Applications of NoSQL Architecture and Graphs in Unstructured Data Warehouse for Testing and Visualization

Introduction

With current advancements and progress in the data analytics field, it’s commonly known that unstructured data has played a great role in all business transactions and will likely continue to grow in both size and scope. Meanwhile, the data warehouse (DW) database system we’ve established in courses like ISTE-724 and research have demonstrated the versatility and great functionality it provides for users seeking to obtain and understand information about data in a fast and efficient manner. And while the on-line analytical processing (OLAP) DW system in its basic form is not equipped to handle unstructured data, there are different methods of implementation that can address this directly, such as the topic dimensional model utilized in this paper.

This paper primarily touches on the background of unstructured data in the data warehouse and some approaches in applying graphs and NoSQL to existing data warehouse structures. It aims to explore techniques in applying the architecture and unique representational capabilities of NoSQL databases to the data mart schema, analyzing different measures of effectiveness, and evaluating the usability based on the principles of data warehouse querying. And while there are many ways to represent this sorted data in other means, it may be best to use tools that are already available to us such as existing NoSQL databases like MongoDB, Redis, RavenDB, Neo4j, and so on.

Two proposed solutions exist in the form of MongoDB and Neo4j, both of which will function as examples of NoSQL implementations that meet our requirements for testing and visualization. MongoDB is a NoSQL database platform that operates using JSON-like files and schemas. It has more similarities to MySQL than Neo4j, and can lead to easier translatability between the two. In Mongo, the data mart schema can be represented in a single document while still maintaining the existing relations for query operations. Meanwhile, Neo4j can visually represent the data warehouse and show how the different dimensions within the data mart schema are interlinked. Performance can be measured not only in terms of accurate mapping, but also size and performance as well as complexity for an end-user.

The Data and DW Model

For this paper, we will be using the topic distribution model implemented by Dr. Kang and the topic dimensional table that enables processing OLAP operations for unstructured data, and particularly for our Amazon reviews data (as we’ve established this process in both RA and Capstone work, I’ve left this section brief for time). The data itself consists of customer reviews for certain electronic and kids’ products on Amazon’s website, complete with review title, text, and rating. The data mart is built around this unstructured data after undergoing preprocessing and the extraction, transformation, and loading (ETL) process through Python and Pentaho Spoon respectively. To build the topic model, we use Latent Dirichlet Allocation (LDA), hierarchical Latent Dirichlet Allocation (hLDA), and cosine similarity to construct a hierarchical topic dimension that links to each individual customer review and fit to the data mart [1] [2].

Diagram

Description automatically generated

Figure 1: The Base Data Mart Schema

Standard SQL vs NoSQL

Contrary to popular belief, NoSQL does not mean “no SQL processes involved”. It is in fact a shorthand notation for “not only SQL”, meaning that it combines elements of SQL with other additional tools and methods of representing and storing data that would be absent in a typical SQL database [3]. Some of the major differences can be defined via the approach and implementation of the two types of databases. Primarily, SQL consists of relational entities and tables, while NoSQL databases provide flexibility (in part due to the various implementations) through document and graph stores, along with options like wide-column or key-value stores.

In order to understand and subsequently successfully transition between the two different systems, the contrasts in database structure can be “connected” in order to match the database sub-components between SQL and NoSQL. In MongoDB, for example, a collection is equivalent to a table in MySQL, where a column would equate to a field [4]. These differences are mostly nominal, and the means of storage are similar. The main difference would be in how the data fits directly into the database; both MongoDB and Neo4j have and utilize the key-value pairs to signify “belonging” to any particular column, or field in NoSQL.

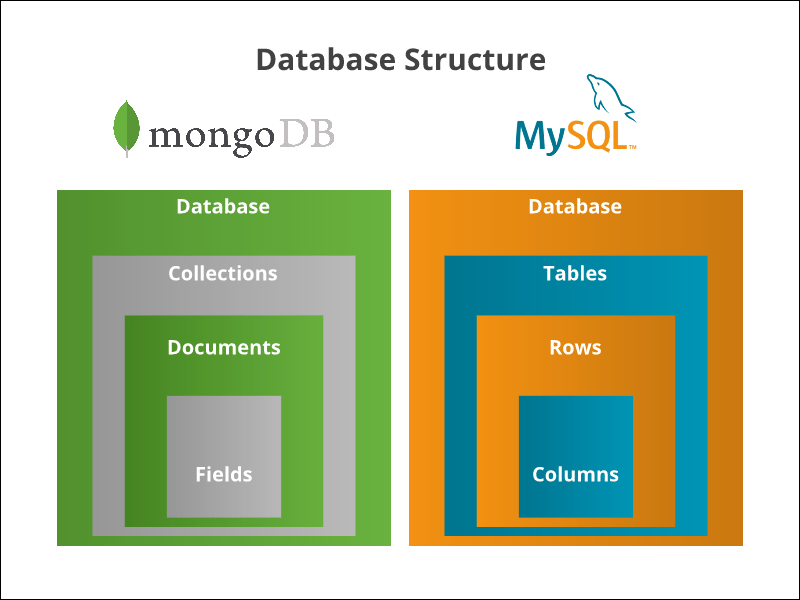


Figure 2: NoSQL vs SQL Architecture (Simplified)

An example of “direct” translation between dimensions in SQL and Neo4j (Figure 3) [5]. The format in the figure will look different from a “star schema” layout, which would be more centralized around one single node.

Diagram

Description automatically generatedDiagram

Description automatically generated

Figure 3: SQL vs Graph Translation

OLAP vs OLTP

OLAP provides a single platform for different types of business needs and analysis for end-user, while OLTP focuses on day-to-day transactions and administration of large numbers of these short online transactions. The data warehouse is the most prominent example of an OLAP database, which functions as an analytics tool to transform data into information. Thus, MongoDB and other OLTP platforms are not able to directly translate the functionality of a data warehouse. However, as discussed in our attached Feasibility Study, so long as the functionality and information are retained, all that is left is to apply our knowledge of the systems to adjust what cannot be directly incorporated. Of course, this will also be a factor in determining the feasibility of our exploration and decide on how well our OLTP platforms can represent a data warehouse without compromising too many of its OLAP functionalities. And while the process of ETL is what bridges the OLTP database and the OLAP DW, our study and feasibility paper show that there is room for an implementation that uses both.

MongoDB

For MongoDB, the records are stored as key-value pairs within records; each “column” is recorded as the key, and there isn’t a “table” or “dimension” that the database entries are slotted into. For a date warehouse, an individual collection (similar to a table) would serve as the dimensional and fact tables in MySQL, and the aggregate “$join” function can be used to query across the two. And for more on-line accessible functionality, we can use both MongoDB Atlas and Compass for utility and visual representation, and while Atlas focuses more on cloud clustering and security while Compass emphasize insight and feedback for querying, they both are more useful alternative platforms for different user needs.

Graphical user interface

Description automatically generated

Figure 3.5 MapReduce Process

MongoDB’s key:value pairs (as seen in Figure 4) also show that for each “entry” the whole purpose is to read and write these pairs. Aggregations are also difficult to perform without using the map-reduce operation. In Mongo, the id for each entry is unique and fits into the role of a Surrogate Key, and a summary on product categories or date could be done using this command. In a basic, command-prompt style of doing things, it will be difficult for the end user to not only understand what they are seeing, but also interpret the data in such a manner that insightful queries can be performed. To address this issue, we can use GUI platforms for MongoDB, MongoDB Atlas and MongoDB Compass, to determine if a solution can be found.

mongoimport --db TestRA --collection Revs --type csv --headerline --file C:MongoDB/Amazon\_Topics.csv



Text

Description automatically generated

Figure 4: Local MongoDB

Graphical user interface, text, application

Description automatically generated

Figure 5: MongoDB Atlas

Atlas’ most prominent usage is for connection, particularly between the MongoDB database and external sources that use this organized data, such as in Javascript or standard JAVA code, as demonstrated in ISTE-610’s final project, where Atlas was a very useful tool. It also incorporates certain aspects of MongoDB Compass, but lacks a good import tool that can translate the data mart, and instead requires manual input that would not be possible on a database of such size.

Graphical user interface, text, application, email

Description automatically generated

Figure 6: MongoDB Compass

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Figure 7: “Aggregation” Mongo Compass tool

MongoDB Compass, meanwhile, serves the purposes of this paper far better, as it enables not only a better way to handle and visualize the data and individual records for fast querying, but it also allows for modified querying that can “imitate” OLAP queries on higher levels that the end-user might take advantage of. In Compass, an aggregation tool is provided that saves the user efforts from searching a query, instead only needing to enter in key-value fields that the user wants to perform queries on. These can range from “matching” fields, which is the most common way of aggregating, to merging, counting, and performing unions on separate documents. Figure 7 shows a sample query that the user can input, choosing a certain product category and Level 2 topic to summarize on and match.

One other way to “fix” the translation issue is to use external software, which could allow for implementing SQL queries and using them to parse MongoDB. This has some advantages compared to simply using MySQL or a designer tool like MySQL Workbench, particularly when it comes to speed, time and scaling (size concerns). This, of course, would require a lot more effort on the development side of things, but can simplify the task for the user.

Neo4j

The inclusion of graphs and visuals can be considered Neo4j’s primary selling point, as it incorporates the elements of an E-R diagram and alters it in such a way that the relations can be represented via the relationships between nodes in the graph database. Compared to SQL, Neo4j is not as versatile or functional in querying and operations, but instead provides a way to visualize relations individually and allow for specifically tailored representation of data in nodes and relations. Loading in the data, therefore, requires a particular and cautious approach, as after loaded in there are few options to manipulate the data itself. Some of the column headers are also renamed due to Neo4j syntax, particularly the columns with periods in the names, which are more difficult to deal with than spaces.

Neo4j is executed via the local connection and the community connector software that links the batch file to the console for the user to edit. By running the line

C:\neo4j-community-4.1.3\bin\neo4j.bat console

Or creating a batch file to perform the operation, Neo4j can be accessed (note, depending on the version of Neo4j a particular version of JAVA SDK may be required). Access can then be granted by entering the following url:

http://localhost:7474/browser/

In the console, a username and password must be provided for login, which should match the username and password used by the local Mongo system. If these don’t match, changes should be made in the Neo4j conf folder.

The end result should look something like this:

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generatedFigure 8: Loaded data in Neo4j output

Loading the csv in is simple, but in Neo4j we need to establish the relationships between the individual “dimensions”, or nodes within the data. Note that the structure of the document follows the key-value pairing that MongoDB uses, as both follow the same NoSQL format. However, no relationships are established yet, and as such an output would only yield nodes, as seen in Figure 8.5.

Background pattern

Description automatically generated

Figure 8.5 A sample of the loaded individual reviews, with no relations established

Graphical user interface, text, application

Description automatically generated

Chart, diagram, schematic, bubble chart

Description automatically generatedChart, bubble chart

Description automatically generated

A picture containing room

Description automatically generated

Figure 9: Query Outputs (see attached for code)

Right off the bat, it’s quite clear that things can get very difficult immediately. Figure 9 shows the outputs for a single topic-distribution, multiple smaller topic-distributions (aka associated with fewer individual reviews), and a holistic representation of all the variables present, aka a fact table representation, respectively. While smaller clusters of distributions can be worked with and easily identified, the more “popular” and frequent products, reviews, and topics can become very muddled very quickly, especially if the user wants a holistic picture including ALL of the relationships present as shown in the third output of Figure 9.

Limitations Analysis and Conclusion

While an in-depth feasibility study/survey paper is included in this report, this section deals more with the “benefit of hindsight” we obtain after the progress we’ve made. First, the issues of translation as described in the beginning of the semester are very present our work. A large amount of this has to do with the simple fact of being built for different functions. While the NoSQL systems share many functionalities with SQL, in part being built atop SQL, trying to make it consistent and imitate a data warehouse in full is a futile endeavor. This is why we focused in on the visualization and representation aspect of using NoSQL as a TOOL to (quoting our proposal) “apply graphs in a sensible way to existing data warehouse structures”. For this, we’ve chosen to create a database within these NoSQL platforms that imitates the standard data mart as close as possible, complete with tables and relationships, and then use these platforms in their capacity to represent our data.

In the beginning, before starting this project in full swing, I’d also made the assumption that as MongoDB and Neo4j had different methods of representation, the means for the end-user to aggregate the data and view summary information was built-in, particularly with Mongos’ Map-Reduce and Neo’s visual nodes that could be “grouped”. Both these systems also possessed the equivalent “functions” perform aggregation, so it was just assumed that the functions COULD be performed in the feasibility report without considering the end-user’s perspective. For MongoDB in particular, querying each time requires a scan through the full collection, and connecting to each individual table is very exhaustive on both the system and user’s time, even if performed on a cloud platform like MongoDB Atlas. After all, 280,000 entries is still a lot, no matter how fast it can be processed, and in NoSQL EACH of these entries contains fields that are independent as a key-value pairing.

Ultimately, these systems have shown that they DO not only possess the ability to load in an unstructured data mart and work with the data in similar ways to the standard SQL practices, but the unique applications they possess can visualize the data in alternative interpretations for the user to work with in different scenarios. Just because it’s doable doesn’t necessarily mean that it should be done, or that it’s optimal. MongoDB is first and foremost an OLTP service and is geared towards such uses, while Neo4j struggles with large amounts of data, and the “clarity” it boasts becomes very quickly muddled up when we are not looking at a subset or extremely summarized version of the data, the latter of which is very difficult to do as we’ve established aggregations are hard to represent in graph form. One major concern that also is problematic with a full NoSQL implementation is the date dimension, which while included is simply a “value”, and version control for time-based querying would have to rely solely on the date and time dimensions without the ability to implement slowly changing dimensions (SCD) to the fold. However, despite all these shortcomings, NoSQL has still demonstrated its ability to not only apply unstructured data, but also maximize its effectiveness and use its strengths in representing a data warehouse.

References (Not including Survey References)

[1] Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. the Journal of machine Learning research, 3, 993-1022.

[2] Griffiths, T., Jordan, M., Tenenbaum, J., & Blei, D. (2003). Hierarchical topic models and the nested Chinese restaurant process. Advances in neural information processing systems, 16.

[3] <https://www.mongodb.com/nosql-explained>

[4] <https://phoenixnap.com/kb/mongodb-vs-mysql>

[5] Vaisman, A., Besteiro, F., & Valverde, M. (2019, September). Modelling and querying star and snowflake warehouses using graph databases. In *European Conference on Advances in Databases and Information Systems* (pp. 144-152). Springer, Cham.

[6] Lawrence, R. (2014, March). Integration and virtualization of relational SQL and NoSQL systems including MySQL and MongoDB. In *2014 International Conference on Computational Science and Computational Intelligence* (Vol. 1, pp. 285-290). IEEE.

[7] Liu, Y., & Vitolo, T. M. (2013, June). Graph data warehouse: Steps to integrating graph databases into the traditional conceptual structure of a data warehouse. In *2013 IEEE International Congress on Big Data* (pp. 433-434). IEEE.