DOI: 10.5748/18CONTECSI/PSE/ISM/6711

PROPOSAL FOR A RELATIONAL DATABASE MIGRATION METHODOLOGY TO NOSQL OF THE MASS DOCUMENT MANAGEMENT SYSTEMS

Edson Marchetti Da Silva - CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS - Orcid: https://orcid.org/0000-0003-4801-0892

Libério Afonso Geraldo Cardoso Da Silva - CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS - Orcid: https://orcid.org/0000-0002-7757-784X

Cristiano Amaral Maffort - CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS - Orcid: https://orcid.org/0000-0001-5624-9006

To propose a methodology for migrating a Relational database to the NoSQL model. . In this sense, this work proposes the migration of a database in a legal context, using Apache NiFi There are some studies in the literature that address the proposed migration between relational and NoSql database, but no references were found to the use of an ETL tool to perform this procedure. To realize tests and trials to define the DBMS NoSQL, ETL tool and the migration process steps. To execute the migration execution and to present the comparative results between the two persistence models evaluating query performance and storage space. From the results presented it is possible to conclude that the methodology proved to be effective for working with large amounts of data, taking only 0.005 seconds to migrate each process. This case study with more than 5 million lawsuits from the Public Ministry of Minas Gerais showed that using Apache NiFi as a good strategy for database migration. It is important to highlight that for the corporate scenario the proposed migration makes it possible to move from a private SQL database to an open source NoSQL database possibility of using less robust hardware and hardware easy horizontal scalability combined with improved document management performance.

Keywords: : Data Migration, NoSQL Database, ETL tools, MongoDB, NiFi

Proposal for a Relational Database Migration Methodology to NoSQL of the Mass Document Management Systems

Abstract

In recent decades, enterprise systems have moved to work with digital documents instead of conventional structured data. Many of these systems have to storage semi-structured information storage, such that using the relational approach is not the most appropriate. This Way, resulting in poor performance in retrieving that information. The objective of this paper is to propose a methodology for migrating a Relational database to the NoSQL model. There are some studies in the literature that address the proposed migration, but no references were found to the use of an ETL tool to perform this procedure. In this sense, this work proposes the migration of a database in a legal context, using Apache NiFi and presents the results found. The results indicate that the proposed methodology is effective and that the NoSQL MongoDB database, used for data loading migrated, is better than the SQL Server, to give support of the data persistency in the context of legal process retrieve. In this way, MongoDB was more efficientily than SQL Server, especially in terms of disk storage and the time taken to fetch a document. The accomplishment of this work made possible the creation of a migration methodology that can be used in corporate environments, contributing to the increased use of NoSQL databases.

Keywords: Data Migration, NoSQL Database, ETL tools, MongoDB, NiFi.

1 Introduction

Over the years, enterprise systems have shifted from working with conventional structured data to work with digital documents, which has led to an accelerated growth in the amount of data stored. This size expansion results problems with in performance and storage space . One of the most commonly used approaches to circumvent this problem is the horizontal distribution of the database servers, ie adding more servers to improve the performance of an application. However, this is not an easy and cheap task because of the difficulty of setting up an application that uses this type of database. (TOTH, 2011).

As stated by Lóscio et al. (2011) To meet the requirements of large, semi-structured or unstructured data systems that seek to improve data availability, performance and scalability, a new database category known as NoSQL has emerged[1].

This paper aims to propose a methodology for relational database migration that deals with massive document management for NoSQL model.

The relational model is currently the most used data persistence model according to Toth (2011). In the other hand, often due to lack of information or distrust from developers, NoSQL technology is discarded. In this sense, this work can serve as a success story to encourage developers and to expand the use and migration to the NoSQL model, as well as to extend the use of Extraction, Transformation and Loading (ETL) tools in the migration process between these models.

2 Theoretical foundation

In this section are presented some of the conceptual foundations that permeate this work, such as: NoSQL database and its types, main concepts about MongoDB and the Apache NiFi ETL tool.

2.1 NoSQL database and its types

For several decades, the most widely used data model for application development has been the relational one. With the emergence of the Web, new applications capable of handling

[1] It is a generic term that represents the database that does not use the relational model.

large amounts of data began to be developed, demanding new persistence models to meet the requirements of these systems. These requirements include lower level of structuring, greater need for scalability, availability and low latency. It from there that the NoSQL models become needed. (LÓSCIO; OLIVEIRA; JONAS, 2011).

Nonrelational databases were initially developed by small businesses and open source communities. To differentiate and categorize these new solutions, the term NoSQL was used. This classification is given to DBMS that do not adopt the relational model as a storage architecture. As a result, they are more flexible in ACID properties. This flexibility is justified by the need for greater scalability to handle large volumes of data as well as ensuring the its availability. (LÓSCIO; OLIVEIRA; JONAS, 2011).

NoSQL DBMS, in addition to not making use of the relational model generally do not make use of the SQL query language, which can bring more complexity for developers to adapt the systems. Usually NoSQL databases have a simplified data structure, without the presence of relationship structure, with natural support for data replication. For this reason they are also known as scalable database (DE SOUZA, 2014).

NoSQL databases are recommended for applications that handle large amounts of data and need a fast response time when searching for information, such as in: mobile devices, web applications and games. The following describes the main features of NoSQL databases.

a) **Flexibility:** flexible schemes that allow rapid and iterative development, enabling the use of semi-structured or unstructured data.
b) **Horizontal** scalability: This is the increase in the number of servers available to provide greater data processing and storage. Horizontal scalability tends to be the most viable solution as increasing the number of machines is cheaper than making existing hardware more powerful. A well-known alternative to horizontal scalability is Sharding, which consists of dividing data into several parts and storing it between a network node (LÓSCIO, 2011). The problem with that technique is that if one of the nodes stops working, the information to be fetched from the database could be corrupted.
c) **Native Replication:** Promote scalability through replication; Its purpose is to decrease the time taken for information retrieval and to increase availability. According to Lóscio et al. (2011) there are two main approaches to replication:
    - Master-Slave: each write in the bank results in N writes, where N is the number of slaves available. In this architecture writing is always done first on the master

node, which is responsible for passing the data to its slaves. This approach speeds up the retrieval of information and increases availability. However, it is not recommended for large data volumes.

- Multi-Master): In this approach, the database has several masters to reduce the bottleneck that exists in writing of the master-slave approach. However, having too many masters can cause a data conflict problem.

d) **Eventually consistent:** It is a feature of NoSQL databases that ensures that all nodes in the database will have consistent data at the end, the time taken to achieve consistency may or may not be defined (LÓSCIO; OLIVEIRA; JONAS, 2011). This feature is based on the Consistency, Availability and Partition Tolerence (CAP) theorem, which says that at any given time it is only possible to guarantee two of the three properties between consistency, availability and partition tolerance (DE SOUZA, 2014).

Oliveira et al. (2017) classified the main types of NoSQL data models into:

1. Key-Value;
2. GraphDB;
3. Column Family;
4. Document.

2.1.1 – Key-Value

According to Lóscio, Oliveira and Jonas (2011), in the key-value model the data is partitioned into a large hash table. The database consists of a set of keys, which are associated with a single string or binary value. This structure is easy to implement and enables data to be quickly accessed through your key, which contributes to increased data accessibility on systems with high scalability. The disadvantage of this model is that it does not allow retrieving objects through more complex queries. The main banks using this model include: Memcachedb[2], Voldemort[3], Redis[4], SimpleDB[5], HBase[6] e DynamoDB[7].

2.1.2 - GraphDB

[2] Available in: <https://memcached.org/> Acesso em Nov. 2018.
[3] Available in: <https://www.project-voldemort.com/> Acesso em Nov. 2018.
[4] Available in: <https://redis.io/> Acesso Nov. 2018.
[5] Available in: <https://aws.amazon.com/simpledb/> Acesso em Nov. 2018.
[6] Available in: <https://hbase.apache.org/> Acesso em Nov. 2018.
[7] Available in: <https://aws.amazon.com/dynamodb/> Acesso em Nov. 2018.

In the graph-oriented model, the idea is to represent the data as directed graphs, or with a structure that generalizes the notion of graphs. This model becomes more interesting when information about data interconnectivity or topology is more important, or as important as the data itself. (NEO4J, 2018).

Also, according to Oliveira (2017), that model has three basic components: nodes, relationships, and properties of nodes and relationships, which will be detailed below:

a. **Nodes:** represent entities, such as people, movies, cities, or anything else that needs to be kept under control. They are the equivalent of the record or row in a relational database, or the document in a document-oriented database;
b. **Relationships:** it is the lines that connect the nodes to each other, that is, they represent the relationship between them. Significant patterns emerge as we examine the connections and interconnection of nodes, properties, and relationships;
c. Properties: relevant information contained within nodes and relationships. Among the main databases that use the graph model we can highlight the Neo4j[8] and the OrientDB[9].

2.1.3 Column Oriented

The column-oriented model resembles the key-value model, but with a higher level of complexity. In this model data is indexed by a triple (row, column and timestamp), where rows and columns are identified by braces and timestamp allows differentiating different versions of the same data. In a column-oriented database read and write operations happen atomically, that is, all values associated with a row are considered in the execution of these operations, regardless of the columns being read or written. Another concept associated with the model is the column family, which is used to group columns that store the same data type (LÓSCIO; OLIVEIRA; JONAS, 2011).

Column-oriented storage delivers great performance for analytic querying as it considerably reduces disk input and output requirements and decreases the amount of data to be loaded onto disk. This model is designed to scale up horizontally database servers to increase throughput, that is, to increase the amount of data returned in a query over the same timeframe, by using distributed clusters, which results in low cost hardware. This model is

[8] Available in: <https://neo4j.com/> Acesso em Nov. 2018.
[9] Available in: <https://orientdb.com/> Acesso em Nov. 2018.

ideal for data warehousing and big data processing (AMAZON, 2018). Among the main databases that use the column-oriented model stand out:BigTable[10] e Cassandra[11].

2.1.4 Documents Oriented

The document-oriented model basically stores document collections. A document is usually a unique identifier object, consisting of a set of fields, which can be strings, lists, or aligned documents. The fields found in the document-oriented model are very similar to the key-value model, the difference is that in the key-value model we have a single hash table created for the entire bank, whereas in the document-oriented model we have a set of documents and For each document we have a set of key-value pairs, that is, we have a distinct hash table for each document (LÓSCIO; OLIVEIRA; JONAS, 2011).

Documents persisted in this template are usually stored in JSON type, which makes them intuitive for developers, giving them the power to persist data using the same template format as they use in their application source code. A document may or may not contain the same data structure as the others, and each document is not necessarily dependent on any other document (AMAZON, 2018).

2.2 MongoDB

MongoDB is a fully scalable and flexible open source document-oriented database implemented in the C ++ language. One of its peculiarities is data storage made using the BSON format, which is basically a way to represent JSON data in binary. MongoDB works in a distributed manner, enabling high availability, horizontal scaling and geographic distribution in a bank-integrated and easy-to-use manner (MONGODB, 2018).

The main reason MongoDB deviates from the relational model is that it makes scaling easier, but there are other advantages as well. The basic idea is to replace the concept of a "table" with a more flexible model, the "document". By enabling embedded documents and arrays, the document-oriented approach makes it possible to represent complex hierarchical relationships in a single record. MongoDB is also schema free: document keys are not predefined. New or missing keys can be handled at the application level instead of forcing all data to have the same shape. This gives a lot of flexibility to developers working with evolving data. (CHODOROW, 2013).

[10] Available in: <https://cloud.google.com/bigtable> Acesso em Nov. 2018
[11] Available in: http://cassandra.apache.org/ Acesso em Nov. 2018.

2.3 - Apache NiFi

The Apache NiFi is an open source ETL tool used to automate data flows between systems (NIFI, 2018). It provides real-time control that makes it easy to manage the movements that the data follows during a flow, so it is possible to see where the data is at an exact moment and where it goes later. It is a data source independent tool supporting heterogeneous and distributed data sources that carry different data formats.

According to Hortoworks (2018), a software company that supports Apache Nifi, the tool was based on a technology called "NiagaraFiles", previously developed by the United States National Security Agency (NSA), which was made available to the Apache Software Foundation through the NSA Technology Transfer Program. Designed to be flexible, extensible and suitable for a wide range of devices, it can work since small business networks to large data clusters in a corporate as well as cloud environment.

Apache NiFi is a tool based on flow-oriented programming concepts, supporting powerful and scalable routing graphs for routing and data transformation. Following are showed some features taken from the Nifi User Guide**[12]**.

- **FlowFile:** represents a unitary piece of data in NiFi, is here that the data enters NiFi and is housed in FlowFile until the end of its processing. In addition to data content, FlowFile has attributes to facilitate identification of this data during the flow, or to assist in the development of some data handling logic. FlowFile is a very important component in NiFi as it persists in memory even if there is a connection in the system preventing failures during the flow of execution.
- **Processor: t**his is a NiFi component used to listen to extracted data from external sources, publish data to external sources, route, transform, and extract information from FlowFile**.**
- **Relationship:** Each Processor has zero or more Relationships defined for it. These relationships are named to indicate the result of processing a FlowFile.
- **Controlling services:** These are extension points that will start when NiFi is started and will provide information for use by other components (such as processors or other controlling services).
- **Interface NiFi:** It provides mechanisms for creating automated data streams, as well as view, edit, monitor and manage these data streams. This interface will configure the set of processors and relationships that will give rise to the modeled flow.

[12] Available in: <https://nifi.apache.org/docs/nifi-docs/html/user-guide.html> Acesso em Mai. 2019.

3 Related works

In the literature it was possible to find several studies based on relational database migration to NoSQL in different scenarios and making use of different technologies. In this section 3 of the most relevant and adherent to the main idea of this work will be listed.

Murari (2014) presents the development the of software for migrating a Firebird database to a nonrelational MongoDB. The proposed software is developed in Python language. The author models data coming from Firebird to the MongoDB base and, after migration, performs validations to ensure that data is not lost during execution. After validation, data in MongoDB goes through an optimization process that aims to remove columns with no or identical content in all records to fit the data, ensuring that information is available and accessible quickly and easily.

Finally, it performs a comparative analysis between migrated database, in order to evaluate databases' performance in terms of information retrieval time, concluding that non-relational database performed better than relational ones.

In the work of Gomes (2011), he highlights that changing from nonrelational to relational paradigm is a major challenge. New data models and user interfaces force a radical change in the mindset of programmers, and bring new responsibilities to the programmer due to the lack of transactional guarantees, the loss of explicit relationships between data, and the control of parameters such as definitions of coherence by operations. Thus, he proposes to try to resolve the separation between relational and non-relational paradigms, by means of object mapping, which presents the reading, writing and updating codes of a relational database as opposed to the equivalent code for a nonrelational database. An analysis is also performed between read, write and update operations for both databases.

Oliveira (2017) aims in his study to validate the existing migration methods from relational to nonrelational systems and perform an adaptation to them, in order to understand which method is most efficient. The paper also shows how to implement and analyze the environment to be migrated.

4 Methodology

The methodology used in this work consists of five steps: (1) literature review, addressing nonrelational database models and ETL tools; (2) search for main works related to the proposed theme; (3) tests and trials to define the DBMS, ETL tool and the migration process steps; (4) migration execution; (5) presentation of comparative results between the two persistence models evaluating query performance and storage space.

With the chosen tools the migration methodology will be detailed in the topics presented follow.

5 – Application of proposed methodology

This section presents the scenario the step-by-step of the migration process and the analysis of the results obtained.

5.1 – Migration scenario

This work was carried out in the context of the Public Prosecution Service of Minas Gerais (MPMG) to migrate a massive legal process management system, with about 5,181,621 legal processes, from the relational persistence model to NoSQL. Simply put, each process consists of a set of attributes stored in multiple tables in a SQL Server database.

Through the theoretical study done during this work, two possible ways to perform the migration process between database systems were verified.

The first of these is the use of a programming language in which the developer is responsible for extracting data from a source, manipulating the extracted data as needed and loading it onto the target storage system. However, the disadvantage of this method is that the optimizations made during the process depend on the choices made by the developer and the resources offered by the chosen language, which can slow the process.

The second way, which was adopted in this paper, is the migration through an ETL tool, after all, by using a tool the migration process becomes more agile and safer. In this sense, the Pentaho and NiFi tools were analyzed. In addition, some ETL tools still allow the use of programming languages at some stages of the process, allowing for a good migration customization.

The tool chosen was Apache NiFi, your choice is justified by:

1. Be an open source*;*
2. Present a good performance proven by tests performed by the authors of this work;

3. Be a flow-oriented tool, which provides more simplicity to assemble the model and transparency in visualizing exactly the path that the data took to reach the destination;
4. Have a user-friendly web interface to assist in mounting flows and executing the migration process.

MongoDB was the NoSQL database chosen by:

1. Be an open source;
2. Have the data model that best fits the proposed scenario, the document-oriented model;
3. Be the most widely used document-oriented database in the world.

The hardware used to perform the migration process is presented in Table 1.

Table 1 - Hardware used to migration

Servers	**Memory RAM**	**Processor**	**Operational System**	**Version**
SQL Server	132GB	Intel Xeon, 12 cores, 2.67 Ghz	Microsoft Windows Server 2019	Microsoft SQL Server 14.0.1000.169
Apanhe NiFi	16 GB	Intel Xeon, 4 cores, 2.2 Ghz	Red Hat Enterprise Linux 7	Nifi 1.8
MongoDB	16 GB	Intel Xeon, 4 cores, 2.2 Ghz	Red Hat Enterprise Linux 7	MongoDB 4.0

Source: the authors.

5.2 Application of proposed methodology

To facilitate the understanding of the proposed methodology for migration it is divided into three steps: Data Extraction; Data transformation; Data Loading.

5.2.1 Data extraction

The first step of migration is to extract data from the SQL Server database. To perform that procedure, flow into the NiFi tool with the 'ExecuteSQL' processor is started. In order for this processor to work, the following properties must be set: ‘SQL select query’ which contains the SQL command responsible for extracting data from the source database; Database Connection Pooling Service is responsible for defining the data for connection to the SQL Server database.

The SQL statement is intended to gather all information from migrated documents. Therefore the idea is to join all the tables necessary for the composition of the documents by means of their foreign keys, with the aid of joins of the relational model. In this sense, from the DOCUMENT_JUDICIAL table joins with the other tables, most of them through the Left Join clause, after all there are several partial relationships, in which foreign keys accept nulls.

5.3.2 Data Transformation

The second part of the migration process is about transforming and manipulating the data that was extracted in the previous step. The ExecuteSQL processor returns through the success relationship a FlowFile with Avro-type content, containing all processes together, as if they were a table, where each row is equivalent to one process. The SplitAvro processor was then used to split this content into multiple FlowFiles containing only one process each. The return of this processor is transferred to the split relationship, which has added the ConvertAvroToJSON processor to convert the contents of Avro format to JSON, a more user friendly format for handling data in the stream sequence.

As MPMG processes have different categories, that is, different structures because they contain different attributes, it was therefore necessary to route this flow to treat processes differently by category. This routing was given through an attribute that is stored in the content of the lawsuit. This information was extracted in JSON format making it a FlowFile attribute to be used in the EvaluateJsonPath processor, which transfers its return in the matched relationship.

Based on the category attribute the 'RouteOnAttribute' processor can direct each process to the path specified in 'Route Strategy'. Eleven process categories were identified. Because NiFi allows more than one route specified in Routing Strategy to be directed to the same path in the flow. For the scenario in question, more than one category can be treated in the same way. Thus, the number of relationships leaving the ‘RouteOnAttribute’ processor has been reduced to five categories.

The next step is the modeling of routed data. For the proposed model it was necessary five different models, although very similar. To accomplish this task was used the processor

'ExecuteScript', which allows to use some programming language to model the data. The language of choice was JavaScript[13]. It was chosen because of its lightness and ease of handling JSON format data.

For the 'ExecuteScript' processor to work, it was necessary to set the‘ Script Engine ’property, which defines the language , and the 'Script Body' property, where the JavaScript script responsible for data manipulation was added.

In the elaborated JavaScript code, FlowFile is received through the session.get() function, and the modeling part happens within the session.write() function, such that JSON is extracted from FlowFile. The data modeling should be done so that data in a document is stored logically in MongoDB. For the scenario of this work, for example, the properties 'TRADID' and 'TRADOM' contained in the JSON object, gave rise to a subobject' newObj ['county'] = {'id': TRADID, 'name': TRADOM} . So the main idea of manipulation is basically to join attributes that in SQL Server were part of the same table into a JSON subobject that will be stored in MongoDB. That is, a main object to represent the process, with several subobjects that represent the data that was stored in the other tables of SQL Server and are part of the composition of this process. At the end of the manipulation, the new JSON is inserted into FlowFile and transferred to the processor success ratio via the session.transfer() function.

5.2.3 Data Loading

The third and final step of the proposed methodology is to store the transformed data in MongoDB.

The processor responsible for entering data into MongoDB requires that FlowFile content be in Avro format. According to Vohra (2016) Avro is a binary data serialization format that provides a varied data structure. Avro uses a JSON schema to serialize and deserialize its data. For this reason, the process data that was converted to JSON in the data transformation and manipulation step should be returned to Avro format. To perform this task will use the processor 'ConvertJSONToAvro’.

To convert JSON to Avro format you need to define a schema using the 'Record Schema' property. That schema is used as a form of JSON validation that will be converted. It states the type and format of each JSON attribute that will enter the processor.

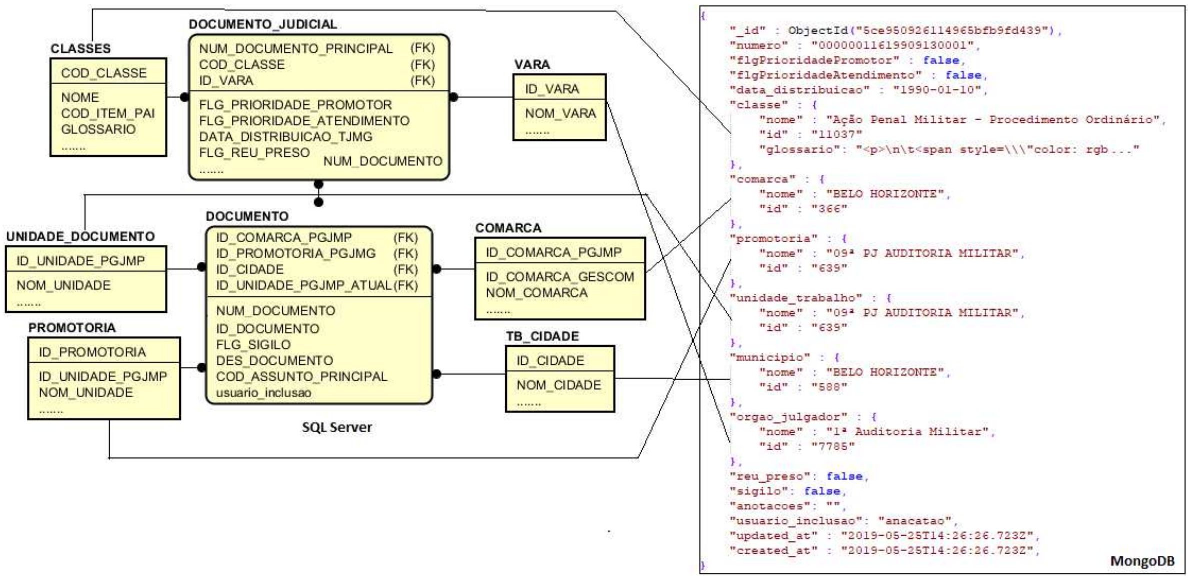
[13] Available in: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript> Acesso em Mai. 2019.

The last step is to insert the processes into MongoDB through the 'PutMongoRecord' processor. For the 'PutMongoRecord' processor to work you must set 'Mongo Database Name', with the database name and 'Mongo Collection Name' property as the MongoDB collection name.

Additionally, as this processor performs an external data publication it is necessary to specify the 'Client Service' property, which connects with the MongoDB server and the 'Record Reader' which is responsible for taking the Avro schema present in the received FlowFile, in order to to validate Avro and prepare it to be inserted into MongoDB.

The Figure 1 shows the mapping of part of a SQL Server process migrated to MongoDB, through the image it is possible to understand how the document was in the relational model and how it was in the document oriented model.

Figure1 – Relational model Mapping for NoSQL



```
{
    "_id" : ObjectId("5ce950926114965bfb9fd439"),
    "numero" : "00000011619909130001",
    "flgPrioridadePromotor" : false,
    "flgPrioridadeAtendimento" : false,
    "data_distribuicao" : "1990-01-10",
    "classe" : {
        "nome" : "Ação Penal Militar - Procedimento Ordinário",
        "id" : "11037"
        "glossario": "<p>\n\t<span style=\\\"color: rgb..."
    },
    "comarca" : {
        "nome" : "BELO HORIZONTE",
        "id" : "366"
    },
    "promotoria" : {
        "nome" : "09ª PJ AUDITORIA MILITAR",
        "id" : "639"
    },
    "unidade_trabalho" : {
        "nome" : "09ª PJ AUDITORIA MILITAR",
        "id" : "639"
    },
    "municipio" : {
        "nome" : "BELO HORIZONTE",
        "id" : "588"
    },
    "orgao_julgador" : {
        "nome" : "1ª Auditoria Militar",
        "id" : "7785"
    },
    "reu_preso": false,
    "sigilo": false,
    "anotacoes": "",
    "usuario_inclusao": "anacatao",
    "updated_at" : "2019-05-25T14:26:26.723Z",
    "created_at" : "2019-05-25T14:26:26.723Z",
}
```

Source: The authors.

5.3 Migration processing and result analysis

The migration was divided into 6 steps due to NiFi server hardware limitations and to avoid congestion in the MPMG SQL Server database. In the first 5 steps, 1 million processes were migrated in each one, and in the sixth stage, the remaining processes were migrated.

The following are the results of this migration, which compares the information fetch performance and storage space used in the SQL Server and MongoDB databases. SQL Server queries were executed using the Microsoft SQL Server Management Studio tool and MongoDB commands were executed using the Roto3T[14]**,** both free tools.

5.3.1 Time consumed in migration

The first analysis performed is the time spent by the migration process. The time spent at each migration step takes into account the moment the migration is started in NiFi, until the last process for this step is entered in MongoDB. The times spent per migration step is presented by Table 2.

Table 2 – Spent time of the migration.

Step of migration	Time spent in seconds
1	4.487
2	4.815
3	4.374
4	4.874
5	4.966
6	2.834
Total	26.350

Source: The authors.

It can be observed that the total time spent by the migration was 26,350 seconds, taking into account the total number of processes migrated. The time taken to migrate each process is approximately 0.005 seconds.

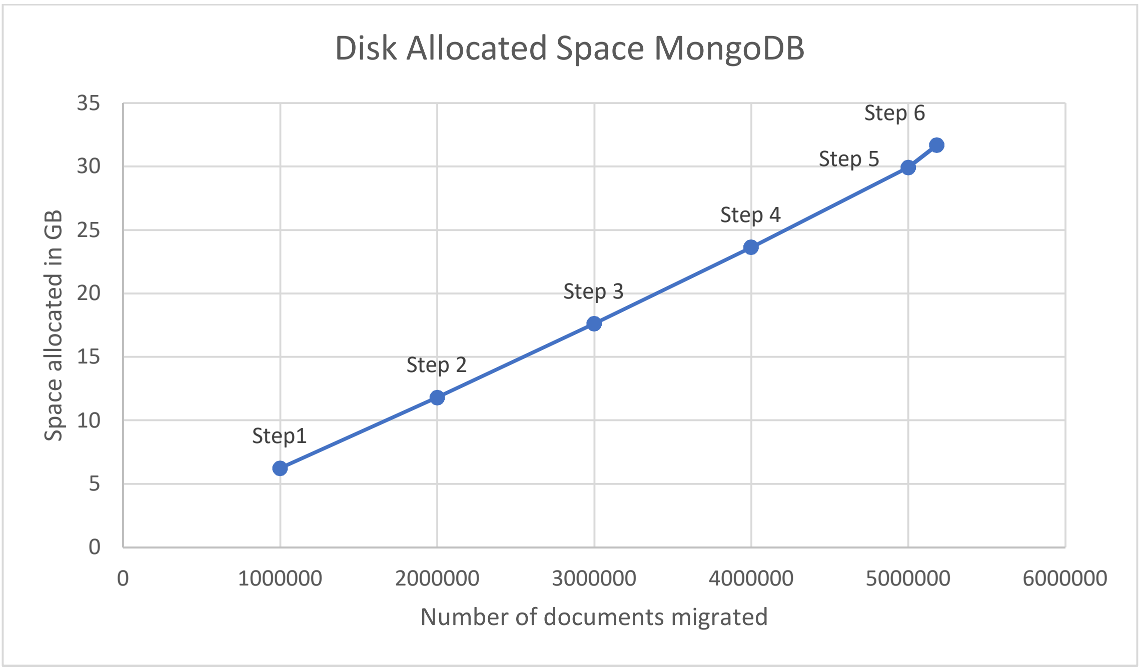
5.3.2 Espaço alocado em disco

A segunda análise a ser feita refere-se à comparação do espaço alocado em disco no SQL Server e no MongoDB.

At each migration step, the disk space allocated by MongoDB was recovered through the command: ‘db.processos.storageSize()’. At the end of the migration the database had an allocated disk space of 31.68 GB. The Figure 2 shows the evolution of allocated disk space in MongoDB by number of documents migrated in each step..

[14] Available in: <https://robomongo.org/> Disponível em Mai. 2019.

Figura 1 - Evolution of disk space allocate d by MongoDB in migration



Source: The authors.

To obtain disk space allocated by SQL Server, the space allocated by all process-related tables obtained by the command: ‘sp_spaceused’ in SQL Server has been summed.

The SQL Server has allocated a space of 45,720,744 Kb and MongoDB has allocated a space of 31,680,000 KBs. Taking into account the total amount of processes, SQL Server spends an average of 8.82 Kb per process while MongoDB spends 6.11 Kb per process, resulting in approximately 30.7% disk space savings even though stored data in a non-standard way.

5.3.3 Search response time

The third analysis aims to evaluate the performance of SQL Server and MongoDB Banks. Five tests were performed based on process-finding situations that are considered important and costly for MPMG SQL Server, according to information obtained from database administrators, which are:

1. Search for lawsuits by district: Its purpose is to search for lawsuits that belong to a particular county;
2. Search prosecution by prosecution: Its purpose is to search for processes that belong to a particular prosecution;

3. Search for lawsuits by the judging body: Its purpose is to search for processes that belong to a particular judging body;
4. Search for lawsuits by class: It has the purpose of searching lawsuits based on the judicial classes they have;
5. Search for lawsuits by municipalities: Its purpose is to search for lawsuits that were initiated in a particular municipality.

For all tests the worst case was used, that is, the search that returns as many processes as possible to have a more reliable analysis of bank performance. SQL Server search times were obtained through 'Client Statistics', provided by the tool: Microsoft SQL Server Management Studio. MongoDB's search times were obtained through the ‘explain (" executionStats ")’ command provided by the database it self.

The tests were performed using the same search parameters for both databases and the times of this procedure, in milliseconds, is presented in Table 3.

Table 3 - SQL Server *vs* MongoDB search time.

Number	**SQL Server Time (ms)**	**MongoDB Time (ms)**	**Number of process returned**	**% response time reduction**
1	793.225	49.686	640.033	93,74
2	524.179	12.041	230.811	97,70
3	20.114	528	14.190	97,37
4	30.439	15.103	172.654	50,38
5	225.015	25.988	285.907	88,45
Avarage	318.594,4	20.669,2	268.719	93,51

Source: The authors.

As you can see, for all searches MongoDB had a considerably shorter response time than SQL Server. MongoDB had an average time of about 0.077 ms to fetch a process, while SQL Server had an average time of about 1.19 ms. That is, MongoDB is on average 93.51% faster than SQL Server for the tests presented.

This performance difference can be explained by the structure of the indices adopted in each DBMS. SQL Server normally uses keys stored in a B^+ tree, according to Cormen (2009) this algorithm has a constant performance O (log n) to search for information. On the other hand, MongoDB uses keys stored in a hash table, which under reasonable assumptions, according to the same author, can achieve O(1) performance in the search for information.

5.3.4 Response block size

The fourth and final analysis aims to evaluate the size of banks' returns to perform a search.

The searches used for this test were the same as the five used for the response time test shown earlier. To calculate the size of an object returned in MongoDB is used the method ‘Object.bsonsize()’, so to calculate the total size of the response was used the sum of the sizes of all objects that make up the response. In SQL Server the total size of the response was obtained through the 'Client Statistics', provided by the tool: Microsoft SQL Server Management Studio, after the query execution. Table 4 presents the results obtained.

Table 4 - Response Size SQL Server vs MongoDB.

Number	SQL Server (GB)	MongoDB (GB)	Number of documents
1	11,18	11,46	640.033
2	5,15	5,08	230.811
3	0,30	0,26	14.190
4	0,41	0,42	172.654
5	4,14	4,23	285.907
Average	4,24	4,29	268.719

Source: The authors

As you can see the results presented, without the size of the response, banks are technically tied. MongoDB has an average response size of 15.97 KB per process, while SQL Server has an average response size of approximately 15.76 KB per process, ie SQL Server shows a response size reduction of approximately 1, 31% compared to MongoDB.

A smaller response size may cause a loss of packets transferred over the network or perform a database request, which may decrease traffic from application bottleneck changes, but this response size analysis between MongoDB and SQL O server is not meaningful, or makes it indifferent to choose one of the two base types in this regard.

6 Conclusion

From the results presented it is possible to conclude that the methodology proved to be effective for working with large amounts of data, taking only 0.005 seconds to migrate each process. It is noteworthy that this time could have been better had the migration occurred all at once, which was not possible in this work due to hardware limitations.

The Apache NiFi tool makes the methodology very robust, because if the migration process crashes and gets interrupted, FlowFiles are persisted to disk and ensures that when the migration process resumes it starts where it left off.

The tests also concluded that MongoDB outperforms SQL Server for storing massive data in document format, with 30.7% less footprint and 93.51% faster search time, even with hardware well below SQL Server. What was expected because MongoDB is a document-oriented bank is therefore extremely optimized for storing massive documents.

In analyzing the size of the responses returned by MongoDB and SQL Server, banks were technically tied, meaning that the network infrastructure used to communicate with the database tends to continue with the same traffic after migration.

Another noteworthy aspect is the decreased complexity to look for a process from SQL Server to MongoDB after migration. While SQL requires several joins and a more complex query, MongoDB can retrieve a process with a simple JSON query.

Finally, it is important to highlight that for the corporate scenario the proposed migration makes it possible to move from a private SQL database to an open source NoSQL database, saving money by not having access license agreements and the possibility of using less robust hardware and hardware. easy horizontal scalability combined with improved document management performance.

As future work, migration can be performed with larger disk storage space, which will allow the entire migration to be performed at one time to check if there will be an improvement in the total migration time, which was not done in this work due to this hardware limitation. Also conducting a study that uses similar hardware to persist DBMS models, thereby eliminating this bias when comparing performance.

REFERENCES

AMAZON, Web Services. Available in: <https://aws.amazon.com/>. Accessed in: Nov 01, 2020.

AVRO, Apache Software fundation. Available in: <https://avro.apache.org/>. Accessed in: May 18, 2021.

CHODOROW, Kristina. **MongoDB: the definitive guide: powerful and scalable data storage.** " O'Reilly Media, Inc.", 2013.

CORMEN, Thomas H. et al. **Introduction to algorithms**. MIT press, 2009.

DE SOUZA, Alexandre Morais et al. Critérios para Seleção de SGBD NoSQL: o Ponto de Vista de Especialistas com base na Literatura. **Anais do X Simpósio Brasileiro de Sistemas de Informação (Londrina–PR, Brasil.** May 27-30**, 2014)**.

FERREIRA, João et al. O processo ETL em sistemas data warehouse. In: **INForum**. 2010. p. 757-765.

GOMES, Pedro Filipe Linhares. **Migração de aplicações legadas para bases de dados NOSQL**. 2011. Master Dissertation.

HORTOWORKS, **Apache Nifi**. Available in: <https://br.hortonworks.com/apache/nifi/>. Accessed in: Nov 27, 2018.

LÓSCIO, Bernadette Farias; OLIVEIRA, Hélio Rodrigues de; JONAS, CSP. NoSQL no desenvolvimento de aplicações Web colaborativas, 2011.

MACÁRIO, Carla Geovana do N.; BALDO, Stefano Monteiro. O Modelo Relacional. **Instituto de Computation Institute of State University of Campinas. Campinas, São Paulo**, 2005.

MONGODB, Document Database. Available in: <https://www.mongodb.com/>. Accessed in: Nov 01, 2020.

MURARI, Mauro André. **Desenvolvimento de um software para migração de um banco de dados relacional Firebird para o não relacional MongoDB**. 2014.

NEO4j, Graph Platform. Available in: <https://neo4j.com/>. Accessed in: Nov 01, 2020.

NIFI, Apache. Disponível em < https://nifi.apache.org/>. Accessed in: Nov 01, 2020.

OLIVEIRA, Fábio Vieira del. **Migração de bases de dados relacionais para NoSQL: métodos de análise**. 2017. Master Dissertation.

TOTH, Renato Molina. Abordagem NoSQL – uma real alternativa. Federal **University of São Carlos,** Apr. 2011.

VIEIRA, Marcos Rodrigues et al. Bancos de Dados NoSQL: conceitos, ferramentas, linguagens e estudos de casos no contexto de Big Data. **Simpósio Brasileiro de Bancos de Dados**, 2012.

VOHRA, Deepak. **Practical Hadoop Ecosystem: A Definitive Guide to Hadoop-Related Frameworks and Tools**. Apress, 2016.