United Software Associates
Best Minds Together®

HIGH PERFORMANCE BENCHMARKING: MongoDB and NoSQL Systems

Contents

Overview

In recent years alternatives to relational databases have emerged that provide advantages in terms of performance, scalability, and suitability for cloud environments. While they vary significantly in terms of capabilities, many in the industry have adopted the term “NoSQL” to describe these products. In this paper we evaluate the performance of the three leading products – Cassandra, Couchbase, and MongoDB – using an industry standard benchmark created by Yahoo! called YCSB.

Selecting the appropriate database technology for a project involves careful consideration of many different criteria. While performance is important, it must be considered along with functionality, operational tooling, ease of use, availability of skills, technology ecosystem, security controls, reliability, and other factors. Our goal in this paper is to take a closer look at one of these criteria – performance – and we encourage readers to use our findings as one of many inputs to their own evaluations.

An important consideration in our evaluations is data durability. For most databases, performance is limited by the speed of the storage subsystem either in terms of throughput, latency, or both. The three databases we evaluate each implement different approaches to durability, allowing users to optimize performance, usually at the cost of durability. For each system, when optimized for throughput, there exists a window of time (usually measured in seconds) in which writes are committed to memory but not yet committed to non-volatile storage. During this window, power loss or a server crash would result in data loss. As these systems are capable of processing in excess of 100,000 writes per second, tens of seconds of data loss represents millions of records.

While in some cases data loss may be acceptable, we believe most applications will prioritize durability over performance, and will be unwilling to tolerate data loss. We wanted to understand the performance of these systems with different levels of durability. In our tests we evaluated three configurations to understanding these tradeoffs.

Methodology

The three technologies we evaluate in this report provide very different feature sets. Comparisons are difficult, but each product provides a core set of capabilities that can be used as the basis of a performance evaluation.

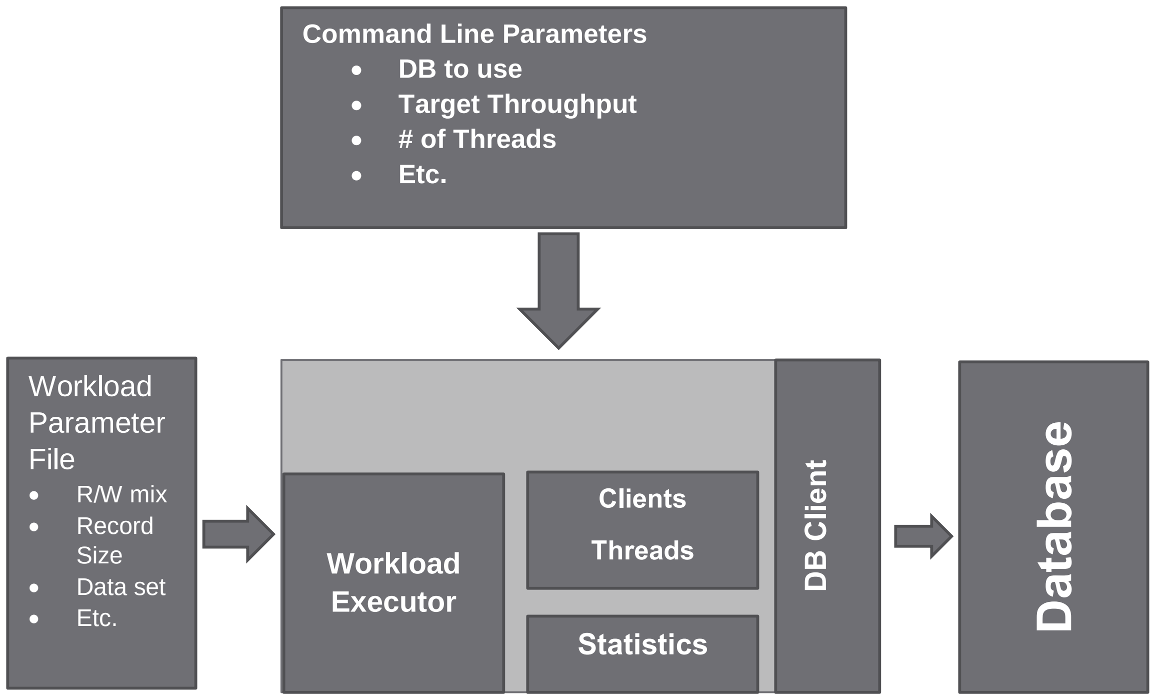
In 2010, Yahoo! published a benchmark called the Yahoo! Cloud Serving Benchmark (YCSB). This benchmark tests CRUD operations. Many organizations have used YCSB to evaluate databases and different hardware environments. It has become popular and well understood.

The vendors associated with the three databases in this report have developed forks of YCSB that provide optimal use of their products. We have incorporated their forks into our tests with the latest version of each database to ensure that the best possible performance for each product was observed, and to prevent the fork of any single vendor from disadvantaging the other products.

YCSB is a framework and common set of workloads for evaluating the performance of different databases. YCSB consists of:

- The Client – an extensible workload generator
- The Core Workloads – a set of workload scenarios to be executed by the generator

The following diagram illustrates how YCSB works with a database:



We used YCSB as the basis for all our tests. We used the same number of records, record size, number of fields, and number of operations in all of our tests to provide a common baseline. For each of the tests, we performed multiple independent runs, increasing the number of threads until the maximum throughput was reached without sacrificing latency. For measuring results, we recorded throughput, and latencies for reads and updates (average, 95th percentile, and 99th percentile).

Our setup consisted of one database server and one client server to ensure the YCSB client was not competing with the database for resources. Both servers were identical.

To ensure the integrity of all tests, we removed the data files and restarted the database between each run. We performed separate runs for each of the three configurations we tested, each of the YCSB workloads (Workload A, Workload B), and each of the thread counts we tested.
Each run was performed as follows:

- Start with an empty database instance (data files wiped, server restarted)
- Load 20M records using the “load” phase of YCSB
- Allow the database to stabilize after the load
- Run workload
- Record results

Durability Settings for the Databases

All three databases provide flexibility in terms of configuring the balance of durability and throughput that is best for a given application. In the following table we describe the three configurations we tested:

In these tests we define durability as the write having been successfully written to persistent storage (disk). Like many databases, both Cassandra and MongoDB use a write ahead log (WAL) to record operations against the database.

- From the Cassandra documentation: *Like other modern systems, Cassandra provides durability by appending writes to a commitlog first. This means that only the commitlog needs to be fsync'd, which, if the commitlog is on its own volume, obviates the need for seeking since the commitlog is append-only.*
- From the MongoDB documentation: *MongoDB stores and applies write operations in memory and in the on-disk journal before the changes are present in the data files on disk. Writes to the journal are atomic, ensuring the consistency of the on-disk journal files.*

In both systems, writes are first committed to the WAL in memory, and then the WAL is written to disk to ensure durability. This approach allows the database to optimize writing the data files to disk, while still ensuring durability. The window of possible data loss is a function of how frequently the WAL is flushed to disk.

Worst Case Data Loss in Case of Server Crash			
	Cassandra	**Couchbase**	**MongoDB**
Throughput Optimized (no WAL; write acknowledged after written to RAM, written to disk asynchronously)	-Lose everything written since Memtables last persisted	-Lose everything in the disk write queue up to the size of your RAM	-Lose everything written since the last successful checkpoint
Durability Optimized (waits for WAL or data to be written to disk)	-Waits for commitlog flush -No possible data loss	-Waits for persist to master -No possible data loss	-Flushes journal to disk -No possible data loss
Balanced (write acknowledged when written to WAL in RAM, WAL written to disk more frequently than data files)	-Commit log flushed to disk every 10 seconds -Lose up to 10 seconds of data	No equivalent option available	-Journal flushed to disk every 100MB -Lose up to 100MB of data

Couchbase does not use a WAL. Instead, writes are written to RAM and then added to a queue that eventually writes the documents to disk in batches of 250K.

- From the Couchbase documentation: *Since Couchbase Server is an asynchronous system, any mutation operation is committed first to DRAM and then queued to be written to disk. The client is returned an acknowledgment almost immediately so that it can continue working.*

Because the size of the queue is unlimited, the potential data loss is equal to the size of RAM. The mechanism Couchbase provides to ensure the durability of a write is to wait for the write to be written to disk.

All three systems provide the ability to wait for the writes to be written to disk before acknowledging success to the application. With this approach there is no possible data loss.

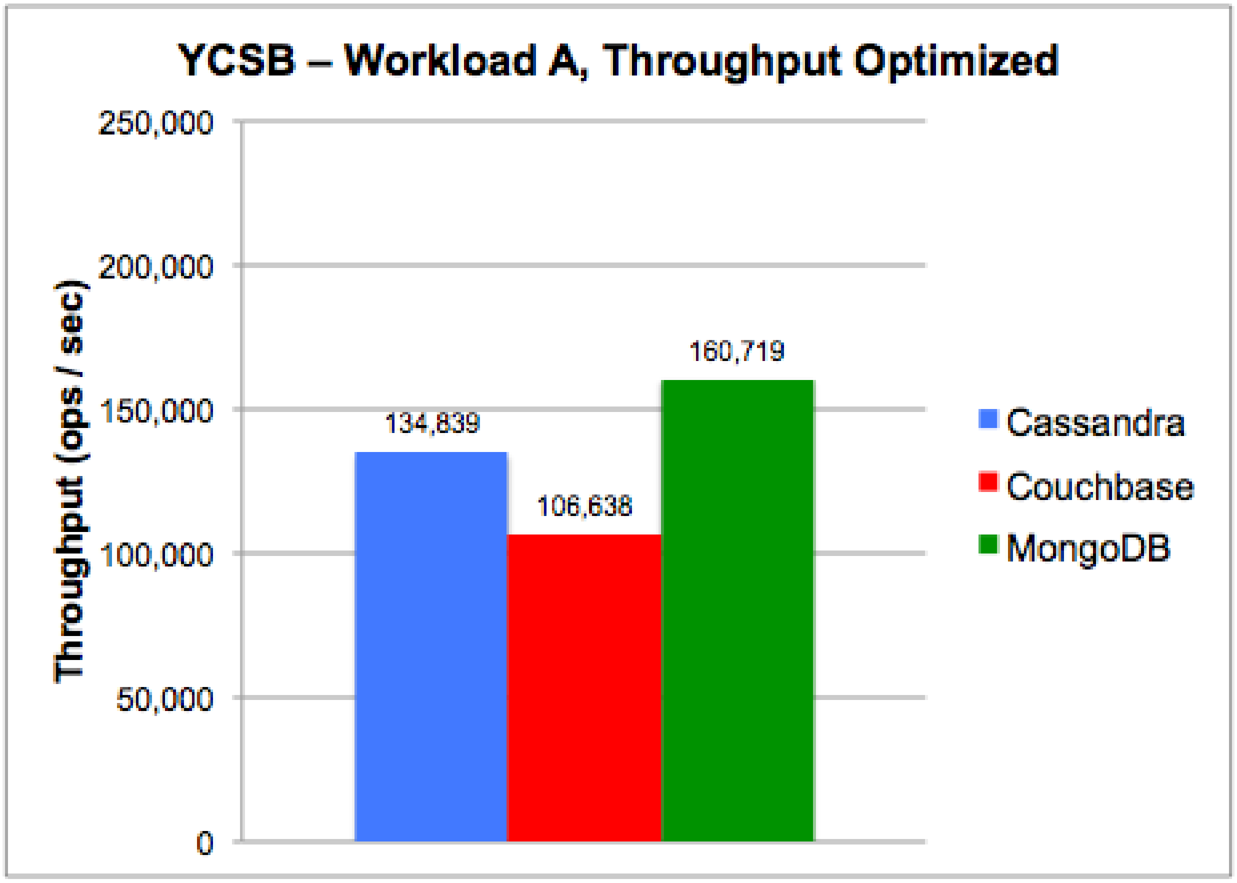
Test Results

For each of the durability configurations we will discuss how the databases performed for the YCSB workloads.

Throughput Optimized

In these tests each system has been optimized for throughput, including no use of a WAL. This is the default configuration of Couchbase. In all three databases, writes are first committed in memory then written to disk asynchronously. The time between the write to memory and the write to disk represents a window of potential data loss in the case of a power failure or server crash. This configuration is suitable for applications where some data loss is acceptable. We believe data loss is acceptable only for a small minority of applications, and these results are not representative for “real world” applications.

Workload A (50% read, 50% update)

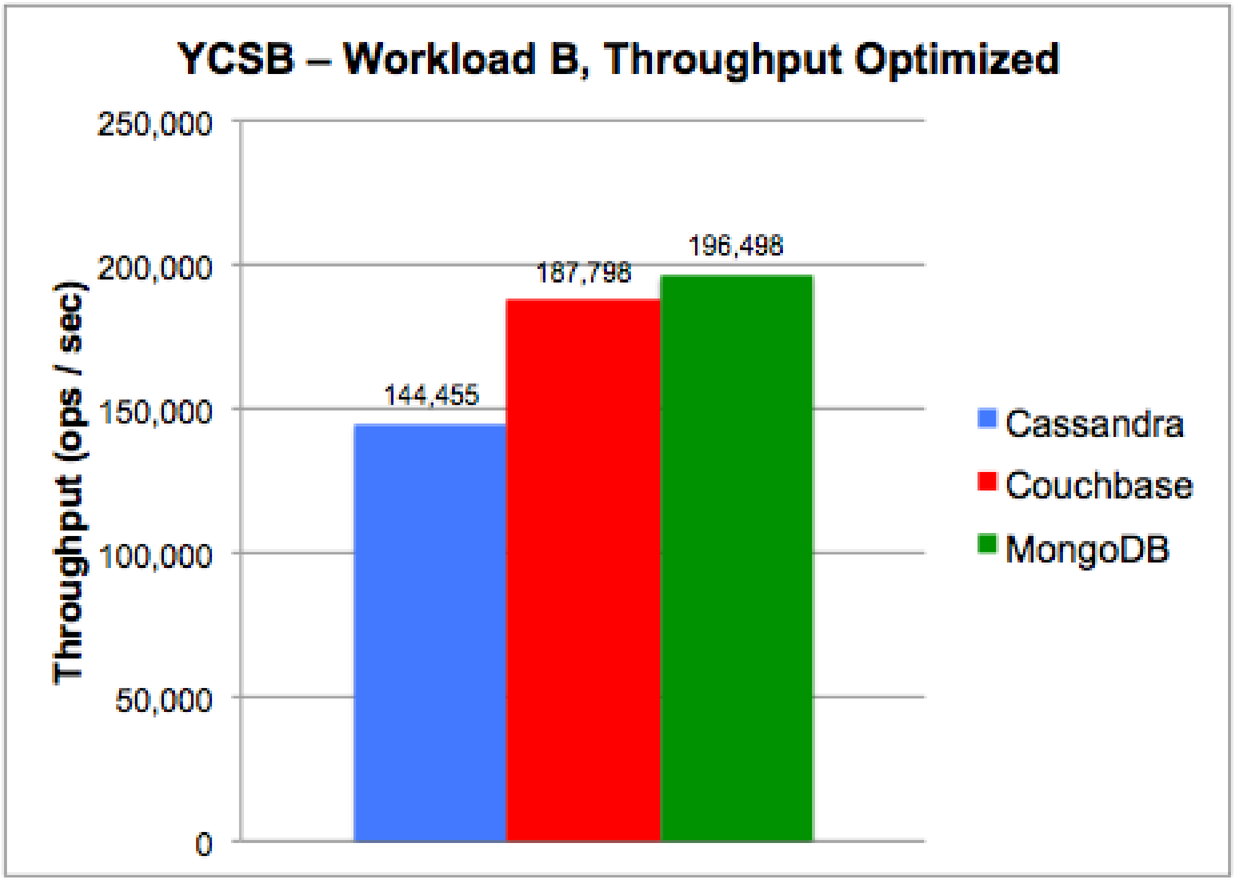


With a configuration optimized for throughput, the 50/50 workload in these tests demonstrates that MongoDB provides about 50% greater throughput than Couchbase, and about 20% greater throughput than Cassandra.

Latency falls within a narrow range for all three databases:

YCSB (Latencies) – Workload A, Throughput Optimized		
	99th (Read)	**99th (Update)**
Cassandra	4ms	3ms
Couchbase	<1ms	3ms
MongoDB	1ms	1ms

Workload B (95% read, 5% update)



With a configuration optimized for throughput, the read-heavy workload (95% reads) shows MongoDB provides about 35% greater throughput than Cassandra, and slightly better throughput than Couchbase.

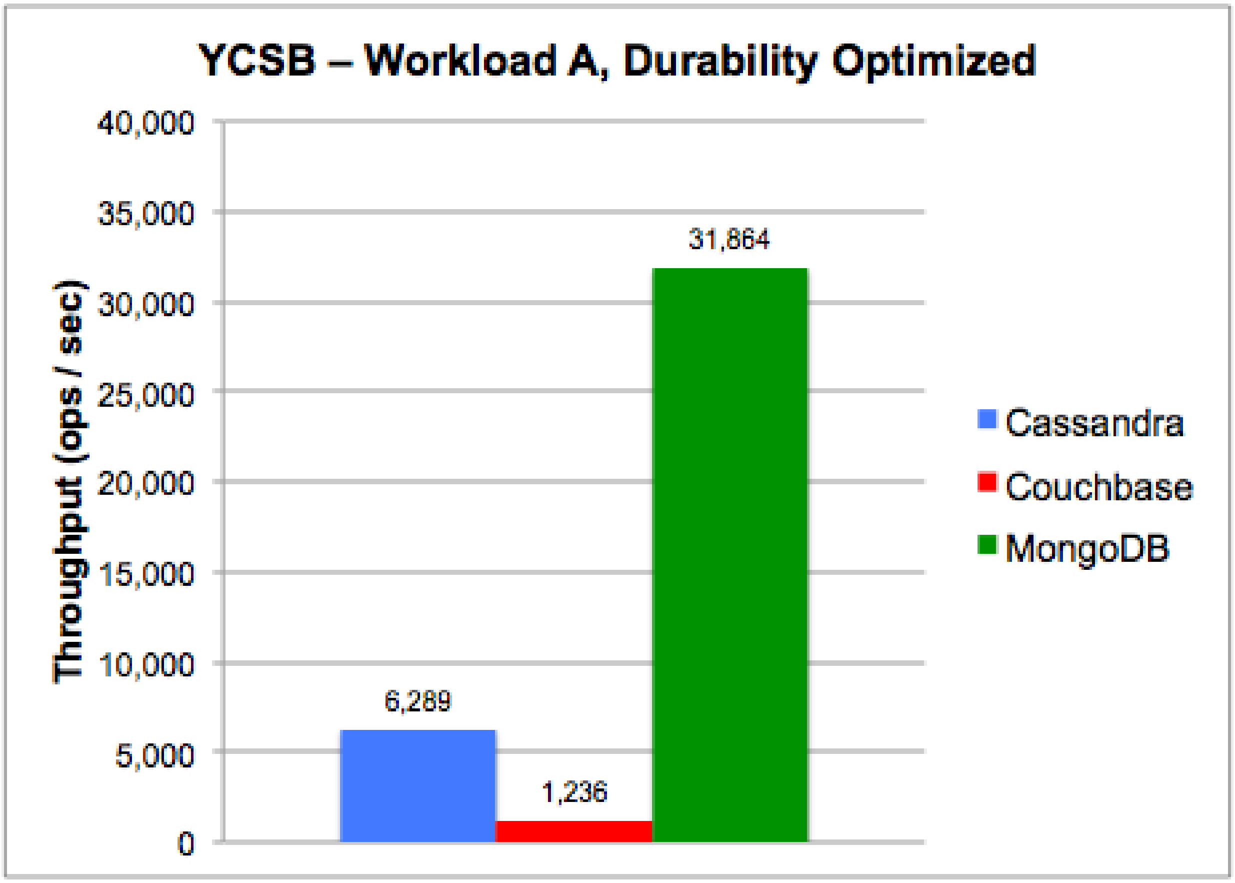
As with the 50/50 workload, latency for the 95th and 99th percentiles falls within a similar narrow range across the databases:

YCSB (Latencies) – Workload B, Throughput Optimized		
	99th (Read)	**99th (Update)**
Cassandra	1ms	1ms
Couchbase	2ms	5ms
MongoDB	1ms	1ms

Durability Optimized

In these tests each database has been optimized for durability. Writes are acknowledged after they are written to disk. There is no possible data loss. This configuration is suitable for applications where durability is more important than performance.

Workload A (50% read, 50% update)

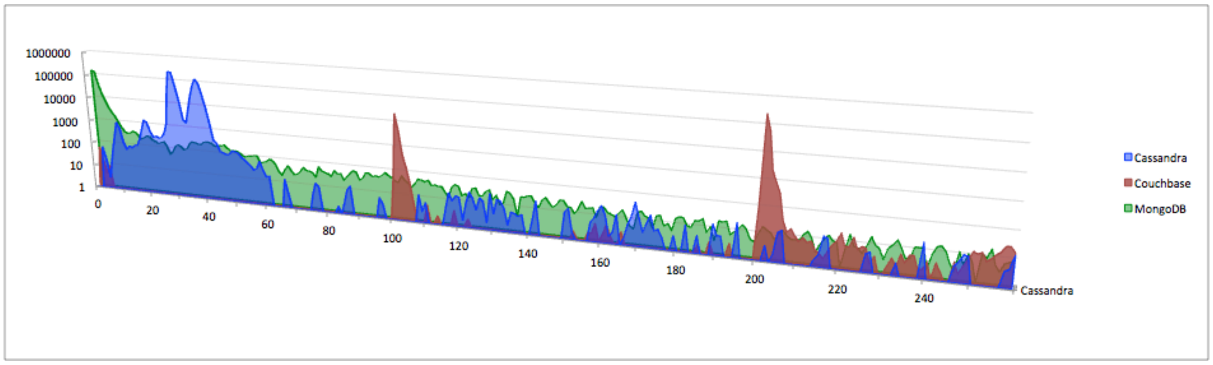


With a configuration optimized for durability, the 50/50 workload in these tests demonstrates that MongoDB provides more than 25x greater throughput than Couchbase, and 5x greater throughput than Cassandra. The lower throughput observed for all systems when compared to the results for the throughput optimized configurations is a result of each write operation waiting for confirmation writes have been persisted to disk.

Latency is relatively low for all three databases for reads, but for updates there are significant differences:

YCSB (Latencies) – Workload A, Durability Optimized		
	99th (Read)	**99th (Update)**
Cassandra	8ms	41ms
Couchbase	<1ms	203ms
MongoDB	1ms	5ms

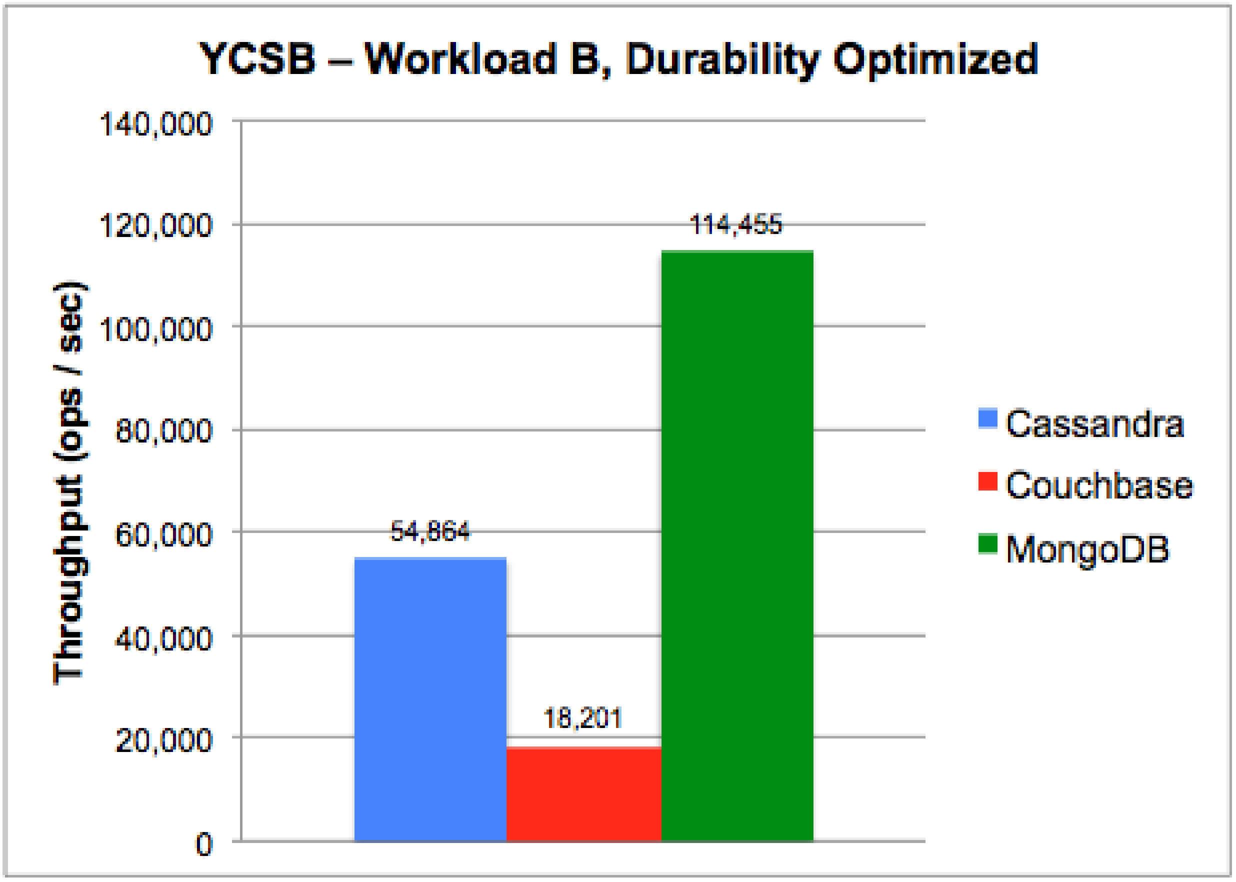
Write latency is considerably higher for Cassandra and Couchbase in this configuration, so let's take a closer look at the histogram of write latencies from the test:



X axis is latency in milliseconds, Y axis is number of updates for each latency (log scale)

Ideally, we would see a distribution with a majority of the updates close to the Y axis, representing 0 latency. A distribution that is more to the right indicates greater latency. We can see that for Cassandra most update operations take between 10ms and 50ms. For Couchbase, results are narrowly distributed around 200ms, and then a much smaller number around 100ms. For MongoDB virtually all of results are under 5ms.

Workload B (95% read, 5% update)



With a configuration optimized for durability, the read heavy workload (95% reads) shows MongoDB providing about 6x greater throughput compared to Couchbase, and about 2x greater throughput compared to Cassandra.

We observed there is much less of a degradation in throughput for MongoDB for Workload B when optimizing for durability – MongoDB provides about 40% less throughput compared to the configuration optimized for throughput. Couchbase provides about one tenth the throughput, and Cassandra provides less than half the throughput when compared to their configurations optimized for throughput.

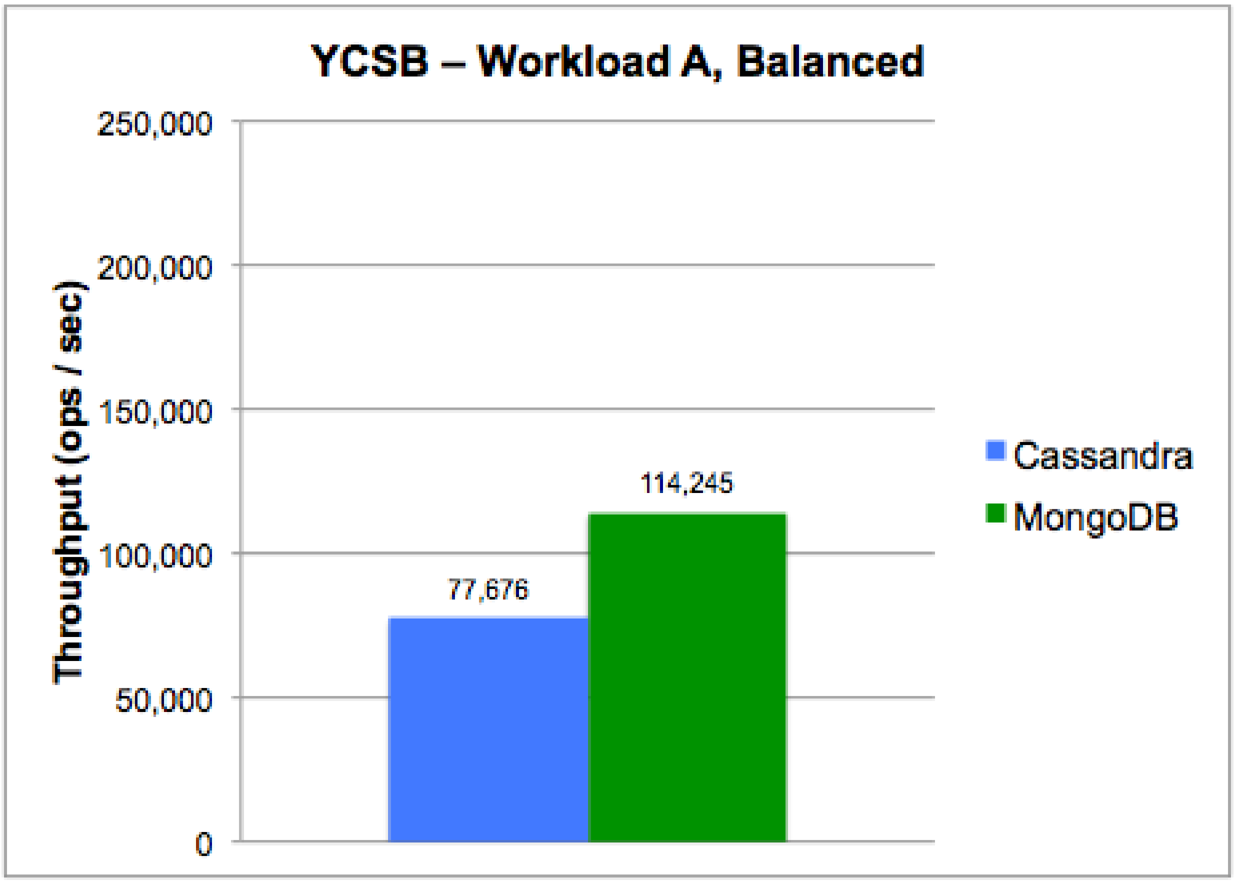
Read and write latencies were similar to those observed in Workload A: read latencies at the 99th percentile were quite low, but write latencies were about 6x higher in Cassandra, and about 20x higher in Couchbase compared to MongoDB.

YCSB (Latencies) – Workload B, Durability Optimized		
	99th (Read)	**99th (Update)**
Cassandra	15ms	66ms
Couchbase	<1ms	238ms
MongoDB	1ms	10ms

Balanced

We believe most applications are best served by a configuration that is balanced – where throughput is good, while potential data loss is minimized. The developers of Cassandra and MongoDB would seem to agree as this is the default configuration of their products. Couchbase, on the other hand, has no equivalent configuration. Users must choose between a configuration that is optimized for throughput, where potential data loss is up to the size of RAM, or durability optimized, where throughput is 99% worse, and where latency is over 200x worse as measured in our tests.

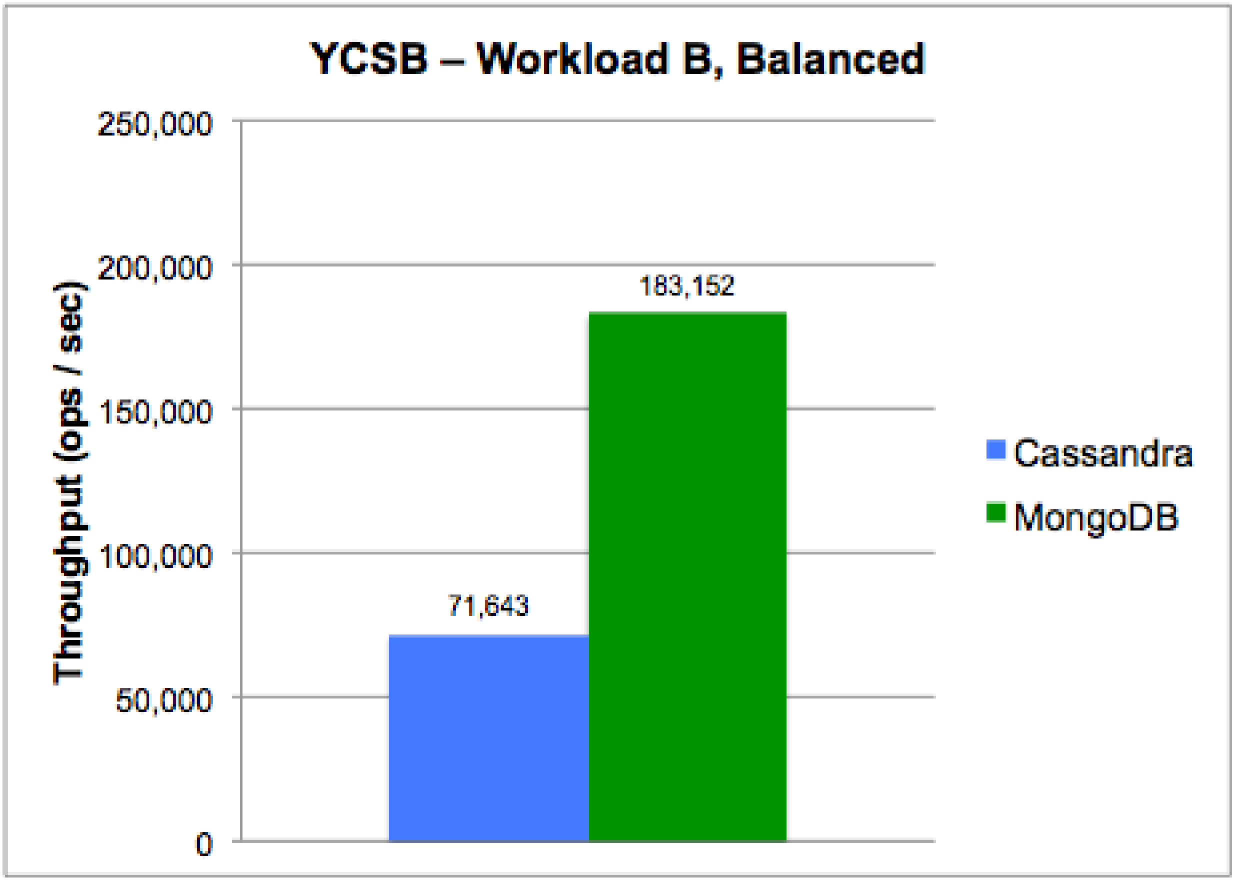
Workload A (50% read, 50% update)



With a configuration balanced for throughput and durability, the 50/50 workload in these tests demonstrates that MongoDB provides almost 50% greater throughput than Cassandra. As noted above, Couchbase does not support an equivalent configuration.

YCSB (Latencies) – Workload A, Balanced		
	99th (Read)	**99th (Update)**
Cassandra	6ms	6ms
MongoDB	<1ms	1ms

Workload B (**95**% read, 5% update)



With a configuration balanced for throughput and durability, the read heavy workload (95% reads) shows MongoDB providing over 2.5x greater throughput than Cassandra. In terms of latency, the results are similar to what we observed in the 50/50 workload.

YCSB (Latencies) – Workload B, Balanced		
	99th (Read)	**99th (Update)**
Cassandra	4ms	4ms
MongoDB	2ms	6ms

Clearly additional durability comes at a cost. Read and update latency is generally higher for databases in the balanced configuration as compared to the throughput optimized configuration. However, we feel that with the balanced configuration latency is sufficiently low for most applications, and the cost of additional durability may be acceptable, depending on the application.

In mixed workloads, MongoDB's throughput is about 30% less than the throughput optimized configuration, and in read-intensive workloads it provides merely 10% less throughput. Cassandra provides about 40% less throughput for mixed workloads, and about 50% less in read intensive workloads when compared to its throughput optimized configuration.

We believe most users will find these tradeoffs worthwhile, and will favor the default configuration over the throughput optimized configurations or the durability optimized configurations.

Conclusions

Based on these tests, we have reached the following conclusions:

- MongoDB provides greater performance than Couchbase or Cassandra in all the tests we ran, in some cases by as much as 25x.
- When optimized for throughput (Couchbase's default), MongoDB outperforms Couchbase by over 50% in mixed workloads, and provides better throughput in read-dominant workloads. MongoDB handles write conflicts in the database. Couchbase documentation instructs application developers to detect and handle conflicts in their application code, requiring additional round trips to retry updates. We believe this accounts for the throughput difference between the two.
- When optimized for a balance of throughput and durability (the default of Cassandra and MongoDB, unavailable in Couchbase), MongoDB provides over 50% greater throughput in mixed workloads, and 2.5x greater throughput in read-dominant workloads compared to Cassandra.
- MongoDB provides the most flexibility for ensuring durability for specific operations: users can opt for the durability optimized configuration for specific operations that are deemed critical but for which the additional latency is acceptable. For Cassandra this change requires editing a server config file and a full restart of the database. For Couchbase, users must choose between high throughput with the potential for significant data loss, or no potential data loss with low throughput and very high latency.
- We focused on single server performance in these tests. Multi-server deployments address high availability and scale out for all three databases. We believe that this introduces a different set of considerations, and that the trade offs may be quite different. We hope to visit this topic in future research.
- Insofar as these tests only evaluate the most basic operations of these databases, users should also consider the other features that are important to their application, and they should develop tests that evaluate those capabilities as well.

Environment

These tests were conducted on bare metal servers. Each server had the following specification:

- CPU: 2 x 3.06GHz Intel Xeon-Westmere (X5675-Hexcore)
- RAM: 96GB: 6x16GB Kingston 16GB DDR3 2Rx4
- Disk: (2) 960GB SanDisk CloudSpeed 1000 SSD
    - drive controller: Adaptec 71605
- OS: Ubuntu 14.10
- Network: 10gigE

The following configurations were made to optimize the environment for each database:

- readahead was reduced to 64 blocks
- transparent huge pages, defrag, numa were disabled

The following YCSB forks are used for each database:

- Cassandra: https://github.com/jbellis/YCSB
- Couchbase: https://github.com/usaindev/YCSB [1]
- MongoDB: https://github.com/10gen-labs/YCSB

Versions of each database:

- Cassandra 2.1.2
- Couchbase 3.0.2
- MongoDB 3.0.1

Driver versions:

- Cassandra: CQL cassandra-driver-core 2.1.5
- Couchbase: couchbase-client 1.1.8, spymemcached 2.8.9
- MongoDB: mongo-java-driver 2.13.0

The following **throughput optimized** configurations were used for each database. All other settings default:

[1] https://github.com/couchbaselabs/YCSB fork was enhanced by Thumbtack and we applied one additional bug fix.

- Cassandra
  - create keyspace … with durable_writes=false
  - create table with COMPACT STORAGE and leveledCompactionStrategy[2]
- Couchbase
  - Default settings
- MongoDB
  - --storageEngine=wiredTiger --nojournal

The following **durability optimized** configurations were used for each database. All other settings default:

- Cassandra
  - in config/cassandra.yaml commitlog_sync; batch[3]
  - in config/cassandra.yaml commitlog_batch_window_in_ms: 10 [4]
- Couchbase
  - YCSB setting couchbase.PersistTo=MASTER
- MongoDB
  - YCSB mongodb.writeConcern=journaled
  - --storageEngine=wiredTiger

For the balanced configuration, all settings were default for Cassandra and MongoDB. The following configurations were made:

- For Cassandra, the commitlog was placed on a separate device
- For MongoDB the journal was placed on a separate device
  - --storageEngine=wiredTiger

YCSB config

- 20 Million documents, each with 10 fields
- 20 Million operations
- Zipfian request distribution
- readallfields=true
- writeallfields=false
- In some configurations where running 20 million operations would be prohibitely slow the workload was run with additional parameter "maxexecutiontime=900" which limited the run to 15 minutes.

[2] Other compaction strategies were tested, and this configuration produced the best results.

[3] Changing commitlog_sync setting requires restarting Cassandra server and applies to all write operations.

[4] Tests were run with 50ms and 10ms and performance was significantly better with 10ms batch window.