Benchmarking Couchbase Server for Interactive Applications

By Alexey Diomin and Kirill Grigorchuk
1. Introduction ........................................................................................................................................ 3
2. A brief overview of Cassandra, MongoDB, and Couchbase ......................................................... 3
3. Key criteria for a DB used in interactive applications ................................................................. 4
4. Infrastructure and settings for benchmarking............................................................................... 6
5. Results: latencies for reads, inserts, and updates ........................................................................ 8
6. Results: 95th-percentile time for reads, inserts, and updates ..................................................... 9
7. Analysis ........................................................................................................................................... 11
8. Conclusion ...................................................................................................................................... 11
9. Additional links ............................................................................................................................. 12
10. About the authors ......................................................................................................................... 12

*The benchmark research and testing presented in this whitepaper were commissioned by Couchbase, Inc.
1. Introduction

Interactive applications need high-performance and scalability, calling for a different kind of database. If your application is not fast enough, users may quickly abandon it and look for alternatives. For example, in paid online social games, players are extremely demanding and will drop out, even if there is a slight delay. To deliver the best user experience, you must pick the right database that is fast, scalable, and robust.

Unfortunately, the capabilities of traditional RDBMS for working with big data are limited—they cannot provide the necessary scalability and performance. NoSQL databases have become a viable alternative to RDBMS—particularly, for applications that need to read or write enormous amounts of data rapidly. They provide high throughput, low latency, and horizontal scaling. However, with so many different options around, choosing the right NoSQL database can be tricky.

Recently we took the time to review and benchmark several NoSQL databases. This whitepaper provides an overview of three popular NoSQL solutions: Cassandra, MongoDB, and Couchbase. The test scenario simulated workloads typical of interactive Web applications, such as social and mobile games, real-time ad platforms, etc. In use cases like these, most, if not all of the working data set is often stored in RAM to ensure low latencies under heavy loads. So, this whitepaper is mainly focused on testing how the three systems behave in such a scenario.

2. A brief overview of Cassandra, MongoDB, and Couchbase

If you are already familiar with these NoSQL databases, you might want to skip the rest of this section and go directly to the performance evaluation.

**Cassandra** is a distributed columnar key-value database with eventual consistency. It is optimized for write operations and has no central master—data can be written or read to and from any of the nodes in a cluster. Cassandra provides seamless horizontal scaling and has no single point of failure. If a node in the cluster fails, another node steps up to replace it. At the moment, Cassandra is an Apache 2.0 licensed project supported by the Apache Community.

**MongoDB** is a schema-free, document-oriented, NoSQL database that stores data in the BSON format. A BSON document is essentially a JSON document in the binary format, which allows for easier and faster integration of data in certain types of applications. This database also provides horizontal scalability and has no single point of failure. A MongoDB cluster is different from a Cassandra or Couchbase Server cluster, as it includes an arbiter, a master, and multiple slaves. Since 2009, MongoDB is an open source project with the AGPL license supported by 10gen.

**Couchbase** is an open source NoSQL document database for interactive Web and mobile applications. Documents in Couchbase Server are stored as JSON. With built-in caching, Couchbase provides low-latency read and write operations with linearly scalable throughput. The architecture has no single point of failure. It is easy to scale-out the cluster and support live
cluster topology changes. This means, there is no application downtime when you are upgrading your database, software, or hardware using rolling upgrades. Couchbase, Inc. develops and provides commercial support for the Couchbase open source project that is Apache 2.0 licensed.

3. Key criteria for a DB used in interactive applications

When choosing a database for interactive applications, the following are the most important factors to keep in mind:

- **Scalability.** It is hard to predict when your application needs to scale, but when your website traffic suddenly spikes and the database does not have enough capacity, you need to scale quickly, on demand, and without any application changes. Similarly, when your system is idle, you should have a possibility to decrease the amount of resources used. Scaling your database must be a simple operation—you should not need to deal with complicated procedures or make any changes to the application.

In this white paper, we only speak about horizontal scalability, which involves dividing a system into small structural components hosted on different physical machines (or groups of machines) and/or increasing the number of servers that perform the same function in parallel.

a) **Cassandra** meets the requirements of an ideal horizontally scalable system. Nodes can be added seamlessly as you need more capacity. The cluster automatically utilizes new resources. A node can be decommissioned in the automatic or semi-automatic mode.

b) **MongoDB** has a number of functions related to scalability. They include: automatic sharding (auto-partitioning of data across servers), reads and writes distributed over shards, and eventually-consistent reads that can be distributed over replicated servers. When the system is idle, cluster size can only be decreased manually. The administrator uses the management console to change the system’s configuration. After that, the server process of MongoDB can be safely stopped on the idle machines.

c) **Couchbase** scales horizontally. All nodes are identical and easy to setup. Nodes can be added or removed from the cluster with a single button click and no changes to the application. Auto-sharding evenly distributes data across all nodes in the cluster without any hotspots. Cross-datacenter replication makes it possible to scale a cluster across datacenters for better data locality and faster data access.

- **Performance.** Interactive applications require very low read and write latencies. The database must deliver consistently low latencies regardless of load or the size of data. In general, the read and write latency of NoSQL databases is very low because data is shared across all nodes in a cluster while the application’s working set is in memory.
Interactive applications need to support millions of users and handle different workloads—read, write, or mixed. In the next section, we share the results of the performance tests that compare Couchbase to both Cassandra and MongoDB in a scenario simulating an interactive application. The tests measured average latency versus varying levels of throughput.

**Availability.** Interactive Web applications need a highly available database. If your application is down, you are simply losing money. To ensure high availability, your solution should be able to do online upgrades, easily remove a node for maintenance without affecting the availability of the cluster, handle online operations, such as backups, and provide disaster recovery, if the entire datacenter goes down.

Below are examples of how availability is achieved in Couchbase, Cassandra, and MongoDB:

**a) Cassandra:** Every node in a Cassandra cluster, or “ring,” is given a range of data for which it is responsible. When Cassandra receives a write operation designated to be stored in a node that has failed, it will automatically route the write request to another node, which saves the write operation with a hint. The hint is a message that contains information about the failed node. The node that holds the hint monitors the cluster for the recovery of the node that missed the write request. If the failed node comes back online, the node that holds the hint will handoff the hint message to the recovered node, so that the write requests can be persisted in their proper location. When a new node is added to the cluster, the workload is distributed to this new node as well.

**b) MongoDB:** In MongoDB, data is spread across several shards (replica sets). Typically, each shard consists of multiple MongoDB instances, including an arbiter node, a master node, and multiple slaves. If a slave node fails, the master node automatically re-distributes the workload to the rest of the slave nodes. In case the master node crashes, the arbiter node elects a new master. If the arbiter node fails and there are no instances left in the shard, the shard is dead. In MongoDB, a replica set can span across multiple datacenters but writes can only go to one primary instance in one data-center (master-slave replication).

**c) Couchbase:** Couchbase Server maintains multiple copies (up to three replicas) of each document in a cluster. Each server is identical and serves active and replica documents. Data is uniformly distributed across all the nodes and the clients are aware of the topology. If a node in the cluster fails, Couchbase Server detects the failure and promotes replica documents on other live nodes to active. The client cluster map is updated to reflect the new topology, so the application continues to work without downtime. When capacity is added, data is re-balanced automatically, also without any downtime.
Ease of development. Relational databases require a rigid schema and, if your application changes, your database schema needs to change as well. In this regard, NoSQL databases have the following advantages:

a) Flexible schema. You do not have to modify any of the existing structural elements when new fields are added to a document. New and existing documents can co-exist without any additional changes.

b) Simple query language. Because data in a NoSQL document is stored in a de-normalized state, you can get and update a document with the put and get operations.

4. Infrastructure and settings for benchmarking

Our test infrastructure consisted of four extra-large instances on Amazon EC2 for the NoSQL databases and one instance for the client. Each instance had four virtual CPU cores with two Amazon compute units per core, 15 GB of RAM, and four EBS 50 GB volumes with RAID 0 striping. We used 64-bit Amazon Linux as the OS. Networking was all 10GigE.

The client used the Yahoo! Cloud Serving Benchmark (YCSB), which was modified to suit our needs—we added a warm-up phase and adjusted working-set load generation that simulates different users accessing different data objects with meaningful data amounts and runtime. As shown in Figure 1, the YCSB client consists of two main parts: the workload generator and workload scenarios.

The benchmark had 30 parallel client threads to drive the test, generating a mixed read-write workload with 5% of creates, 33% of updates, 2% of deletes, and 60% of reads. For all of the tests, we used 1.5 KB documents (15 fields and 100 bytes each)—a typical document size across several NoSQL database use cases. The total number of documents in the cluster was 30 million—15 million of active and 15 million of replica documents for each database.
We ran each test five times for every NoSQL database and compared the average data access latency against different throughput levels. The NoSQL databases were set up using the following configuration:

**Cassandra 1.1.2**

Cassandra JVM settings:

1. `MAX_HEAP_SIZE` (the total amount of memory dedicated to the Java heap)—6 GB
2. `HEAP_NEWSIZE` (the total amount of memory for a new generation of objects)—400 MB

Cassandra settings:

1. RandomPartitioner that uses MD5 Hashing to evenly distribute rows across the cluster
2. Memtable of 4 GB in size
MongoDB

1. Four shards, each with one replica; each shard is a set of two nodes—primary and secondary
2. Journaling disabled
3. Each node was running two Mongo Daemon processes and four Mongo Router processes.

Couchbase 2.0, Beta build 1723

1. One replica setting
2. 12 GB used as per node RAM quota using the Couchbase bucket type

5. Results: latencies for reads, inserts, and updates

Figures 2 and 3 show the average latency at varying throughput levels for read, insert, and update operations measured from the client to the server and back against varying levels of throughput. The lower the latency values, the better.

![Read latencies against throughput](image)

*Figure 2. Reads (average time)*
6. **Results: 95\textsuperscript{th}-percentile time for reads, inserts, and updates**

We also calculated the 95th-percentile time taken for a request to execute read, insert, and update operations measured from the client to the server and back against varying levels of throughput.

**Figure 3. Writes (average time)**
Figure 4. Reads (95th percentile)

Figure 5. Writes (95th percentile)
7. Analysis

While not exhaustive, this brief analysis aims to explain the results of the performance tests in the scenarios that simulate an interactive application.

**MongoDB** processed read requests slightly faster than Cassandra (see figures 2 and 4) but slower than Couchbase. Cassandra and Couchbase demonstrated better results, when processing writes (see figures 3 and 5) compared to MongoDB. The reason is that, due to coarser locking, writing data is a common performance bottleneck in MongoDB—especially when a lot of read and write operations are involved.

**Cassandra** uses the key and row types of cache while MongoDB relies on memory-mapped files. Despite this difference, the two databases demonstrated almost equal reading speeds (see figures 2 and 4). When writing data, Cassandra showed better results than MongoDB (see figures 3 and 5) since it first adds data to an in-memory structure called Memtable. Then, if the configured thresholds have been exceeded, it asynchronously flushes data to disk-based SSTables.

**Couchbase** had the lowest latencies in our test scenarios for interactive applications because of the built-in object-managed cache. Fine-grained locking at the document level provide high throughput for both reads and writes (see figures 2, 3, 4, and 5).

The results of this performance benchmarking can be easily replicated. To do so, use the configuration provided in the section, “Infrastructure and settings for benchmarking.” The modified YSCB benchmarking tool with custom connectors developed for this test can be downloaded from Github.

8. Conclusion

Choosing the right NoSQL database for your application can be complicated because not all NoSQL solutions are the same. Every solution is optimized for a particular type of workload and use case. Therefore each has its own pros and cons.

For instance, though the advantages of MongoDB include ad-hoc querying, increasing cluster size in Mongo involves a lot of manual operations done through the command line. So, it is mandatory that you have a highly skilled system administrator for this database.

Unlike MongoDB, Cassandra is rather flexible when a cluster needs to be resized. However, its extreme flexibility designed to sustain performance in highly distributed environments results in additional limitations. In particular, the database supports no transactions and cannot block separate records.

In its turn, Couchbase provides an admin console with flexible settings for changing cluster size. Each document is hashed to a node in the cluster. Read and write requests for a particular document are routed to the server holding the active copy of the document. This architecture
makes Couchbase fully consistent. Additionally, Couchbase comes with an administrative UI that provides the status of the whole cluster as well as an individual cluster node.

Your final decision may be influenced by these as well as many other factors. Still, the most important things you must consider when working with big data are latencies, throughput, availability, horizontal scaling, and ease of development.

9. Additional links

Cassandra’s Website: http://cassandra.apache.org
Couchbase’s Website: http://www.couchbase.com
MongoDB’s Website: http://www.mongodb.org
A modified version of YCSB at Github: https://github.com/Altoros/YCSB

10. About the authors

Altoros Systems, Inc. provides technology enablement and integration services around Hadoop and NoSQL to infrastructure software vendors, IaaS service providers and information-heavy enterprises. Focused on creating large-scale systems and custom data science solutions, the company brings years of experience in utilizing and benchmarking NoSQL databases, automation of deployment, cluster management and performance optimization. Headquartered in Silicon Valley (Sunnyvale, California), Altoros is a 250+ employee strong team across 8 locations (California, Massachusetts, Norway, Denmark, UK, Switzerland, Argentina, and Eastern Europe). The company has a proven track record serving technology leaders, such as RightScale, Couchbase, NuoDB, Joyent, Cisco, and other businesses across various industries. For more, please visit www.altoros.com.