A Qualitative Comparison of NoSQL Data Stores

Sarah H. Kamal[1]
Information Systems Department
Akhbar Elyom Academy
6th October City, Egypt

Hanan H. Elazhary[2]
Computers and Systems Department
Electronics Research Institute
Cairo, Egypt

Ehab E. Hassanein[3]
Information Systems Department
Faculty of Computers and Information, Cairo University
Cairo, Egypt

***Abstract*—Due to the proliferation of big data with large volume, velocity, complexity, and distribution among remote servers, it became obvious that traditional relational databases are unsuitable for meeting the requirements of such data. This led to the emergence of a novel technology among organizations and business enterprises; NoSQL datastores. Today such datastores have become popular alternatives to traditional relational databases, since their schema-less data models can manipulate and handle a huge amount of structured, semi-structured and unstructured data, with high speed and immense distribution. Those data stores are of four basic types, and numerous instances have been developed under each type. This implies the need to understand the differences among them and how to select the most suitable one for any given data. Unfortunately, research efforts in the literature either consider differences from a theoretical point of view (without real use cases), or address performance issues such as speed and storage, which is insufficient to give researchers deep insight into the mapping of a given data structure to a given NoSQL datastore type. Hence, this paper provides a qualitative comparison among three popular datastores of different types (Redis, Neo4j, and MongoDB) using a real use case of each type, translated to the others. It thus highlights the inherent differences among them, and hence what data structures each of them suits most.**

***Keywords*—Document datastores; graph datastores; key-value datastores; MonoDB; Neo4j; NoSQL datastores; Redis**

I. Introduction

In the past few years, we experienced a tremendous growth in the amount of data resulting in what is called "big data." Big data is generally distinguished by large volume, which may reach petabytes or much higher; high velocity, possibly from several locations; large variety, structured, semi-structured, and/or unstructured; and distribution, in different locales, data centers, or cloud geo-zones [1] [2]. This entitled the need to store such complex data, and it was obvious that traditional relational databases were not suitable to meet those requirements [3]. This led to the emergence of a new breed of data management systems, referred to as NoSQL datastores.

NoSQL, which means "Not only SQL" is a generic term of database management systems (DBMS), which provide a mechanism for storing and retrieving data different from that of relational DBMS, and hence, traditional SQL queries over the data cannot be applied to them. A basic feature of most NoSQL datastores is the "shared nothing" horizontal scaling, which allows them to execute a huge number of read/write operations per second [4]. Non-relational databases are generally known for their schema-less data models, improved performance and scalability. We summarize the importance and genuine need of NoSQL data stores as follows [2]:

- extendibility to handle future growth of data
- efficiency and ability to deal with fast data
- flexibility of data formats
- ability to handle data partitioned across multiple servers to meet the growing data storage requirements
- remote access
- the continuous availability of such datastores online

There are four basic types of NoSQL data stores in the broad sense: key-value, document, graph, and column. A huge number of cloud datastores have been developed under each category. This implies the need to understand the differences among such data stores, and which is more suitable to any given data. Unfortunately, research efforts towards this issue are either theoretical (without showing real implementations), or deal with performance issues such as speed, which are characteristics of the specific studied datastores. The goal of this paper is to present a qualitative comparison among three popular datastores of different types (Redis, Neo4j, and MongoDB) using a real use cases of each type, translated to the others. It thus highlights the inherent differences among them with respect to their data definition strategy, and hence what data structures each of them suits most.

The rest of the paper is organized as follows: Section II presents a discussion of the different types of NoSQL data stores and a popular example of each. Section III presents related work in the literature to highlight our contribution. Section IV provides a qualitative comparison of three popular data stores of different types; in addition to a discussion of the results. Finally, Section V presents the conclusion of the paper and directions for future research.

II. Types of NoSQL Datastores

In this section, we discuss the four basic types of NoSQL data stores and a popular example of each.

A. Key-Value Datastores

The use of key-value datastores indicates that the stored values guide to a specific key, and the only appropriate way to query about data is through the key. Those datastores use a data structure similar to those employed in maps and dictionaries, where data can be manipulated and handled using

a unique key [4]. The flexibility of those datastores makes it convenient to store data in unstructured format. They also allow fast and huge random read/write requests, and highly scalable retrieval of requested data [5]. Such datastores are used by Facebook to store posts with unique Ids. The value of a given unique id contains a real message, identity of the user and time of sharing the corresponding post [6]. Key-value datastores are appropriate in cases when you want to store a user's session or a user's shopping cart or to get information about favorite products. Fig. 1 illustrates a simple example data structure of a key-value datastore. As shown in the figure, three users are identified by their Ids, and the only stored values indexed by those Ids are their first names.

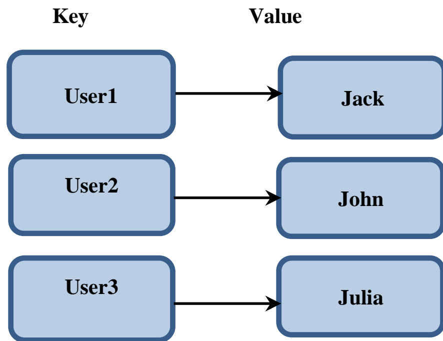

Fig 1. Simple Data Structure of a Key-Value Datastore.

One of the most popular key-value datastores is Redis developed by Salvatore Sanfilippo [7]. This open-source datastore has the ability to provide fast and huge random read/write requests. It can handle more than 100,000 read or write operation per second. It also supports different types of data structures such as strings, hashes, lists, sets, sorted sets, bitmaps, and geospatial indexes. It also has built-in replicas that can be replicated using the master-slave model, and a master can have multiple slaves [8].

B. Document Datastores

Document datastores are used to store and organize data in the form of documents. The documents allow storing and retrieving data in numerous formats such as XML (Extensible Markup Language), PDF and JSON (Java Script Object Notation). Those datastores are very flexible in nature since they are schema-less. They are also characterized by the ability to add a large number of different fields to one or more documents without wasting space by adding the same empty fields to other documents [9] [10]. Documents are grouped together into collections. Though a collection is composed of many documents, each document can have different schemas and different types of stored data. Each document holds a unique Id within its corresponding collection. Document datastores are suitable for web applications, which involve storage of semi-structured data and the execution of dynamic queries. Fig. 2 depicts a simple example data structure of a document datastore.

MongoDB is one of the most popular open-source document datastores, written in C++ programming language and developed by Software Company 10gen [11]. It is a high performance and efficient datastore. It is also a flexible, schema-less datastore that can include one or more collections of documents. It can be used to store and customize large files like images and videos. It also has a complex query language and supports MapReduce to process distributed data [2]. The documents in the figure store information regarding products, their branches and their corresponding orders.

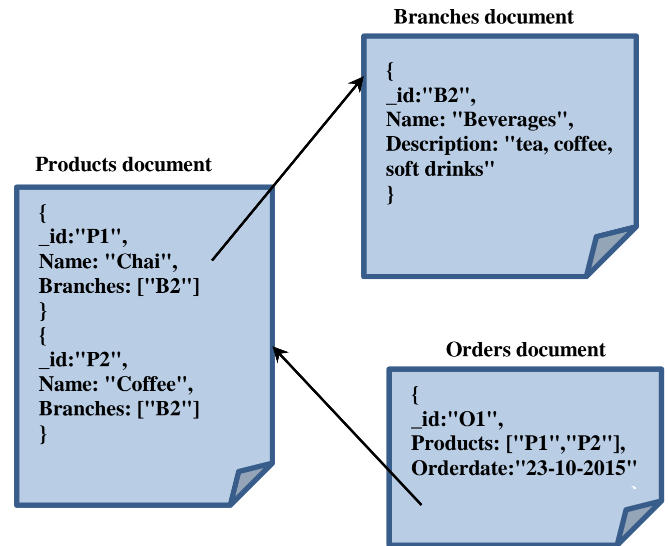

Fig 2. Simple Data Structure of a Document Datastore.

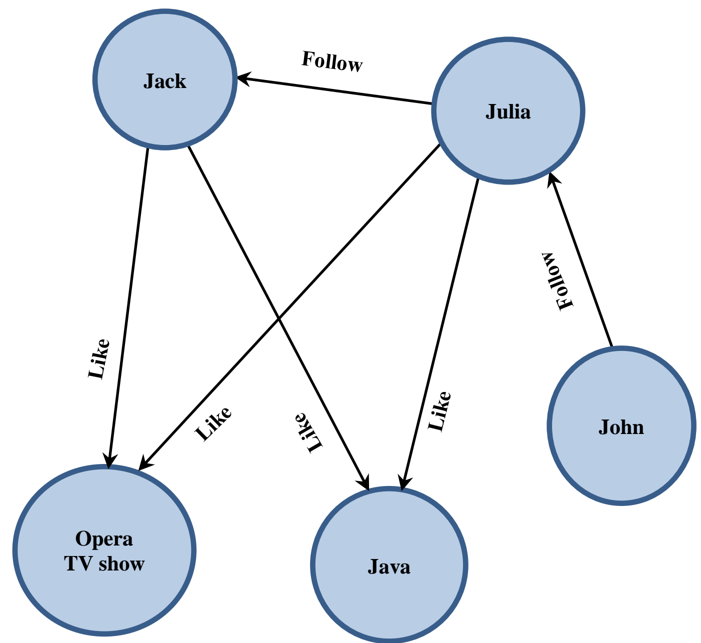

Fig 3. Simple Data Structure of a Graph Datastore.

C. Graph Datastores

Graph datastores are designed around the idea of a graph structure which contains nodes, properties and connecting edges. Nodes represent entities, properties describe real information about the entities and edges represent the relationships between nodes. Graph datastores use sophisticated shortest path algorithms to make the process of querying data more efficient. Most of those datastores are

schema-less and few of them support horizontal scaling because it is difficult to traverse and manipulate graph when connected nodes are spread on clusters. Graph databases are specialized in path finding problems in navigation systems [12]. They are also designed to be suitable for representing heavily linked data such as social relations, geographic data, social networking sites, bioinformatics and cloud management [13]. Fig. 3 depicts a simple example data structure of a graph datastore, with its nodes and directed edges. It shows a number of users, and what they like and who they follow.

Neo4j is one of the most popular and powerful graph datastores, written in Java [14]. It is a high performance graph databases which can provide a flexible network structure. It is highly available and scalable since it has the ability to store and organize massive numbers of nodes and relations between them effectively. It has a cypher query language, which is used for fast querying and efficient traversal. It also offers a representational state transfer (REST) interface and Java application program interfaces (APIs) [10].

D. *Column Datastores*

Column datastores are designed to store huge numbers of columns. Data is stored based on column values. Though those datastores are the most similar to their traditional relational counterparts, they are able to overcome the drawbacks of the latter databases, as they remove null values from columns, when values are unknown. They support high scalability since column data can be distributed on several clusters easily. They are also most suitable for data mining and analytics applications [15]. Most of those datastores employ MapReduce framework to speed up processing of large amounts of data distributed on numerous clusters [10]. Fig. 4 illustrates a simple example data structure of a column datastore. It stores information similar to that of the document datastore in Fig. 2, but in a different column-oriented format.

Product_id
1
2
3
4

Branch
Beverages
Seafood

Product
Soft drinks
coffee
tea
shrimp

Order_date
23-10-2015
25-10-2015
27-10-2015

Fig 4. Simple Data Structure of a Column Datastore.

One of the popular column data stores is Cassandra, which was developed by Apache Software Foundation, and implemented in Java. It is based on both Amazon's DynamoDB key-value datastore and Google's Bigtable column datastore, so it includes concepts of both datastore types. It supports high availability, partitioning tolerance, persistence and high scalability. It also has a dynamic schema. It can be used for a variety of applications like social networking websites, banking and finance, and real time data analytics [16].

III. Related Work

This section discusses research studies dedicated to comparisons involving NoSQL datastores. Some authors were mainly concerned with the differences between relational databases and non-relational alternatives especially NoSQL datastores. For example, Makris et al. [4] reviewed the concepts of relational and NoSQL datastores and the differences between them based on schemas, transaction methodologies, complexity, fault tolerance, consistency and dealing with storage of big data. Nayak et al. [16] also provided a comparison between both parties, and concluded that a lot of effort is needed to introduce a standard query language for NoSQL datastores. Sahatqija et al. [17] also reviewed the pros and cons of NoSQL datastores over relational databases. Corbellini et al. [18] provided a similar comparison, using a set of examples. Kumar et al. [19] provided a discussion of the problem of relational databases and how NoSQL datastores are the best solution for handling them by discussing and comparing two popular document datastores MongoDB and CouchDB.

Other researchers were mainly concerned with comparing the different types of NoSQL datastores, but as previously noted, without showing implementations of real use cases. For example, Srivastava et al. [6] discussed the pros and cons of six popular NoSQL datastores. Padhy et al. [11] provided a thorough discussion of NoSQL storage technology, types of NoSQL datastores, and the differences among them. Han et al. [20] provided a comparison from a totally different point of view, which is the dependency on the CAP theorem. They described the basic characteristics and data models of NoSQL datastores, and classified them according to this theorem.

Another research direction is concerned with studying the performance of NoSQL and SQL databases. For example, Parker et al. [21], experimented with MongoDB as an example document NoSQL datastore, and SQL Server as a traditional relational database. They compared the performance of both parties. The results proved that MongoDB is faster in terms of insert, update and simple queries; whereas SQL Server performs better in terms of update, queries with non-key attributes, and aggregate queries. Li and Manoharan [22], examined the performance of some NoSQL datastores and SQL databases. They compared the read, write and delete operations, and observed that not all NoSQL datastores perform better than the SQL databases. Specifically, RavenDB and CouchDB do not perform well in terms of read, write and delete operations. Cassandra is slow on read operations, but good for write and delete operations. Additionally, Couchbase and MongoDB are the fastest in general for read, write and delete operations. Okman et al. [23] provided a comparison

from a different point of view, which is data security. The authors focused only on MongoDB and Cassandra as two of the most popular NoSQL datastores. They found that both of them lack encryption support for data files, have weak authentication, and very simple authorization.

Adding two documents to branches, showing their names and specializations

```
> db.Branches.insert([
.. {_id:"B1",
..  Name:"seafood",
..  description:"fish"
.. },
.. {_id:"B2",
..  Name:"beverages",
..  description:"tea,coffee,softdrinks"
.. }
.. ]);
```

Adding two documents to suppliers, showing their names and locations

```
> db.suppliers.insert([
... {_id:"S1",
... companyname:"tokyo traders",
... country:"Japan"
... },
... {_id:"S2",
... companyname:" new orleans"
... country:"USA"
... }
... ]);
```

Adding three documents to products showing their names; and referencing their suppliers and branches

```
> db.Products.insert([
.. {_id:"P1",
... productname:"Chai",
... suppliers:["S2"],
... Branches:["B2"]
... },
.. {_id:"P2",
... productname:"shrimp",
... suppliers:["S1"],
... Branches:["B1"]
... },
. {_id:"P3",
.. productname:"coffee",
.. suppliers:["S1","S2"],
.. Branches:["B2"]
.. }
. ]);
```

Adding two documents to orders referencing the included products and showing their dates

```
> db.orders.insert([
... {_id:"O1",
... products:["P1","P3"],
... orderdate:"23-10-2015"
... },
... {_id:"O2",
... products:["P2","P3"],
... orderdate:"25-10-2015"
... }
... ]);
```

Fig 5. MongoDB Example.

IV. Comparative Study

According to the above discussion, the main contribution of this paper is to conduct a qualitative comparison based on intensive experimentation with three popular NoSQL datastores using real use cases for each type, translated to the others. Specifically, we selected Redis as an example key-value datastore, MongoDB as an example document datastore, and Neo4j as an example graph datastore.

A. *MongoDB Example*

A document datastore example was implemented using MongoDB. The example involves relating documents of a set of collections like products, branches, suppliers and orders. Fig. 5 shows the implementation of this example. As shown in the figure, MongoDB uses a set of *db.<collection>.insert ()* instructions to add document(s) to collections. Each document has a number of fields (attributes). The _id field is automatically generated for a new document if the field is not defined. In this example, we employ document references to relate documents in different collections.

Forming sets and adding values for branches and suppliers

```
> sadd Branches "Branch:1" "Branch:2"
(integer) 2
> hmset Branch:1 name seafood description fish
OK
> hmset Branch:2 name beverages description "tea,coffee,softdrink"
OK
> sadd suppliers "supplier:1" "supplier:2"
(integer) 2
> hmset supplier:1 companyname tokyotraders country Japan
OK
> hmset supplier:2 companyname neworleans country USA
OK
```

Forming sets and adding values for products and orders

```
> sadd products "product:1" "product:2" "product:3"
(integer) 3
> hmset product:1 productname chai
OK
> hmset product:2 productname shrimp
OK
> hmset product:3 productname coffee
OK
> sadd orders "order:1" "order:2"
(integer) 2
> hmset order:1 orderdate "23-10-2015"
OK
> hmset order:2 orderdate "25-10-2015"
OK
```

Fig 6. Translation Example of MongoDB to Redis.

1) Document to key-value datastore: In Redis, the sadd instruction is used to add one or more member keys to a set, while the hmset instruction is used to add values to fields in the hash stored at a given key. Fig. 6 shows the translation of the MongoDB example to Redis using those instructions. To illustrate, as shown in the figure, we use sadd to add the keys of two branches to a single set. We then use hmset to add two fields and their respective values to the hash stored at each of them. For example, in case of Branch:1, the name is "seafood" and the description is "fish." In order to implement relationships between entities, we need to use the sadd instruction to implement each relationship and its inverse, as shown in Fig. 7. It is obvious that representing relationships is an overwhelming task in Redis.

Relating products to suppliers, orders, and branches; suppliers to products; branches to products; and orders to products

```
> sadd product:1:suppliers 2
(integer) 1
> sadd product:1:Branches 2
(integer) 1
> sadd product:1:orders 1
(integer) 1
> sadd product:2:suppliers 1
(integer) 1
> sadd product:2:Branches 1
(integer) 1
> sadd product:2:orders 2
(integer) 1
> sadd product:3:suppliers 1 2
(integer) 2
> sadd product:3:Branches 2
(integer) 1
> sadd product:3:orders 1 2
(integer) 2
> sadd supplier:1:products 2 3
(integer) 2
> sadd supplier:2:products 1 3
(integer) 2
> sadd Branch:1:products 2
(integer) 1
> sadd Branch:2:products 1 3
(integer) 2
> sadd order:1:products 1 3
(integer) 2
> sadd order:2:products 2 3
(integer) 2
```

Fig 7. Translation Example of MongoDB to Redis (cont.).

It is worth noting that we considered representing such relationships as attributes as in the case of MongoDB, but in Redis, keys added as values of *relationship* attributes will not reference their corresponding entities.

2) Document to graph datastore: Finally, Fig. 8 shows the translation of MongoDB example to Neo4j. As shown in the figure, each entity regardless of its type is represented as a node in a graph (with hidden attributes), and the relationships are represented using directed arrows. It is clear that Neo4j does not normally consider collections or sets of entities. We can, for example, add a node representing each collection and let it point to its members as shown in Fig. 9. Though this would do the job, the graph will become too cumbersome. Alternatively, we can add the name of each collection as an attribute to each of its members. Nevertheless, to find the members of a given collection, we will have to inspect each and every entity in the graph.

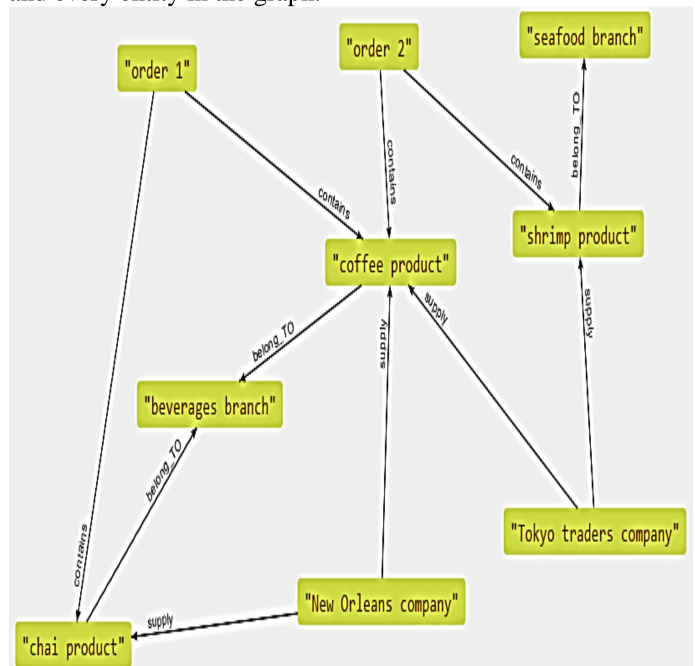

Fig 8. Translation Example of MongoDB to Neo4j.

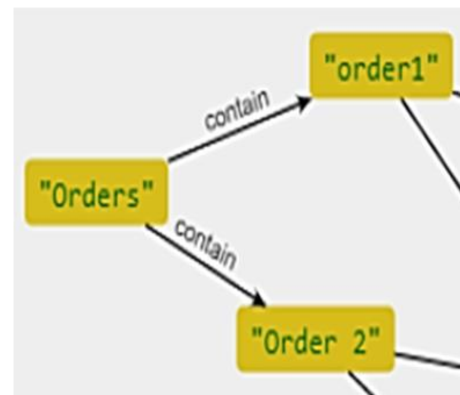

Fig 9. Example of Implementing Collections as Relationships in Neo4j.

B. Redis Example

Next, we discuss a key-value datastore example using Redis. Since Redis does not support relationships efficiently, we selected an example that does not involve any relationships. As shown in Fig. 10, we form sets of categories and users. Each category has a name and a specific set of attributes (that may differ from the others); while each user has a name, age and country. In this specific example, we need to store information about merely the users and categories of items in a given organization, without relating them.

1) Key- value to document datastore: To translate the above example from Redis to MongoDB, we represent the users and categories as collections including documents as shown in Fig. 11. It is clear that MongoDB was able to smoothly represent all the information of Redis, though the instructions of Redis are simpler. To assess the difference between them further, as future work, quantitative analysis will be conducted to compare storage space, for example.

2) Key-value to graph datastore: Finally, we translate this specific example to Neo4j. As shown in Fig. 12, we represent the users and categories as nodes. We also add nodes representing their sets, each pointing to its respective members. As in the case of MongoDB, Redis instructions are simpler, and a qualitative comparison will be conducted for further comparison between Redis and Neo4j.

Forming sets and adding values for categories and users; without relating them

```
> sadd categories "category:1" "category:2" "category:3"
(integer) 3
> hmset category:1 name opera description music year 1573
OK
> hmset category:2 name snowsports description sport Tmembers groups
OK
> hmset category:3 name java description language year 1995
OK
> sadd users "user:1" "user:2" "user:3"
(integer) 3
> hmset user:1 name jack age 23 country france
OK
> hmset user:2 name john age 25 country Uk
OK
> hmset user:3 name julia age 27 country england
OK
```

Fig 10. Redis Example.

C. Neo4j Example

Finally, we discuss the Neo4j example. As shown in Fig. 13, this example represents a social network of users and films/shows, with hidden attributes. The relationships relate users to what they watched, on what they commented, and what they like. They also relate uses to the friends they follow.

Adding three documents for users showing their attributes

```
> db.users.insert([
.             {
..              _id:"U1",
..              user1name:"jack",
..              age:"23",
..              country:"france"
..            },
.             {
..              _id:"U2",
..              user2name:"john",
..              age:"25",
..              country:"UK"
...           },
..            {
...           _id:"U3",
...           user3name:"julia",
...           age:"27",
...           country:"england"
...           }
..        ]);
```

Adding three documents for categories showing their attributes

```
> db.Categories.insert([
.. {
... _id:"C1",
... category1name:"opera",
... description:"music",
... year:"1573"
... },
.. {
... _id:"C2",
... category2name:"snowsports",
... description:"sport",
... Tmembers:"groups"
... },
.. {
... _id:"C3",
... category3name:"java",
... description:"language",
... year:"1995"
... }
.. ]);
```

Fig 11. Translation Example of Redis to MongoDB.

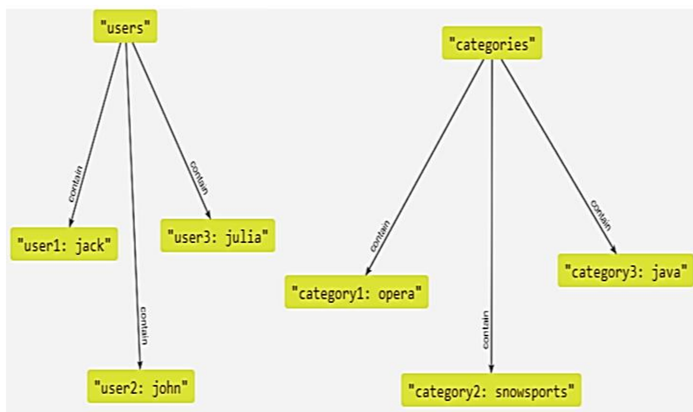

Fig 12. Translation Example of Redis to Neo4j.

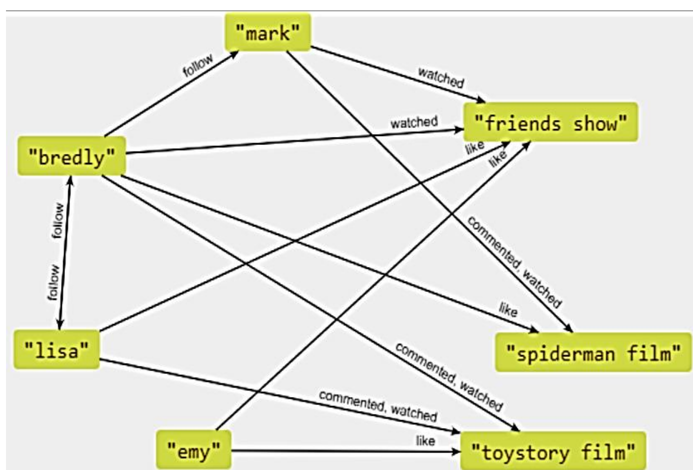

Fig 13. Neo4j Example.

Adding four documents for users showing their attributes

```
>
> db.users.insert([
... {
... _id:"U1",
... name:"bredly",
... age:"25",
... surname:"green"
... },
... {
... _id:"U2",
... name:"lisa",
... age:"15",
... surname:"adams"
... },
... {
... _id:"U3",
... name:"mark",
... age:"22",
... surname:"cooper"
... },
... {
... _id:"U4",
... name:"emy",
... age:"15",
... surname:"mark"
... }
... ]);
```

Adding one document for TV shows showing their attributes

```
> db.tvshow.insert([
... {
... _id:"ts1",
... genre:"comedy",
... seasons:"10",
... name:"friends show"
... }
... ]);
```

Adding two documents for films showing their attributes

```
> db.films.insert([
... {
... _id:"F1",
... genre:"animation",
... name:"toystoryfilm",
... won:"two oscars"
... },
... {
... _id:"F2",
... genre:"animation",
... name:"spidermanfilm",
... won:"three oscars"
... }
... ]);
```

Fig 14. Translation Example of Neo4j to MongoDB.

Collections for relating the various entities

```
> db.comment.insert([
... {
... _id:"U1",
... films:["f1"]
... },
... {
... _id:"U2",
... films:["f1"]
... },
... {
... _id:"U3",
... films:["f2"]
... }
... ]);
```

```
> db.like.insert([
... {_id:"U1",
... films:["f2"],
... },
... {_id:"U2",
... tvshow:["ts1"]
... },
... {
... _id:"U4",
... films:["f1"],
... tvshow:["ts1"]
... }
... ]);
```

```
> db.follow.insert([
... {
... _id:"U1",
... users:["U2","U3"]
... },
... {
... _id:"U2",
... users:["U1"]
... }
... ]);
```

```
> db.watched.insert([
... {_id:"U1",
... films:["f1"],
... tvshow:["ts1"]
... },
... {_id:"U2",
... films:["f1"]
... },
... {_id:"U3",
... films:["f2"],
... tvshow:["ts1"]}
... ]);
>
```

Fig 15. Translation Example of Neo4j to MongoDB (cont.).

1) Graph to document datastore: Fig. 14 and 15 show the translation of this example to MongoDB. As shown in the figures, nodes are converted to documents. Nevertheless, we have to explicitly group some nodes into collections even if this is not intended. Additionally, since MongoDB supports only abstract relationships, we had to create separate collections for each relationship type.

2) Graph to key-value datastore: Finally, we translated Neo4j example to Redis. As shown in Fig. 16, each node is represented using a key (in a corresponding set), and hash data structures are used to store attributes and values as discussed earlier. Similar to the case of MongoDB, we need to group nodes into sets even if not intended. In addition to the fact that representing relationships is overly cumbersome, a major problem is that we can only represent abstract relationships. In other words, we are unable to represent the named relationships. We can add them as attributes, but as discussed earlier, in Redis, keys added as values of *relationship* attributes will not reference their corresponding entities.

Forming sets and adding values for users

```
> sadd users "user:1" "user:2" "user:3" "user:4"
(integer) 4
> hmset user:1 name bredly age 25 surname green
OK
> hmset user:2 name lisa age 15 surname adams
OK
> hmset user:3 name mark age 22 surname cooper
OK
> hmset user:4 name emy age 15 surname mark
OK
```

Forming sets and adding values for films

```
> sadd films "film:1" "film:2"
(integer) 2
> hmset film:1 genre animation name toystoryfilm won "two oscars"
OK
> hmset film:2 genre animation name spidermanfilm won "three oscars"
OK
```

Forming sets and adding values for TV shows

```
> sadd tvshow "tvshow:1"
(integer) 1
> hmset tvshow:1 genre comedy seasons 10 name "friends show"
OK
```

Fig 16. Translation Example of Neo4j to Redis.

D. Discussion

According to the above qualitative comparison, and the illustrated translation from one datastore to another, we can conclude the following findings:

- Graph datastores are designed to be suitable for representing heavily linked data and intensive relationships such as social networks, geographical data, and bioinformatics. We could not effectively represent named relationships in MongoDB and Redis.
- Document datastores are suitable for managing collections with abstract relationships. Representing such relationships is cumbersome in case of Redis. In case of Neo4j, representing collections is not a normal practice and we had to rely on creating new relationships for this issue.
- Key-value datastores are suitable when relationships are not our issue, such as retrieving information about favorite product names of customers, shopping carts, and a user's session. In this case its instructions are much simpler than those of MongoDB and Neo4j.
- We may consider combining more than one datastore type to meet more than one of the above objectives.

V. Conclusion

This paper presented a qualitative comparison of three popular NoSQL datastores of different types (Redis, Neo4j, and MongoDB) using a real use case of each type, translated to the others. The goal was to assess the inherent differences between them in defining data rather than merely comparing their data structures (without showing real use cases) or their performance, as in other research studies in the literature. It was shown that graph data stores are the best choice in case of intensive relationships. Document datastores are better when it comes to collections and abstract relationships. Finally, key-value datastores are the best when relationships are not our issue. As future work, we intend to complement this study with a study of data retrieval queries in each datastore, in addition to their performance. The aim is to assist organizations to find suitable NoSQL datastores that suit their needs.

References

[1] H. Elazhary, "Cloud computing for big data," MAGNT Research Report, vol. 2, no. 4, pp. 135-144, 2014.

[2] A. Angadi, A. Angadi, and K. Gull, "Growth of new databases & analysis of NoSQL datastores," International Journal of Advanced Research in Computer Science and Software Engineering, vol. 3, pp. 1307-1319, 2013.

[3] W. Naheman, and J. Wei, "Review of NoSQL databases and performance testing on HBase," in Proc. 2013 International Conference on Mechatronic Sciences, Electric Engineering and Computer, Shenyang, China, 2013, pp. 2304-2309.

[4] A. Makris, K. Tserpes, D. Anagnostopoulos, and V. Andronikou, "A classification of NoSQL data stores based on key design characteristics," in Proc. 2nd International Conference on Cloud Forward: From Distributed to Complete Computing, Madrid, Spain, 2016, pp. 94-103.

[5] J. Bhogal, and I. Choksi, "Handling big data using NoSQL," in Proc. IEEE Conference on Advanced Information Networking and Applications Workshop, Gwangju, Korea, 2015, pp. 393-398.

[6] P. Srivastava, S. Goyal, and A. Kumar, "Analysis of various nosql database," in Proc. 2015 IEEE International Conference on Green

Computing and Internet of Things (ICGCIoT), Nodia, India, 2015, pp. 539-544.

[7] Redis Labs, “Redis FAQ” Internet: http://redis.io/topics/faq, [Online, Available: 31/1/2019].

[8] Redis, http://redis.io/http://www.neo4j.org/, [Online, Available: 31/1/2019].

[9] G. Deka, “Fine A Survey of Cloud Database Systems,” in IT Professional, vol. 16, no.2, pp. 50-57, 03 January 2013.

[10] K. Kaur, and R. Rani, "Modeling and querying data in nosql databases," in Proc. 2013 International Conference on IEEE on Big Data, Silicon Valley, CA, USA , Oct. 2013, pp. 1-7.

[11] R. Padhy, M. Patra, and S. Satapathy, "RDBMS to NoSQL: Reviewing some next-generation nonrelational database’s," International Journal of Advanced Engineering Sciences and Technologies, Vol. 11, PP.15-30, 2011.

[12] J. Miller, "Editor Graph database applications and concepts with Neo4j," in Proc. 23rd-24th Southern Association for Information Systems Conference, Atlanta, GA, USA, 2013, pp.141-147.

[13] R. Angles, and C. Gutierrez, “Survey of graph database models” ACM Computing Survy, no.1, pp.1-39, Feb. 2008.

[14] http://www.neo4j.org/ [Online, Available: 31/1/2019].

[15] G. Bathla, R. Rani, and H. Aggarwal, “Comparative Study of NoSQL Databases for Big Data Storage,” International Journal of Engineering & Technology, vol. 7, p. 83, 2018.

[16] A. Nayak, A. Poriya, D. Poojary, “Type of NOSQL Databases and its Comparison with Relational Databases,” International Journal of Applied Information Systems, vol. 5, pp. 16-19, 2013.

[17] K. Sahatqija, J. Ajdari, X. Zenuni, B. Raufi, and F. Ismaili, "Comparison between relational and NOSQL databases," in proc. 2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Croatia, 2018, pp. 0216-0221.

[18] A. Corbellini, C. Mateos, A. Zunino, D. Godoy, and S. Schiaffino, “Persisting Big Data: The NoSQL landscape,” Information Systems, Elsevier Science, Vol. 63, pp. 1-23, 2017.

[19] K. Kumar, S. Sundhara, and S. Mohanavalli, "A performance comparison of document oriented NoSQL databases," in proc. 2017 International Conference on Computer, Communication and Signal Processing (ICCCSP), Chennai, India, 2017, pp.15-19.

[20] H. Jing, E. Haihong, G. Le, and J. Du, "Survey on NoSQL database," in proc. 2011 IEEE 6th international conference on Pervasive computing and applications (ICPCA), Port Elizabeth, South Africa , 2011, pp. 363-366.

[21] Z. Parker, S. Poe, and S. Vrbsky, “Comparing NoSQL MongoDB to an SQL DB, ” in Proc. 2013 51st ACM Southeast Conference (ACMSE), ACM, New York, NY, USA, , Apr. 2013, pp.4-5.

[22] Y. Li, and S. Manoharan, "A performance comparison of sql and nosql databases," in proc. 2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM), Victoria, BC, Canada, Aug. 2013, pp. 15-19.

[23] L. Okman, N. Gal-Oz, Y. Gonen, E. Gudes and J. Abramov,“Security Issues in NoSQL Databases,” in proc. 2011 IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications, China, Nov. 2011, pp. 541-547.