

Evaluation of Couchbase, CouchDB and MongoDB using OSSpal

André Calçada¹ and Jorge Bernardino^{1,2} ^a

¹*Polytechnic of Coimbra – ISEC, Rua Pedro Nunes, Quinta da Nora, 3030-199 Coimbra, Portugal*

²*CISUC - Centre of Informatics and Systems of University of Coimbra, Pinhal de Marrocos, 3030-290 Coimbra, Portugal*

Keywords: NoSQL, Document Store, OSSpal, CouchDB, Couchbase, MongoDB.

Abstract: Document stores are a NoSQL (Not Only Structured Query Language) database type, made to deal with big amounts of data and considered to be very developer-friendly, but they have also attracted a large interest from researchers and practitioners. In this paper, we analyze and evaluate three of the most popular document stores Couchbase, CouchDB, and MongoDB, the evaluation is based on their functionalities. In this evaluation, we use the OSSpal methodology, which combines quantitative and qualitative measures to evaluate the software. We conclude that of these open-source document stores the best is MongoDB, followed by Couchbase, and CouchDB.

1 INTRODUCTION

There are many applications of document stores, but we do not know which one is better and so in this document, we will compare and evaluate three document stores (Couchbase, CouchDB, and MongoDB) through OSSpal evaluation methodology, this evaluation is based on their functionalities.

Document stores are a NoSQL database type. NoSQL databases were developed to deal with big amounts of data, there are four main types of NoSQL databases, Key-Value Store, Column-Oriented Database, Document Store and Graph Database (Sareen & Kumar, 2015) (Abramova et al., 2014). The OSSpal methodology combines quantitative and qualitative measures for evaluating software in several categories, resulting in a quantitative value that allows the comparison between tools.

The objective of this study is to help users to know what database application to choose, in a PC, through the comparison and evaluation of these document stores.

To choose these three documents stores we resorted to db-engines ranking (<https://db-engines.com/en/ranking>), a popularity ranking of databases.


The rest of this paper is structured as follows. Section 2 introduces document stores. Section 3 describes each document store and shows a

comparison of these. Section 4 shows us the evaluation of these three document stores. Finally, section 5 presents the main conclusions and future work.

2 DOCUMENT STORES

SQL databases (relational databases) were developed in the 1970s to deal with the first data storage applications and with the problems of navigational databases that are easily not consistent (easily lose data). The storage in this model is made by individual records, stored in rows of a table, if there is the need to add or change a type of data (table) we must change the entire database, in these databases the scaling is vertical (Abramova et al., 2015). Document stores are dynamic unlike relational databases and the scaling is horizontal which allows them to store more data, keeping a high-performance level (Sareen & Kumar, 2015).

If the scaling is vertical it means that to improve the performance of the database, we have to improve our machine (computer or server) hardware otherwise if the scaling is horizontal it means that to improve the performance of the database, we have to distribute the different tasks of the database into more machines (computer or server) (Lourenco et al., 2015).

^a  <https://orcid.org/0000-0001-9660-2011>

A document store is a NoSQL database, developed to store documents. NoSQL databases were developed in 2000 to deal with the limitations of SQL databases. In this paper, we are focused only on document stores. These rely on the internal structure of the document to extract data that the database engine uses for further optimization. Document stores store the data of an object is an instance of the database, and every stored object can be different from every other. This eliminates the need for object-relational mapping while loading data into the database (Nayak, Poriya, & Poojary, 2013).

2.1 Architecture

Documents are addressed in the database via a unique key that represents that document. This key is a simple identifier, typically a string, a URI (Uniform Resource Identifier), or a path. The key can be used to retrieve the document from the database. Typically, the database retains an index on the key to speed up document retrieval, and in some cases, the key is required to create or insert the document into the database.

Document stores implementations offer a variety of ways of organizing documents, including notions of:

- Collections: groups of documents, where depending on implementation, a document may be enforced to live inside a collection, or may be allowed to live in multiple collections;
- Tags and non-visible metadata: additional data outside the document content;
- Directory hierarchies: groups of documents organized in a tree-like structure, typically based on path or URI.

These databases offer an API or query language that allows the user to retrieve documents based on content (or metadata). For example, we may want a query that retrieves all the documents with a certain field set to a certain value. To update or edit the data of a document, the database either allows the entire replacement of the document or individual structural pieces of the document (Nayak et al., 2013), (Sareen & Kumar, 2015).

Figure 1 represents the architecture of a document store. It is like a tree that has branches and each branch as more branches, the root is the database engine and the branches are the documents that the database stores and each branch as its values/objects.

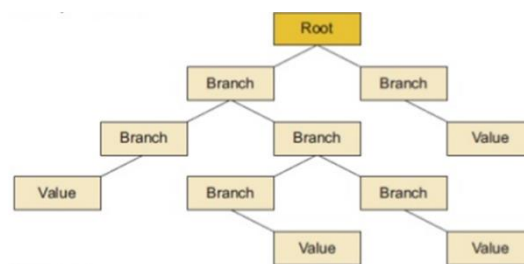


Figure 1: Document store architecture (“slideshare,” 2016).

2.2 Advantages and Disadvantages

The main advantages are the following:

- Storage of unstructured data becomes easy: This happens because the document will have all the keys and values required by application logic. There is no heavily priced migration involved as it does not need to know any information beforehand;
- Efficiency at retrieving data: Efficient at retrieving information about an object with a minimum of disc operations;
- Flexibility: Document stores are very flexible, they operate over a wide variety of data types, like JSON (JavaScript Object Notation), HTTP (Hypertext Transfer Protocol), REST (Representational State Transfer), and JavaScript;
- Big evolutionary rhythm: Due to the growing amounts of data these needs to be constantly improved and updated;
- Efficiency at writing data: Most efficient when writing a new row if all the row data is provided at the same time so the entire row can be written with a single seek.

The main weaknesses are the following:

- Lack of efficiency: These systems are not efficient at performing operations that apply to the entire data set as opposed to specific records;
- Model limitations: Query model is limited to keys and indexes, this means that to get a value we have to access the document where it is stored, and the document can only be reached by their key or index;
- Lack of consistency: Document stores are considered immature because they do not have to use SQL.

3 TOP 3 DOCUMENT STORES

In this section, we describe three popular document stores: Couchbase, CouchDB, and MongoDB. These were chosen according to a ranking (db-engines, the ranking of February of 2019) (“db-engines,” 2019).

3.1 Couchbase

Couchbase is an open-source document store, developed by CouchBase Inc. Couchbase first release was in 2010, it is designed for interactive web applications and mobile applications and the documents are stored in JSON files. In support of these kinds of application needs, Couchbase is designed to provide easy-to-scale key-value or JSON document, with integrated caching. Couchbase offers low latency read and write operations, providing linearly scalable throughput. Couchbase supports live-cluster topology changes, this means that there is no application downtime when updating the database.

Couchbase architecture consists of the following core components: Cluster Manager, Data service, Index service, and Query service, like it is described in Figure 2. The runtime behaviors such as replication, storage, caching, and so on, can be tuned to the needs of the different services. Couchbase has 4 types of applications (web, mobile, cloud and kubernetes) each with one application, each application has a variation for each type of system it will operate in, Couchbase can be downloaded at <https://www.couchbase.com/downloads> (“Couchbase,” 2019a), (Bazar & Iosif, 2014).



Figure 2: Couchbase Server architecture (“Couchbase Server architecture,” 2018).

Couchbase uses Triggers (procedure used when a determined action occurs), Secondary Indexes, Server-Side Scripts and MapReduce (model to process and generate big data sets with a parallel, distributed algorithm on a cluster), to improve database performance, it also has Typing and a SQL-like query language (N1QL), and it is not ACID compliant (ACID stands for Atomicity, Consistency, Isolation, Durability, it is a

set of properties of database transactions) (“db-engines,” 2019).

In Figure 3 we can see the interface of Couchbase Enterprise Edition, as we can see in the upper side of the interface there is a tab where we can access the database features, like query and indexes, and that the “overview” tab is open, and it shows us the performance of the database.

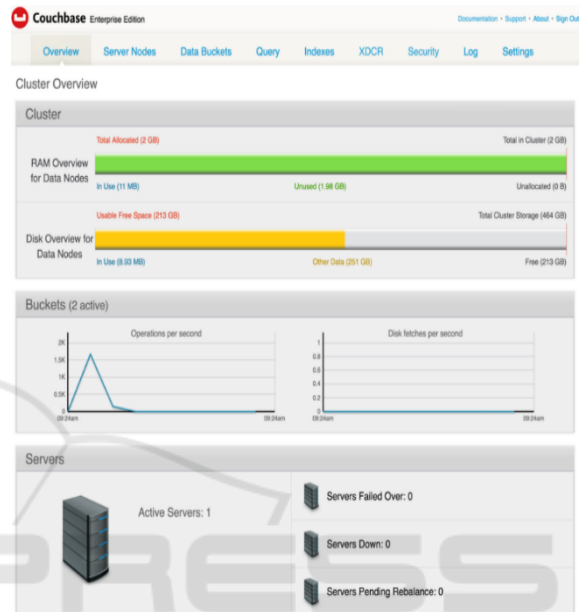


Figure 3: Couchbase enterprise edition interface (“Couchbase,” 2016).

3.2 CouchDB

CouchDB is an open-source document store, developed by Apache Inc. CouchDB first release was in 2005, it was developed in Erlang, which is a language meant for distributed systems, it uses JSON documents to store data and RESTful HTTP API to create and update database documents and JavaScript is used as a query language. CouchDB provides a built-in web application called FULTON which can be used for administration. CouchDB works with and without a network connection, but the application must keep on working. It is compatible with CRM (Customer Relationship Management) and CMS (Customer Management Systems) systems. Some of the drawbacks of CouchDB are temporary views, there is no support for ad-hoc queries. CouchDB focuses on ease of use and having a scalable architecture. CouchDB implements a form of multi-version concurrency control (MVCC) so it does not lock the database files during writes. Conflicts are left to the application to resolve. Resolving a conflict

generally involves first merging data into one of the documents, then deleting the stale one (Nayak et al., 2013), (“CouchDB,” 2019).

CouchDB like Couchbase uses Triggers, Secondary Indexes, Server-Side Scripts and MapReduce, to improve database performance, but it does not have Typing or a SQL-like query language, and it is not ACID compliant (“db-engines,” 2019).

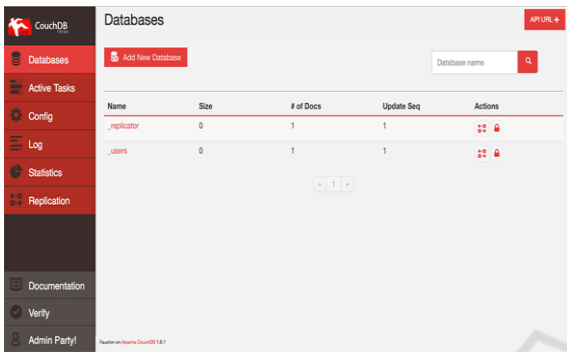


Figure 4: CouchDB interface (“masteringionic,” 2016).

In Figure 4 we can see the interface of CouchDB, as we can see in the left side of the interface that we have a tab where we can access the database features, in Figure 7 the opened tab is “Databases” that shows us the databases that we currently have and a few details about them.

The application has one variation for each operating system, that can be downloaded at <http://couchdb.apache.org>.

3.3 MongoDB

MongoDB is a free open-source document store, developed by MongoDB Inc. MongoDB was first released in August of 2009 and it was developed in C++. MongoDB is well suited for applications like content management systems, archiving, real-time analytics, etc, it has 4 different types applications (cloud, analytics, services and software) and in each type there are a few different applications and each application have a variation for each type of system it will operate in, MongoDB can be downloaded at <https://www.mongodb.com>.

MongoDB is a schema-free database, that stores data in JSON-like files (BSON), BSON is a binary format of JSON that allows quick and easy integration of data. MongoDB has a flexible structure, its databases are very easily scalable, and it has a very friendly interface.

MongoDB, unlike CouchDB and Couchbase, does not use triggers, it uses Secondary Indexes

Server-Side Scripts and MapReduce, to improve database performance, it also has Typing, it is ACID compliant and does not have an SQL-like query language (“db-engines,” 2019).

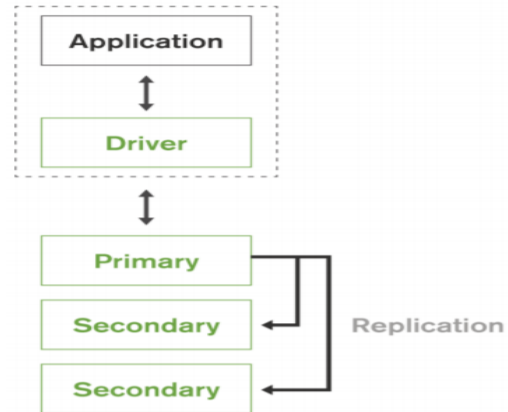


Figure 5: Replica set (“MongoDB Architecture Guide,” 2018).

MongoDB cluster is different from the CouchBase cluster because it includes an arbiter, a master node and multiple slave nodes. MongoDB provides a replica set, a failover mechanism. There is only one Primary database that allows a write operation and multiple secondary servers only for read operations. this mechanism needs three servers: Primary, Secondary and, Arbiter. Arbiter does not store any data; it is only used during failover to decide which server will be the next primary server.

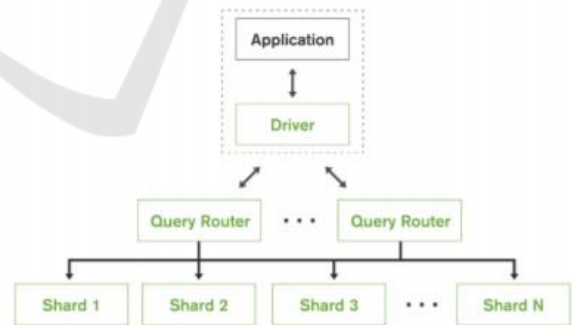


Figure 6: MongoDB Sharding (“MongoDB Architecture Guide,” 2018).

MongoDB architecture is based on four essential capabilities in meeting modern application needs: Availability; Workload isolation; Scalability; Data locality.

To fulfill these, each database is at least 3 databases as we can see in Figure 5. Unlike relational databases, MongoDB sharding (Partition method that separates the database horizontally) is automatic and

built into the database. Developers do not face the complexity of building sharding logic into their application code, which then needs to be updated as data is migrated across shards. They do not need to integrate additional clustering software or expensive shared-disk infrastructure to manage the process and data distribution, or failure recovery. Figure 6 shows us MongoDB sharding. One disadvantage of this document store is that it can be unreliable (data can easily be eliminated by mistake due to the lack of relations) and indexing takes up a lot of RAM (Read All Memory)(“MongoDB,” 2019), (“MongoDB Architecture Guide,” 2018).

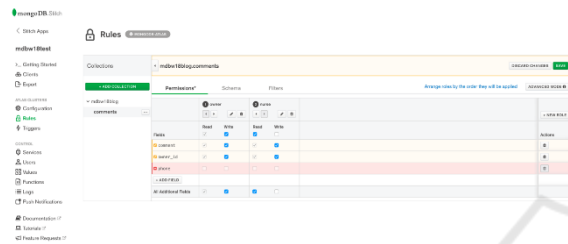


Figure 7: MongoDB Stitch interface (“MongoDb,” 2019).

In Figure 7 we can see the interface of MongoDB Stich, which stores the database in machine where it is running, as we can see on the left side of the interface there is a tab where we can access the database features, triggers, rules and others, in Figure 7 the “rules” tab is opened and as we can see that here we can add a new collection or change the existing ones.

3.4 Comparison

An advantage that Couchbase has over CouchDB and MongoDB is that it uses a SQL like a query language which makes it easier, for the user, to get the data/information that he wants (“Couchbase,” 2019b). Figures 8 and 9 are a comparison of MongoDB and Couchbase queries.

MongoDB big advantages over Couchbase and CouchDB are that it supports transactions that make it is data saver, and that, as referenced before, it uses BSON files that are faster to load (“db-engines,” 2019).

MongoDB as two major disadvantages over Couchbase and CouchDB, one is that the performance of the database rapidly degrades as the cluster size or number of users increases, the other is that since it only uses master-slave replication it is easier for it to lose data, another disadvantages are that it has limited data size (documents cannot have

more than 16MB) and has limited nesting (maximum nesting level is 100) (“data-flair,” 2018).

A big disadvantage of CouchDB over Couchbase and MongoDB is that it uses arbitrary queries, which are expensive, in terms of performance because these queries need a temporary view to be executed (Atkin, 2014).



Figure 8: MongoDB query (“Couchbase,” 2019b).

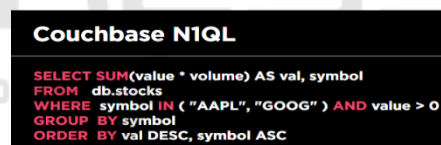


Figure 9: Couchbase, N1QL, query (“Couchbase,” 2019b).

4 EVALUATION

In this section will be presented the evaluation of these document stores through the OSSpal methodology.

OSSpal is an evaluation that helps companies and other organizations to find high-quality open-source software to match their needs. It is the successor of the Business Readiness Rating (BRR) methodology (Marinheiro et al., 2015), classified as one of the best methodologies to evaluate open-source software, combining quantitative and qualitative measures (Wasserman, 2017).

To do such an evaluation, we need to choose features and weights.

Table 2: Weight assigned the categories.

Category	Weight
Overall Quality	30%
Functionalities	20%
Stability	15%
Security	15%
Robustness	10%
Scalability	10%
Usability	10%

Table 2 summarizes the assigned weights to the categories, we gave more weight to Overall Quality than to all the other features because Overall Quality has a bit of everything in the application. We attributed more weight to Functionalities than remaining features because it is more important what the application can do then if it is stable or save. We gave more weight to Security, Stability and Usability because without these the Scalability and Robustness is compromised.

After the features are chosen and the weights are given, it is time to evaluate each feature in each application. The values to be assigned will mostly be based on g2crowd ranking, except for Robustness that is evaluated by us, given the information, we have about these document stores (“G2crowd,” 2019).

Table 3: Evaluation of the applications through the OSSpal evaluation method.

Features	Couchbase	CouchDB	MongoDB
Functionalities	3.5	3.5	4
Overall Quality	4	3.5	4.5
Robustness	5	5	4.5
Scalability	4	4	4
Stability	4	4	4
Security	3	3	4
Usability	4.5	4	4.5

In Table 3 we gave 5 to robustness in CouchDB and Couchbase because both, has referred above, support live-cluster topology changes and we gave 4.5 to MongoDB because it uses BSON files which makes reading and writing operations faster, in the other features we never attributed 5 values to any because, we believe there is always space for improvement in those features, in the evaluation we only took in account these databases and so the attributed values were given according to the data presented above.

After the evaluation of each category, the last step in this methodology is to calculate the final score. For each category, it is necessary to multiply the score with the respective weight assigned.

- **Couchbase** = $4*0.15 + 4.5*0.1 + 5*0.1 + 3*0.15 + 4*0.1 + 3.5*0.2 + 4*0.3 = 4.3$
- **CouchDB** = $4*0.15 + 4*0.1 + 5*0.1 + 3*0.15 + 4*0.1 + 3.5*0.2 + 3.5*0.3 = 4.1$
- **MongoDB** = $4*0.15 + 4.5*0.1 + 4.5*0.1 + 4*0.15 + 4*0.1 + 4*0.2 + 4.5*0.3 = 4.7$

As we can see, MongoDB is the application that obtained the best final score with the application of the OSSpal methodology, with a final score of 4.7 (from 1 to 5), Couchbase with the score of 4.3 and then CouchDB with the worst score of 4.1.

5 CONCLUSIONS AND FUTURE WORK

In this paper we can conclude that MongoDB is best open-source document store with a score of 4.7, followed by Couchbase with the score of 4.3, and in last we find CouchDB with the worst score of 4.1, but we must take in consideration that these applications were developed for different types of systems, meaning that this evaluation is made according to their evaluation on a computer operative systems like Windows. These applications are not so different, the main differences lie in what these applications were designed for, for example, CouchDB was designed for web/mobile while MongoDB was designed as a PC application.

As future work, we intend to evaluate these applications through their performance in each basic operation (creation, updating and elimination of data), through the YCSB benchmark, these tests will have in consideration the number of records, number of operations per second and the number of threads.

REFERENCES

- Abramova, V, Bernardino, J and Furtado, P (2015). Experimental evaluation of NoSQL databases. International Journal of Database Management Systems (IJDMS), Vol.6, No.3, pp. 1-16.
- Abramova, A., Bernardino J., and Furtado, P. (2014). Evaluating Cassandra scalability with YCSB,” in 25th International Conference on Database and Expert Systems Applications (DEXA), pp. 199–207, Munich, Germany, September 1-4, 2014.
- Atkin, B. (2014). Quora. Retrieved from <https://www.quora.com/What-are-the-pros-and-cons-of-CouchDB>
- Bazar, C., & Iosif, C. S. (2014). The Transition from RDBMS to NoSQL. A Comparative Analysis of Three

- Popular Non-Relational Solutions: Cassandra, MongoDB and Couchbase. *Database Systems Journal*, V(2), 49–59. Retrieved from http://www.dbjournal.ro/archive/16/16_5.pdf
- Couchbase. (2016). Retrieved from <https://blog.couchbase.com/wp-content/original-assets/2016/november/getting-comfortable-with-couchbase-mobile-installing-couchbase-server/adminpanelstart.png>
- Couchbase. (2019a). Retrieved from <https://www.couchbase.com>
- Couchbase. (2019b). Retrieved from <https://www.couchbase.com/comparing-couchbase-vs-mongodb>
- Couchbase Server architecture. (2018). Retrieved from <https://docs.couchbase.com/server/4.0/architecture/architecture-intro.html>
- CouchDB. (2019). Retrieved from <http://couchdb.apache.org>
- data-flair. (2018). Retrieved from <https://data-flair.training/blogs/advantages-of-mongodb/db-engines>
- db-engines. (2019). Retrieved from <https://db-engines.com/en>
- G2crowd. (2019). Retrieved from <https://www.g2crowd.com/categories/document-databases>
- Lourenço JR, Abramova V, Vieira M, Cabral B, Bernardino J (2015) Nosql databases: A software engineering perspective In: *New Contributions in Information Systems and Technologies*, 741–750. Springer
- Marinheiro, A., Bernardino, J. (2015). Experimental evaluation of open source business intelligence suites using OpenBRR. *IEEE Latin America Transactions* 13(3), 810–817.
- masteringionic. (2016). Retrieved from <http://masteringionic.com/perch/resources/tutorials/couchdb-database-list.png>
- Mongodb. (2019). Retrieved from <https://www.mongodb.com/assets/images/index/stitch-ss.png>
- MongoDB. (2019). Retrieved from <https://www.mongodb.com>
- MongoDB Architecture Guide. (2018). *MongoDB White Paper*, (June), 1–16.
- Nayak, A., Poriya, A., & Poojary, D. (2013). Type of NoSQL Databases and its Comparison with Relational Databases. *International Journal of Applied Information Systems*, 5(4), 16–19. <https://doi.org/10.5120/ijais15-451326>
- Sareen, P., & Kumar, P. (2015). NoSQL Database and its Comparison with SQL Database. *Int J Comput Sci Commun Networks*, 5(5), 293–298. Retrieved from <http://www.ijcscn.com/Documents/Volumes/vol5issue5/ijcscn2015050506.pdf>
- slideshare. (2016). Retrieved from <https://image.slidesharecdn.com/nosqlch4-160301045310/95/nosql-data-architecturepatterns-53-638.jpg?cb=1456808054>
- Wasserman, A. (2017). *Open Source Systems : Towards Robust Practices*.