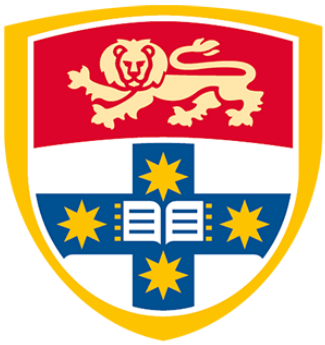
ADVANCED DATA MODELS
ASSIGNMENT 3

Lecturer: Ying Zhou

Tutor: Chenhao Huang



THE UNIVERSITY OF
SYDNEY

The Document Data Model as implemented in MongoDB and CouchDB: A comparative study

Anshu Kumar (490517666)
Geogy Sabu Jose (490556492)
Sanna Nazir (490517677)

November 2020

Contents

1 Abstract

As famously said by Clive Humby, "Data is the new oil." The modern world runs on data and its immense potential when refined, cleaned and analyzed. With each passing day, this data keeps on increasing and taking various forms. These include the basic structured data to the more recent forms of unstructured data such as document stores, graphs, images, etc. An important step to reaching the treasures that data can unlock is the modelling and storage of this data. It is important to create various different data models that can provide unique ways of analysis and handling. These models are then implemented across various diverse database systems. The purpose of this report is to analyze one such data model: the document store. Followed by a comparison of two database systems (MongoDB and CouchDB) that implement the said data model. The report is based on an integrated literature review and identifies key differences across the two. It also postulates our understanding of the disparities and the potential reasons and design choices for the same. The report concludes with an overarching reflection of the literature analyzed.

2 Introduction

2.1 Aims/Objectives

The aim of this study is driven by the objectives for Assignment 3 of study unit COMP5338 (Advanced Data Models) taught at the University of Sydney during Semester 2, 2020. The primary objective of the same is to analyze a post-relational data model and investigate two database systems that support the said model. The primary aim to encourage the student to perform an in-depth investigation of a post-relational data model, its implementation across systems and to draw a comparison between the two.

2.2 Data Model

For the purpose of our study, we have chosen the Document Data Model. Also known as the Document Store Database Model.

Document oriented databases started as a subclass of key value databases [9]. In time, they have become a NoSQL database class on its own due to the fact that they can store more complex data types [9]. A document data model is comprised of the simplest element: a document. A document is may be implemented in different ways across different database systems. However, commonly used encapsulation methods include JSON (JavaScript Object Notation) format, XML (eXtensible Markup Language) format, YAML (YAML Ain't Markup Language) format etc. To put it simply, a document stores data with some form of a header/label that can then be used for retrieval and analysis. However, unlike relational data storing, it does not use rows, columns and tables to store this data. Data is stored in a much more free-flowing format that allows for immense flexibility and thus, drives the popularity of this model for modern unstructured data storage. Another flexibility feature related to the way data is stored is that a document can contain other nested documents [31].

The choice of this data model was guided by its ease of use and understanding, its universality, its flexibility and its significant potential for storage of unstructured text data.

Customer Document

```
"customer" =
{
   "id": "Customer:1",
   "firstName": "John",
   "lastName": "Wick",
   "age: 25,
   "address": {
        "country": "US",
        "city": "New York",
        "state": "NY"
        "street": "21 2nd Street",
      },
   "hobbies": [ Football, Hiking ],
   "phoneNumbers": [
      {
        "type": "Home",
        "number": "212 555-1234"
      },
      {
        "type": "Office",
        "number": "616 565-6789"
      }
    ]
}
```

Figure 1: A sample Document used in a document store data model) *Image sourced from Google Images*

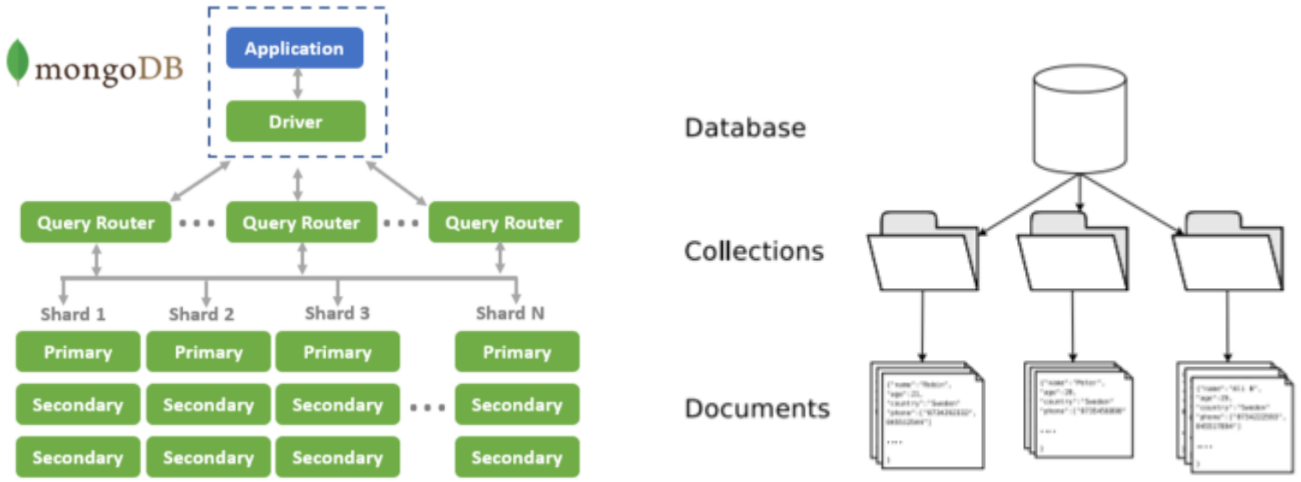
2.3 Database Systems

For the purpose of our study, we have chosen the MongoDB and CouchDB Database systems and consequently compare and analyze their implementation of the document data model.

2.3.1 MongoDB

As described on their website: MongoDB is a general purpose, document-based, distributed database built for modern application developers and for the cloud era [22]. The MongoDB database system follows the Database-As-A-Service (DBaaS) business model. Its community edition with limited features is a public, open-source software that can be used by one and all. However, the enterprise product with enhanced features is a paid product that can be used by organizations as a computing service.

MongoDB stores data in flexible, JSON-like documents, meaning fields can vary from document to document and data structure can be changed over time [22]. It implements the document store model using BSON-based documents. These documents are the elementary data storage essentials and allow for metadata which can then be used for querying and retrieval. These documents are aggregated at a collection level. A Collection consists of multiple documents that may share some similar characteristics. Further, collections are aggregated to the database level. Thus, each database can consist of multiple collection that in turn may consist of multiple documents within them.



(a) MongoDB Architecture

(b) MongoDB Document Model Implementation

Figure 2: The MongoDB Database (Images sourced from [35])

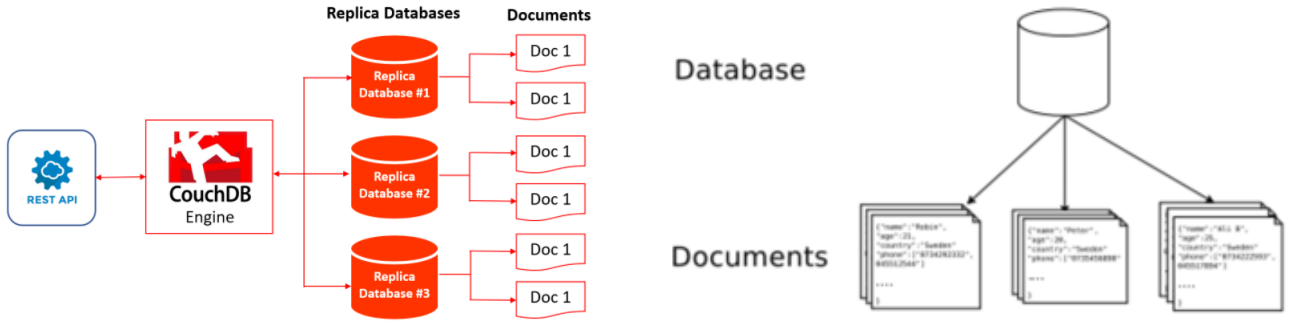
Some key features of this database are as below [22]:

- High availability through built-in replication and failover
- Horizontal scalability with native sharding
- End-to-end security
- Native document validation and schema exploration with Compass
- Management tooling for automation, monitoring, and backup
- Fully elastic database as a service with built-in best practices

2.3.2 CouchDB

As described on their website: CouchDB is an open-source, document-oriented NoSQL DB, providing seamless multi-master sync, that scales from Big Data to Mobile [5]. The CouchDB database also follows the DBaaS model. It can used as a single-node database or even as a clustered cloud database system. A CouchDB cluster improves on the single-node setup with higher capacity and high-availability without changing any APIs [5]. Unlike MongoDB, CouchDB is completely open-source and free to use. It can be easily integrated with existing database management systems as well.

CouchDB stores data in the form of JSON-based documents. These documents are the building blocks of the database and aggregate directly to the database level. Thus, it doesn't feature collections like MongoDB.



(a) CouchDB Architecture

(b) CouchDB Document Model Implementation

Figure 3: The CouchDB Database (Images sourced from [35])

Some key features of this database are as below [14]:

- Easy replication of a database across multiple server instances
- Fast indexing and retrieval
- REST-like interface for document insertion, updates, retrieval and deletion
- JSON-based document format (easily translatable across different languages)
- Multiple libraries for your language of choice (show some of the popular language choices)
- Subscribe-able data updates on the _changes feed

2.4 Features Investigated

For the purposes of this study, we have identified three important features to be investigated. These are:

1. **Design and Architecture**
2. **Performance**
3. **Security**

These features are further analyzed across various sub-verticals to draw over-arching conclusions. Detailed explanations about the feature descriptions and their verticals are provided under each respective section for the feature.

3 Feature 1: Design and Architecture

For the purpose of our study, the first feature we have investigated is titled "Design and Architecture." As the name suggests, this feature mainly focuses on the database design and the underlying architecture of how a document model is implemented. A comparative analysis of the same is done for our chosen databases.

The most significant characteristics that dictate the design of a distributed database are consistency, availability and partition-tolerance. And often, a database has to trade-off one characteristic for the other. This is known as the "CAP Theorem." Formally, CAP Theorem or Eric Brewer's theorem states that we can only achieve at most two out of three guarantees for a database: consistency, availability and

partition-tolerance [3]. This theorem highlights the limitations of a NoSQL database in terms that it will always have to chose whether it wants be highly available or highly consistent. Both cannot be achieved at the same time, together.

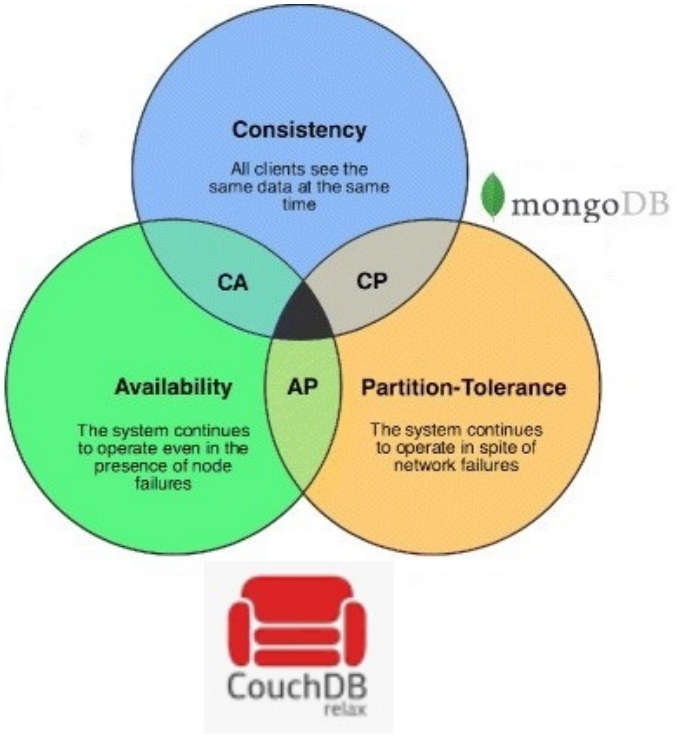


Figure 4: The CAP theorem Constraints. *Image sourced from Google Images*

For the purposes of our analysis, we have chosen the below four verticals to define the design and architecture of a database:

1. Consistency
2. Availability
3. Underlying Architecture
4. Handling of Parallel Versions

Consistency:
Consistency means, if you write data to the distributed system, you should be able to read the same data at any point in time from any nodes of the system or simply return an error if data is in an inconsistent state [16]. In simpler terms, consistency helps maintain the credibility of data by making sure that whenever accessed, returned data is always accurate, up-to-date and correct. Database systems may employ various algorithms for maintaining consistency across their data.

Availability:
Availability means the system should always perform reads/writes on any non-failing node of the cluster successfully without any error [16]. This is availability is mainly associated with network partition. i.e. in the presence of network partition whether a node returns success response or an error for read/write operation [16]. Thus, a highly available database would always allow the user to read/write data from it, no matter which and how many nodes in the distributed network may have failed.

Underlying Architecture:
The architecture of a distributed database system typically refers to the way in which the system arranges its servers and storage. A server may be considered as the smallest element of the database network and a collection of these can be designed in multiple ways to achieve varying objectives. Various databases follow a Master-Slave architecture where one server acts as the Master handling control requests while the rest act as slave nodes that serve the Master's requests. Another architecture design is the Peer-to-Peer or Master-Master design where each node is a master in itself. Replication of the data held within these servers is done in the same manner as the server distribution. That is, a Master-Master system will replicate data across the system in a peer-to-peer setting.

Handling of Parallel Versions:
A distributed database system is built with the intent of supporting multiple users at the same time. To achieve this objective, the database must have a protocol for handling parallel requests from simultaneous users. If many users have access to a data source in parallel, strategies for avoiding inconsistency based on conflicting operations are necessary [9]. This is also known as "Concurrency Control." Some common methods for concurrency control are Lock-Ins, Multi-Version Concurrency Control (MVCC) method etc.

3.1 Comparison (Literature Review)

For the purposes of this feature, we compared MongoDB and CouchDB functionalities across the above mentioned four verticals. The comparison is largely based on review of relevant papers in the field and the original documentation provided by the two database systems. The below findings can be identified across the four verticals of comparison for the feature:

3.1.1 Consistency

MongoDB is considered to be a strictly consistent database system. Strict consistency is the strongest consistency model [37]. Under this model, a write to a variable by any node needs to be seen instantaneously by all nodes [37]. In MongoDB, replica sets provide strict consistency which implies that the nodes are divided into two types: primary and secondary [17]. Primary ones are used to write and read the operations while secondary ones are used to provide redundancy in case of hardware failure [17]. This claim is supported by many studies analyzing the consistency property for MongoDB ([18],[15],[9],[29]).

CouchDB is considered to be an eventually consistent database system. Eventual consistency is a consistency model used in distributed computing to achieve high availability that informally guarantees that, if no new updates are made to a given data item, eventually all accesses to that item will return the last updated value [38]. In CouchDB, eventual consistency is incorporated where the functions can be performed on the nodes without the agreement of other nodes and further, it copies the changes made between the two nodes in the document incrementally so that they are synced continuously [17]. This claim is supported by many studies analyzing the consistency property for CouchDB ([18],[15],[9],[29]).

While both are considered consistent systems in their own ways, MongoDB prefers consistency over availability under the constraints of the CAP theorem.

3.1.2 Availability

MongoDB ensures availability by incorporation of replica sets which provides backup during failures and also is highly robust [17]. To ensure consistency, MongoDB by default sends every read/write request only to the Master Node. Because of this default behavior, Mongo DB is a consistent system but not available due to the below reasons [16]:

- If a leader disconnects from the cluster, it takes a few seconds to elect a new leader. So, making it unavailable for writes and reads [16].
- A client can always disconnect from the leader due to network partition even if both client and leader node is running fine. Hence making it unavailable [16].

This consistent but less available nature of MongoDB is highlighted across various research studies analyzing the same ([25],[30],[15],[9],[17],[18]).

CouchDB ensures availability by following the underlying Master-Master architecture. This peer to peer architecture allows each node to act as a master in itself and answer every query. One of CouchDB's defining features is bi-directional replication, which enables synchronization of data across multiple servers and devices via bi-directional replication [11]. This replication enables enterprises to maximize systems availability, reduce data recovery times, geo-locate data closest to end users, and simplify backup processes [11]. Thus, it may be concluded that CouchDB's design has been intentional in maintaining availability across the system. This finding is supported by various other studies as well ([18],[15],[9],[29]).

While both are considered available systems in their own ways, CouchDB prefers availability over consistency under the constraints of the CAP theorem.

3.1.3 Underlying Architecture

MongoDB supports a basic Master-Slave (Primary-Secondary) architecture. This architecture forms the basis for fault tolerance and replication in the database. MongoDB manages replication using a form of asynchronous master/slave replication called Replica Sets [29].

CouchDB supports a Master-Master (Peer-to-Peer) architecture. This architecture forms the basis for fault tolerance and replication in the database. For fault-tolerance, CouchDB supports both master/master and master/slave replication [29]. Replication can be finely tuned via replication filters [29].

3.1.4 Handling of Parallel Versions

MongoDB ensures control of parallel versions for data by using in-place update. It's In-place Update method uses lock-in protocol to make sure that an update is completed for a variable and all other operations are locked for the same till the update process gets completed. MongoDB delivers very high write performance, especially for updates [26,27]. It fits best if needed by a scenario that requires high update rate of objects or if the data is to be dumped in large volume [1]. It also allows for lazy writes, meaning that MongoDB writes to disk only when it has to [10]. Since primary memory is many times faster than secondary memory, MongoDB groups changes together and writes them to disk together [10].

CouchDB ensures control of parallel versions for data by using MVCC protocol. The Multiversion Concurrency Control (MVCC) makes sure that information is shown in different version for different users [10]. Every user gets a "snapshot" of the current state of the database to work with [10]. Changes made to the data will not be seen by other users until the transaction is fulfilled [10]. Thus, while using CouchDB, a user may see out of date information if parallel versions are running and an update process is incomplete somewhere.

Both MongoDB and CouchDB have their methods for handling parallel versions of data. While MongoDB's concurrency methods ensure that only correct data is retrieved every time (strict consistency), CouchDB's concurrency control methods focus on enabling updates even when (sometimes) incorrect/out-of-date data may be retrieved (eventual consistency).

3.2 Tabulated Summary

DESIGN AND ARCHITECTURE VERTICAL	MONGODB	COUCHDB
CONSISTENCY	• Eventual Consistency + Immediate Consistency • Default Behaviour: Data is always consistent as it is always read and written by master.	• Eventual Consistency: Clients can write to a single database node, and this information is guaranteed to eventually propagate to the rest of the database.
AVAILABILITY	• Crash and recover: Might lose consistency. • Less available due to more dependency on master.	• Crash-only: Crash and remain consistent. • More available due to master-master architecture.
HANDLING OF PARALLEL VERSIONS	• Locks (Instant update)	• MVCC (Multi Version Concurrency Control)
UNDERLYING ARCHITECTURE	• Master-Slave	• Master-Master and Master-Slave (Peer-based)

Figure 5: A tabulated summary of findings for Design and Architecture Feature. *Image created by Anshu Kumar*

3.3 Understanding the disparity

Analysing the overall design and architecture of MongoDB and CouchDB, it was found that they both differ in their approaches. On one hand MongoDB favours consistency and partition-tolerance, on the other hand CouchDB favours availability and partition tolerance. MongoDB was found to be immediate/strictly consistent because of their trade-off of availability whereas CouchDB was eventually consistent. However, CouchDB was found to be highly available, but at the cost of consistency. It allows all clients to perform read and write operations in any situation. For concurrency control, MongoDB uses lock-in feature which is even helpful in maintaining consistency of data. Whereas, in CouchDB MVCC is used to manage concurrent access to the database. This helps to overcome the bottleneck of most of the databases where the data being changed becomes locked to other users until it is completely modified. CouchDB follows master-master architecture for replication and fault-tolerance whereas MongoDB follows Master-Slave architecture for the same.

4 Feature 2: Performance

The performance of databases is generally measured in terms of time take to execute various operations on the data. Most studies conclude MongoDB's superiority over CouchDB in terms of these database operations. Deeper look into the design decisions of these databases show the various trade-offs made to achieve performance. Understanding these tradeoffs is essential to evaluate the suitability of these databases in various applications. As we will see there are many instances where database performance is secondary to achieving other goals such as device scalability or ease of use. The following will illustrate the difference in performance between MongoDB and CouchDB and examine the different perspective to distributed database design that are the underlying cause of the disparity in their performance.

4.1 Comparison (Literature Review)

Database operations are key to many web applications and good performance would reduce the latency for these applications and improve user satisfaction. The most basic functions of a databases are the

CREATE, READ, UPDATE and DELETE operations (CRUD). Improving the performance these operations should improve the overall performance of the database system and hence the application that employees it. However, these operations are fundamentally different for NoSQL databases. The nature of these operations has been changed in NoSQL databases to better fit the internet and cloud environment. The reason being that these databases need to be replicated across multiple servers to serve a wide network and therefore the performance of these operations will be compromised if databases are to maintain ACID compliance. Both MongoDB and CouchDB makes various trade-offs to provide acceptable level of database performance while operating in these environments. These-tradeoffs have different impact on the performance of each of the basic operations.

There are many studies that examine the performance of MongoDB and CouchDB under varying loads and environment. Studies have been carried out with varying loads and users [6]. Studies have also examined supplementary operations such as iterating through all the keys and iterating through all values [19]. A study has even examined the performance of these database in an IoT environment [13]. Generalizing these studies, we have identified MongoDB to be like CouchDB performance in terms of write and superior in terms of read operations. However, these studies do not have high degree of concurrent users for their testing which is what CouchDB is more suited for. However, we can see that as the studies look at smaller server devices, more concurrent users and more distributed networks the differences in performance become less wide.

Despite small improvements in performance at more extreme conditions there is a clear indication that MongoDB can provide better read performance and comparable write performance in test environments. It is difficult to extrapolate these results to actual performance on real-life system especially in situations of high load, small servers, widely distributed networks and with poor connections initial analysis of studies on test environment suggest better performance for MongoDB in CRUD operations.

4.2 Tabulated Summary

PERFORMANCE VERTICAL	MONGODB	COUCHDB
DATABASE OPERATIONS	• 10x faster in terms of database operations • Design Decisions that focuses on performance of database operations	• Trades-off performance for other features • Design Decisions that focuses on usability, scalability and running on web
IMPLEMENTATION	• C++ based database • BSON based storage • Binary protocol and custom protocol over TCP/IP	• Erlang based database • JSON based storage • RESTful HTTP API
REPLICATION AND CONCURRENCY CONTROL	• Sharding with Update in place • Sacrifices Durability for Performance	• Asynchronous Replication with MVCC • Ensures ACID property
CONCLUSION	• MongoDB was designed from ground up to provide best database performance in a distributed environment. It achieves it by removing constraint on durability	• CouchDB's value proposition is that web applications needs databases that are reliable databases that are scalable across varying devices on the web.

Figure 6: A tabulated summary of findings for Performance Feature. *Image created by Geogy Sabu Jose*

4.3 Understanding the disparity

When examining the implementation of the two databases the various philosophies and their impact on the performance becomes clearer. MongoDB is implemented using C++ and BSON data serialization. These design choices provide better performance as compared to CouchDB which is implemented in Erlang and JSON. C++ is a much more performance-oriented language with native compilation while Erlang was designed to be used to build massively scalable soft real-time systems with requirements on high availability. These differences might only make minimal impact on the database performance but might have greater impact in development of web applications. Another important distinction is the use of BSON instead of JSON, which has a strong impact on query performance. BSON is just the bin¬ary-en¬coded seri¬al¬iz¬a¬tion of JSON-like doc¬u¬ments [2]. Using BSON instead of JSON enables more powerful indexing and querying features. BSON objects are better optimized for data storage and retrieval [21]. However, JSON has become ubiquitous in the web ecosystem and it is mainly since it is both human and machine readable. It has become a key part of the REST architecture which is a dominant architecture for the web. CouchDB is therefore able to allow developers to access the database through a REST based HTTP interface. This makes the database much more accessible to web developers and easily incorporated with traditional web application stacks [8]. This is also much more familiar to web developers as well. MongoDB uses a custom TCP/IP protocol with its BSON serialization [23]. MongoDB's TCP based communication will have much less latency than CouchDB's HTTP protocol. However, by using JSON and implementing a RESTful API, CouchDB is more compatible with the web ecosystem and can be scaled across devices even onto mobile and embedded devices.

Another important design decision that has an impact database operation is the degree to which these systems comply with ACID properties of atomicity, consistency, isolation and durability. Both MongoDB and CouchDB takes different compromises in terms of ACID properties and these compromises affect various database operations to different degree. The main difference between the system is that MongoDB does not ensure durability and CouchDB does not ensure strong consistency. MongoDB ensure consistency through locking which has an impact on write and update operations as the database using single-master system. This causes a bottleneck for write operations and affects the scalability of the system in concurrent environment. Couch DB uses Multi-Version Concurrency Control (MVCC) to manage concurrent access to the database instead of locks [27]. This enables CouchDB to provide master-master replications that can handle writes in a highly distributed environment in exchange for eventual consistency. In terms of durability, CouchDB ensures that write operations are durable by ensuring data is written into disk before confirming the application. MongoDB however forsakes durability to improve the write speed. MongoDB will ensure data is written to the memory but will not wait for confirmation from disk operations. This makes more sense for traditional server architecture where server failure is handled separately but in the case of a distributed mobile server would need greater importance to durability [24].

In the end the difference in performance is a deliberate decision made by both databases in service of different design beliefs. From the underlying languages and tools used to the difference in replication, concurrency control and fault tolerance the two databases make different trade-offs when it comes to database performance. Next, we will talk about the underlying philosophies for these design decisions and its value in the current web ecosystem.

4.4 Re-evaluating Importance of Performance

The underlying difference in philosophy can be boiled down to the trade-off between database performance and features that cater to developer needs. MongoDB design philosophy is directed at efficiency of database operations. Using C++ and BSON with TCP for data communication makes its querying very efficient. Despite maintaining strong concurrency, MongoDB can provide satisfactory write performance by not waiting for disk operation and thereby sacrificing durability. These design decisions show the

optimization of database performance considering the current web application architecture and servers. However, these studies fall short of being able to judge the performance of the databases at higher loads and with high number of concurrent users. It is also important to consider if the database operations performance is significant in the overall latency of web applications and distributed systems. The success of the database might not depend entirely on performance but also on other features such as being able to handle high level of concurrent users, providing high availability in wide networks with low connectivity or even being able to be scale your database server to smaller devices.

This has been the perspective of CouchDB. They argue that database operations are such an insignificant section of the latency that a web application process encounters that it makes sense to deprioritize database performance to achieve other goals. The provide multiple features to users such as being better designed for concurrent users using Erlang and master-master replication that is possible by using MVCC. The MVCC also provides higher availability and support in wide network systems with poor connection to servers. However, this also results in large memory usage in the event of rapid updating as CouchDB needs to log these updates for reaching consistency which hinders its use in real-time Iot devices that need to log a lot of data for updating. Using Erlang, disk-based storage and master-master replication allows CouchDB to be deployed on mobile servers and even on embedded devices. There have even been experiments of JavaScript based alternative that runs on a web browser [33]. The amount of flexibility and scalability of CouchDB makes it a perfect choice for applications such as a peer to peer mobile application or an offline friendly web app.

Another feature that CouchDB has focused on is on how friendly the app is for developer to use. Using JSON and having a HTTP REST API allows developers to quickly learn and integrate the database with their technology stack and offers an easy to use and simple tool for managing data for a web application in a distributed manner. This approach reminds about the rising popularity of ruby on rails which become popular for it ease of use and flexibility. The ease for the developers is present in human readable JSON data serialization that can be extracted with intuitive HTTP REST API's as well as ability the easily interface with the database with web application stack. The intuitiveness of the database is somewhat hindered by the fact that CouchDB does not have SQL query support and uses map-reduces that are more flexible but difficult to learn.

To summarize, when considering the performance of database operation to select between MongoDB and CouchDB it is important to carefully consider the importance of database efficiency on the overall performance on the application and the trade-offs that need to be made to achieve better database performance. It is also important to consider external factors such as the ease of learning and use which will also have an impact on the development of the application.

5 Feature 3: Security

Database security refers to the range of tools, controls, and measures designed to establish and preserve database confidentiality, integrity, and availability [12]. The security of a database is a crucial element to its success and usage. Without essential security measures in place, the data stored within the database may be prone to tampering, leakage and/or theft. Consequently, defeating the very purpose of having a database for securely storing data.

With the increasing amount of data being stored across cloud services that are often based on NoSQL database systems such as MongoDB and CouchDB, the concern for security of this data is also increasing. It is essential to maintain the confidentiality and privacy of this data as more and more sensitive information is being uploaded to them with each passing day.

For the purposes of this report, we have analyzed database security across four verticals:

1. Authentication
2. Access Control
3. Denial of Service Attacks
4. Data Encryption

Authentication:
Authentication is the process of verification of a user's identity that aims to access the data or resources of a certain system or organization [26]. In simpler terms, the authentication process is used to judge whether a user trying to access the resources of a system is a legitimate user or not. In terms of a database system, this is an important security measure that only allows genuine users to access the data of a database. Thereby ensuring privacy and confidentiality of data. Databases may provide their own authentication mechanisms or leverage third party systems to verify their users. A few of the well-known authentication techniques include Password based authentication, Multi-Factor authentication, Certificate based authentication, authentication protocols like SSL, SSH and Kerberos etc. [39]

Access Control:
Access Control is the process of restricting the access of a certain user/s to certain pre-defined set of database resources. In other words, by using access control processes, the database system can define which user can access which resource and which ones are inaccessible to them. Access control can be applied at system, database, object and content level depending on the configurations of the database administrator [32]. Various access control models such as Role Based Access Control (RBAC), Policy Based Access Control (PBAC), Task Based Access Control (TBAC) etc. are used by database systems to ensure appropriate access for users.

Denial of Service Attacks:
A denial-of-service (DoS) attack is a type of cyber-attack in which a malicious actor aims to render a computer or other device unavailable to its intended users by interrupting the device's normal functioning [4]. As the name suggests, put in simple terms, a DoS attack aims to break down a computing system so that it is unable to provided desired service to its legitimate users. Most computing systems build security measures to ensure they are not vulnerable to such attacks. However, hackers often find new and innovative ways to hack into systems and disrupt their services.

Data Encryption:
Data encryption translates data into another form, or code, so that only people with access to a secret key (formally called a decryption key) or password can read it [7]. Data Encryption is a security measure often used to mitigate data hacking and theft attacks. The idea is to make the data unreadable, even if it is stolen or intercepted while transferring. Encryption can be done on both static data (that is not transferred over the network) or for data in-transit (that is while it travels over communication channels). A plethora of different encryption techniques are available in the computing world and database systems often implement one of these to provide an additional level of security to their data.

5.1 Comparison (Literature Review)

For the purposes of this feature, we compared MongoDB and CouchDB functionalities across the above mentioned four verticals. The comparison is largely based on review of relevant papers in the field and

the original documentation provided by the two database systems. The below findings can be identified across the four verticals of comparison for the feature:

5.1.1 Authentication

MongoDB has two authentication mechanisms: The Salted Challenge Response Authentication (SCRAM) which is implemented by default and is based on password authentication, and the x.509 certificate protocol which requires a public key infrastructure [36]. [28] terms this as a weak authentication mechanism that can only be activated in the standalone mode or replica-set mode. It is based on the key or password called 'pre-shared secret' which has been hashed with MD5 algorithm before being stored in the key file [28]. The MongoDB Enterprise Server/Product supports two authentication mechanisms: LDAP (Lightweight Directory Access Protocol) and Kerberos, which allows taking advantage of existing authentication infrastructure and technologies ([36], [20]).

On the other hand, CouchDB provides a number of different yet simple authentication mechanisms. It supports basic password-based authentication as well as cookie-based authentication for its users ([39], [36], [20]). Passwords in CouchDB are hashed using PBKDF2 hashing algorithm ([39], [36]). CouchDB also provides the Admin Party Authentication Process in which any user who has access to the database has the privilege of an administrator ([36]). The authentication through cookies process allows the user to login through a Web interface and the HTTP protocol in which case a unique token is generated for a defined period of time ([36]). This last mechanism may cause some problems when this time is very long because it could allow attackers to obtain that information and carry out identity-theft attacks ([36]).

5.1.2 Access Control

MongoDB uses a Role Based Access Control (RBAC) Method ([36], [20], [34]). This implies that MongoDB uses system-defined and custom-defined roles within a sharded cluster to enable appropriate access control [39]. However, this is not enabled by default [36] and is unsupported in the un-sharded mode of usage [20]. MongoDB treats roles as a group of privileges with each privilege composed of a resource and the permitted operations upon this resource ("operations" are referred to as "actions" in MongoDB's documentation) [34]. The database system comes with a bunch of different pre-defined roles that can be used as is. It also allows for the customization of said roles or creation of new ones. Given the fact that roles are a group of privileges, and that each privilege is made of a resource and its permitted actions, all that is needed in order to create a new role is compose it with your desired privileges [34].

CouchDB also uses a Role Based Access Control (RBAC) framework to ensure access control ([39], [36], [20], [34]). However, this authorization is implemented only at the database level ([39], [34]) and is highly basic as compared to that of MongoDB's. It comes only with two built-in classes: member and admin [34]. Users that are members can read, edit and create documents inside the database, except the ones called design documents [34]. Admins, are able to create, read and modify any type of document, besides configuring other users' roles and the database itself [34]. CouchDB allows the creation of custom roles as well. However, these are limited and have to be assigned to one of the two pre-defined classes of authorization. To implement anything apart from these, authorization functions have to be created separately that are to be run before each user action [34].

5.1.3 Denial of Service Attacks

Existing literature points to the fact that MongoDB is vulnerable to DoS attacks and documented cases of the same exist as well. [28] highlights that MongoDB versions 2.4.0 – 2.4.4 always have a problem with the denial of service attack in which the remote authenticated users can perform the attack through the vulnerabilities of uninitialized pointer and crash the server.

Similar to MongoDB, it appears that CouchDB also has vulnerabilities to the DoS attacks and has faced these attacks in the documented past. A review of literature in [28] highlights two distinct vulnerabilities identified in the database. The first is specific to the Apache CouchDB 1.5.0 version while the second is a one-line command that can crash the CouchDB servers.

5.1.4 Data Encryption

MongoDB community edition does not provide data encryption at rest ([39], [36]) while MongoDB Enterprise product provides at-rest encryption using Application Level Encryption [36]. This is implemented using AES-256 (Advanced Encryption Standard) and Base64 (binary-to-text encoding) protocols [20]. In transit encryption for MongoDB Community Edition is enabled using TLS (Transport Layer Security) and SSL (Secure Sockets Layer) certification protocols ([39], [36]). While the same is implemented in Enterprise Edition using FIPS (Federal Information Processing Standards) protocol ([39], [36]).

On the other hand, CouchDB does not provide any data encryption at rest [36]. The only at-rest encryption is enabled for the authentication passwords. This is implemented using PBKDF2 (Password-Based Key Derivation Function 2) ([39], [20]). In transit encryption of data is implemented by means of TLS (Transport Layer Security) and SSL (Secure Sockets Layer) certification protocols ([39], [36]).

5.2 Tabulated Summary

SECURITY VERTICAL	MONGODB	COUCHDB
AUTHENTICATION	• Salted Challenge Response Authentication (SCRAM) [by default and based on password authentication] • x.509 certificate protocol (requires a public key) • LDAP (Lightweight Directory Access Protocol) [enterprise product] • Kerberos [enterprise product]	• Basic password-based authentication (hashed using PBKDF2 hashing algorithm) • Cookie-based authentication (time governed) • SSL/TSL certification for network communication
DENIAL OF SERVICE ATTACKS	• Various report DoS attacks • Can be performed through remote authenticated users • Exploits vulnerabilities of the uninitialized pointer and crashes the server	• Various reported methods for DoS attacks known vulnerability in version 1.5.0 that can be exploited to crash the server • Another known vulnerability can cause a crash with single-line commands
ACCESS CONTROL	• Comes with a wide variety of built-in roles split into defined categories e.g., database user roles, backup and restoration roles etc. • New custom roles can be easily created due to the resource-privilege setup	• Only comes with two defined classes: user and admin • Custom roles can be created (but are assigned to these two classes only)
DATA ENCRYPTION	• At rest provided only in enterprise product, uses AES-256 (Advanced Encryption Standard) and Base64 Encoding • In transit encryption with TLS/SSL certification. FIPS (Federal Information Processing Standards) mode available in enterprise product	• At rest, no encryption is provided (except for passwords using PBKDF2 (Password-Based Key Derivation Function 2)) • In transit, SSL certification is used

Figure 7: A tabulated summary of findings for Security Feature. *Image created by Sanna Nazir*

5.3 Understanding the disparity

Analyzing the overall security features of MongoDB and CouchDB, various disparities can be identified. While each use different protocols to implement various different security measures, it may be said that MongoDB has a more secure data environment. This conclusion can be based on its superiority in terms of data encryption at rest and better in-transit security protocols for its enterprise edition. Furthermore, MongoDB uses superior methods of authentication that are less vulnerable to attack along with providing

better access control measures. CouchDB seems to be lagging in these domains. This nature of CouchDB can be attributed to its deliberate design decision of placing high significance on large-scale distribution and scalability. In turn this can be attributed to its primary goal of being a mobile-friendly, highly distributed database system.

6 Reflection and Conclusion

Following a thorough research and analysis of the various features for MongoDB and CouchDB, we can conclude that both have their own pros and cons. Each database system has been designed with certain end objective in mind and their features are consequently catering to that objective. MongoDB outperforms CouchDB in terms of query workloads and overall performance. However, CouchDB is an availability-dominant database while MongoDB favors consistency. In terms of device scalability, CouchDB can be scaled to even mobile devices while MongoDB requires high-memory environments. For security protocols, MongoDB outperforms CouchDB metrics. Overall, MongoDB favors high performance rates while CouchDB trades off performance for improved usability, device scalability and operations in distributed environment.

6.1 Limitations of Findings

There are various limitations to our current study. The first and foremost is that our analysis is purely based on an integrated literature review. Experimentation and validation of these findings has not been carried out. Furthermore, for performance analysis, various benchmarks have been used in the referenced studies. While these benchmarks are industry standards, they do not always simulate real-world distributed environments. They may lack in terms of the actual loading, number of users and so on. Therefore, their results must be consumed with this fact in mind. In terms of security, only documented cases of attacks and vulnerabilities have been analysed. This is not to say that more vulnerabilities cannot be identified in the future.

7 Appendix

For this study, individual contributions are as below:

- Feature 1 Section researched and written by Anshu Kumar
- Feature 2 Section researched and written by Geogy Sabu Jose
- Feature 3 Section researched and written by Sanna Nazir

The remaining collective sections were completed together.

References

[1] AU Essays. Comparision between mongodb and couchdb information technology essay. https://www.auessays.com/essays/information-technology/comparision-between-mongodb-and-couchdb-information-technology-essay.php.

[2] BSON. Bson. http://bsonspec.org/.

[3] Cloud Lab. Nosql - cap theorem. https://cloudxlab.com/assessment/displayslide/345/nosql-cap-theorem.

[4] CloudFlare. What is a denial-of-service (dos) attack? https://www.cloudflare.com/learning/ddos/glossary/denial-of-service/.

[5] CouchDB. Couchdb website. https://couchdb.apache.org/.

[6] Niteshwar Datt. Comparative Study of CouchDB and MongoDB - NoSQL Document Oriented Databases. *International Journal of Computer Applications*, 136(3):24–26, February 2016.

[7] Digital Guardian. What is data encryption? definition, best practices more. https://www.cloudflare.com/learning/ddos/glossary/denial-of-service/.

[8] J. Edstrom and E. Tilevich. Reusable and extensible fault tolerance for restful applications. In *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 737–744, 2012.

[9] R. Hecht and S. Jablonski. Nosql evaluation: A use case oriented survey. In *2011 International Conference on Cloud and Service Computing*, pages 336–341, 2011.

[10] Robin Henricsson. Document oriented nosql databases: A comparison of performance in mongodb and couchdb using a python interface, 2011.

[11] IBM. Apache couchdb. https://www.ibm.com/cloud/learn/couchdb.

[12] IBM. Database security. https://www.ibm.com/cloud/learn/database-security, Last accessed on 2020-11-20.

[13] Lorenzo Incipini, Alberto Belli, Lorenzo Palma, Roberto Concetti, and Paola Pierleoni. Databases performance evaluation for iot systems: The scrovegni chapel use case. In *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 463–468. IEEE, 2019.

[14] InfoQ. What is couchdb and why should i care? https://www.infoq.com/articles/warner-couchdb/.

[15] Jing Han, Haihong E, Guan Le, and Jian Du. Survey on nosql database. In *2011 6th International Conference on Pervasive Computing and Applications*, pages 363–366, 2011.

[16] Bikas Katwal. What is the cap theorem? mongodb vs cassandra vs rdbms, where do they stand in the cap theorem? https://medium.com/@bikas.katwal10/mongodb-vs-cassandra-vs-rdbms-where-do-they-stand-in-the-cap-theorem-1bae779a7a15.

[17] Harpreet Kaur and Kamal Kaur. A review:study of document oriented databases and their security. *International Journal of Advanced Research in Computer Science*, 4(8), 2013.

[18] Subita Kumari. Implementation of map-reduce paradigm in mongodb and couchdb. *International Journal of Advanced Research in Computer Science*, 9(2), 2018.

[19] Yishan Li and Sathiamoorthy Manoharan. A performance comparison of sql and nosql databases. pages 15–19, 08 2013.

[20] Petar Milić, Kristijan Kuk, Slaviša Trajković, Dragan Ranelović, and Brankica Popović. Security analysis of open source databases in web application development.

[21] MongoDB. Json and bson. `https://www.mongodb.com/json-and-bson`.

[22] MongoDB. Mongodb website. `https://www.mongodb.com/`.

[23] MongoDB. Mongodb wire protocol. `https://docs.mongodb.com/manual/reference/mongodb-wire-protocol/`.

[24] MongoDB. What about durability? `https://www.mongodb.com/blog/post/what-about-durability`.

[25] ABM Moniruzzaman. Nosql database: New era of databases for big data analytics–classification, characteristics and comparison. 2013; 13: 6, 2019.

[26] Valarie Moore. Oracle database security guide, 10g release 1 (10.1) part no. b10773-01 copyright© 2003 oracle corporation. all rights reserved. primary authors: Laurel p. hale, jeffrey levinger contributing authors: Ruth baylis, michele cyran, john russell. 2003.

[27] NA. Eventual consistency. `https://guide.couchdb.org/draft/consistency.html`.

[28] Preecha Noiumkar and Tawatchai Chomsiri. A comparison the level of security on top 5 open source nosql databases. In *The 9th International Conference on Information Technology and Applications (ICITA2014)*, 2014.

[29] Ahmed Oussous, Fatima-Zahra Benjelloun, Ayoub Ait Lahcen, and Samir Belfkih. Comparison and classification of nosql databases for big data. 05 2015.

[30] Rabi Prasad Padhy, Manas Ranjan, P. Suresh, C. Satapathy, and Oracle India Pvt. Rdbms to nosql: Reviewing some next-generation non-relational database's. 2011.

[31] Luc Perkins, Eric Redmond, and Jim Wilson. *Seven databases in seven weeks: a guide to modern databases and the NoSQL movement*. Pragmatic Bookshelf, 2018.

[32] Mario G Piattini and Eduardo Fernandez-Medina. Secure databases: state of the art. In *Proceedings IEEE 34th Annual 2000 International Carnahan Conference on Security Technology (Cat. No. 00CH37083)*, pages 228–237. IEEE, 2000.

[33] PouchDB. The database that syncs! `https://pouchdb.com/`.

[34] Bruno Jorge Siqueira Rodrigues. Security of a nosql database: authentication, authorization and transport layer security. 2019.

[35] SeveralNines. The battle of the nosql databases - comparing mongodb and couchdb. `https://severalnines.com/database-blog/battle-nosql-databases-comparing-mongodb-and-couchdb`.

[36] Irving L Solsol, Héctor F Vargas, and Gloria M Díaz. Security mechanisms in nosql dbms's: A technical review. In *International Conference on Smart Technologies, Systems and Applications*, pages 215–228. Springer, 2019.

[37] Wikipedia. Consistency model. `https://en.wikipedia.org/wiki/Consistency_model#Strict_consistency`.

[38] Wikipedia. Eventual consistency. https://en.wikipedia.org/wiki/Eventual_consistency.

[39] Anam Zahid, Rahat Masood, and Muhammad Awais Shibli. Security of sharded nosql databases: A comparative analysis. In *2014 Conference on Information Assurance and Cyber Security (CIACS)*, pages 1–8. IEEE, 2014.