University of Tromsø

CAP theorem in Cassandra and MongoDB

Camilla Stormoen & Nina Angelvik

Department of Computer Science

May 2017

1 Abstract

The shifting trends in consumer behaviour and the emerging of new data classes, require databases to be flexible and scalable, and to manage massive data flows. Corporate business can sometimes find that their relational databases do not satisfy these requirements and must look for other options, such as NoSQL databases. In this paper we will compare two of the most common NoSQL databases, MongoDB and Cassandra, to highlight the differences and when they are benificial to use.

2 Introduction

Imagine you are the CTO of a big corporate business. The market is changing fast and in order to be the best, the company has developed an idea for a new, innovative mobile application. If all goes well, the application is expected to generate massive amounts of unstructured data rapidly, and you understand that the traditional relational database model will fail to provide the scalability the application requires. It is your job to find a database management system that can provide both flexible and scalable solutions and you turn to NoSQL databases as they seem to fulfill these requirements. You soon understand that there are a number of different non-relational databases, but what is the difference between them and which one should you choose?

Non-relational databases use new and possibly unfamiliar architectures and data models. To satisfy the requirements of modern online applications, these databases must make trade-offs governed by the CAP theorem [1]. In this paper we will compare the two NoSQL databases MongoDB and Cassandra to highlight differences and when they are beneficial to use. We will do this by looking at how they address the infamous CAP theorem, as well how they handle data scalability in terms of read and update operations on an increasing data set.

3 Background

Relational databases have been the *de facto* standard since the 1970s since they have been the lone option available for developers and infrastructure teams. In the late 90s to early 2000, applications went from serving thousands of company-employees to having millions of users. The need for highly-available, scalable and high-performance datastores drove companies like Facebook, Amazon and Google to create new technologies. This new technology is called NoSQL databases [2][3]. NoSQL is also known as “non sql” or “not only SQL” databases. They provide mechanisms for storage and retrieval for data without the tabular relations used in relational databases.

3.1 NoSQL

Today there are many different types of NoSQL databases. They are all based on the same principles, but own different characteristics and can typically be divided into four categories [2]. Table 1 contains an overview of the four categories, their definitions and some example databases.

Type	**Definition**	**Examples of this type**
Key-Value store	All data is stored as a set of key and value. All keys are unique, and data access is done relating those keys to values.	Riak, Couchbase, Dynamo, HyperDex
Graph store	Databases that are used when data can be represented as a graph. E.g social networks.	Apache Giraph, AllegroGraph
Column store	This type is similar to the relational database model. Data is structured in columns that may be countless.	Cassandra, Accumulo
Document store	These databases pair each key with a complex data structure known as a document. Documents can contain different key-value pairs or nested documents.	Apache CouchDB, MongoDB

Table 1: An overview of the four categories of NoSQL databases.

3.2 The CAP theorem

The CAP theorem states that there are three basic requirements that exists in a relation when designing applications for a distributed architecture:

- **C**onsistency: all nodes have same data at the same time
- **A**vailability: all requests must result in a non-error response
- **P**artition tolerance: if part of the system fails, the system as a whole will not collapse

In theory it is impossible to fulfil all 3 requirements and therefore the current NoSQL databases can only satisfy one of the following combinations of consistency, availability and partitioning [4]

- **CA** - Single site cluster. All nodes are always in contact. When a partition occurs, the system blocks.
- **CP** - Some data may not be accessible, but the rest is still consistent.
- **AP** - System is still available under partitioning, but some of the data returned may be inaccurate.

4 Cassandra

Cassandra[1] is a distributed open-source storage system for handling large amounts of structured data across many commodity servers, while providing high availability with no single point of failure [5]. The database is developed by Apache Software Foundation and written in Java. Cassandra partitions data across the cluster using consistent hashing, but uses an order preserving hash function to do so.

Cassandra is part of the column-store family and is similar to the usual relational model which is made of columns and rows. It is also designed to store large amount of data and deal with huge volumes in an efficient way. Data stored in Cassandra can be distributed all over the world and deployed on large numbers of nodes across numerous data centers. When a node is added or removed, the data is automatically distributed over other nodes. The failed nodes can be replaced with no downtime. Since all the nodes have the same role there is no need for a master node. This is also called a decentralized architecture.

[1]http://cassandra.apache.org/

5 MongoDB

MongoDB[2] is a distributed open-source document-oriented data store developed in C++ [2]. It is a schemaless database where documents are grouped into collections according to their structure. It is possible to store some documents with different structure, but similarity is advised in order to achieve the best level of efficiency.

MongoDB documents are stored in BSON, Binary JSON, and each document is identified by a defined type (e.g timestamp) as well as an id field. In order to increase performance, data files are mapped in memory and when a new file is created, everything is flushed to disc. This releases memory, which is necessary especially for 32 bit MongoDB, as MongoDB's memory use results in the nodes only being able to store 2GB each. A 64 bit OS will be able to address more memory, thus it is not limited by this.

MongoDB use Master-Slave replication mechanism. A master can write or read files while slaves are backups with read-only permission. When a master fails, the slave with the most recent data is promoted to master. The replicas are asynchronous, meaning that performed updates are not spread instantly.

6 Comparison

In this section we will do a comparison of Cassandra and MongoDB in terms of how they address the CAP theorem.

6.1 Consistency, avalability and partitioning

Neither Cassandra nor MongoDB violates the CAP theorem, thus we know that they only fulfill two of the three properties.

In MongoDB these are consistency and partitioning. MongoDB supports a large variety in consistency levels, from strict consistency where you always read your own writes, to weak consistency where you read data of unknown oldness [6]. These varying levels of consistency are achieved by keeping a queue of requests for each connection. Every new request is appended to the queue, and subsequent requests on the connection occur only after the enqueued operation is processed. This results in each connection having a consistent view of the database, enabling it to always read its own writes.

In terms of partitioning, MongoDB supports *autosharding* [6]. Sharding is another term for partitioning and autosharding tries to abstract the architecture away from the application. This way the application is, to some extent, allowed to ignore the fact that it is not communicating with a standalone server. On the server side, MongoDB automates load balancing across the machines in the cluster (shards) and simplifies the process of adding and removing capacity.

The CAP properties fulfilled by Cassandra are availability and partitioning. Cassandra uses consistent hashing to partition data across the cluster [5]. The used hash function is order preserving and the output range of the function is treated as a circular space where the largest and smallest output value overlap. Each node in the cluster is assigned a position in the ring and becomes responsible for the value space between itself and its preceding node. Each data item is identified by a key which is hashed and then assigned to the node responsible for the key space in which the data item belongs. This partitioning method makes it easy to add/re-

[2]http://www.mongodb.org

move nodes as only the immediate neighbors will be affected by the arrival/departure. The bootstrapping (adding/removing a new node to the ring) is initiated by an operator using the Cassandra web dashboard or a command line utility, but can be executed by any other node in the system. The nodes will also handle transferring of data between nodes by themselves.

Cassandra achieves high availability through replication and each data item is replicated at a given number of hosts. As already mentioned, each node in the cluster is responsible for data items within a given range of the total key-value space. Given a replication factor N, each node is also responsible for replicating all items that fall within its range to all N-1 nodes in the ring. Cassandra uses a system called ZooKeeper to elect a leader amongst its nodes. Whenever a node joins the cluster, it will contact the leader and the leader will inform it about the ranges it is replica for. The leader will also make an effort to make sure that no node is replicating more than N-1 node ranges in the ring. Each node locally caches the metadata about the ranges it is responsible for, as well as it is stored in ZooKeeper in case of node failure. To handle data center failures, Cassandra also replicates each row across multiple data centers which are connected through high speed network links.

Even though both Cassandra and MongoDB have one propriety which they cannot fulfill by the standards of the CAP-theorem, Cassandra does provide eventual consistency, while MongoDB uses replication and failover to provide high availability [7].

7 Discussion

Now that we have presented the two databases and compared them in terms of the CAP theorem, the remaining question is: Which one should you choose? As the CTO of the big corporate business, you know that the new application is likely to generate massive amounts of unstructured data rapidly and the new database must be both scalable and flexible.

Both Cassandra and MongoDB fulfils the partitioning property of the CAP theorem and are built to scale well. Both systems also handle scaling without requiring much administration, as MongoDB provides autosharding and Cassandra has the nodes handle the bootstrapping of new nodes. Cassandra is however the only one specifically designed to handle massive amounts of data.

The application is only expected to generate large amounts data if it is used by many. It is therefore important that the application is both reliable and available to the users, even at a large scale. Both databases provide fault-tolerance and in Cassandra faulty nodes can have their data transferred to other functional nodes and be removed without any downtime. MongoDB handles fault-tolerance using failover and replication, by promoting the slave with the most recent data to master if the original master should fail. They both also claim to be highly available, even though only Cassandra is available by the definition of the CAP theorem. If availability is the most important factor of the application, then Cassandra is the best choice.

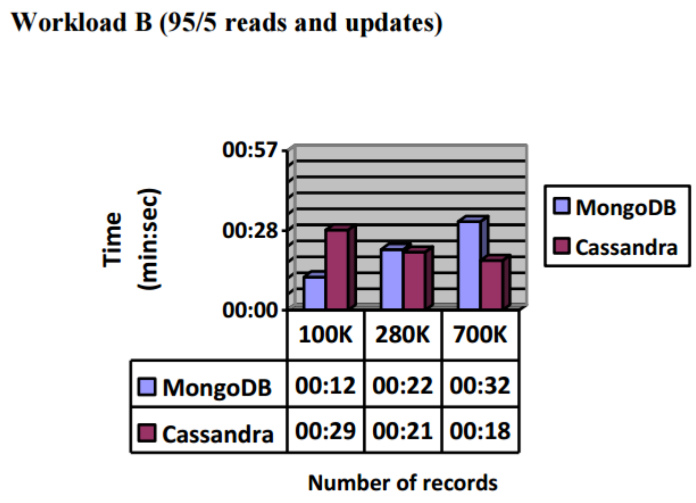
However, depending on the data generated by the users, consistency might also be a desired functionality. In this case, Cassandra will not be a good choice as it has chosen to compromise both latency and consistency for its availability, and only can guarantee eventual consistency. MongoDB on the other hand, is not only able to provide consistency, but it can do so with a variety of consistency levels, making it possible to modify the database consistency to the requirements of the application. If the users should be guaranteed read-your-writes consistency, you should choose MongoDB.

If the application is to become popular amongst the users, the database servers must reply to the clients in a timely manner. In [2] they perform a YCSB benchmark of MongoDB and Cassandra using eight different scenarios (referred to as workloads), which are different combinations of read, write and update operations performed on randomly chosen records.

Tables 1 and 2 show the results of the benchmarking using the two scenarios: 5% reads/ 95% updates and 95% reads / 5% updates. With a 95/5 reads/update mix, the execution time for MongoDB kept increasing with the number of queried records. On the other hand, Cassandra showed a decrease in execution time as the number of queried records increased. At 100,000 records Cassandra was 17 seconds slower than MongoDB, while at 700,000 records, Cassandra had surpassed MongoDB, spending 18 seconds executing, compared to MongoDB's 32 seconds.

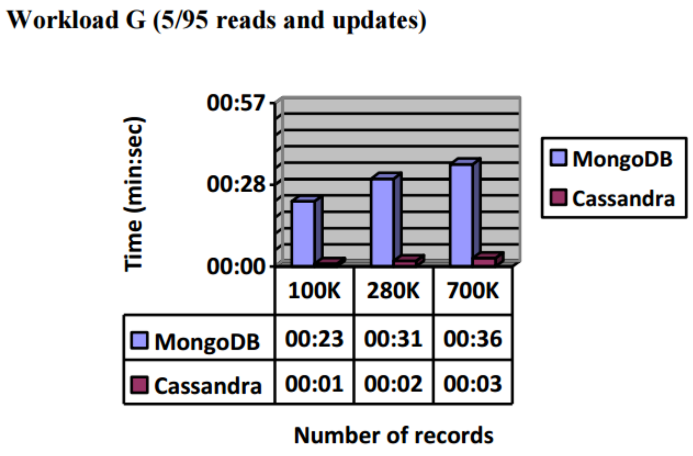
With a 5/95 reads/updates mix, the results show the superiority of Cassandra over MongoDB for all database sizes. Even though both databases kept increasing in execution time as the data volume grew, Cassandra finished executing 12 to 23 times faster than MongoDB, depending on the size of the database volume.

In both cases, Cassandra outperforms MongoDB as the size of the data volume in the database increases, especially if the majority of the client requests are update operations. From a performance perspective, seeing as the application will generate such large amounts of data, Cassandra will be the better choice independent of whether the majority of client requests are read or write.



	100K	280K	700K
MongoDB	00:12	00:22	00:32
Cassandra	00:29	00:21	00:18

Figure 1: YCSB results testing 95% reads and 5% updates



	100K	280K	700K
MongoDB	00:23	00:31	00:36
Cassandra	00:01	00:02	00:03

Figure 2: YCSB results testing 5% reads and 95% updates

8 Conclusion

In this paper we have evaluated two NoSQL databases; Cassandra and MongoDB in terms of the CAP theorem and when they are beneficial to use. Cassandra provides availability and partitioning to its database while MongoDB can provide consistency and partitioning. Cassandra and MongoDB are both great data stores, and there is no right answer to which database you should choose since they provide different approaches and have their advantages and disadvantages in respect to the CAP theorem. If we look at performance, Cassandra outperforms MongoDB when the size of data volume increases in the database, independent of whether the majority of client requests are read or write.

References

[1] D. Academy, "A brief introduction to apache cassandra." `https://academy.datastax.com/resources/brief-introduction-apache-cassandra`. [Online; accessed 20-March-2017].

[2] V. Abramova and J. Bernardino, "Nosql databases: Mongodb vs cassandra," in *Proceedings of the International C* Conference on Computer Science and Software Engineering*, C3S2E '13, (New York, NY, USA), pp. 14–22, ACM, 2013.

[3] IBM, "Why nosql? your database options in the real non-relational world." `https://cloudant.com/wp-content/uploads/Why_NoSQL_IBM_Cloudant.pdf`, March 2015. [Online; accessed 20-March-2017].

[4] Eureka!, "Hbase vs cassandra vs mongodb - choosing the right nosql database." `https://www.slideshare.net/EdurekaIN/no-sql-databases-35591065`, Jun 7, 2014. [Online; accessed 20-March-2017].

[5] A. Lakshman and P. Malik, "Cassandra: A decentralized structured storage system," *SIGOPS Oper. Syst. Rev.*, vol. 44, pp. 35–40, Apr. 2010.

[6] K. Chodorow, *MongoDB: The Definitive Guide: Powerful and Scalable Data Storage.* O'Reilly Media, 2013.

[7] B. K. Sahu, "A real comparison of nosql databases hbase, cassandra and mongodb." `https://www.linkedin.com/pulse/real-comparison-nosql-databases-hbase-cassandra-mongodb-sahu`, Jun 14, 2015. [Online; accessed 20-March-2017].