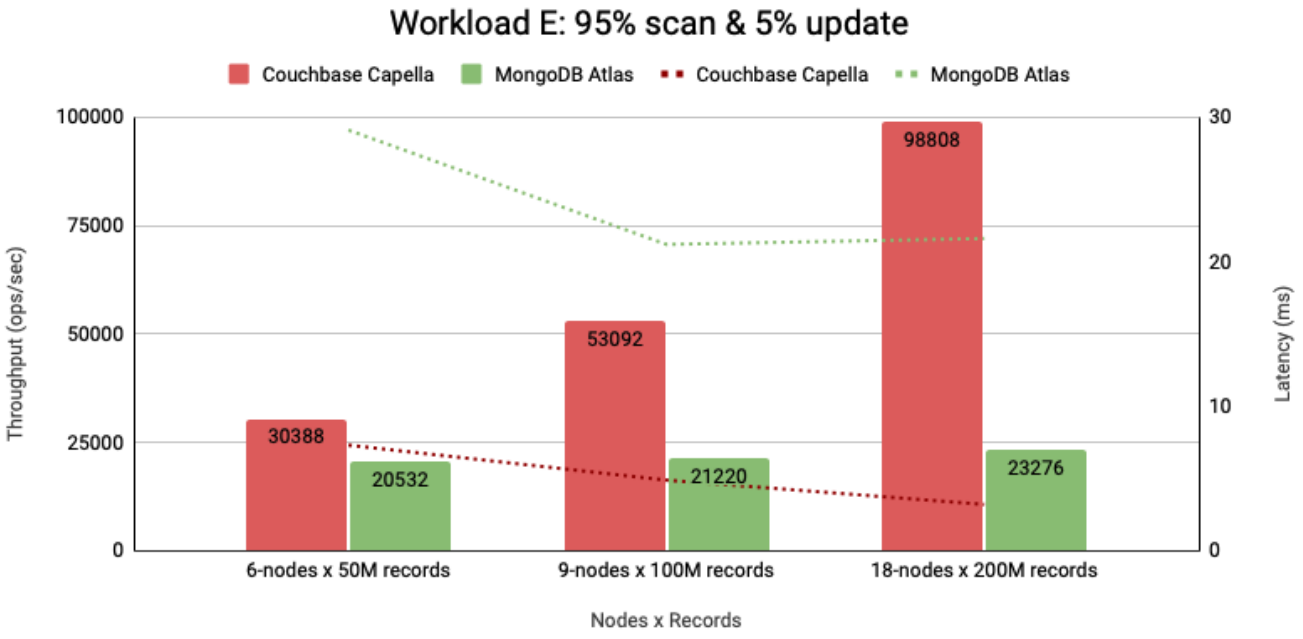


Figure 4.2.3 Performance results under Workload E on 6-, 9-, and 18-node clusters

4.2.4 Summary

Under *Workload E*, Couchbase Capella demonstrated the best results, increasing in throughput as the number of nodes grew. Latency also decreased with more nodes from 7 ms on a 6-node cluster to 3.2 ms on an 18-node cluster, which is the lowest and most stable. For MongoDB Atlas, increasing the amount of shards positively affected performance slightly as scan operations became more efficient with additional shards. On the other hand, the results seem to be close to the throughput limit, since increasing count of records per shard leads to lower amount of operations and higher latency. MongoDB Atlas demonstrated the lowest latency on 18 nodes at 21 ms, and it decreased from 29 ms on 6-node clusters.

4.3 Pagination Workload: Filter with OFFSET and LIMIT

4.3.1 Workload definition and model details

Pagination Workload is a query with a single filtering option, an offset, and a limit. The workload simulates a selection by field with pagination. This scenario is used for listings, such as e-commerce category pages or search engine results. The scenario was executed under the following settings.

- The read ratio was 100%.
- The size of a data set was scaled in accordance with the cluster size: 5 million customers (each 4 KB in size) on a 6-node cluster, 25 million customers on a 9-node cluster, and 100 million customers on an 18-node cluster.
- The maximum of a query length reached 100 records.
- Uniform was used as a query length distribution.
- The maximum query offset reached 5 records.
- Uniform was used as a query offset distribution.

The primary index of Couchbase Capella allows for querying any field of a document. However, this type of querying is rather slow, since it retrieves all the documents of all types in the bucket, whether or not a query eventually returns them to the user. For the sake of fast query execution, secondary indexes are created for specific fields by which data is filtered. Couchbase Capella provides two index storage modes: memory- and disk-optimized. The latter is the default mode.

Memory-optimized indexes use an in-memory database with a lock-free skip list, which has a probabilistic ordered data structure and, thus, performs at in-memory speeds. The search is similar to a binary search over linked lists with the `O(log n)` complexity. The lock-free skip list is used to provide nonblocking reads/writes and maximize utilization of the CPU cores. On top of a lock-free skip list, there is a multiversion manager responsible for regular snapshotting in the background. Memory-optimized indexes reside in memory and require the amount of RAM available to fit all the data inside of it. The indexes on a given node will stop processing further mutations, if a node runs out of index RAM quota. The index maintenance is paused until sufficient memory becomes available on the node. Since the data set was required to fit the available memory, memory-optimized indexes fit the requirements well.

Memory-optimized global secondary indexes were created for filtering fields with index replication on each cluster node.

```
CREATE INDEX `query1` ON `bucket`(`address`.`country`) USING GSI;
```

MongoDB Atlas uses mongos instances to route queries and operations to shards in a sharded cluster. If the result of the query is not sorted, the mongos instance opens a result cursor from all cursors on the shards using a round robin method. If a query limits the size of the result set using the `limit() cursor` method, the mongos instance passes that limit to the shards and then reapplies the limit to the result before returning it to the client. If a query specifies a number of the records to skip using the `skip() cursor` method, the mongos cannot pass the skip to the shards. Instead, the mongos instance retrieves unskipped results from the shards and skips the appropriate number of documents when assembling the complete result. However, when used in conjunction with `limit()`, the mongos instance will pass the limit plus the value of `skip()` to the shards to improve the efficiency of these operations. For better performance, an additional secondary index was added to a filtered field as shown below.

```
db.customer.ensureIndex( { "address.country": 1 } );
```

4.3.2 Query

The following queries were used to perform Pagination Workload.

***Table 4.3.2** Evaluated queries*

Couchbase SQL++	MongoDB Query

SELECT meta().id
FROM `bucket`
WHERE address.country='$1'
OFFSET $2
LIMIT $3
	db.customer.find({
	address.country: $1
	}, {
	_id: 1
	})
	.skip($2)
	.limit($3)

4.3.3 Evaluation results

On a 6-node cluster, MongoDB Atlas had the lower throughput of 36,392 ops/sec, while Couchbase Capella had 51,952 ops/sec. MongoDB Atlas slightly decreased throughput on a 9-node cluster with a rate of 35,325 ops/sec. This happened due to the amount of records per shard. In comparison, Couchbase Capella increased throughput to 60,680 ops/sec. Couchbase Capella performed best on 18-node clusters with 65,228 ops/sec, while MongoDB Atlas managed a throughput of 29,300 ops/sec on the same cluster size.

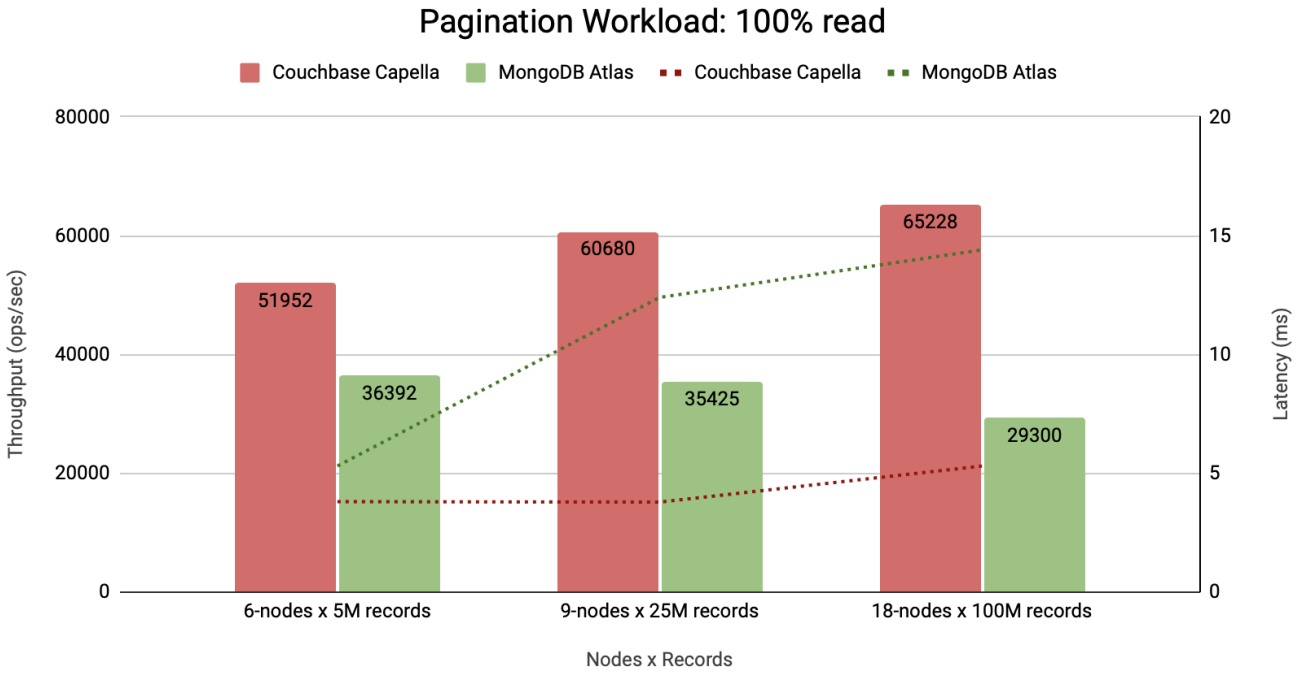


Figure 4.3.3 Performance results under Pagination Workload on 6-, 9-, and 18-node clusters

4.3.4 Summary

Couchbase Capella demonstrated good results for *Pagination Workload*, gradually increasing in throughput as the number of nodes grew. However, latency rose from 3 ms on a 6-node to 6 ms on an 18-node cluster. MongoDB Atlas had the inverse result, performing progressively worse as the cluster size increased. This is due to the size of the data set and indexes being doubled, while the cluster size increased by only 50%. The latency for MongoDB Atlas also got gradually worse from 11 ms on a 6-node cluster, to 12.4 ms on a 9-node cluster, and finally 14.4 ms on an 18-node cluster.

4.4 JOIN Workload: JOIN operations with grouping and aggregation

4.4.1 Workload definition and model details

JOIN Workload is a `JOIN` query with grouping and ordering applied. The workload simulates a selection of complex child/parent relationships with categorization. The scenario was executed under the following settings.

- The read ratio was 100%.
- The size of a data set was scaled in accordance with the cluster size: 25 million customers and 25 million orders (each 4.5 KB in size) on a 6-node cluster, 25 million customers and 25 million orders on a 9-node cluster, and 100 million customers and 100 million orders on a 18-node cluster.
- The maximum of a query length reached 100 records.
- Uniform distribution was used for a query length and query offset selection.
- The maximum of a query offset reached 5 records.

There are different types of JOIN operations available the SQL++ query engine in Couchbase out of the box:

- Index JOIN is used when one of the two JSON documents represents a document key(s) employing the ON KEYS statement.
- ANSI JOIN is applicable to arbitrary expressions on any field in a document, standard JOIN statement, with a nested loop under the hood. SQL++ supports the standard INNER, LEFT OUTER, and RIGHT OUTER JOINs.
- ANSI HASH JOIN creates an in-memory hash for one of the tables in the JOIN query (usually, the smaller one) used by the other table to find matches. Performance can be optimized under suitable conditions.

Only the first two types—Index JOIN and ANSI JOIN—were evaluated under this benchmark. In addition, a dedicated covering index was used, as it contained all the fields required by the query. This way, a query engine skips the whole document retrieval from data nodes after the index selection is made. Therefore, the query execution plan consists of only index resolutions without time-consuming document retrieval over the network, which results in a significant query performance boost.

The following secondary index was created for Couchbase:

```
CREATE INDEX `query2` ON `bucket`(address.zip, month, order_list,
sale_price) USING GSI;
```

MongoDB ensures the $lookup aggregation out of the box to apply a left outer JOIN over an unsharded collection in the same database. It helps to filter document keys from the "joined" collection for further processing. Unfortunately, MongoDB v5.0 did not support the $lookup aggregation on sharded collections when the evaluation was carried out. So, in order to evaluate *JOIN Workload*, an alternative solution was employed. One way to work with JOIN operations on a nonrelational database is to denormalize a data model, embed the elements into the parent objects, and perform a regular query. Still, this approach invokes additional redundancy and extra storage costs, as well as impacts the read/write performance.

Another way is to model the dedicated "joining table" and query its elements by a partition key, which generally becomes identical to *read by key*. This approach leads to data duplication and an increase in write complexity through the necessity to support consistency between models, which also causes a significant write performance downgrade. Furthermore, the approach brings along additional storage costs. The same specific data modeling approach can be applied to all the databases under evaluation, but it leads to dramatically varying results. Due to this, we considered a similar business case with two different models available: *customers* and *orders*. In this case, a JOIN operation was a simple two-phase read with filtering, which had a significant impact on the overall JOIN operation performance.

4.4.2 Query

The following queries were used to perform JOIN Workload.

***Table 4.4.2** Evaluated queries*

Couchbase SQL++	MongoDB Query

SELECT o2.month,
c2.address.zip,
SUM(o2.sale_price) as
sale_price
FROM `bucket` c2
INNER JOIN `bucket` o2
ON (meta(o2).id IN
c2.order_list)
WHERE c2.address.zip = $1
AND o2.month = $2
GROUP BY o2.month,
c2.address.zip
ORDER BY SUM(o2.sale_price)
	$r1 = db.customer.find({
	address.zip: $1
	}, {
	address.zip: 1,
	order_list: 1
	})
	
	$r2 = db.order.aggregate([
	{
	$match: {
	$and: [{
	_id: {
	$in: $r1.order_list
	}
	}, {
	month: $2
	}]
	}}, {
	$group: {
	_id: null,
	sum: {
	$sum: "$sale_price"
	}
	}}])

4.4.3 Evaluation results

On a 6-node cluster, Couchbase Capella reached a throughput of 34,448 ops/sec, while MongoDB had 884 ops/sec. The highest throughput of Couchbase Capella demonstrated on an 18-node cluster was 56,828 ops/sec. The performance of MongoDB Atlas decreased on a 9-node cluster and continued to fall on an 18-node cluster with a throughput of 60 ops/sec.

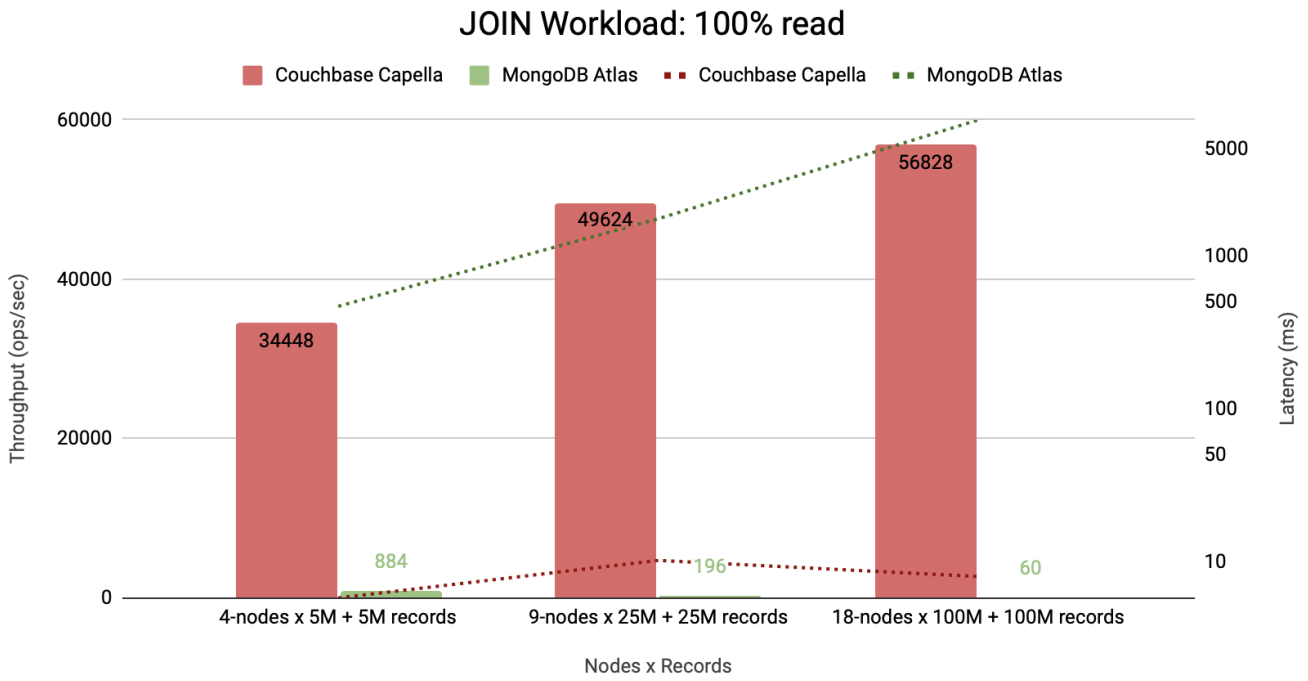


Figure 4.4.3 Performance results under JOIN Workload on 6-, 9-, and 18-node clusters

4.4.4 Summary

Pagination Workload can be processed by searching an index in the `O(log n)` time, while JOIN Workload is much more expensive consisting of multiple steps: index search, values lookup of the other index for `JOIN` matching, and then sort-based aggregation using `ORDER BY`. This results in at least `O((n*m) log(n*m))` complexity and, thereafter, shows lower numbers compared to Pagination Workload.

Couchbase Capella outperformed MongoDB Atlas in JOIN Workload by demonstrating higher throughput and lower latencies on all cluster sizes. Couchbase Capella had a consistent latency of about 9 ms across cluster sizes, while MongoDB Atlas had about 10,400 ms (above 10 seconds) on an 18-node cluster.

5. Conclusion

No single database as a service is perfect for meeting all the requirements of any given scenario. Each solution has its advantages and disadvantages that become more or less important depending on the specific criteria to meet. Despite this, DBaaS helps engineers to reduce the time for deployment, configuration, and support. Though DBaaS does not offer broad system tools for configurations, the databases have been optimally tuned for each workload. Therefore, configurations can be adjusted based on workloads.

Couchbase Capella showed better performance across all four evaluated workloads in comparison to MongoDB Atlas for all three cluster sizes. In the case of queries, Couchbase Capella provides flexible functionality to handle the deployed workloads. Furthermore, the query engine of Couchbase Capella supports aggregation, filtering, and `JOIN` operations on large data sets without the need to model data for each specific query. As clusters and data

sets grow in size, Couchbase Capella ensures a high level of scalability across these operations.

MongoDB Atlas produced comparatively decent results. The database is scalable enough to handle increasing amounts of data and cluster extension. Under this benchmark, the only issue we observed was that MongoDB Atlas did not support `JOIN` operations on sharded collections out of the box. This resulted in poor performance in JOIN Workload.

6. Appendix: Additional Queries and Indexes

6.1 MongoDB Atlas shard collection and indexes

MongoDB Atlas shard collection

```
sh.shardCollection( "ycsb.usertable", { _id: 1 }, false )
```

MongoDB Atlas index

```
db.usertable.ensureIndex({_id: "hashed"})
```

6.2. Indexes for the SCAN query

Couchbase indexes

```
CREATE PRIMARY INDEX on `bucket` WITH {"num_replica":
NUMBER_OF_INDEX_NODES - 1}
```

6.3. Indexes for Pagination Workload

Couchbase indexes

```
CREATE PRIMARY INDEX ON `bucket`;
CREATE INDEX `query1` ON `bucket`(`address`.`country`);
```

MongoDB Atlas indexes

```
db.customer.ensureIndex( { "address.country": 1 } );
```

6.4. Indexes index for JOIN Workload

Couchbase indexes

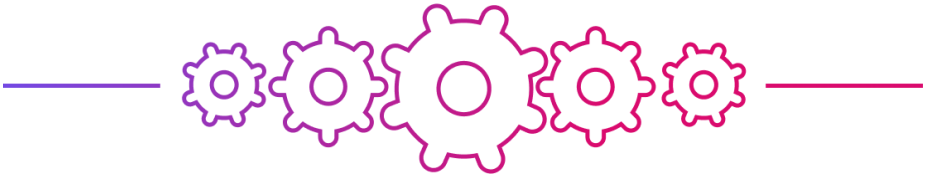
```
CREATE INDEX `query2` ON `bucket`(address.zip, month, order_list,
sale_price)
```

MongoDB Atlas indexes

```
db.customer.ensureIndex( { "address.zip": 1 } );
db.order.ensureIndex( { "month": 1 } );
```

7. About the Author

Ivan Shyrma is Data Engineer at Altoros with extensive hands-on experience in high-load, scalable applications and web services development. Ivan has worked as a full-stack engineer for several years designing durable distributed systems. He is able to create complex architecture solutions, adopt systems for production use, and is keen on resolving any engineering problems.



Altoros is an experienced IT services provider that helps enterprises to increase operational efficiency and accelerate the delivery of innovative products by shortening time to market. Relying on the power of cloud automation, microservices, AI/ML, and industry knowledge, our customers are able to get a sustainable competitive advantage. For more, please visit www.altoros.com.

Want more?

To download other research papers and articles like that:

- check out our resources page
- subscribe to the blog
- or follow @altoros for daily updates

Feel free to contact us if you'd like to discuss your project.