RESEARCH PROJECT

MONGO DB VS COUCH DB

COMP5338: ADVANCED DATA MODEL

GROUP -7 T20A
AKANKSHA PATIL (490323780)
MOHAMMED SAIF (490278701)
SAMARTH SEHGAL (490528857)

ABSTRACT

In this paper we outlined the NoSQL databases and explained more on their documentation and existing literature. This paper reviews the detailed comparative analysis between two most popular NoSQL databases – MongoDB and CouchDB. The comparative analysis is done with respect to three main features of both databases. Features include storage architecture, CAP trade-off and indexing. The paper is then concluded with recommendations on the type of system to be chosen based on these features.

Contents

INTRODUCTION 3
FEATURE 1 - STORAGE ARCHITECTURE 4
MONGODB 4
COUCHDB 5
COMPARISON OVERVIEW 6
FEATURE 2 - CAP TRADE-OFF 7
MONGODB 8
COUCHDB 9
COMPARISION OVERVIEW 11
FEATURE 3 – INDEXING 11
MONGODB 12
COUCHDB 13
COMPARISION OVERVIEW 14
CONCLUSION 14
Limitations to our research 15
APPENDIX 16
Individual Contribution 16
Table of Figures 16
Table of Tables 16
References 17

INTRODUCTION

NoSQL databases have completely disrupted the computing world. They provide better scalability and performance compared to other alternatives like the traditional relational database system. While relational databases focus on consistency and availability of the data, NoSQL databases focus on partition tolerance and chooses either of consistency or availability. There are several NoSQL databases that are readily available, and each have a different use case scenario (Robin Funck, 2011). They are generally divided into four categories, namely Key-Value Stores, Document Stores, Column Stores and Graph databases (Leavitt, 2010), (A. Schram, 2012), (Cattell, 2010). These are divided in such a way owing to the fact that each kind of database provides a different set of solution for specific contexts. In other words, the "one size fits all" approach of the traditional relational database system does not apply any longer.

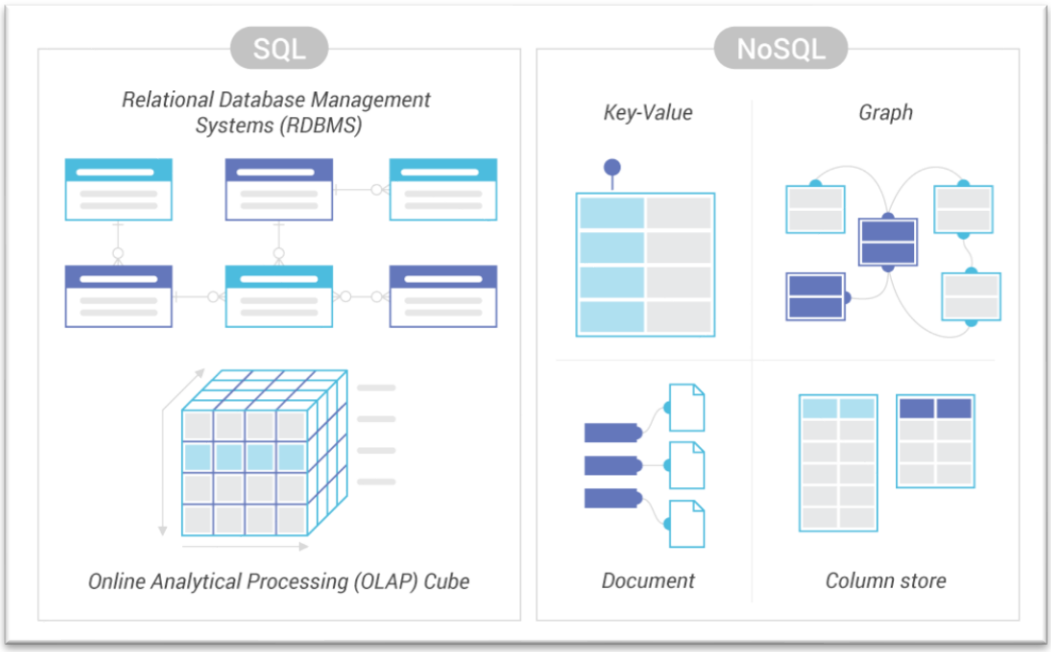

Figure 1: SQL vs NoSQL (ScyllaDB, n.d.)

A document database is usually used to store the document format of semi-structured data along with its detailed description. There are a number of different document-oriented NoSQL databases available, but all have different mechanisms to store and query the data in document format. In our report, we'll be focussing on two such database systems: MongoDB and CouchDB.

MongoDB is an open source NoSQL database developed in C++ and managed by MongoDB Inc. It was developed in 2007 by 10gen. It is a document store database where documents based on their structure are grouped into collections (Veronika Abramova, 2017). These documents are stored in BSON format (Binary JSON). Some of the main features of MongoDB includes indexing, sharding, replication, aggregation, ad-hoc queries, high performance.

CouchDB on the other hand, is also an open source document-oriented NoSQL database. It is implemented in Erlang and makes use of JSON. It was released in 2005 and is developed and managed by Apache Software Foundation. It is a single node database and stores data in the form of collection of documents instead of tables (Rupali Kaur, 2019).

Some of the basic comparative features between MongoDB and CouchDB as described by Rupali Kaur and Jaspreet Kaur are as follows (Rupali Kaur, 2019):

Features	MongoDB	CouchDB
Developmental Languages	C++	Javascript
Database Structure	BSON	JSON
CAP Features	Consistency and Partition Tolerant	Availability and Partition Tolerant
Replication	Master-Slave	Master-Master
Map Reduce	Javascript	HTTP and REST API
Mobile Support	No mobile support	Support for iOS and Android

Table 1: Basic comparative features between MongoDB and CouchDB (Rupali Kaur, 2019).

Based on our research, we decided to conduct further research on three features namely – storage architecture, CAP trade-off and indexing. We will be drawing an in-depth comparative analysis between MongoDB and CouchDB on grounds of these three features.

FEATURE 1 - STORAGE ARCHITECTURE

MONGODB

The data in MongoDB documents is stored as BSON (Binary JSON) (Gupta, 2018), which are Binary encoded JSON like objects. It is a NoSQL, schema free database. Each document in MongoDB has an ID field that can be used as a primary key. MongoDB also provides "multikeys" support for array-type indexes. In MongoDB, documents can further be organized into "collections," which are analogous to tables in RDBMS (W3Resource, 2020). These collections can store different structured documents, owing to their schema-free property. However, in MongoDB, queries and indexes can be made only in one collection at a time.

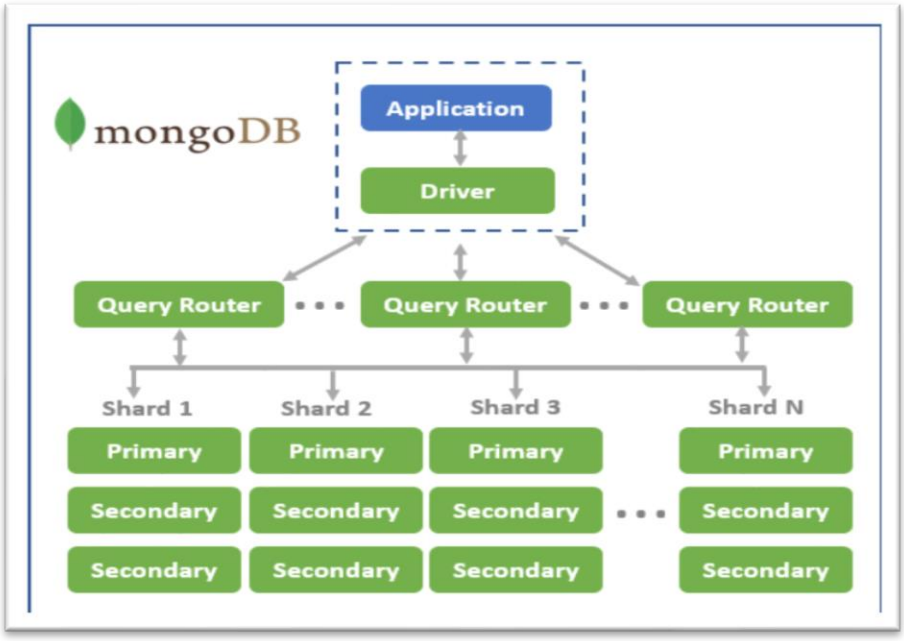

Figure 2: MongoDB Architecture (SeveralNines, 2020)

As can be seen from the diagram, the application interacts with the driver which in turn interacts with queries on multiple servers that communicate with the data distributed on shards and have primary and secondary replicas. Due to usage of multiple servers, MongoDB offers load balancing to maintain the workload between different servers. The architecture of MongoDB is further described in the next section.

Architecture

A cluster in MongoDB consists of three key components (Rabi Prasad Padhy, 2011), that is, the Shard nodes, the configuration servers and the routing services, also known as mongos.

Shard nodes: MongoDB makes use of sharding and shard nodes to support deployments with very large data sets and high throughput operations (MongoDB, 2008). There can be either one node or a pair of replication nodes in each shard. The shards contain the database cores called mongod (Mongo Daemon, the host process for the database).

Configuration servers: Since MongoDB runs on multiple servers, it has various information, such as the metadata and routing information, that needs to be stored. The configuration servers are used to store such information. These can be accessed from the shard nodes as well as the routing services.

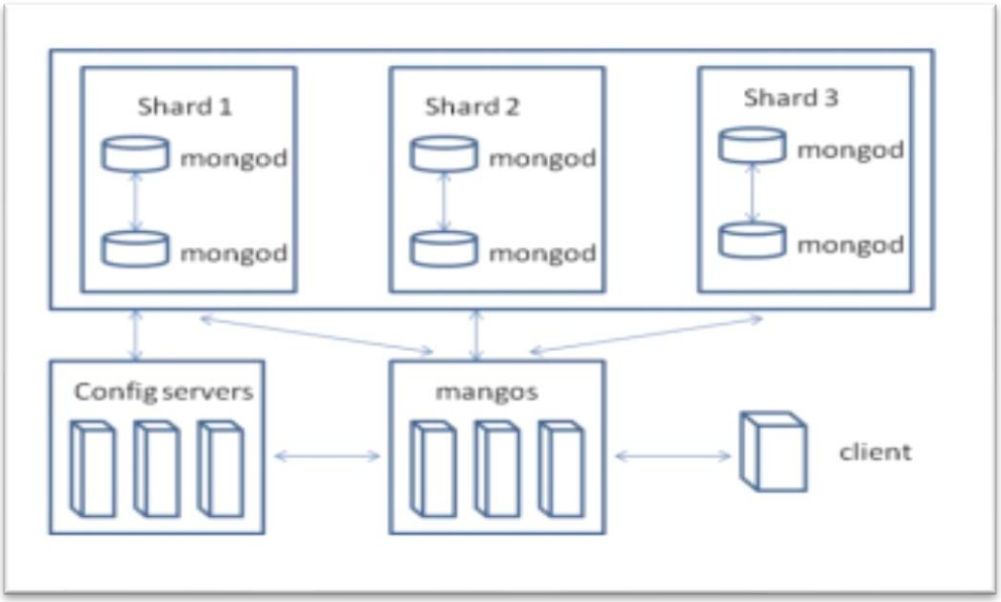

Figure 3: In-detail architecture of a MongoDB cluster (Rabi Prasad Padhy, 2011)

Routing services or mongos: When clients request some tasks, it becomes the core responsibility of the mongos, or the routing services, to perform them efficiently. Owing to their stateless property, mongos can be run in parallel. Depending on the type of task, they interact with the shard nodes and merge all the results before returning them to the client.

MongoDB makes use of **memory mapped files** for storage. This is done to utilize as much of the memory available so as to boost performance of the system. The memory mapped files allow the system's virtual memory management (VMM) tool decide which parts of the database are required to be stored on the disk and which parts are required to be stored in memory. Similar to other databases, the indexes are stored as B-Trees in MongoDB.

COUCHDB

Unlike MongoDB, data in CouchDB is stored as JSON. CouchDB has a document-type structure with schema (in contrast to MongoDB, which is schema free). It is noteworthy to say that these documents are not stored in collections in CouchDB. Unlike relational database, CouchDB does not have data and relationships in the table.

While MongoDB uses master-slave replication, CouchDB makes use of both master-master and master-slave replication and due to this feature (further explained in the CAP Trade-off section), CouchDB provides versioning of the documents. For this reason, all CouchDB documents have unique identifier

as well as a revision id. The revision id updated by CouchDB every time a document is written. CouchDB also makes use of RESTful HTTP API to help applications read, delete and edit documents. The presence of lockless mechanism in CouchDB makes it highly scalable.

The architecture of CouchDB is further explained in the next section.

Architecture

Just like MongoDB has three main components in its structure, CouchDB also has three components, though they differ in their function when compared to MongoDB. The three key components are Storage Engine, View Engine and Replicator (Rabi Prasad Padhy, 2011).

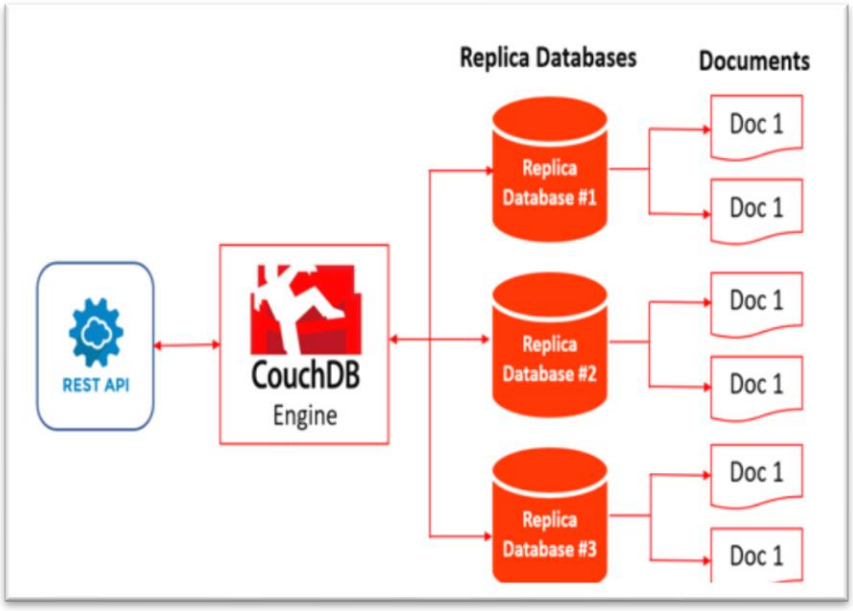

Figure 4: CouchDB Architecture (SeveralNines, 2020)

Storage Engine: The storage engine is the core of the CouchDB system and follows a B-Tree implementation. It manages the internal data of CouchDB, documents and views. The data in CouchDB is accessed by keys or a range of keys that map directly to the underlying B-Tree operations. This direct mapping of the B-Tree operations improves the speed of the system significantly.

View Engine: The view engine in CouchDB helps it create views that are made out of the MapReduce jobs. The design documents contain the definitions of the views. In CouchDB, indices are created using these views, which is further explained in the Indexing section of this report. The view engine is written in JavaScript and is based on Mozilla SpiderMonkey. When a user reads data in a view, CouchDB makes sure that the data is up to date.

Replicator: The replicator serves a very important purpose of replicating data to a remote or local database in CouchDB. It also helps in synchronizing the design documents.

COMPARISON OVERVIEW

Having studied storage architectures of both types of the system, what database system should one prefer? Although it totally depends on the readers personal preference, it is recommended that:

1. If one is making transition from a relational database to a NoSQL database, MongoDB might be a better option. This is because CouchDB, although offering a schema, does not offer data and relationships in a table. So, it becomes easier to choose a schema free database like MongoDB instead.

2. In general, CouchDB, since it uses JSON, is faster, efficient and offers less computation time as well as a smaller data size. So, in all other cases, CouchDB can be preferred.

FEATURE 2 - CAP TRADE-OFF

The CAP theorem is a trade–off between consistency, availability and partition tolerance. It implies that a system can only have at most two of three properties i.e. availability, consistency or partition tolerance. In case of a network partition, a system has to choose between consistency and availability (Kleppmann, 2015). There can be a critical condition in the system in which system must give up either its availability or its consistency.

Consistency states that all the clients see the same view of data right after any update or write operation is performed. If all the clients see the same data after any update operation, the system is considered to be highly consistent.

Availability states that the system is successfully able to respond to a client request even in case of any hardware failure. If the system is highly available, clients can find at least one response even if some of the machines are down.

Partitioning indicates the system's ability to act when the nodes are dynamically added or removed from the database. It defines how the system acts in case of network partitions.

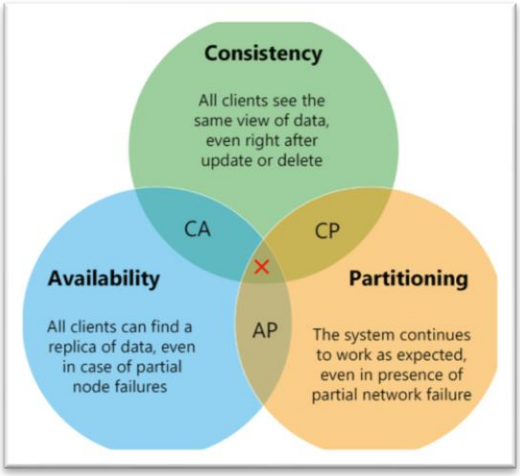

Figure 5: CAP Theorem (Hamzeh Khazaei, 2015)

Different databases fit in different views of CAP theorem. Below is the list for the same.

Type	Databases
CA (Consistency and Availability)	Redis, RDBMS, HBase, Memcached
AP (Availability and Partition Tolerance)	Cassandra, Dynamo, CouchDB, Voldemort, Elastic Search
CP (Consistency and Partition Tolerance)	MySQL, Postgres, Bigtable, MongoDB

Table 2: CAP Theorem and Databases

Both MongoDB and CouchDB have different approaches to CAP theorem. MongoDB favours consistency and partition tolerance while CouchDB on the other hand favours availability and partition tolerance.

MONGODB

As already discussed, MongoDB has a master-slave architecture which indicates that it has only one primary node and all the other nodes are secondary nodes. The primary node is responsible for all the read and write operations. As only one node handles all the read and write operations, the system becomes highly consistent. The secondary nodes duplicate the changes of primary nodes and apply it to themselves (IBM Cloud Education, 2019).

MongoDB uses a "strict consistency" model for replication. It makes use of replica set to provide redundancy in case of any hardware or system failures. The replica set represents the whole collection.

There are two cases that can happen in case of any hardware or network failure, either the primary node fails, or the secondary nodes are not able to duplicate the changes of the primary nodes (Bhamra, 2017).

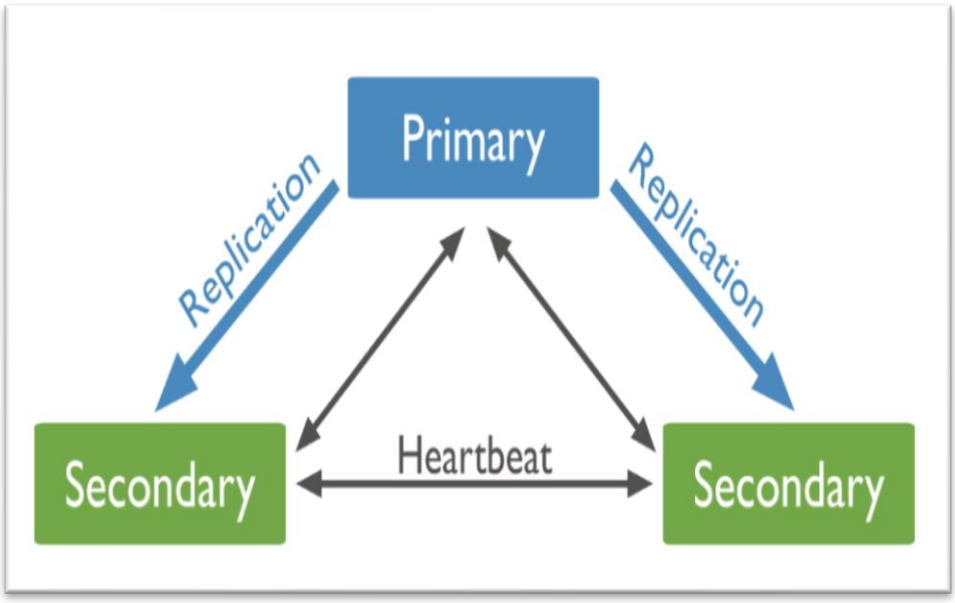

Figure 6: Replication in MongoDB (Katwal, 2019)

In the first case, if the primary node fails or is partitioned from the system, it stores all the writes that have been sent to the primary node but not yet propagated to the secondary nodes in a local rollback directory. In such a case, system elects a new primary node to be responsible for all the reads and writes. The new node elected as the primary node is the secondary node which has the most recent transaction. This makes the system unavailable to handle any requests as there is inconsistency between the new primary node and the old primary node until the system repair from the failure (Abadi, 2009). The system becomes unavailable until all the nodes catch up with the newly elected primary node. As there cannot be any write operation during this time, the data is consistent across the entire network of systems.

In the other case, i.e. if the secondary nodes are not able to duplicate the primary nodes' changes, the data itself is not available to access. This makes the system unavailable as it cannot process the client's requests.

In both the above cases, the availability of the system is sacrificed. This implies that in case of a partition, MongoDB is not highly available (IBM Cloud Education, 2019). Hence, MongoDB is referred to as a consistent and partition tolerant system, i.e. it favours C (consistency) and P (partition tolerance) of the CAP theorem.

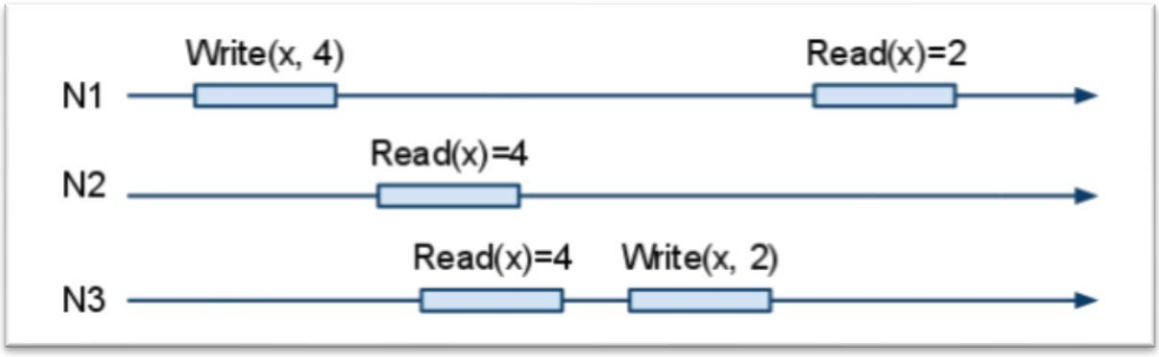

Figure 7: Strict Consistency in MongoDB (Unge, 2011)

Node 1 updates the value of x as 4. Node 2 and node 3 reads the value of x as 4. Node 3 again updates the value of x as 2. When node 4 now performs the read operation, it gets the value of x as 2 i.e. it's latest value.

To maintain consistency, MongoDB provides traditional update-in-place mechanism using locks. If first client wants to perform an operation, it will get the lock and everybody else have to wait. Second client can only perform an operation after the first client's request has been processed. This delivers a very high write performance especially in case of updates. If the data is dumped in high volume or there is a high update of multiple objects, MongoDB fits in well.

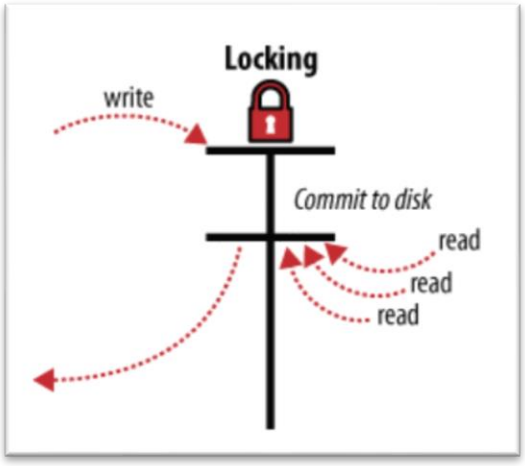

Figure 8: Locking in MongoDB (Apache CouchDB, n.d.)

COUCHDB

Although MongoDB uses "Strict Consistency" model, CouchDB provides the simplest form of replication, that means it uses an "Eventual Consistency" model. This model is considered to be the weaker form of consistency model. It implies that the updated object will have replicated to all the nodes in the database and any access request will provide the latest update. However, because of network delays, there might be inconsistencies in the nodes (Unge, 2011).

Unlike MongoDB, in this model, the data can be written to a single database node. This can be done without waiting for other nodes to come into agreement (Gorst, 2014). The data is propagated to the rest of the DB (Dearmer, 2020) implying that they will be synchronized eventually. The nodes are self-sufficient and independent, leaving no single point of disagreement across the system (O'Reilly, n.d.). The master-master and master-slave architecture in CouchDB ensures that the users can access the data from various different locations even without the slightest delay.

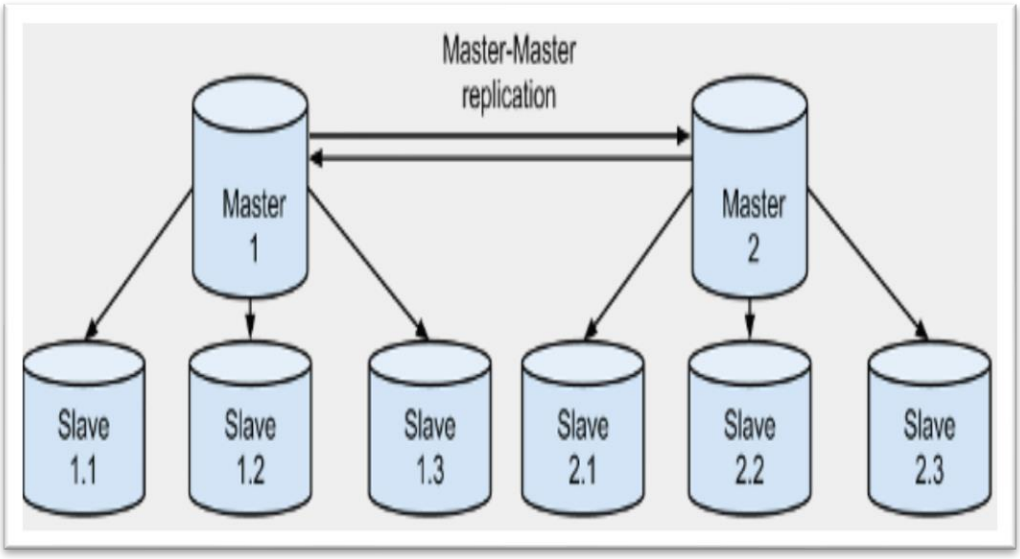

Figure 9: Master-master and master-slave replication in CouchDB (Alamin, 2016)

Due to the time between performing the update and the update being reflected on all nodes, inconsistency in the replicas may exist. Because of this, the, the result returned to the client might not be the latest one. Also, two different nodes might have different information for the same request. This implies that consistency is sacrificed for availability. Hence, CouchDB is referred to as an available and a partition-tolerant system, i.e. it favours A (availability) and P (partition tolerance) of the CAP theorem.

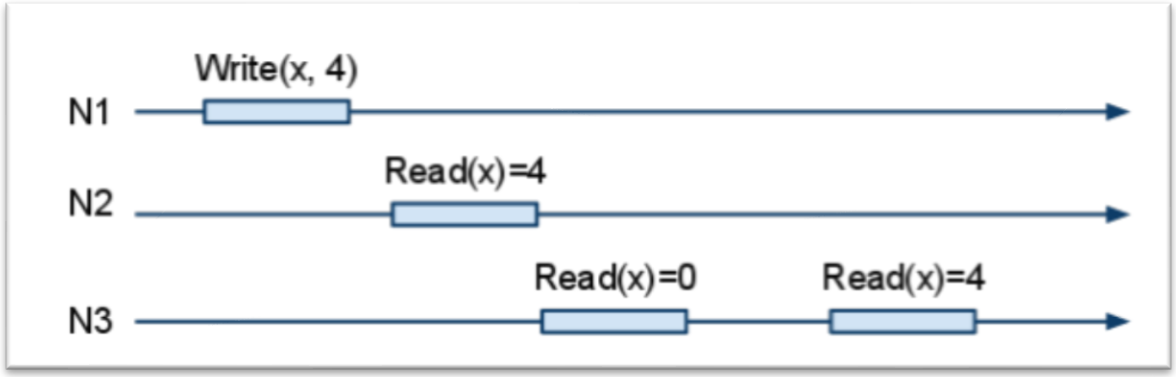

Figure 10: Eventual Consistency in CouchDB (Unge, 2011)

As shown in the above figure, the initial value of x is 4 as updated by node 1. When node 2 performs the read operation, it reads the value of x as 4, its latest value. Node 3 reads the value of x as 0 which is the old value, but when node 3 performs another read operation, it gets the value of x as 4, i.e. its latest value.

Unlike MongoDB's update-in-place method, CouchDB uses MVCC i.e. multi version concurrency control to provide concurrent access to the database. In MVCC, the data is updated by creating its newer versions. That means if there is an update operation, the old data is not overwritten or deleted, instead the new data is added. This makes the old data obsolete. This old data can obviously be deleted later. MVCC is suitable in case of master-master slave architecture, where the replication is needed for a large amount of data. In case of any failure in MVCC, it needs manual handling of data unless there is a locking mechanism implemented. (AUEssays, 1970)

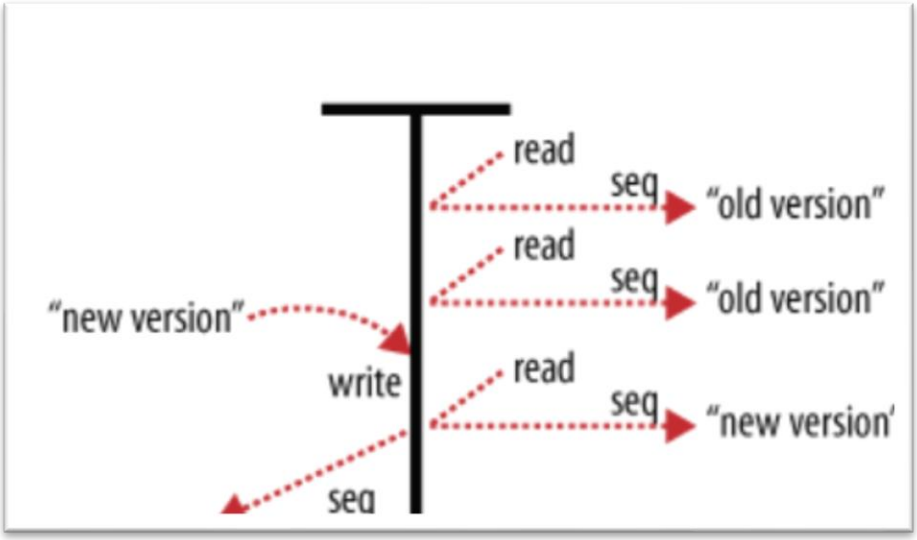

Figure 11: MVCC in CouchDB (Apache CouchDB, n.d.)

In case of multiple requests to access a document, if the second request changes the document when the first request is reading it, CouchDB will create a newer version of that document without even waiting for its first request to finish. Now, if the third request comes in, it will be redirected towards the newest version of the document by CouchDB. Whereas, the first request would still be reading the older version of document. This implies that the read request can see the document that was there at the beginning of the request (Apache CouchDB, n.d.).

COMPARISION OVERVIEW

A system neither guarantees nor provides 100 percent consistency, the system might be unavailable at times or the data can be out-of-date. Some systems don't want to sacrifice availability in any case. They want their systems to be responsive all the time. But if the system is spread across vast locations, the level of availability will be deteriorated. The designers sacrifice consistency in such situations as they want a response even if its incorrect. On the other hand, it may take time for the content to propagate through all the servers to guarantee consistency (Seth Gilbert, 2000).

MongoDB can always be consistent depending on how you configure your data, but it cannot always be available. It can always be available for read operations but not for write operations because the system becomes unavailable in case of election of primary nodes or disconnection between primary node and secondary node (Katwal, 2019). Couch DB on the other hand will always be available because of its master-master slave architecture where each node acts as a master node.

As mentioned above, MongoDB provides traditional Update-in-place while CouchDB uses MVCC (Multi version concurrency control). In case of high-performance requirements and very huge datasets, MongoDB is preferable over CouchDB.

So, if you are looking for a highly consistent system, for example, when performing some trades, MongoDB should be your choice. But if you want a highly available system, CouchDB fits in best even if you want to access the out-of-date data.

FEATURE 3 – INDEXING

Indexes are used to accelerate query computation and reduce the overall time required to compute a query. Indexes help in quickly locating the row that contains the data without having to scan the entire database. It works by storing partial data (say only name column in employee database) as a key which

links to the rest of the data in that row(Employee ID, DOB, Designation etc fields). Indexes improve the performance of read queries and are therefore ideal for scenarios where frequent read queries are performed. The drawback of adding indexes is that it reduces the write query performance.

Index storage

In both – MongoDB and CouchDB, index fields are stored in the form of BTrees. The column data is stored in the leaf nodes and the non-leaf nodes have index pointers. BTrees can be traversed using binary search in O(Log(n)) time and which makes them the obvious choice for index storage.

(MongoDB, 2008)

MONGODB

According to MongoDB documentation, MongoDB defines indexes at collection level and supports indexes on any field or sub-field of the documents. MongoDB supports indexing and allows you to create indexes on different types of data. These indexes are physically stored in key value pairs inside the collection. The different types of indexes that are available on MongoDB are as follows- Compound, Multikey, Geospatial, Text and Hashed indexes.

Multikey index can be used to speed up queries for databases where array data is embedded in the documents. It works by creating separate index entries for every element in the array. The creation of multikey index is handled automatically by MongoDB.

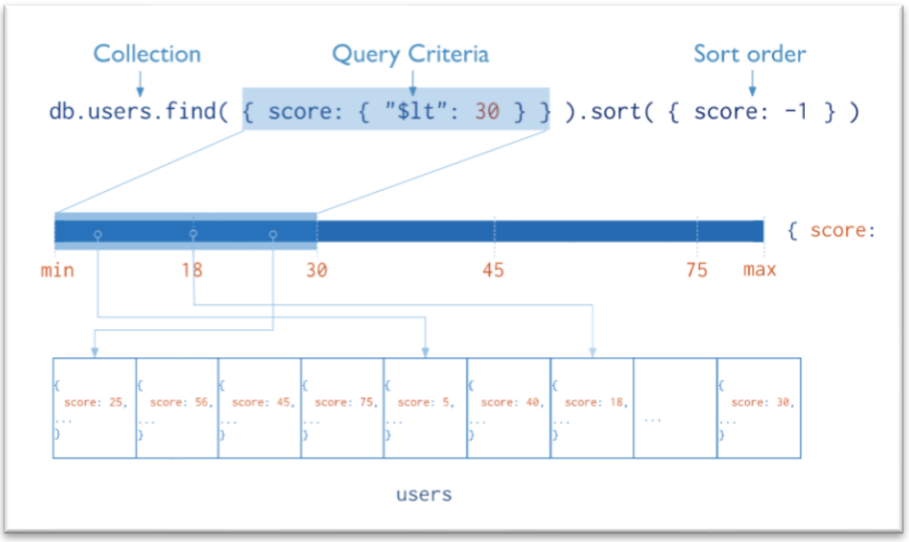

Figure 12: Indexing in MongoDB (MongoDB, 2008)

If the DB requires indexes that are complex in nature, then a compound index can be used as it can be user-defined. Geospatial indexes accelerate queries on geospatial data and work by either using 2d indexes deploying planar geometry or 2dsphere indexes that run on spherical geometry. MongoDB also allows users to search on text string by supporting the creation of text index on text data. These indexes are created from tokenised root words. However, these indexes are generally larger in size and should only be created when there is a large enough difference in performance when implemented. Hashed indexes support hash based sharding and therefore cannot work with range-based queries.

As per our research, we realised that that there is no automatic creation of indexes in MongoDB. One needs to first determine the fields/columns in the document that will be queries by read queries frequently. Once the columns are decided, the user then needs to explicitly create indexes. Indexes

accelerate read performance but at the cost of write queries. Therefore, a performance trade off needs to be considered before adding indexes.

In DBs where there are a balanced number of read and write queries, indexes may do more harm than good. In these use cases, MongoDB's query optimiser reduces the computation time without the presence of an index. It functions but comparing different candidate plans and choosing the one that needs the least amount of work. This query plan is cached so that subsequent queries of similar shape can benefit from it.

Index during insert operations

Index needs to be updated whenever there is an update/ insert operation in the DB. MongoDB updates the index, but depending on the number of indexes, the time required to update the index may vary. It performs the update by traversing the BTree using binary search. For applications where there are fewer writes than reads, there should be minimal degradation in performance. However, it should be noted that a read request right after a write request can face a performance impact. MongoDB has the option to set the write concern level which can help in these cases. Best practice for creation of index is when the data is loaded onto the DB.

COUCHDB

CouchDB doesn't directly have the concept of Indexes but does support it via a different implementation. The DB consists of all the documents and in order to query the documents, views need to be designed. Views are logical windows that are built on top of the documents in the DB. These views are created via JavaScript map function, which accepts key and value as parameters. The records in the view are ordered according to the key and the queries need to be run on top of these views. These sorted keys provide the index functionality in CouchDB. Views are built when a query is run on them and remain unchanged until the underlying data changes. After the first query is run, a BTree is created for all the keys in that view. This BTree accelerates the queries. Multiple views can lead to the creation of multiple keys pointing to the same data, which can have storage implications. The map function can be customized to write customizable queries.

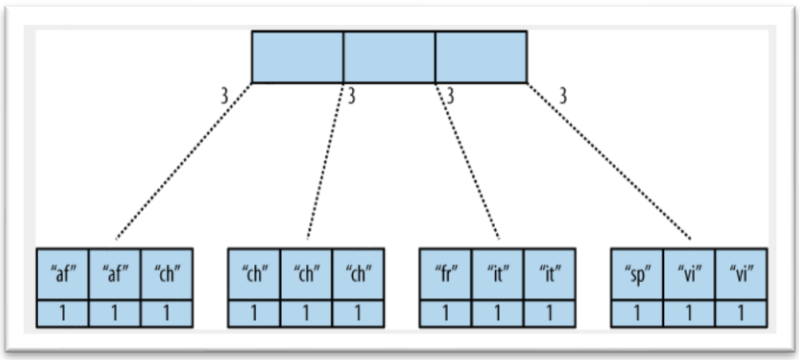

Figure 13: Indexing in CouchDB (Apache CouchDB, 2020)

Unlike MongoDB, initial CouchDB didn't have native support for text, geospatial, multikey and compound indexes. It has plug ins that were released later on to support creation of text and geospatial indexes. Subsequent versions of CouchDB developed by cloudant allow the creation of multikey indexes on Mango.

Since the indexes are created automatically, the user doesn't have to worry about explicitly creating them. Moreover, different views can be designed on top of the data which focus on the performance of different columns. Query performance is instantaneous when it's performed on a view that has already been built. This can be compared to cached query plans of MongoDB.

Index during insert operations

Views are built on underlying data. When new data is added, it doesn't necessarily update the view. The view is updated when a query is run on it. When a read query is run after new data is inserted, its execution time is higher. This is because the insert has triggered a view update, which is time consuming. However, if different views are accessing the same columns, there may be a possibility that some of them don't have the updated version of data simply because they haven't been queried yet, resulting in different index values across the views.

COMPARISION OVERVIEW

MongoDB stores indexes in the same DB alongside all the collections and updates it whenever new data is inserted. This update happens in memory. When it comes to read after write performance, MongoDB performs better than CouchDB (Yacine Atif, 2008). This can be partly attributed to the option of setting write concern level in MongoDB. The benefit of views in couch DB is that new data can be added onto the couch and if it doesn't contain fields pertaining to a certain view then it doesn't trigger an update (Matthew R. Hanlon, 2011).

MongoDB allows users to create indexes on null as well as embedded documents (Robin Henricsson, 2011). However, CouchDB excels in applications where read and write queries are balanced, and it's automatic indexing works well in those environments (Yacine Atif, 2008).

Geospatial datasets benefit greatly from geospatial indexes. MongoDB does support these types of indexes, but CouchDB natively doesn't support it. Couchbase, a hosted CouchDB from cloudant allows this through Mango libraries.

When new documents are added in CouchDB, it finds all the rows of data in the view that were affected by the update and marks them invalid. Once the document is run through the map function, it simply adds the new line data in the Btree in its intended spots (Apache CouchDB, 2020). MongoDB on the other hand drops the entire index and creates a new index whenever new data in inserted (MongoDB, 2008).

If there is a need for a general-purpose application with balanced read and writes, CouchDB should be preferred. CouchDB doesn't need explicit creation of indexes and offers better execution speeds because of the implementation of views. On the other hand, If the application warrants faster read queries and has fewer write queries, then MongoDB is the better option. It is also more suited for DBs where type of queries is known, because those can benefit from explicitly created indexes.

CONCLUSION

Despite being document stores, the systems differ in the way they store the data. CouchDB stores it in the form of JSON while MongoDB stores it in BSON, which is a subset of JSON. They also differ in the way they replicate data among servers- Master-Master or Master-Slave. The choice of DB depends on the requirements of the system. MongoDB queries are similar to SQL in structure and individuals familiar with relational databases will find it easier to work with. On the other hand, CouchDB stores data in JSON which is widely popular and de facto standard for data representation.

Another aspect to consider when choosing DB systems is to consider the CAP trade-offs. Applications where consistency is important, for example disaster management, it would be advisable to store the data on a MongoDB system as it updates in place. However, it should be noted that different domains have different requirements, and some of them might need higher availability. Verizon is a Telecom company and due to the nature of its industry has high availability requirements. Therefore, they store their data on CouchDB. (HGInsights, 2020).

Lastly, if you're an expert DB admin and you're building a system with known query types, you'd prefer a system that enables you to do just that. MongoDB is one such system that allows you to create indexes on fields which will accelerate queries in a read heavy DB. However, if your data size is relatively small and you're in need of a general-purpose DB with balanced queries, CouchDB will work well.

Limitations to our research

All the research that we have conducted is based on the existing literature for MongoDB and CouchDB. We have not practically run the queries to test the existing findings. Most of the literature that we have cited is based on scholarly articles till 2019, so it does not cover any changes that might have happened in year 2020.

APPENDIX

Individual Contribution

Name	Student ID	Feature worked on	Contribution
Samarth Sehgal	490528857	Storage Architecture	33.33%
Akanksha Patil	490323780	CAP Trade-off	33.33%
Mohammed Saif	490278701	Indexing	33.33%

Table 3: Individual Contribution

Even though we assigned individual ownerships for each of the features, everyone collaborated together on the entire report. We all put in efforts in researching the documentation and existing literature for every feature.

Table of Figures

Figure 1: SQL vs NoSQL (ScyllaDB, n.d.) 3
Figure 2: MongoDB Architecture (SeveralNines, 2020) 4
Figure 3: In-detail architecture of a MongoDB cluster (Rabi Prasad Padhy, 2011) 5
Figure 4: CouchDB Architecture (SeveralNines, 2020) 6
Figure 5: CAP Theorem (Hamzeh Khazaei, 2015) 7
Figure 6: Replication in MongoDB (Katwal, 2019) 8
Figure 7: Strict Consistency in MongoDB (Unge, 2011) 9
Figure 8: Locking in MongoDB (Apache CouchDB, n.d.) 9
Figure 9: Master-master and master-slave replication in CouchDB (Alamin, 2016) 10
Figure 10: Eventual Consistency in CouchDB (Unge, 2011) 10
Figure 11: MVCC in CouchDB (Apache CouchDB, n.d.) 11
Figure 12: Indexing in MongoDB (MongoDB, 2008) 12
Figure 13: Indexing in CouchDB (Apache CouchDB, 2020) 13

Table of Tables

Table 1: Basic comparative features between MongoDB and CouchDB (Rupali Kaur, 2019). 4
Table 2: CAP Theorem and Databases 7
Table 3: Individual Contribution 16

References

- A. Schram, K. M. A., 2012. *MySQL to NoSQL: data modeling challenges in supporting scalability.* [Online]
  Available at: https://www.semanticscholar.org/paper/MySQL-to-NoSQL%3A-data-modeling-challenges-in-Schram-Anderson/e873fea9fce1a0ae7383f6678980c14d84ff2456

- Abadi, D. J., 2009. *Consistency Tradeoffs in Modern Distributed Databse System Design.* [Online]
  Available at: http://www.cs.umd.edu/~abadi/papers/abadi-pacelc.pdf

- Alamin, 2016. *Why do we need CouchDB?.* [Online]
  Available at: https://imalamin.wordpress.com/tag/couchdb/

- Apache CouchDB, 2020. *Introduction to Views.* [Online]
  Available at: https://docs.couchdb.org/en/stable/ddocs/views/intro.html

- Apache CouchDB, n.d. *Eventual Consistency.* [Online]
  Available at: https://docs.couchdb.org/en/latest/intro/consistency.html

- AUEssays, 1970. *Comparision Between Mongodb And Couchdb Information Technology Essay.* [Online]
  Available at: https://www.auessays.com/essays/information-technology/comparision-between-mongodb-and-couchdb-information-technology-essay.php#:~:text=One%20big%20difference%20between%20MongoDB,transactional%20memory%20in%20programming%20languages.

- Bhamra, K., 2017. *A Comparative Analysis of MongoDB and Cassandra.* [Online]
  Available at: http://bora.uib.no/bitstream/handle/1956/17228/kb-thesis.pdf?isAllowed=y&sequence=1

- Cattell, R., 2010. *Scalable SQL and NoSQL data stores.* [Online]
  Available at:
  https://www.researchgate.net/publication/220415613_Scalable_SQL_and_NoSQL_data_stores

- Dearmer, A., 2020. *CouchDB vs. MongoDB: What You Need to Know.* [Online]
  Available at: https://www.xplenty.com/blog/couchdb-vs-mongodb/

- Gorst, D., 2014. *MongoDB vs CouchDB.* [Online]
  Available at: https://blog.scottlogic.com/2014/08/04/mongodb-vs-couchdb.html#:~:text=CouchDB%20uses%20a%20replication%20model,will%20eventually%20be%20in%20sync.

- Gupta, K., 2018. *COUCHDB VS MONGODB: UNDERSTANDING THE DIFFERENCE.* [Online]
  Available at: https://www.freelancinggig.com/blog/2018/04/19/couchdb-vs-mongodb-understanding-difference/

- Hamzeh Khazaei, M. F. S. Z. N. B., 2015. *How do I choose the right NoSQL solution? A comprehensive theoretical and experimental survey.* [Online]
  Available at:
  https://www.researchgate.net/publication/282679529_How_do_I_choose_the_right_NoSQL_solution_A_comprehensive_theoretical_and_experimental_survey

- HGInsights, 2020. *CouchDB.* [Online]
  Available at: https://discovery.hgdata.com/product/couchdb
- IBM Cloud Education, 2019. *CAP Theorem.* [Online]
  Available at: https://www.ibm.com/cloud/learn/cap-theorem
- Katwal, B., 2019. *What is the CAP Theorem? MongoDB vs Cassandra vs RDBMS, where do they stand in the CAP theorem?.* [Online]
  Available at: https://medium.com/@bikas.katwal10/mongodb-vs-cassandra-vs-rdbms-where-do-they-stand-in-the-cap-theorem-1bae779a7a15
- Kleppmann, M., 2015. *A Critique of the CAP Theorem.* [Online]
  Available at:
  https://www.cl.cam.ac.uk/research/dtg/www/files/publications/public/mk428/cap-critique.pdf
- Leavitt, N., 2010. *Will NoSQL Databases Live Up to Their Promise?.* [Online]
  Available at: https://ieeexplore.ieee.org/document/5410700
- Matthew R. Hanlon, R. D. S. M. D. P. N. P. H., 2011. *A Case Study for NoSQL Applications and Performance Benefits: CouchDB vs. Postgres.* [Online]
  Available at: https://www.diva-portal.org/smash/get/diva2:832580/FULLTEXT01.pdf
- MongoDB, 2008. [Online]
  Available at: https://docs.mongodb.com/manual/indexes/
- MongoDB, 2008. *Indexes.* [Online]
  Available at: https://docs.mongodb.com/manual/indexes/
- MongoDB, 2008. *Manage Indexes.* [Online]
  Available at: https://docs.mongodb.com/manual/tutorial/manage-indexes/
- MongoDB, 2008. *Sharding.* [Online]
  Available at: https://docs.mongodb.com/manual/sharding/
- O'Reilly, n.d. *Eventual Consistency.* [Online]
  Available at: https://www.oreilly.com/library/view/couchdb-the-definitive/9780596158156/ch02.html
- Rabi Prasad Padhy, M. R. P. S. C. S., 2011. *RDBMS to NoSQL: Reviewing Some Next-Generation Non-Relational Database's.* [Online]
  Available at:
  https://liacs.leidenuniv.nl/~stefanovtp/courses/StudentenSeminarium/Papers/DB/3.IJAEST-Vol-No-11-Issue-No-1-RDBMS-to-NoSQL-Reviewing-Some-Next-Generation-Non-Relational-Database's-015-030.pdf
- Robin Funck, S. J., 2011. *NoSQL evaluation: A use case oriented survey.* [Online]
  Available at:
  https://www.researchgate.net/publication/254048347_NoSQL_evaluation_A_use_case_oriented_survey
- Robin Henricsson, G. G., 2011. *A comparison of performance in MongoDB and CouchDB using a Python interface..* [Online]
  Available at: https://www.diva-portal.org/smash/get/diva2:832580/FULLTEXT01.pdf

- Rupali Kaur, J. K. S., 2019. *A review of comparison between NoSQL Databases: MongoDB and CouchDB.* [Online]
  Available at: https://www.ijrte.org/wp-content/uploads/papers/v7i6s/F03820376S19.pdf
- ScyllaDB, n.d. *NoSQL vs SQL.* [Online]
  Available at: https://www.scylladb.com/resources/nosql-vs-sql/
- Seth Gilbert, N. A. L., 2000. *Perspectives on the CAP Theorem.* [Online]
  Available at: https://groups.csail.mit.edu/tds/papers/Gilbert/Brewer2.pdf
- SeveralNines, 2020. *The Battle of the NoSQL Databases - Comparing MongoDB and CouchDB.* [Online]
  Available at: https://laptrinhx.com/the-battle-of-the-nosql-databases-comparing-mongodb-and-couchdb-1101513385/
- Unge, S., 2011. *Implementing a eventual consistency job distribution with CouchDB.* [Online]
  Available at: http://uu.diva-portal.org/smash/get/diva2:431609/FULLTEXT01.pdf
- Veronika Abramova, J. B., 2017. *NoSQL Databases: MongoDB vs Cassandra.* [Online]
  Available at: https://www.silvercross.is/wp-content/uploads/2017/03/p14-abramova.pdf
- W3Resource, 2020. *Databases, documents and collections in MongoDB.* [Online]
  Available at: https://www.w3resource.com/mongodb/databases-documents-collections.php#:~:text=A%20collection%20is%20analogous%20to,of%20data%20in%20a%20database.
- Yacine Atif, H. G., 2008. *A COMPARISON BETWEEN MONGODB & COUCHDB ON SEARCH PERFORMANCE.* [Online]
  Available at: https://www.diva-portal.org/smash/get/diva2:1215436/FULLTEXT01.pdf