# Performance Analysis of NoSQL Databases with Large Volumes of Open Educational Data

Felipe F. de Lima Melo
Federal Rural University of Pernambuco UFRPE
Dom Manuel Medeiros Street, Dois Irmos, Recife-PE, Brazil

Roberta M. Marques Gouveia
UFRPE

Andrêza L. De Alencar
UFRPE

Maria da Conceição M. Batista
UFRPE

Ademir B. Santos Neto
UFRPE

Tiago A. E. Ferreira
UFRPE

## ABSTRACT

Non-Relational Databases, also known as NoSQL (Not Only Structured Query Language), emerged in the face of new requirements of Web 2.0 computer applications. Relational databases, although consolidated as a data storage and manipulation model for decades, began to face performance limitations when dealing with large volumes of data. NoSQL databases have flexible data structure, and when associated with distributed computing provide a good scalability, being indicated in the Big Data scenario. In this context, this work evaluates the performance of three NoSQL databases, in order to verify their performance in large volumes of educational data. The experiments were performed with school census data, available in the repository of the Ansio Teixeira National Institute for Educational Studies and Research (INEP) in Brazil. For this case of study, the following databases were adopted: DynamoDB (whose data model is key-value oriented), MongoDB (whose data model is document-oriented), and Cassandra (whose data model is column-oriented). Therefore, among the investigated databases, MongoDB was more efficient, presenting lower processing times in the operations of inserts/loads, queries, updates, and removals of basic educational data.

## Keywords

Nonrelational Databases, Data Processing, Data Models, NoSQL Evaluation, Performance

## 1. INTRODUCTION

Relational databases have been used as data storage and manipulation model for decades and widely used by Database Management Systems (DBMS). However, in the face of new requirements of Web 2.0 computer applications, the relational model began to face limitations in performance and efficiency in relation to the manipulation of large volumes of data. According to Harish *et al.* [1], NoSQL DBs (databases) have existed since the late 1960s, however, initially, they were not called "no sql", until the early 21st century when there was an increase in their popularity, motivated by companies such as Facebook, Google, and Amazon.

The need to handle larger and larger volumes of data motivated the adoption of NoSQL databases, as this emerging and fast-growing technology tends to create systems with better performance, scalability, and easier to program [2]. NoSQL has been consolidating in this new context of different kinds of persistence, where several storage models coexist, allowing developers to choose the most suitable and optimized technology for each type of data access.

Bathla *et al.* [3] affirmed that NoSQL is complementary to SQL technology, as there are other ways to represent data than just in the tabular form of rows and columns. Non-relational models offer various storage formats that can easily support high speed, large volumes and varieties of data. Among the main NoSQL DB models are: key-value oriented, column-oriented, document-oriented, graph-oriented, and hybrid/multi-model. This paper aims to evaluate the performance of three of these non-relational DB models to verify how they perform in the scenario of large volumes of open educational data.

The NoSQL DBs adopted in this study are Cassandra (column-oriented model), DynamoDB (key-value oriented model), and MongoDB (document-oriented model). The experiments were run on school census data, which is available in the repository of the Ansio Teixeira National Institute for Educational Studies and Research (INEP). This repository contains open governmental data[1] on primary, secondary, and higher education, with information collected annually on students, classes, teachers, and educational institutions throughout the Brazilian territory. Having free access to information about basic and higher education contributes to several studies that can be applied, both in the interdisciplinary area of Data Science, and in reflections about educational public policies, epistemological aspects, and the use of information and communication technologies, as is the case of the work of Sossai *et al.* [4].

The paper is structured as follows: Section 1.1 discusses some related work; Section 1.2 discusses the three non-relational database

[1]The repository is available in: https://www.gov.br/inep/pt-br/acesso-a-informacao/dados-abertos/microdados Acessed on: March 10, 2021.

models and systems used in the experiments; Section 2 describes the methodology of the work, focusing on the experiments; Section 3 presents the performances of the three non-relational DBs and statistical analysis of the results obtained; and Section 4 brings the final considerations, with proposals for future works.

### 1.1 Non-Relational Database Systems

This section addresses the BASE (Basically Available, Soft State, Eventual Consistency) paradigm and CAP ( Consistency, Availability, and Partition tolerance) theorem. The three non-relational data models used in the experiments (DynamoDB, MongoDB, and Cassandra) are conceptualized in this section.
As Sadalage and Fowler [2], NoSQL DBs involve schema-free or flexible schema, aggregate, new distribution, and MapReduce data models. Non-relational DBs make use of the properties of the BASE paradigm (basically available, lightweight state and eventual consistency), whose system does not need to be consistent all the time, but it is considered consistent when all nodes in the cluster present the same data. Therefore, it may eventually be inconsistent, because it would be waiting for an update to be propagated to all nodes. Non-relational DBs also make use of the CAP theorem, also known as Brewer's Theorem [5], which at any given time, it is only possible to guarantee two of these three properties simultaneously in a distributed processing environment of large volumes of data. So it is necessary choose two characteristics between strong consistency (C - Consistency), high availability (A - Availability), and network partition tolerance (P - Network Partition Tolerance).
The key-value model, also known as distributed hash table, is considered easy to handle in relation to the use of stored data, has high scalability and easy implementation, since the data is stored and located by means of keys, using the concept of map or data dictionary, where these keys work as an identifier for different values. In this model, queries and insertions are performed over the key, and there is no specific size for the data, *i.e.* the data can have random size without having a specific value and can be represented by any type of data [6]. The following items are examples of DB systems that use this model: Redis, Riak, DynamoDB, Azure Table Storage, BerkeleyDB and Chordless.
In the present paper, DynamoDB was selected as the database system that implements the key-value model. DynamoDB is maintained by Amazon and was initially created to meet the needs of e-commerce operations, such as the shopping cart. The system has eventual consistency, but high availability, and high storage and retrieval power for large amounts of data. There is no need to set up clusters and it has low administration costs, as well as a flexible data model without a predefined structure.
In relationship to the document-oriented model, data is stored through a collection of documents, whose data can be located by its unique identifier or by any record that is in the document. A document has a unique identifier object and a set of fields that can be lists, strings, or nested documents [6]. In this paper, MongoDB was selected as the document-oriented database system. MongoDB is an open-source database, written in C++, and has a language that makes it easy to manipulate the data. MongoDB stores collections of documents, so it doesn't require a rigid schema, unlike relational databases. Documents in MongoDB are JSON objects and documents are stored on disk in the serialized Binary JSON (BSON) format, which is a binary representation of a JSON document.
In relation to the column-oriented model, data is stored in columns of a table, but unlike relational databases, these tables do not use the concept of relationship and are stored separately. This type of database uses the column family concept, which is used to group columns that store data of the same nature. This model was created with the intention of storing and processing large amounts of data distributed over several machines.
Shehata and Abed [7] state that column-oriented NoSQL databases are indicated when the data set is unstructured or semi-structured, and consistency can be relaxed for a while in situations where performance must be prioritized. In the column-oriented model, a large number of user requests can be answered with eventual consistency, unlike the relational DB, which focuses on having strong consistency, but at the cost of scale, and performance speed. Combining the benefits of SQL with NoSQL, referred to as NewSQL, has been researched in recent years, as exemplified by Pavlo and Aslett [8]. In that paper, Cassandra was selected as the column-oriented database system. Cassandra is an open-source database, developed in Java, created by Facebook, but currently maintained by the Apache Software Foundation. Cassandra consists of a distributed database for storing large amounts of data and is based on Google's BigTable model and Amazon's DynamoDB storage system.
The graph-oriented DB can be seen as a set of labeled and directed graphs, allowing the storage of entities and the relationships that occur between these entities, providing performance gains and better query response times. The data to be stored in a graph-oriented DB must present three basic elements: the vertices of the graphs (nodes) in which the data are stored; the edges (relationships) that represent the type of association existing between these nodes; and the attributes, which are the properties of the nodes and relationships. For Angles and Gutierrez [9] this data model became popular in the 1980s and early 1990s, and its influence gradually declined with the emergence of other DB models, however the need to manage information of a graphical nature, re-established the relevance of the graph-oriented model. The data used in this work does not follow graph structures or generalizations of them, and the data manipulation is not expressed by graph-oriented operations, so this model is not suitable for this study.

### 1.2 Related research papers

Some relevant studies regarding comparative performance analysis of NoSQL databases served as reference for the present work, such as: Davoudian *et al.* [10], Lira *et al.* [11], Soares *et al.* [12], Barros *et al.* [13], Kuhlenkamp and Ross [14], Manoharan [15], Abramova *et al.* [16], Carniel *et al.* [17], Loscio *et al.* [18], and Hecht and Jablonski [19].
The Davoudian's *et al.* [10] work's addresses the challenges of storage and query demands in the Big Data landscape, revealing several shortcomings of traditional relational DB systems. The authors aim to explain design decisions about NoSQL storage. The four principles of distributed database systems are addressed: data model, consistency model, data partitioning, and CAP theorem. Through academic and industrial NoSQL technologies, each principle is explained as to available features, strengths, and drawbacks.
Lira *et al.* [11] performed a comparative analysis between non-relational (NoSQL) and multidimensional (Data Warehouse) data models to verify which one was better suited to processing large volumes of retail sales data. For the DW, MySQL was used for the data storage, Pentaho for the ETL process - Extract, Transform and Load (ETL), and Saiku as Online Analytical Processing (OLAP) tool. Regarding NoSQL, HBase was used, a column-oriented DB, with Hadoop as the distributed system, and Hive was used to perform the queries. The results showed that the multidimensional model obtained a superior performance in relation to the non-relational model, that is, the DW used only 36% of the time

spent by HBase to perform the same queries with the market data. The authors concluded that DW was superior because a large number of complex queries are required for data analysis, which is an inherent characteristic of multidimensional models.

The work of Soares and Matos [12] analyzed the performance of three non-relational DBMSs in the context of the Internet of Things (IoT), being two document-oriented (MongoDB and Couchbase) and one key-value oriented (Redis). These DBMSs used a real IoT database - Air Quality Data Set, made available in the UCI Machine Learning Repository. The tests evaluated response time, throughput, error rate, and consumption of CPU, RAM, and storage. The results showed that each DBMS has characteristics that make it recommended for a specific scenario, depending on the application's objective, reaching the following conclusions: MongoDB is recommended for applications with reliability and disk storage constraints, Couchbase is recommended for applications that demand high speed, and Redis for applications in which there are processing consumption constraints.

Barros *et al.* [13] evaluated the performance of different storage strategies in three DBs belonging to two different paradigms: MySQL - representative of Relational DBs; Cassandra and MongoDB - representative of NoSQL DBs. In this study, genomic data (DNA sequences) were used, so that the sequencing of a single organism can generate files with gigabytes of information. The results showed that relational DBs have limitations when inserted into an environment with large amounts of data.

Kuhlenkamp *et al.* [14] warns that investigations of distributed database system performance depend on reliable experimental data, so they made use of benchmarks. The authors reproduced benchmark experiments that were conducted by previous research in order to verify the performance and scalability of the NoSQL systems HBase and Cassandra, then compare the results by measuring the speed of scaling and the impact on performance during scaling.

Li and Manoharan [15] investigated the performance of NoSQL and SQL DBs with respect to key-value storage. Read, write, instantiate, and delete operations of the following DBs were compared: MongoDB, RavenDB, CouchDB, Cassandra, Hypertable, Couchbase, Microsoft SQL Server Express. The results showed that not all NoSQL DBs perform better than SQL DBs, and in some cases were worse. Furthermore, the performance varies with each operation for each DB, *i.e.* some are slow to instantiate but fast to read, write and delete, while others are faster in other instantiate but slow at the other operations.

The work by Abramova and Bernardino [16] conclude that relational DBs are inefficient at storing and processing data in the context of Big Data. The authors compared and evaluated two NoSQL DBs: MongoDB and Cassandra.

Carniel *et al.* [17] investigated and compared DW implementations using relational and NoSQL DBs. Query processing response times, memory usage and CPU usage were evaluated considering Star Schema Benchmark queries. As a result, the column-oriented model, implemented by FastBit software, showed time reduction gains of 25.4% to 99.8% compared to the other NoSQL and relational models in query processing.

Lscio *et al.* [18] contemplated the requirements of managing large volumes of unstructured and semi-structured data. The paper presented the fundamentals of non-relational DB technology, emphasizing its main characteristics and differentials, as well as its application areas. The authors performed a case study using the MongoDB non-relational DB, and developed an application in PHP focused on using MongoDB's features, such as creating a database, inserting, removing, updating, and viewing data.

Hecht and Jablonski [19] evaluated NoSQL DB techniques considering applicability for certain requirements. The DBs were compared by their data models, query possibilities, concurrency controls, partitioning, and replication.

The differential of the present study in relation to the related works cited here refers to the fact that it performs a comparative analysis of the performance of three NoSQL DBs - column-oriented, key-value, and document-oriented, in the scenario of large volumes of open educational data, more specifically data from the Brazilian school census. It is relevant mentioning that several works already make comparisons between relational and non-relational data models, therefore, in this work the scope of relational DBs was not contemplated.

## 2. METHODOLOGY

The methodology of the work follows the following steps: (I) choose of the database in the context of large volumes of data; (II) pre-processing the data with data cleaning, standardization, and integration; (III) selection of the non-relational data models; (IV) configurations of the NoSQL database systems; (V) executions of the operations of loads/inserts, removals, queries and updates in each NoSQL database.

The database used in the experiments refers to the school census, which brings information about basic education - early childhood education, elementary school, high school, and youth and adult education. The school census consists of a nationwide survey of data on students, classes, teachers, and public and private schools carried out every year and coordinated by INEP, an agency linked to the Ministry of Education.

To clean the data, it was necessary to implement a script in PHP language, whose goal is to go through all instances of the .CSV (Comma Separated Values) file, which contains the school census 2014 data, in order to perform the automatic selection of attributes. The school census database is divided into: teacher, student (entitled "enrollment"), and school. For the experiments performed in this work, the teacher and student databases of the northern region were considered, as well as the complete school database, *i.e.*, all five regions of Brazil. Table 1 presents an overview of the school census database.

The school census database referring to schools (approximate size of 87 MB) has 143 attributes, however, after cleaning, 136 attributes were selected. The school census database for teachers in the northern region (size approximately 336 MB) originally has 126 attributes, 108 of which were selected in this study. Finally, the school census database referring to the students of the northern region (the size of approximately 1.13 GB) has 85 attributes, and 64 attributes were used after the pre-processing stage.

In this step, modifications were also made in the attributes whose instances had empty content, and the null value was replaced by a certain chosen value. The following pattern was adopted: when the empty attribute is of type string, it assumes the value "undefined", and when the empty attribute is of type integer, it was replaced by "100". The value "100" was chosen because there was no conflict with other integer values present in the other attributes, so it was chosen a value not used anywhere in the database, in order not to cause inconsistencies. This replacement of the null data was performed because the DynamoDB database does not allow the insertion of empty values.

To clean the data it was necessary to perform data integration. Due to the huge size of the teacher and enrollment files, it was necessary to divide them into several smaller files to run the PHP script developed to modify the null values. After cleaning the data, it was nec-

Table 1. Data regarding the number of instances and attributes of the School Census.

| Data Base | Number of Instances | Number of Attributes | Number of selected Attributes |
|---|---|---|---|
| School | 276.331 | 143 | 136 |
| Midwestern Teacher | 892.550 | 126 | 108 |
| North Teacher | 1.115.009 | | |
| Northeast Teacher | 3.086.414 | | |
| Southeast Teacher | 4.832.921 | | |
| South Teacher | 1.841.301 | | |
| Central West Registration | 4.125.612 | 85 | 64 |
| North Registration | 6.000.886 | | |
| Northeast Registration | 17.146.344 | | |
| South Registration | 7.310.357 | | |
| Southeast Registration | 21.481.476 | | |

essary to regroup each of these previously split tables. This process occurred in the tables Northeast teacher, Southeast teacher, North registration, Northeast registration, South registration, and Southeast registration. Figure 1 shows part of the script implemented for cleaning the data.

```
<?php
ini_set('memory_limit', '-1');
set_time_limit(0);
$infos = array();
$file_name = "../Sheets/North_Teachers.csv";
if(($handle = fopen($file_name, "r")) !== FALSE){
  while (($data = fgetcsv($handle, 0, "|")) !== FALSE){
                for ($i = 0; $i <= 125; $i++){
                        if ($data[28] == ""){
                                $data[28] = "Undefined";
                        }
                                [...]
                        if ($data[$i] == ""){
                                $data[$i] = "100";
                        }
                }
        $infos[] = $data[1].";".$data[2].[...].";".$data[114];
        }fclose($handle);
}
$fp = fopen('North_Teachers_New.csv','w');
foreach($infos as $info){
        fputcsv($fp, array($info), ';', ' ');
}
fclose($fp);
```

Fig. 1. Part of the data pre-processing script.

For the execution of the experiments, three infrastructures were defined, for each of the non-relational data models addressed in this work, which are: key-value oriented (DynamoDB), document-oriented (MongoDB), and column-oriented (Cassandra). In the case of DynamoDB the cloud computing infrastructure offered by Amazon was used. Initially, the local version provided by Amazon was used, but this version is very limited and does not provide good performance compared to the cloud environment. For this reason, it was decided to use DynamoDB in Amazon's cloud environment since it is possible to modify the machine's resources. The MongoDB test environment was created on a Notebook with 4 GB of memory, an Intel i5 processor, and 120 GB SSD HDD. Figure 2 illustrates the DynamoDB and MongoDB environment.

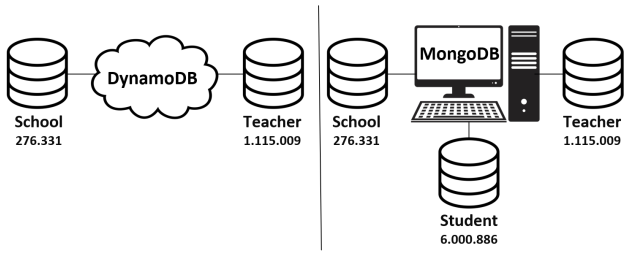


Fig. 2. DynamoDB and MongoDB infrastructures used in this work.

Figure 3 shows part of the implemented code for creating the table and inserting it into DynamoDB, respectively. When creating the table in DynamoDB, the following parameters were used: TableName, AttributeDefinitions, KeySchema, and ProvisionedThroughput. To insert the records, the `putItem` command was used, passing a data dictionary as a parameter.

In the case of the test environment of the Cassandra DB it was used the notebook with the specifications reported above, plus four machines instantiated on Amazon - Amazon Web Services (AWS) cluster. Three machines called "t2.medium 2 vCPUs" correspond to the nodes of the distributed system, and have the following configurations: 2.5 GHz, Intel Xeon Family processor, 4 GB memory, EBS, 50 GB HD, and Ubuntu Server 14.04 LTS OS (HVM). A fourth machine, called "t2.large 2 vCPUs", corresponding to the Cassandra client application, was also instantiated. The machine "t2.large 2 vCPUs" has the following configurations: 2.4 GHz, Intel Xeon Family, 8 GB memory, EBS, 50 GB HD, and Ubuntu Server 14.04 LTS (HVM). Figure 4 illustrates the Cassandra environment. Each of the chosen NoSQL DBs (DynamoDB, MongoDB and Cassandra) has particular characteristics with respect to the CAP theorem, discussed in section 1.1. All three systems were designed with a focus on scalability, aiming to support large volumes of data, however, the focus of the Cassandra system is an environment with high availability and high partitioning tolerance (AP). While MongoDB uses an environment with high consistency and high availability (CA), compromising on partitioning tolerance. DynamoDB uses an environment with strong consistency and high partitioning tolerance (CP).

Table 2 presents the comparison between the chosen NoSQL databases (DynamoDB, MongoDB and Cassandra), analyzing the following metrics: installation process, dependency on other programs, integration with operating systems (OS), database usage, integration with the PHP programming language and file size of each DB.

```php
<?php
$dynamodb->createTable(array(
        'TableName' => 'school',
        'AttributeDefinitions' => array(
                array(
                        'AttributeName' => 'id',
                        'AttributeType' => 'N'
                        )
                ),
                'KeySchema' => array(
                        array(
                                'AttributeName' => 'id',
                                'KeyType'       => 'HASH'
                        )
                ),
                'ProvisionedThroughput' => array(
                'ReadCapacityUnits'  => 5,
                'WriteCapacityUnits' => 5
                )
));
```

(a)

```php
<?php
if (($handle = fopen($file_name, "r")) !== FALSE){
   while (($data = fgetcsv($handle, 0, ";")) !== FALSE){
        $count++;
        $result = $dynamodb->putItem(array(
        'TableName' => 'escola',
        'Item' => array(
                'id' => array('S' => "$count"),
                'PK_COD_ENTIDADE' => array('S' =>  $data[0]),
                'NO_ENTIDADE' => array('S' => utf8_encode("$data[1]")),
                        [...]
                "ID_PROPOSAL_PEDAG_ALTERNATION" => array('S' =>  $data[135])
                )
        ));
   }
}
```

(b)

Fig. 3. (a) Part of the code for creating the table in DynamoDB. (b) Part of the data insertion code in DynamoDB.

Table 2. Comparison of the analyzed NoSQL databases.

| | | **Cassandra** | **DynamoDB** | **MongoDB** |
|---|---|---|---|---|
| Metrics | Install | Command line | Install via setup | Install via setup |
| | Compatibility with OS | Linux, OS X, Solaris, Windows | BSD, Linux, OS X, Windows | *Hosted* in Windows and Linux |
| | Usability | High | High | High |
| | Dependency of other software | Have | Not Have | Have |
| | Setup of the variables environment | Have | Have | Not Have |
| | Integration with PHP language | Yes | Yes | Yes |
| | Availability of tutorials for the developers | Yes | Yes | Yes |

In the installation process the availability of tutorials, ease of installation, dependency on other programs or environment variable settings, and compatibility between operating systems are analyzed. This analysis aims to provide basic, yet fundamental information about the three databases investigated in this work.

## 3. RESULTS AND DISCUSSION

The experiments of the present work began with the creation and loading of three School Census databases into the non-relational DB systems DynamoDB, Cassandra, and MongoDB. The three databases are: (I) students in the northern region, with 6,000,886 records; (II) teachers in the northern region, with 1,115,009 records; (III) schools in the 5 regions of Brazil, with 276,331 records.

These databases were chosen because they have different amounts of records and attributes, approximately 6 million, 1 million, and 276,000 records, which favored a differentiated analysis regarding the behavior of the three non-relational databases selected. Databases larger than 6 million records were not used, since the data insertion time in DynamoDB was difficult to perform experiments with this expressive volume of data. For example, for the Northeastern students' database, which has approximately 17 million records, and the Southeastern students' database, which has approximately 21 million records, it would take several days just to enter the data into DynamoDB. Even with the increased machine computational resources, and increasing the writing capacity of 100 records per second, there was no way to reverse this scenario, since the insertion time did not present a favorable situation for this DB.

Continuing the experiments, comparisons were made regarding the data creation and loading operations on the DBs. In this step a quantitative analysis was elaborated, comparing the processing time necessary for the creation and insertion of data in the "student", "teacher" and "school" data bases. Aiming at a more specific analysis, the insertions were done three times, having their times verified.

After this step, non-trivial queries were performed, seeking knowledge discovery in the non-relational databases. The tests used in order to compare the 3 data models refer to the time of: (I) Executions of seventeen queries on the school base, such as: "School type (public) x Locality (urban) x Basic Structure (Sanitation: water and sewage; Electricity, Garbage disposal, etc.)"; "School type (public) x Locality (urban) x Accessibility (disabilities)". (II) Executions of six queries on the teacher base, such as: "Teacher's Schooling x Type of Teaching School (public) x Locality of Teaching School (urban)"; "Teacher's Disability(ies) x Type of Teaching School (public) x Locality of Teaching School (urban)". (III) Executions of four queries in the student base, such as: "Teaching Stage of Enrollment x School of Attachment (public) x Locality (urban) x Gender/Color/Race"; "Teaching Stage of Enrollment x School of Attachment (public) x Locality (urban) x Age".

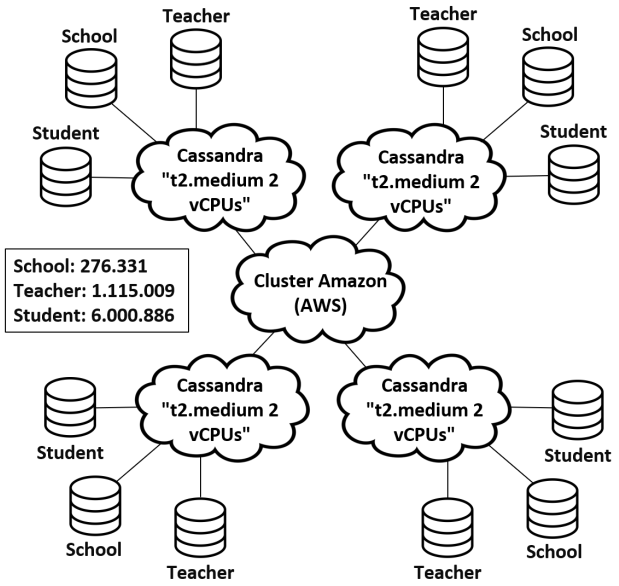


Fig. 4. Cassandra infrastructure used in this work.

Finally, the performance regarding the updating and deletion of data in the student, teacher, and school bases is evaluated, in order to diagnose which of the databases analyzed has better performance and shorter times for these operations. The following are examples of changes and deletions made in the teacher base: modify the teacher's role considering "Teacher's schooling x Type of hiring x Race x Gender x Course status". Examples of changes and removals made in the school base: modify the information on reading room, library, broadband internet access, use of indigenous materials, considering "School type x School locality x Computer lab x cultural diversity". Examples of changes and removals made to the student base: modify information about school transportation, considering "Gender/Color/Race x Pupil with disabilities x Public authority responsible for school transportation".

### 3.1 Inserts in DynamoDB, MongoDB, and Cassandra

Figure 5 shows the times obtained by the three NoSQL databases to enter data for schools, teachers and students. Database MongoDB had the best average insertion time, being much faster compared to the others.

It is possible to see in Figure 5, the time of insertion of the student database by DynamoDB is not shown in the image, since the experiment had to be interrupted, as it had been running for more than 2 days. Therefore, the results presented below are disregarding the student database. Figure 6 presents an overview of the longest times obtained when executing the CRUD operations of the school and teacher databases, through the DynamoDB, MongoDB and Cassandra databases.

MongoDB's average insertion time was "5 minutes and 56 seconds", while Cassandra took "1 hour, 50 minutes and 29 seconds" to insert the data. DynamoDB took a long time to perform the insertions, with the worst average time: "1 day, 11 hours, 1 minute and 40 seconds". When comparing Cassandra and DynamoDB, it is possible to see that Cassandra had a much faster insertion time, but could not match MongoDB's time, which was much faster for both databases. Looking at the data insertion by Cassandra and MongoDB, it can be seen that Cassandra took about 18.6 times the average MongoDB insertion time. Making the same comparison between MongoDB and DynamoDB, it can be seen that this time increases to about 353.81 times the average MongoDB insertion time. When comparing Cassandra and DynamoDB, it can be seen that Cassandra performed better than DynamoDB, being 19 times faster than DynamoDB.

### 3.2 Querys in DynamoDB, MongoDB, and Cassandra

According to Figure 6 it can be seen that the MongoDB database had the best query performance, with an average time of "29 seconds" to perform the queries. Cassandra had an average time of "2 minutes and 52 seconds" to perform the same query, while DynamoDB had the worst time, performing the query with an average time of "20 minutes and 29 seconds". When comparing the time of the MongoDB query with the Cassandra queries it can be seen that MongoDB was about 6 times faster than Cassandra. Doing the same comparison between MongoDB and DynamoDB it was found that MongoDB was about 43.5 times faster. Doing the same comparison between Cassandra and DynamoDB it was found that Cassandra was about 7.1 times faster than DynamoDB. The times presented refer to one of the queries performed on the teacher database, obtaining the schooling of the teachers who teach in public schools in the urban region. The result of this query returned 758,926 records, showing that the vast majority of teachers who teach in public schools have a college degree, which is expected.

### 3.3 Updates in DynamoDB, MongoDB, and Cassandra

According to Figure 6, analyzing the changes made to the faculty database, it can be seen that the MongoDB database showed the best update time. The change performed was "Teacher's Schooling versus Type of Hiring versus Race versus Gender". Out of 276,000 records, 85,844 records were modified. MongoDB performed these changes in approximately 13 seconds, Cassandra had an update time of "4 minutes and 56 seconds", and DynamoDB had an update time of "2 hours, 18 minutes and 7 seconds". When comparing the update time of MongoDB with Cassandra it can be seen that MongoDB performed the updates about 22.4 times faster than Cassandra. Doing the same comparison between MongoDB and DynamoDB one can see that MongoDB was about 627.9 times faster than DynamoDB. When comparing Cassandra and DynamoDB, it can be seen that Cassandra was about 28 times faster than DynamoDB.

### 3.4 Delete in DynamoDB, MongoDB, and Cassandra

According to Figure 6, analyzing the removal performed on the faculty database, the database that presented the best removal time was MongoDB. MongoDB showed a removal time of approximately 8 seconds, Cassandra showed the second-best removal time where the removal took "3 minutes and 10 seconds", while for DynamoDB the removal time was "2 hours, 18 minutes and 7 seconds". When comparing the execution time of MongoDB with Cassandra it can be seen that MongoDB was about 24.3 times faster than Cassandra. When comparing MongoDB to DynamoDB one can see that the removal time was about 1,062.4 times faster than DynamoDB. When comparing Cassandra with DynamoDB it can be seen that Cassandra showed a removal time about 43.5 times faster than DynamoDB.

As illustrated in Figure 6, the MongoDB database showed the best times in the 4 operations in which this DB was evaluated, showing better performance than the others. The DynamoDB database

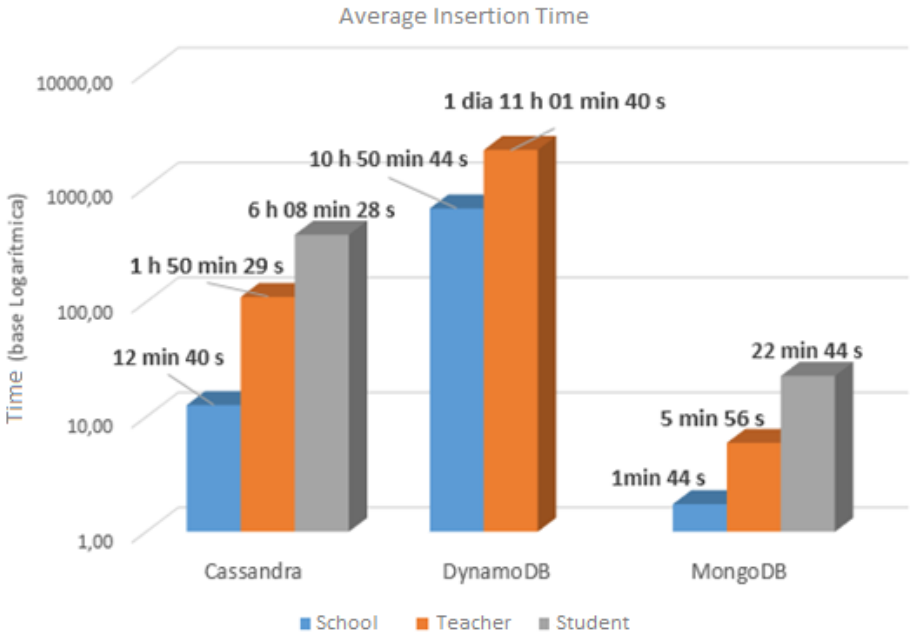


Fig. 5. DynamoDB, Cassandra and MongoDB performance: insertion times of the school, teacher and student databases.

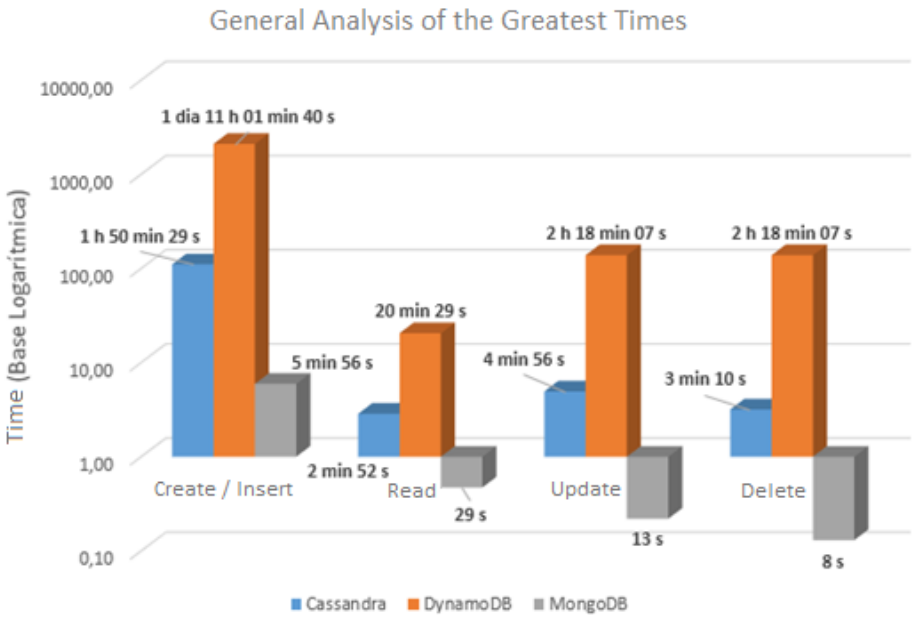


Fig. 6. DynamoDB, Cassandra and MongoDB performance: longer insertion times, queries, updates and deletes.

showed the worst performance in all operations evaluated, not performing well with the established infrastructure and the School Census data. In the case of the Cassandra database, it was always in second place, *i.e.*, it did not show time as bad as DynamoDB, but it also did not outperform MongoDB, which was very fast in performing CRUD (Create, Read, Update and Delete) operations.

You can see that database insertion was the most critical operation for DynamoDB, as it had the worst time. It is also worth noting that insertion was the most critical operation for all databases, however, it should be taken into consideration, in the case of Cassandra and DynamoDB, the network latency, because there was a very high discrepancy between the minimum and maximum value in the insertion time for these DBs. In the Cassandra case, the inser-

tion of the student database had the following results: lowest time corresponding to “4 hours, 31 minutes and 32 seconds” and highest time corresponding to “8 hours, 33 minutes and 13 seconds”. In the case of DynamoDB, this can be verified when inserting the school’s database, which had the following results: the shortest time corresponding to “10 hours and 1 minute” and the longest corresponding to “11 hours, 30 minutes and 18 seconds”.
Although, DynamoDB showed the worst performance, the key-value oriented data model should not be considered the worst model, after all, only one DB system was evaluated, but there are several options in the free software community. The best thing to do would be to perform the analysis on other databases that use this model, in order to diagnose if the low performance in insert, query, update and remove operations is in fact related to the data model or the tool analyzed. Table 3 shows how many more times, approximately, it took the Cassandra and DynamoDB databases to perform the same operation as MongoDB. Figure 7 illustrates the overview of the times obtained by the insert, query, update and remove operations, which proves that MongoDB was more efficient, presenting the shortest times in the 4 operations, followed by the Cassandra DB , and finally, DynamoDB that obtained the longest times, presenting the lowest performance.

Table 3. Comparison with MongoDB.

| | **Cassandra** | **DynamoDB** |
|---|---|---|
| Insert/Load | 18 | 354 |
| Query | 6 | 43 |
| Update | 22 | 628 |
| Delete | 24 | 1.062 |

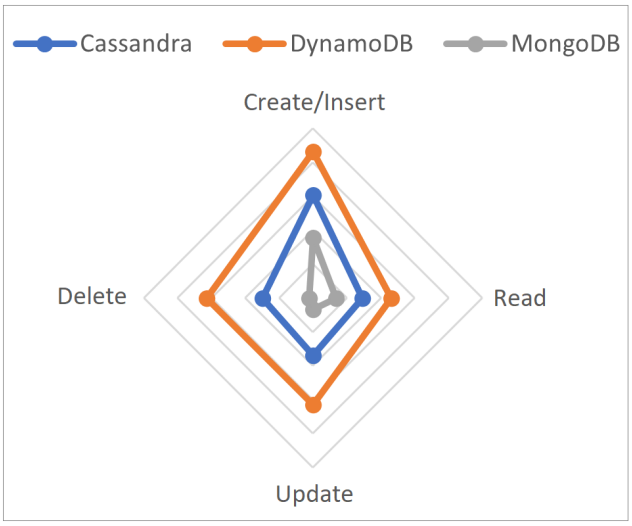


Fig. 7. Overview of the results about the experiment.

## 4. CONCLUSION

The database of the document-oriented model, MongoDB, presented the best results in several scenarios analyzed in this work through CRUD operations. In the inserts and queries, this database got the best times, as well as in the removals and changes of records, so this databse presented a good performance, superior to the others.
According to the analyses performed, the column-oriented database Cassandra also showed good results, but it was not superior to the results obtained by MongoDB. The key-value oriented database, DynamoDB, showed the lowest performance among the three databases analyzed, being the least suitable in this context of the infrastructure adopted in the work and with open data from the school census. MongoDB was 18.6 times more efficient in insertion if compared to the second-best DB (Cassandra), presenting results with 13.5 times more efficient in querying, 22.4 times more efficient in updating, and 24.3 times more efficient in removing. Therefore, among the databases analyzed in this work, MongoDB was the best suited when it comes to storing and manipulating school census data.
As future work, it is suggested to analyze the impact of using more nodes in the Cassandra clusters, in order to verify what the benefits will be for this database, thus evaluating its horizontal scalability, as well as the creation of clusters for DynamoDB and MongoDB. It is also expected to evaluate other databases embedded in the column-oriented data model - such as HBase, key-value oriented (such as Redis and Riak), and document-oriented - such as CouchDB and RavenDB.

## 5. REFERENCES

[1] Harish Kumbhar, Edberg Kinny, Kevin Fernandes, and Shirshendu Maitra. Article: Benefits of nosql databases. *IJCA Proceedings on Leveraging Information Technology for Inter-Sectoral Research*, ICAIM 2017(1):11–13, February 2019. Full text available.

[2] Pramod J. Sadalage and Martin Fowler. *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Addison-Wesley Professional, 1st edition, 2012.

[3] Gourav Bathla, Rinkle Rani, and Himanshu Aggarwal. Comparative study of nosql databases for big data storage. *International Journal of Engineering & Technology*, 7(2.6):83–87, 2018.

[4] Fernando C. Sossai, Viviane Grimm, and Carla C. Loureiro. Highlights on educational technologies and policies in brazil: an analysis of the works published by anped and rbpae (2000-2013) (translated from portuguese). *RELATEC: Latino America Magazine of Educative Technology*, 15(3):27–37, 2016.

[5] E. Brewer. Cap twelve years later: How the ”rules” have changed. *Computer*, 45(2):23–29, 2012.

[6] 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.

[7] Naglaa Saeed Shehata and Amira Hassan Abed. Big data with column oriented nosql database to overcome the drawbacks of relational databases. *International Journal of Advanced Networking and Applications - IJANA*, 11(05):4423–4428, 2020.

[8] Andrew Pavlo and Matthew Aslett. What’s really new with newsql? *SIGMOD Rec.*, 45(2):4555, September 2016.

[9] Renzo Angles and Claudio Gutierrez. Survey of graph database models. *ACM Comput. Surv.*, 40(1), February 2008.

[10] Ali Davoudian, Liu Chen, and Mengchi Liu. A survey on nosql stores. *ACM Comput. Surv.*, 51(2), April 2018.

[11] Maria Camila S. De Lira, Ademir B. Santos Neto, Maria C. Moraes Batista, Roberta Macedo M. Gouveia, and Tiago Alessandro E. Ferreira. Multidimensional and non-relational

data models: A comparison with a big volume of data. *International Journal of Computer Applications*, 175(36):1–7, Dec 2020.

[12] Allexandre S. S. Soares and Pablo F. Matos. A comparative analysis between nosql database management systems in the context of internet of things. (translated from portuguese). In *Brazilian Symposium on Databases - SBBD*, pages 306–311, 2017.

[13] Juccelino Barros, Gustavo Callou, Glauco Gonalves, Victor Wanderley, and Henrique Casteletti. Performance analysis of relational and non-relational databases in genomic data (translated from portuguese). *Theoretical and Applied Computer Magazine*, 24(2):11–27, 2017.

[14] Jörn Kuhlenkamp, Markus Klems, and Oliver Röss. Benchmarking scalability and elasticity of distributed database systems. *Proc. VLDB Endow.*, 7(12):12191230, August 2014.

[15] Y. Li and S. Manoharan. A performance comparison of sql and nosql databases. In *2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, pages 15–19, 2013.

[16] Veronika Abramova and Jorge Bernardino. Nosql databases: Mongodb vs cassandra. In *Proceedings of the International C* Conference on Computer Science and Software Engineering*, C3S2E '13, page 1422, New York, NY, USA, 2013. Association for Computing Machinery.

[17] Anderson Chaves Carniel, Aried de Aguiar Sá, Marcela Xavier Ribeiro, Renato Bueno, Cristina Dutra de Aguiar Ciferri, and Ricardo Rodrigues Ciferri. Experimental analysis of relational databases and nosql in data warehouse queries processing (translated from portuguese). In *Brazilian Symposium on Databases - SBBD*, pages 113–120, 2012.

[18] Bernadette Farias Lscio, Hlio R. Oliveira de Oliveira, and Jonas C. de S. Pontes. Nosql in the development of collaborative web applications (translated from portuguese). *VIII Brazilian Symposium on Collaborative Systems*, 10(1):11, 2011.

[19] 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.