Comparing NoSQL Databases with YCSB Standard Benchmark

MongoDB 3.2 vs Couchbase Server 4.5

Published June 2016

AVALON

Executive Summary

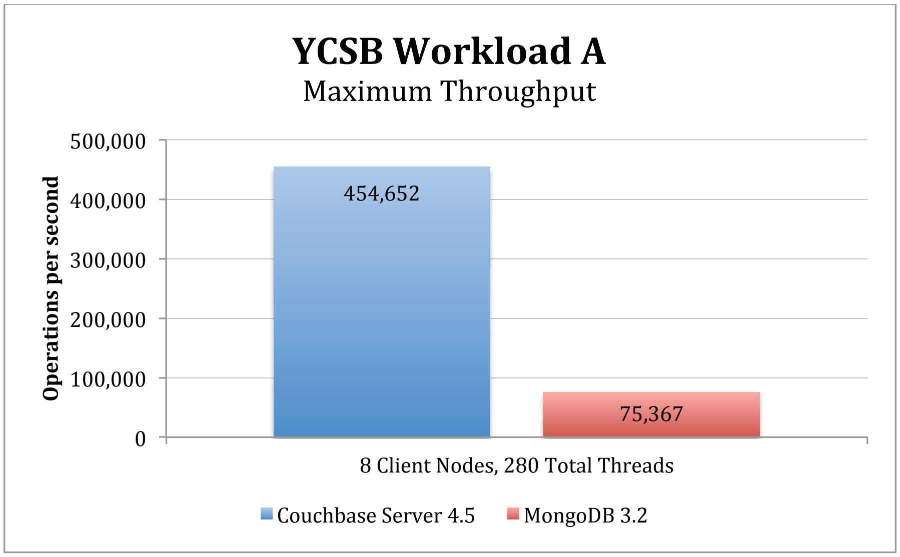
Avalon Consulting, LLC. has conducted a YCSB standard benchmark for a series of comparisons between MongoDB 3.2 and Couchbase Server 4.5. The measurements compared both direct data access with Workload A and querying with Workload E, applying the best practices from both Couchbase Server and MongoDB. As we did in last year's benchmark, we focused on performance. Key to performance is the ability to maintain low latency at high throughput. It is also important to show how these 2 databases perform when the volume of data is too large to reside in memory.

For the measurement, good hygiene was critically important. To achieve this we have applied a few principles to the measurements.

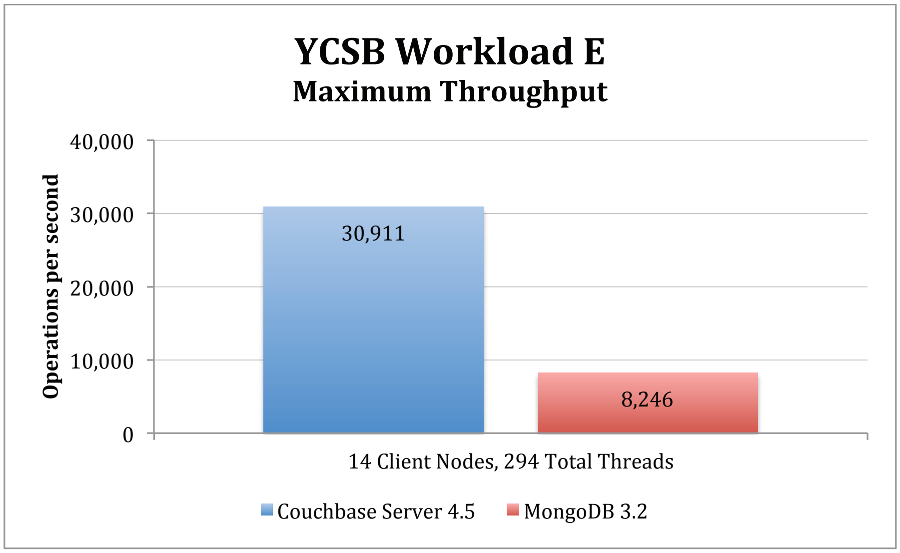
- Stay loyal to the original definition of YCSB workloads: Unlike some of the other YCSB branded studies, we have used the original workload definitions for workload A and workload E without any modifications except the item count in the database: Both runs are executed on 150 million items in the database.
- Use the most popular drivers for both products: we based our tests original github repository with the top "star" and "fork" count (brianfrankcooper/YCSB). (NOTE: We included a pull request (PR #773) from Couchbase. This version is available as a fork at https://github.com/Avalon-Consulting-LLC/YCSB. We expect this to be in the upstream repository soon.)
- We have used official published binaries from both companies.
- Ensure results can be repeated by anyone out there: We have fully disclosed the details of the test in this study to allow repeating the results. Please see the full disclosure details below for detailed instructions and scripts.

Overall, Couchbase Server 4.5 has shown a great deal of improvement over the previous runs, while MongoDB results have been similar to previous measurements. The improvements in Couchbase Server for Workload E (query execution) were due to the new N1QL query execution engine and memory-optimized Global Secondary Indexes. Workload A with Couchbase Server also has shown that direct data access is much faster with efficient direct data access with a caching consolidated database that is capable of performing sub-millisecond latency reads and writes under high throughputs.

The results below show that for both workloads (A and E), Couchbase Server significantly outperformed MongoDB, displaying a far higher maximum throughput for each under both workloads, while maintaining better latency.



Price/Performance	Couchbase Server	MongoDB
Monthly Cost per (Op/sec)[1]	$0.02	$0.15



Price/Performance	Couchbase Server	MongoDB
Monthly Cost per (Ops/sec)[2]	$0.36	$1.34

[1] See cost calculation in appendix
[2] See cost calculation in appendix

Benchmark Overview

In order to deliver the personalized, contextualized experiences that today's customers demand, companies have to harness and utilize the data behind their business and applications. NoSQL promises to power such applications that need real-time, big data interactions in the new Digital Economy.

NoSQL databases provide a variety of different approaches for query and data access. For measurements in this study Couchbase Server and MongoDB were chosen as both support document stores via JSON, providing an agile and flexible approach to data modeling. However most similarities between these two databases end there. Architecturally both products are very different in how they choose to provide data access and query execution. The following table summarizes some of these differences.

	Couchbase Server 4.5	MongoDB 3.2
Query Language	SQL-like language for combining best of NoSQL and SQL	MongoDB Specific API (.find() etc)
Query Execution	Direct Global-Index Access with Subset of Nodes Engaged in Query Execution	Scatter-Gather with all nodes Engaged in Query Execution
Indexing Topologies	Global & Local Indexing	Local Indexing
Indexing Storage	Lock-free Skip-list Indexes	B-tree Indexes
High Availability	Replica Based	Replica Based
Consistency	Consistent Data Access with Master based Read/Writes with Dials for Data Access and Query Consistency	Consistent Data Access with Master based Read/Writes with Dials for Data Access and Query Consistency
Durability	Replication and Disk Based Durability	Replication and Disk Based Durability
Caching for Fast Data Access	Built-in actively managed, in-heap cache that eliminates the need to deploy a separate caching tier	Simple caching that requires an added caching tier for low latency access

It is important to note that you will find other reports based on YCSB and you may notice contradicting results. That is why it was important for this measurement to stay loyal to the definition of the YCSB workloads. Unlike other reports, this measurement did not modify the query, read or write ratios of workloads or the data types defined by the original benchmark. The measurements kept full fidelity with the original Workload A (%50 read and %50 update) and Original Workload E (%95 query and %5 insert).

Test Methodology and Configuration

Test Environment

For the measurements, Avalon used Amazon Web Services EC2 instances. In order to minimize the variances in performance of AWS instances, each measurement was done 3 times. Both the server side and client side resources are kept identical for both Couchbase Server and MongoDB measurements. All instances were hosted in a VPC to avoid variance due to noisy neighbors. In addition, virtual machines were tuned to use AWS enhanced networking to provide maximum network throughput. Avalon created an AMI based on CentOS 6 with tuned network settings for Couchbase Server, MongoDB and YCSB client instances. 2 SSD storage volumes were used for each database instance, with indexes on one volume and data on the other.

In our previous benchmark, we limited results to a 5ms latency cap. For this benchmark, we removed that cap and used a fixed set of node and thread settings, recording the throughput and latency at each setting.

Database Server Resources	
Node Count	9
Node Type	C3.8xlarge 32 virtual CPUs with 60 GB RAM and 2 x 320GB SSD Storage with High Bandwidth Networking
Node OS	CentOS 6
Database Client (YCSB) Resources	
Node Count	1 to 14
Node Type	R3.8xlarge 32 virtual CPUs with 60 GB RAM and 2 x 320GB SSD Storage with High Bandwidth Networking
Node OS	CentOS 6
Data Configuration	
Item Count	150 Million
Data Shape	YCSB Default ~1K JSON documents with 10 fields with 100 bytes per field.
Memory to Data Size Ratio	Target of ~%50 of Data In RAM with %100 of Data on Storage

YCSB Code

There are a number of YCSB repositories publicly available on github. The code used in measurements is critical to the validity of the results and It is important to check the repository used with each measurement when validating results. Many of the published YCSB-branded benchmarks utilize modified repositories that change the underlying code used for measurements. For this measurement, we have used an updated Couchbase driver which has been submitted to the main fork as a pull request (PR #773).

YCSB Configuration	
Repo	https://github.com/brianfrankcooper/YCSB Most popular YCSB repo as of June 2016. Github “Stars” >1200 “Forks” >800
MongoDB YCSB Driver	mongodb https://github.com/brianfrankcooper/YCSB/tree/master/mon-godb
Couchbase YCSB Driver	couchbase2 https://github.com/brianfrankcooper/YCSB/tree/master/couch-base2 pull request PR#773 or https://github.com/ingenthr/YCSB (n1ql-raw branch) Settings: couchbase.epoll=true couchbase.boost=16 couchbase.upsert=true

Workloads

As stated above, for this benchmark we have stayed loyal to the definition of YCSB workloads and picked two representative workloads to measure: Workload A and Workload E.

- Workload A defines a workload that simulates the capture of recent user actions with 50% reads and 50% updates.
- Workload E defines a workload that simulates threaded conversations in social networks with 95% queries looking for a range of items and 5% inserts.

You can find the full definitions of the workloads here: https://github.com/brianfrankcooper/YCSB/wiki/Core-Workloads
We set the record count to 150 million and the operation count to 150 million and ran each iteration for at least 20 minutes.

Reporting Aggregate Results

For reporting accurate results that minimized the impact of the noisy-neighbor problem of public infrastructures like AWS, we have run all tests measurements 3 times.
When reporting throughput numbers we have averaged all 3 measurements per test.

- For both workload A and E, throughput (ops/sec) is calculated as an average of the 3 runs.

When reporting latency, we have again averaged the latency across 3 measurements and calculated operation latency for the workload using the distribution of the operations.

- For workload A, with a 50% read and 50% update distribution, read and update latency is calculated as an average of the 3 runs for each operation using the 95th percentile measurement for Update and Read. Operation latency is calculated as an average of read and update latency as the distribution is 50%/50%.
- For workload E with 95% scan (query) and 5% insert distribution, query and insert latency is calculated as an average of the 3 runs for each operation using the 95th percentile. Operation latency is calculated as a weighted average of the 0.95*scan latency and 0.05*insert latency, aligning with the 95% scan/5% insert distribution ratio.

Couchbase Server Configuration

Couchbase Server 4.5 was used for the tests. (version 4.5.0.2601). The configuration we used is below.

Couchbase Server Configuration	
Data RAM Quota	18GB
Bucket RAM Quota	18GB
Index RAM Quota *	34GB
Services Configured on each node	Data, index, query
storageMode	memory_optimized
indexer.settings.maxVbQueueLength	5000
indexer.settings.max_cpu_percent	400
indexer.settings.wal_size	40960
replicas	1
compaction_number_of_kv_workers **	1
compaction trigger on % fragmentation **	%75

* Memory allocated to data vs index has a fine grain control in Couchbase Server 4.5. 18GB for bucket RAM is applied to keep memory-resident ratio at %50 for data. Index RAM is only used when global secondary indexes are present in the system. Workload A does not use Index Service. Only Workload E needs global secondary indexes.

** In productions systems with heavy mutation load, it is recommended to dial down the aggressiveness of the background compaction.

MongoDB Configuration

MongoDB 3.2 was used for the tests. We used the community version for this benchmark, but there are no advertised performance differences between the community version and the enterprise version. The following configuration parameters were used for the benchmark.

MongoDB Server Configuration	
Storage Engine	Wired Tiger
mongos	On each YCSB client node
Memory *	52GB per instance for workload E 18GB per instance for workload A
Read Preference	nearest
Replicas	1

* Memory allocated is lowered in workload A to maintain memory-resident ratio at %50 for data. With Workload E indexes take up additional space so memory setting is kept higher to allow caching indexes.

Results

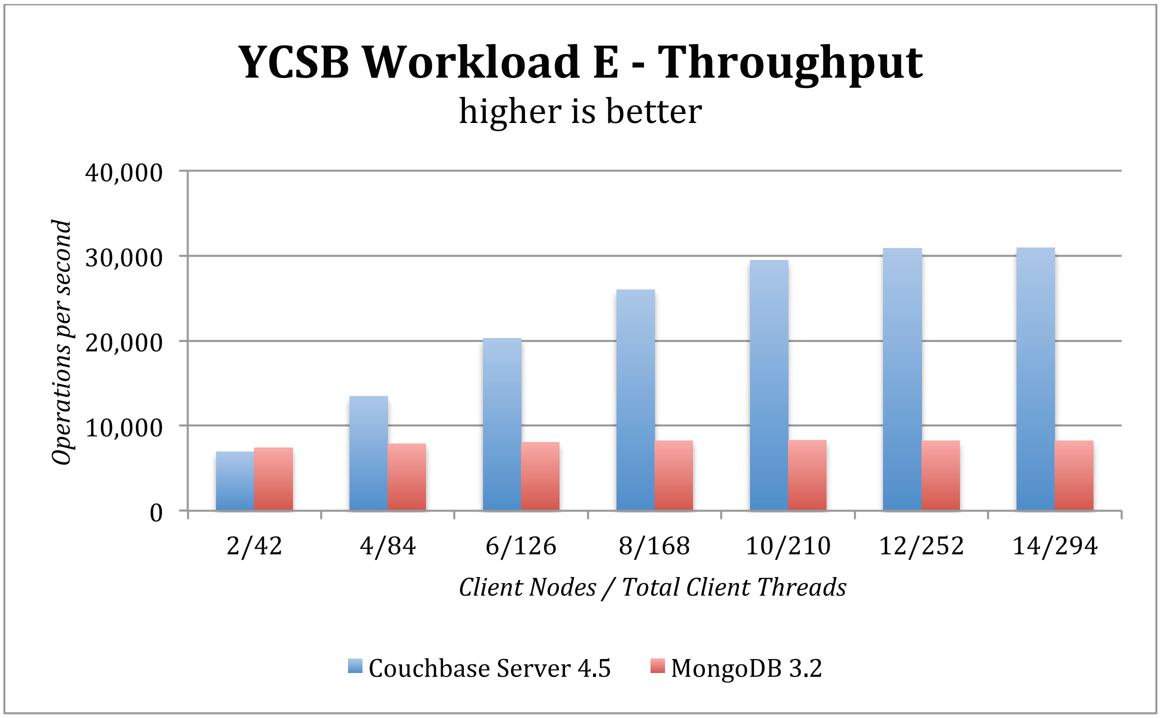
YCSB Workload E

Workload E measures query capabilities in both products. Workload E defines a workload that simulates threaded conversations in social networks with %95 queries looking for a range of items and %5 inserts.

We ran 7 different client loads for YCSB Workload E, increasing both the number of client nodes and the total thread count at each increment.

Throughput Comparison

As illustrated in the graph below, Couchbase Server 4.5 was able to scale to handle the increasing load at each step. MongoDB's throughput capacity remained relatively flat. It's important to note how each database demonstrated increased latency across the load steps, however latency for Couchbase increased roughly 57% where latency for MongoDB increased 589%.



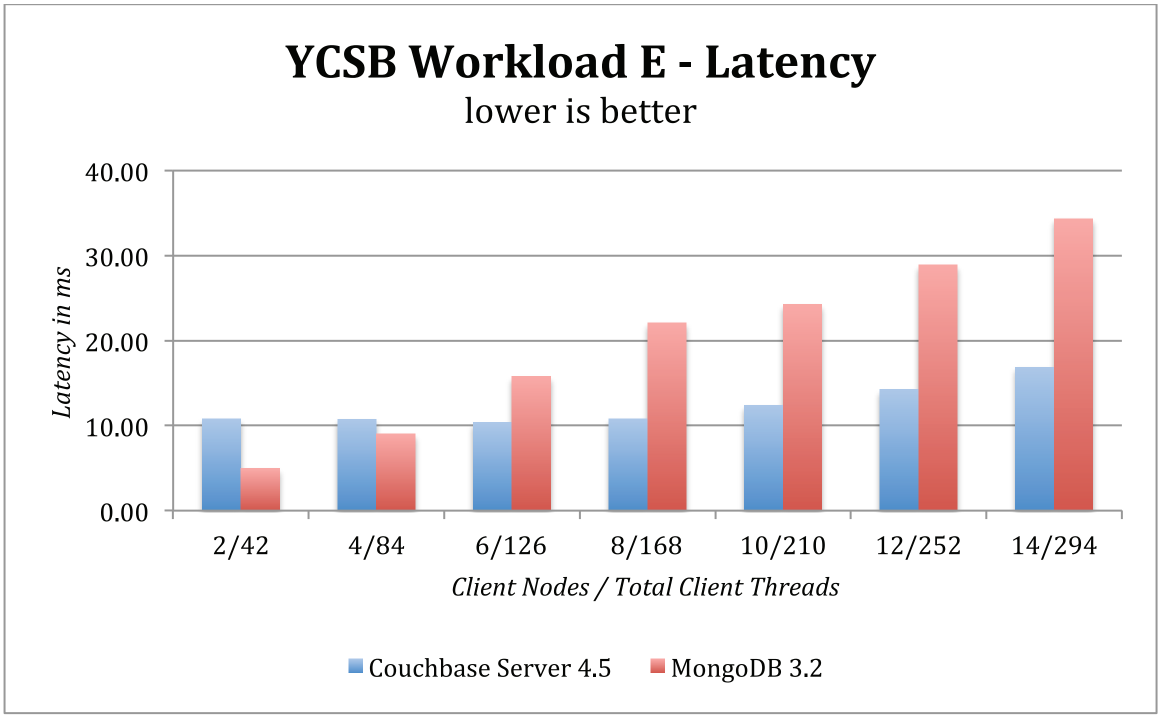
	Couchbase Server	MongoDB
294 client threads	30,911 Ops / Sec	8,246 Ops / Sec

Latency Comparison

The graph below shows the change in latency for Workload E across the YCSB load scale. The latency is a weighted average of average scan time and average insert time, measured as:

$$\Sigma(\,(0.95 \times average(scan))),(0.05 * average(insert))\,)$$

Latency for Couchbase Server increased somewhat modestly across the increasing load, where latency for MongoDB, increased much more quickly despite having a lower initial value than Couchbase Server.



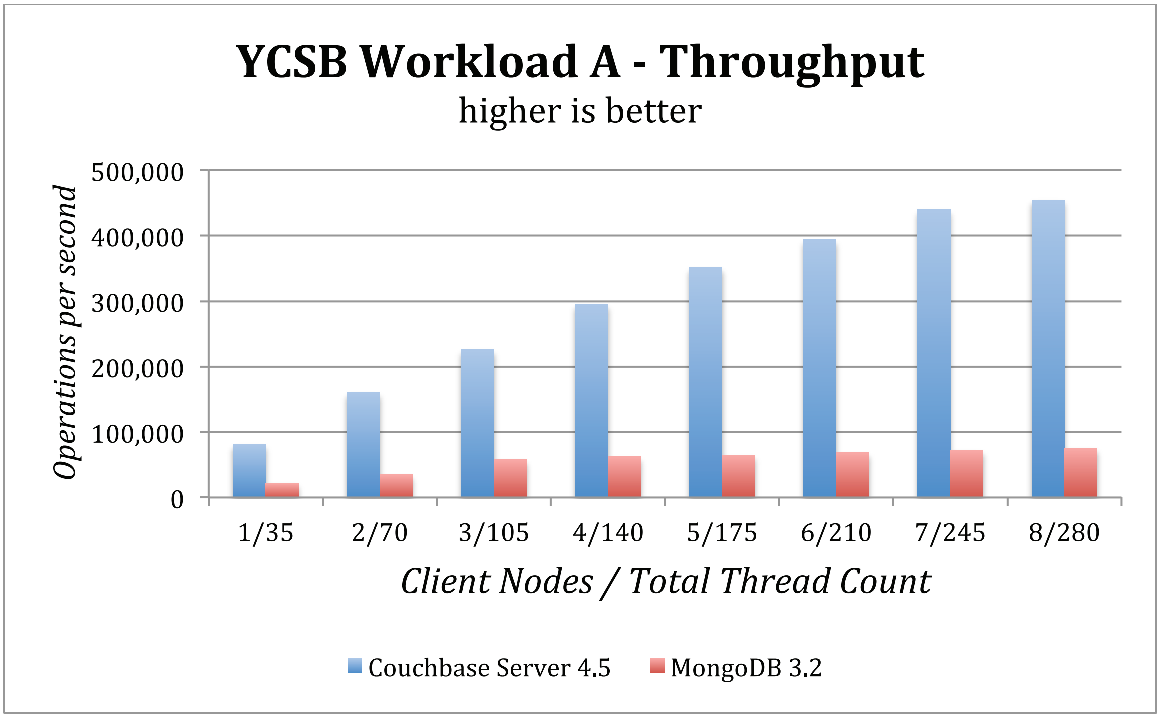
	Couchbase Server	MongoDB
294 client threads	16.9ms	34.36ms

YCSB Workload A

YCSB Workload A was included to demonstrate performance for a typical key-value scenario. It presents a load balanced 50/50 between read and update. We ran 4 combinations of nodes and threads for each database using YCSB Workload A, varying from 2 nodes with 70 total client threads up to 8 nodes with 280 total threads.

Throughput Comparison

For YCSB Workload A, Couchbase Server was able to scale somewhat linearly with the increasing client load. Couchbase throughput increased 264% compared to MongoDB's 186% increase.



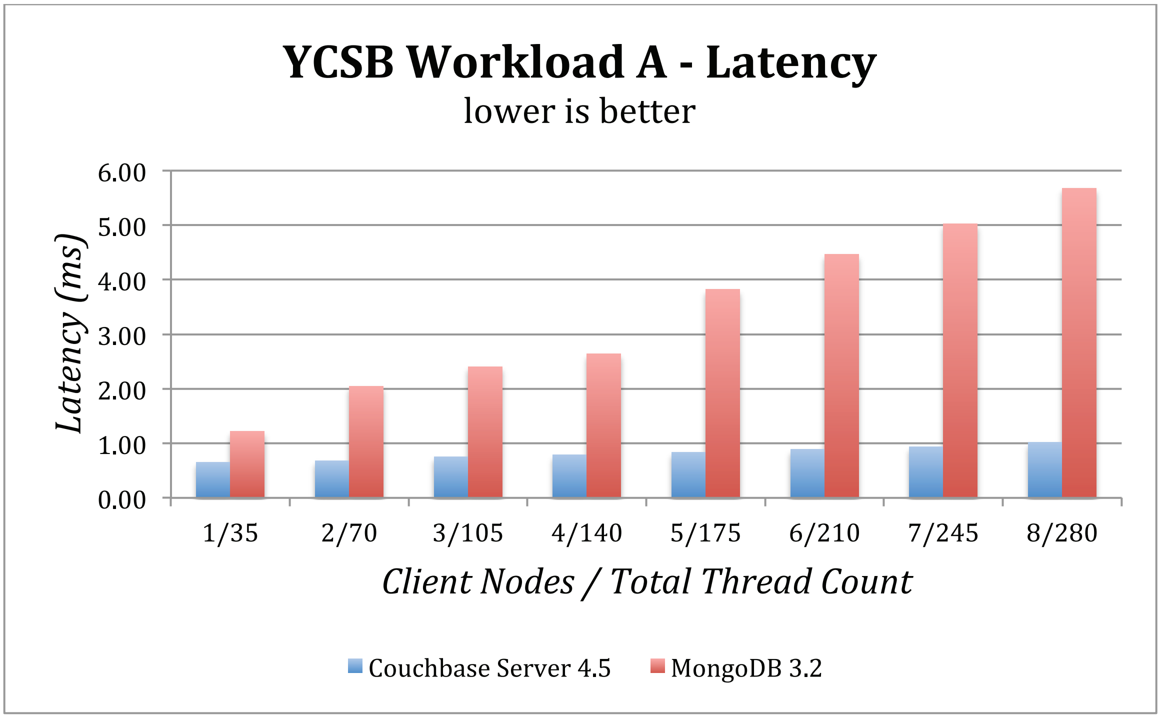
	Couchbase Server	MongoDB
280 client threads	454,652 Ops / Sec	75,367 Ops / Sec

Latency Comparison

The graph below shows the change in latency for Workload A across the YCSB load scale. The latency is a weighted average of average scan time and average update time, measured as:

$$\Sigma(\ (0.5 \times average(read))), (0.5 * average(update))\ )$$

Latency for Couchbase Server increased somewhat modestly across the increasing load showing a 20% increase. Across the same load scale, MongoDB's latency increased 116%.



	Couchbase Server	MongoDB
280 client threads	1.02ms	5.68ms

Conclusion

We attempted to simulate as realistic workloads as possible via the YCSB benchmark suite. We included benchmarks using 2 of the YCSB workloads: Workload A, intended to simulate a standard read/write application profile; and Workload E, intended to simulate a system that is query-intensive (a 95% query /5% insert split). Between these two workload types, we believe that the most common NoSQL usage profiles are covered.

Based on our benchmark results, Couchbase Server 4.5 shows better overall performance for both throughput and latency in both YCSB A and E Workloads. In fact, Couchbase Server appeared to have much more capacity for handling load within a 5ms latency cap for Workload A. For Workload E, with its more intensive query load, Couchbase Server 4.5 also clearly outperformed MongoDB 3.2.

Appendix

Result Data

MongoDB **Workload A (Key-Value)**

YCSB Nodes/Threads (35 threads/client)	Run #1 Throughput/Latency (95th)	Run #2 Throughput/Latency (95th)	Run #3 Throughput/Latency (95th)
1/35	22,101 ops/sec Update: 1.41ms Read: 1.01ms	21,192 ops/sec Update: 1.49ms Read: 1.02ms	22,127 ops/sec Update: 1.40ms Read: 1.01ms
2/70	35,002 ops/sec Update: 2.21ms Read: 1.84ms	34,427 ops/sec Update: 2.32ms Read: 1.89ms	35,143 ops/sec Update: 2.19ms Read: 1.85ms
3/105	57,096 ops/sec Update: 2.35ms Read: 1.99ms	58,002 ops/sec Update: 3.29ms Read: 1.87ms	59,103 ops/sec Update: 3.14ms Read: 1.81ms
4/140	62,953 ops/sec Update: 3.12ms Read: 2.16ms	63,101 ops/sec Update: 3.09ms Read: 2.11ms	61,311 ops/sec Update: 3.19ms Read: 2.19ms
5/175	65,021 ops/sec Update: 3.69ms Read: 3.99ms	65,691 ops/sec Update: 3.65ms Read: 3.91ms	64,802 ops/sec Update: 3.71ms Read: 3.99ms
6/210	69,003 ops/sec Update: 4.91ms Read: 4.01ms	68,892 ops/sec Update: 4.92ms Read: 4.04ms	69,113 ops/sec Update: 4.90ms Read: 4.00ms
7/245	72,911 ops/sec Update: 5.55ms Read: 4.41ms	71,998 ops/sec Update: 5.61ms Read: 4.52ms	72,346 ops/sec Update: 5.59ms Read: 4.47ms
8/280	75,101 ops/sec Update: 6.31ms Read: 5.01ms	75,002 ops/sec Update: 6.39ms Read: 5.12ms	75,997 ops/sec Update: 6.26ms Read: 5.00ms

Couchbase **Workload A (Key-Value)**

YCSB Nodes/Threads (35 threads/client)	Run #1 Throughput/Latency (95th)	Run #2 Throughput/Latency (95th)	Run #3 Throughput/Latency (95th)
1/35	81,163 ops/sec Read : 0.634ms Update : 0.673ms	81,132 ops/sec Read : 0.637ms Update : 0.675ms	81,038 ops/sec Read : 0.637ms Update : 0.677ms
2/70	160,443 ops/sec Read : 0.66ms Update : 0.711ms	160,989 ops/sec Read : 0.658ms Update : 0.705ms	160,291 ops/sec Read : 0.655ms Update : 0.706ms
3/105	226,652 ops/sec Read : 0.728ms Update : 0.787ms	225,248 ops/sec Read : 0.729ms Update : 0.786ms	225,598 ops/sec Read : 0.727ms Update : 0.785ms
4/140	295,533 ops/sec Read : 0.756ms Update : 0.824ms	295,473 ops/sec Read : 0.755ms Update : 0.822ms	296,257 ops/sec Read : 0.749ms Update : 0.818ms
5/175	352,985 ops/sec Read : 0.796ms Update : 0.871ms	352,448 ops/sec Read : 0.804ms Update : 0.88ms	349,661 ops/sec Read : 0.809ms Update : 0.885ms
6/210	398,770 ops/sec Read : 0.855ms Update : 0.931ms	395,697 ops/sec Read : 0.837ms Update : 0.915ms	397,353 ops/sec Read : 0.863ms Update : 0.939ms
7/245	442,551 ops/sec Read : 0.893ms Update : 0.97ms	438,194 ops/sec Read : 0.906ms Update : 0.983ms	439,526 ops/sec Read : 0.9ms Update : 0.977ms
8/280	436,328 ops/sec Read : 1.145ms Update : 0.969ms	456,227 ops/sec Read : 1.02ms Update : 1.001ms	471,400 ops/sec Read : 0.962ms Update : 1.009ms

MongoDB **Workload E (Query)**

YCSB Nodes/Threads (21 threads/client)	Run #1 Throughput/Latency (95th)	Run #2 Throughput/Latency (95th)	Run #3 Throughput/Latency (95th)
2/42	7,501 ops/sec Scan: 5.02ms Insert: 3.01ms	7,421 ops/sec Scan: 5.09ms Insert: 3.04ms	7,316 ops/sec Scan: 5.14ms Insert: 3.12ms
3/63	7,648 ops/sec Scan: 7.94ms Insert: 5.86ms	7,621 ops/sec Scan: 7.99ms Insert: 5.91ms	7,511 ops/sec Scan: 8.05ms Insert: 6.14ms
4/84	7,994 ops/sec Scan: 9.02ms Insert: 7.12ms	7,901 ops/sec Scan: 9.11ms Insert: 7.28ms	7,812 ops/sec Scan: 9.29ms Insert: 7.43ms
5/105	8,029 ops/sec Scan: 13.21ms Insert: 10.02ms	8,001 ops/sec Scan: 13.29ms Insert: 10.06ms	7,992 ops/sec Scan: 13.35ms Insert: 10.11ms
6/126	8,102 ops/sec Scan: 15.83ms Insert: 13.91ms	8,004 ops/sec Scan: 15.92ms Insert: 13.98ms	8,009 ops/sec Scan: 15.94ms Insert: 13.99ms
7/147	8,199 ops/sec Scan: 18.03ms Insert: 15.41ms	8,083 ops/sec Scan: 18.22ms Insert: 15.55ms	8,116 ops/sec Scan: 18.18ms Insert: 15.48ms
8/168	8,251 ops/sec Scan: 22.01ms Insert: 19.12ms	8,183 ops/sec Scan: 22.39ms Insert: 19.54ms	8,201 ops/sec Scan: 22.32ms Insert: 19.42ms
10/210	8,246 ops/sec Scan: 24.35ms Insert: 23.52ms	8,299 ops/sec Scan: 24.27ms Insert: 23.10ms	8,281 ops/sec Scan: 24.31ms Insert: 23.40ms
12/252	8,283 ops/sec Scan: 28.01ms Insert: 24.52ms	8,107 ops/sec Scan: 31.23ms Insert: 26.91ms	8,341 ops/sec Scan: 27.98ms Insert: 25.99ms
14/294	8,303 ops/sec Scan: 31.72ms Insert: 27.01ms	8,103 ops/sec Scan: 36.94ms Insert: 30.06ms	8,332 ops/sec Scan: 35.26ms Insert: 29.94ms

Couchbase **Workload E (Query)**

YCSB Nodes/Threads (21 threads/client)	Run #1 Throughput/Latency (95th)	Run #2 Throughput/Latency (95th)	Run #3 Throughput/Latency (95th)
2/42	7,086 ops/sec Insert : 1.817ms Scan : 11.015ms	6,513 ops/sec Insert : 2.185ms Scan : 11.967ms	7,139 ops/sec Insert : 1.783ms Scan : 10.863ms
3/63	10,374 ops/sec Insert : 1.931ms Scan : 11.076ms	10,759 ops/sec Insert : 1.807ms Scan : 10.69ms	10,667 ops/sec Insert : 1.824ms Scan : 10.764ms
4/84	13,846 ops/sec Insert : 1.893ms Scan : 10.835ms	12,969 ops/sec Insert : 2.213ms Scan : 11.659ms	13,560 ops/sec Insert : 2.08ms Scan : 11.087ms
5/105	17,343 ops/sec Insert : 1.864ms Scan : 10.754ms	17,162 ops/sec Insert : 1.923ms Scan : 10.802ms	17,137 ops/sec Insert : 1.939ms Scan : 10.869ms
6/126	19,928 ops/sec Insert : 2.032ms Scan : 11.12ms	20,601 ops/sec Insert : 1.968ms Scan : 10.578ms	20,293 ops/sec Insert : 1.937ms Scan : 10.767ms
7/147	23,243 ops/sec Insert : 1.961ms Scan : 10.858ms	22,634 ops/sec Insert : 2.01ms Scan : 11.188ms	22,507 ops/sec Insert : 2.104ms Scan : 11.262ms
8/168	26,095 ops/sec Insert : 2.013ms Scan : 11.179ms	25,743 ops/sec Insert : 2.137ms Scan : 11.408ms	26,210 ops/sec Insert : 2.011ms Scan : 11.19ms
10/210	29,752 ops/sec Insert : 2.159ms Scan : 12.826ms	29,359 ops/sec Insert : 2.284ms Scan : 12.96ms	29,241 ops/sec Insert : 2.233ms Scan : 12.941ms
12/252	30,823 ops/sec Insert : 2.928ms Scan : 14.92ms	30,888 ops/sec Insert : 2.922ms Scan : 14.883ms	30,869 ops/sec Insert : 2.898ms Scan : 14.74ms
14/294	31,321 ops/sec Insert : 3.884ms Scan : 17.3ms	30,603 ops/sec Insert : 4.042ms Scan : 17.776ms	30,810 ops/sec Insert : 3.909ms Scan : 17.658ms

Cost Calculation

Instance Count	9	
Instance Type	c3.8xlarge	
OS	CentOS	
Cost per hour per instance	$1.68	
Cost per day	$40.32	
Cost per month per instance	$1,229.76	
Monthly cost for all instances	$11,067.84	
Workload A	Couchbase	MongoDB
Max Throughput (ops/sec)	454,652	75,367
Monthly cost per op/sec	$0.02	$0.15
Workload E	Couchbase	MongoDB
Max Throughput (ops/sec)	30,911	8,246
Operations/sec $s per month	$0.36	$1.34

Full disclosure details

The configuration and scripts to reproduce this benchmark are available on github at https://github.com/Avalon-Consulting-LLC/couchbase_45_mongodb_32_benchmark.

AVALON