Proceedings of the Federated Conference on Computer Science and Information Systems pp. 463–470 DOI: 10.15439/2020F76 ISSN 2300-5963 ACSIS, Vol. 21

How well a multi-model database performs against its single-model variants: Benchmarking OrientDB with Neo4j and MongoDB

Martin Macak[1,2], Matus Stovcik[1,2], Barbora Buhnova[1,2], and Michal Merjavy[2]
[1]*Institute of Computer Science, Masaryk University*
[2]*Faculty of Informatics, Masaryk University*
Brno, Czech Republic
{macak, mstovcik, buhnova, merjavy}@mail.muni.cz

***Abstract*—Digitalization is currently the key factor for progress, with a rising need for storing, collecting, and processing large amounts of data. In this context, NoSQL databases have become a popular storage solution, each specialized on a specific type of data. Next to that, the multi-model approach is designed to combine benefits from different types of databases, supporting several models for data. Despite its versatility, a multi-model database might not always be the best option, due to the risk of worse performance comparing to the single-model variants. It is hence crucial for software engineers to have access to benchmarks comparing the performance of multi-model and single-model variants. Moreover, in the current Big Data era, it is important to have cluster infrastructure considered within the benchmarks.**

In this paper, we aim to examine how the multi-model approach performs compared to its single-model variants. To this end, we compare the OrientDB multi-model database with the Neo4j graph database and the MongoDB document store. We do so in the cluster setup, to enhance state of the art in database benchmarks, which is not yet giving much insight into cluster-operating database performance.

I. Introduction

THE AGE we live in is governed by information. Our society daily produces excessive amounts of raw data. Big Data tools were created as the help to this issue, each being a better fit for a different data set or scenario [1]. Existing surveys describe these tools either generally or are focused on the visualization, processing, or storage options. Focusing on the Big Data storage tools, it is difficult to navigate between them and choose the most effective tool for the given problem [2]. Therefore, for an efficient solution, it is crucial to make the right choice of storage technology, reflecting data variety and other characteristics [3].

Organizations are thus left to either compromise the functionality and use one (i.e., single-model) database strategy or combine more single-model solutions. Implementing and maintaining more solutions means more load on the database engineers and often higher costs [4]. Multi-model databases were proposed as an answer to the need for new, more general, data storing solution [5], offering a possibility to work with one database with multiple types of data.

The ability to effectively store and process data is further emphasized in the context of Big Data, which calls for operating the database solutions in a cluster setup. Even though there are papers comparing databases in a cluster-based scenario [6], the area of comparing multi-model databases with their single-model variants using cluster setup is so far unexplored.

In this paper, we aim to address this gap and contribute to state of the art in the decision about storage technology, specifically choosing among multi-model database and its single-model variants when there is a need to operate them in a cluster setup. In the context of Big Data, NoSQL databases, and specifically the document stores and graph databases, belong among the most popular storage technologies. In our research, we have, therefore, decided to compare these two storage strategies with the multi-model variant (as can be seen in Figure 1). Specifically, we compare the OrientDB multi-model database with the Neo4j graph database and the MongoDB document store. We chose OrientDB, as it is currently one of the most popular and advanced multi-model database [7], [8], whereas MongoDB and Neo4j are suitable representatives of document [9] and graph [10] databases. As for the comparison metric, we use the execution time of queries as it is a standard metric for comparison also in other (non-cluster) benchmarks [11], [12], [13].

To make the benchmarks relevant to the possible real-world Big Data scenarios, we work with a big data set that asks for a storage solution in a cluster of computers. Hence we import this data to distributed versions of the databases, i.e., OrientDB, MongoDB, and Neo4j, exploring how the queries behave in a cluster setup.

The structure of the paper is as follows. Following the related work overview in Section II, we explain our choice of the compared database technologies in Section III. Section IV presents the used cluster configuration, chosen datasets, and designed queries for our experiments. Then, Section V contains the results of the experiments. We provide a summary of these results, including key observations, in Section VI. In Section VII, the threats to validity are discussed, and Section VIII contains the conclusion of this paper.

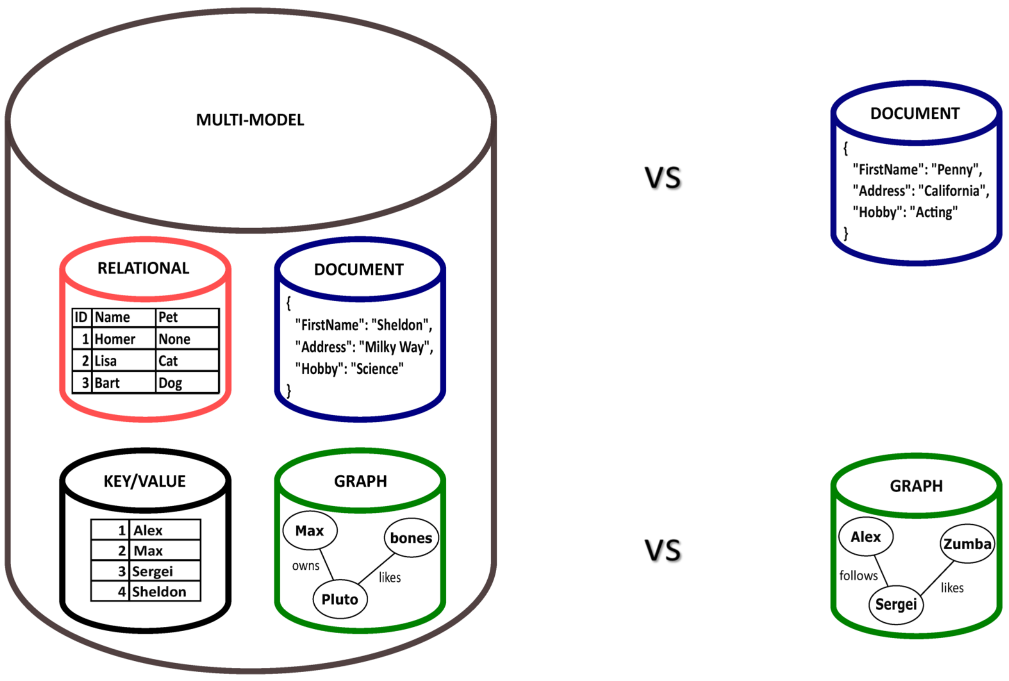


Fig. 1: Compared data structures

II. Related work

Relevant related work considering a multi-model database can be identified in two directions, based on the metric of the comparison. The first direction is based on the efficiency of compared databases (Section II-A), i.e., the comparison of their performance. The second direction is based on the effectiveness of compared databases (Section II-B), i.e., the comparison of their characteristics.

A. Comparisons based on performance

There are many comparisons of multi-model databases with different representatives of its single-model variants. The significant number of these comparisons used OrientDB and compared it with Neo4j and MongoDB [12], [14]. However, none of them used a cluster setup in their comparison.

The work of Ataky et al. [11] presented a paper about performance testing of OrientDB and Neo4j. They focused on the difference between query performance on the index and non-indexed data. They conclude that Neo4j is more suitable for non-indexed data, while MongoDB performed better on data with indexes.

Messaoudi et al. [12] also used Neo4j, MongoDB, and OrientDB in their comparisons. Their research was focused on biomedical data. Their benchmarks covered different depth levels but also some CRUD operations. They found out that OrientDB better handles deeper levels of traversal. A similar study by Messaoudi et al. [14] uses, as in the previous case, Neo4j, MongoDB, and OrientDB. They evaluate databases' ability to manage proteomics data. They observe that MongoDB has better performance for importing protein information in both large and small datasets, but not always in the case when protein fields had a great number of fields. They also found instances in which Neo4j outperformed OrientDB.

Jayathilake et al. [13] have enlarged the scope of focus considering database tools by Cassandra and MemBase. The primary interest was on handling a highly heterogeneous tree. They did a ranking of database tools. Among those tools, MongoDB, Neo4j, and OrientDB stood their place within a higher ranking.

The work performed by Oliveira and del Val Cura [15] focused on benchmarking NoSQL multi-model databases with a polyglot persistence approach. They designed benchmarks with three databases OrientDB, ArrangoDB, and the combination of Neo4j and MongoDB. In contrary to our work, they did not test a multi-model to single-model variant but to a combination of two databases, document and graph database. These tests showed that the combination of Neo4j and MongoDB in some cases performed as good as OrientDB, depending on the size of the dataset, and that ArrangoDB has better performance on more document-based queries with smaller depth levels.

B. Comparisons based on effectiveness

Several studies are extensively comparing the characteristics, features, and benefits of the multi-model databases against their single-model variants.

Fernandes and Bernardino [16] provided characteristics of NoSQL databases, mainly comparing graph database and multi-model database with graph aspect. They discussed and explained the features and benefits of representation of graph data in different databases, including AllegroGraph, ArangoDB, InfiniteGraph, Neo4j, and OrientDB. The recommendation was offered that Neo4j and ArrangoDB are the correct way of representing a graph database in case of a single-model database.

In the paper by Bathla et al. [17], the authors categorized different databases. Among them, we can find Neo4j, MongoDB, and OrientDB. They drew guidelines for choosing the right database tools for users.

The survey by Mazumdar et al. [18] provided a better understanding of choosing the right database solution based non-functional requirements.

III. Compared databases

For our benchmark, we have chosen one representative of a multi-model database, OrientDB, and two representatives of single-model variants: the Neo4j graph database and the MongoDB document store. When selecting the representatives, we considered the popularity of databases based on the DB-Engines website[1]. This ranking of popularity is based on various factors, like the frequency of Google search, relevance in social networks, and a number of job offers. We were looking only for non-commercial open-source databases with the ability to run in a cluster setup.

A. MongoDB

MongoDB[2] version 4.0.12 was chosen as a representative of document store databases. It is classified as a NoSQL database. Data are managed in structure-free storage, with the capability of every collection to be different. Therefore, it stores data in a more flexible way than SQL databases. Its flexibility is one of the reasons for being chosen in our work. MongoDB works with a JSON-like document schema. MongoDB uses collections instead of tables, and it also implements references for faster querying. In Listing 1 is an example query that returns all females in a collection of people.

```
db.people.find( { sex: { "female" } } )
```

Listing 1: MongoDB query example

B. Neo4j

Neo4j[3] version 3.5.12 is an open-source native graph database that is also ACID compliant, highly available, and scalable. Neo4j is one of the leading software solutions in graph databases with active support and development. Its storage was explicitly designed for the management and storage of graphs [19]. Neo4j appears to be the right choice as a representative of graph databases; it has performed well in many comparisons [20], [21], [22]. Neo4j stores data in graphs format using nodes and relationships. Relationships are used for connecting nodes and traversing through data, which is less costly than using a SQL-like approach, i.e., joins. Nodes and relationships are labeled by name, have properties, and are grouped to sets. We can improve the performance of graph traversals by dividing parts of the graphs and using indexes. Data in Neo4j can be accessed through two different query languages.

Neo4j implements cluster setup via core and read-only nodes. This setup does not use the traditional concept of master and slave hierarchy; a leader is voted every period to maintain freshness and availability. Only the leader has the ability to use write operations; followers are used only for read operations. In Listing 2 is a query that returns a number of friends of Jennifer using Neo4j.

```
match (a:Person {name: 'Jennifer'})
      -[:Friend]->
      (b:Person)
return count(b) as count;
```

Listing 2: Neo4j query example

C. OrientDB

OrientDB[4] version 3.0.23 is a multi-model open source NoSQL database management system that supports document, graph, key-value, and object data model. It was released in 2010, is implemented in Java, and is being developed by OrientDB Ltd. It supports distributed architecture with replication and is transactional.

OrientDB uses Paginated Local Storage for storing data [23]. It is disk-based storage that uses a page model to work with data and consists of several components that use disk data trough disk cache. Paginated local storage is a two-level disk cache that works together with a write-ahead log. Files are split into pages, and this allows operations to be atomic at a page level. Two-level disk cache allows OrientDB to cache often accessed pages, separate pages that are not accessed frequently, minimize the amount of disk head seeks during data writes. It also enables the mitigation of pauses that are needed to write data to the disk by flushing all changed or newly added pages to the disk in a background thread. Disk cache consists of two parts read cache and write cache. Read cache is based on the 2Q cache algorithm and write cache is based on WOW cache algorithm. One of the possibilities or manipulating database data is using Java, SQL with extension for graphs and Gremlin. OrientDB supports schema-less, schema-full, or schema-mixed data. OrientDB uses the Hazelcast Open Source project for automatic discovery of nodes, storing cluster configuration, and synchronization between nodes. Distributed architecture can be used in different ways to achieve better performance, scalability, and robustness. OrientDB also provides a web interface that can be used for viewing graphs and data manipulation [16].

OrientDB can use the SQL-like approach for querying. The following query in Listing 3 returns all people with sex equal to 'Female'.

```
SELECT FROM People WHERE sex LIKE 'Female'
```

Listing 3: OrientDB SQL-like query example

[1]https://db-engines.com/en/ranking
[2]https://www.mongodb.com/
[3]https://neo4j.com/
[4]https://www.orientdb.org/

The example of a query that uses an approach of a graph database and returns a number of friends of Jennifer is in Listing 4.

```
SELECT
 both('HasFriend').size() AS FriendsNumber
FROM `Person`
WHERE Name='Jennifer'
```

Listing 4: OrientDB graph query example

IV. Design of experiments

This section presents the used cluster configuration and datasets that were used for testing, together with all queries that were designed. Each query contains a brief description of the results it produces.

Our primary interest was to mirror real-world use cases. Therefore, we designed our queries accordingly in the respective data models. Graph databases stand out with abilities to connect data with relationships and to query data using relationship traversal. Therefore, we wanted to underline these characteristics using queries with a various depth of relationship traversing. Document databases store their data in a structure that offers a suitable environment for filter querying with single or multiple conditions. Hence, it is a natural decision to mirror this into out benchmark queries.

For graph queries, we mostly focused on traversal between nodes because we expect relationships to be more relevant than information stored in individual nodes. In document queries, we concentrated mainly on filtering the data with and without indexes.

A. Setup

Each database is configured in a cluster of three nodes. Nodes are located on the OpenStack cloud platform. All three nodes are running on Ubuntu 18.04.2 LTS. One node runs on dual-core CPU with 2GHz for each core and uses 4GB of RAM. Remaining two nodes run on quad-core CPU witch 2GHz each and use 8GB of ram.

1) Graph dataset: To compare the graph database, we use the 22.3 GB dataset of Twitter followers [24]. Files in this dataset were processed into a format suitable for importing it into the database. We are using CSV format for import as both Neo4j and OrientDB support it. In Neo4j, we use the Neo4j-admin import tool with a file that contains names of all files about to be imported and the name of the database.

For loading data in OrientDB, we utilized the ETL tool that requires a JSON file to define Extractor, Transformer, and Loader. An extractor is responsible for extracting data from a source file and defining other options for extraction, such as separator, columns, and date format. The transformer defines to which class it imports the data and edges that are related to this class. The loader contains the name of the database, type of the database, indexes for classes.

2) Document dataset: As dataset for a document database, we use records of taxi rides in New York City from 2013 [5]. This 18.3 GB dataset is already in the CSV format; hence, it does not need any further alteration. We use this dataset for both MongoDB and OrientDB. To MongoDB, it is imported by the mongoimport tool, which required a path to file, which we want to import and the database name. In OrientDB, we use the same ETL tool as in the case with the graph database.

B. Graph queries

In these queries, we are using relationships for traversal between nodes. Queries are labeled from GQ1 to GQ8. We use a setup described in IV-A, with the exception of queries GQ2 and GQ7, where we used 4GB of RAM on all nodes. The description of the queries is as follows:

- GQ1 counts connected nodes that have less than 1000 followers until depth two,
- GQ2 identical to GQ1 (4GB of RAM on all nodes)
- GQ3 identical to GQ1 depth three,
- GQ4 identical to GQ1 depth four,
- GQ5 identical to GQ1 depth five,
- GQ6 finds the shortest path between two nodes where the desired path is three edges long,
- GQ7 identical to GQ6 (4GB of RAM on all nodes),
- GQ8 finds the shortest path where the path between nodes does not exist.

C. Document queries

In a document database, we focus on queries that deal with filtering data for results that satisfy their requirements, grouping data by some common denominators. Queries are labeled from DQ1 to DQ10. We use a setup described in IV-A, with the exception of queries DQ2 and DQ8, where we used 4GB of RAM on all nodes. The description of the queries is:

- DQ1 counts how many documents fulfill one condition[6],
- DQ2 identical to DQ1 (4GB of RAM on all nodes),
- DQ3 counts how many documents fulfill two conditions,
- DQ4 counts how many documents fulfill three conditions,
- DQ5 counts how many documents fulfill four conditions,
- DQ6 sum of total tip amount on different types of payments,
- DQ7 counts how many documents fulfill one condition on an indexed property where a number of these documents is more than ten million,
- DQ8 counts how many documents fulfill one condition on an indexed property where a number of these documents is more than ten million (4GB of RAM on all nodes),
- DQ9 counts how many documents fulfill one condition on an indexed property where a number of these documents is less than one hundred thousand,
- DQ10 combined index for two properties.

[5]https://chriswhong.com/open-data/foil_nyc_taxi/
[6]conditions are understood as WHERE statements

V. Results of the experiments

This section presents all measurements of queries and interpretation of the results for each comparison between OrientDB and its single-model variant, namely Neo4j and MongoDB. We provide further discussion and interpretation of the results.

The comparison between Neo4j and OrientDB is using a built-in timer within. For comparing MongoDB and OrientDB, we are using Unix command time. Each query was executed five times. The final time is calculated as an average from all runs of the query.

A. Graph queries measurements

TABLE I: Graphs queries

Query	Neo4j	OrientDB
GQ1	**24.586s**	1m16s
GQ2	**30.638s**	4m41s
GQ3	**3m43s**	10m52s
GQ4	30m38s	**15m37s**
GQ5	251m19s	**28m41s**
GQ6	2m7s	**20.436s**
GQ7	3m32s	**1m22s**
GQ8	1m14s	**8.247s**

Neo4j outperformed OrientDB in the first three queries, as shown in Table I. However, in queries GQ4 and GQ5 where the depth was four and five, respectively, OrientDB came out on top with a significant lead in both cases. In our last three queries where the objective was to find the shortest path between selected nodes, OrientDB was also significantly faster than Neo4j.

In queries GQ2 and GQ7, one can see that lowering the size or RAM negatively affects both OrientDB and Neo4j.

B. Document queries measurements

TABLE II: Document queries

Query	MongoDB	OrientDB
DQ1	**9m27s**	38m2s
DQ2	**17m6s**	51m32s
DQ3	**10m15s**	32m13s
DQ4	**12m47s**	32m3s
DQ5	**11m4s**	34m42s
DQ6	**18m42s**	42m57s
DQ7	**2.571s**	50.324s
DQ8	**2.652s**	77.893s
DQ9	**0.673s**	1.078s
DQ10	**0.562s**	1.459s

Queries we tested are focused on filtering a different number of properties and grouping data. In MongoDB filtering, a different number of properties did add some additional time to query executions. As we can see from Table II, in queries DQ2 and DQ3, where we were filtering two and three properties respectively, OrientDB performed better than in DQ1. We suspect that lower times in query DQ3 and DQ4 was due to the lower number of results. In DQ5 execution time for OrientDB increased, we assume it is because the last added property was a string.

In the last four queries, there were indexes created for properties that we used to filter the data. In OrientDB, it was SB-Tree Index; for MongoDB, it was Single Field. Both indexes are based on the B-tree algorithm. Both indexes were chosen based on recommendations found in respective documentations. In query DQ7, we can see that MongoDB outperformed OrientDB, but in the query DQ9, that uses the same index where the number of results was significantly smaller. The difference in these queries in real-time was negligible. In the last query, we used a combined index that uses more than one property for the index, and OrientDB performed a little bit better than MongoDB in this case. Since not in all cases, it is beneficial to create an index on all the data because it requires additional disk space and in most cases, slows inserts it is essential to know how we want to use a database.

In the case of non-indexed data, MongoDB outperforms OrientDB, and in the case of indexed data, OrientDB does not perform very differently to its variant. In query DQ7, where we used only 4GB of RAM instead of 8GB, we also found that having more RAM improved the performance of OrientDB on indexed data significantly, and for MongoDB, it did not have a significant impact in this case. The amount of RAM on non-indexed data improved performance in both cases, as shown in DQ1 and DQ2. This improvement was very similar in all queries for non-indexed data.

VI. Summary of results

This section provides a summary of our results. We provide evaluation and interpretation of the results as well as an explanation for our outcomes. We also include a list of key observations that emerged from our experiments.

A. Results of comparison with a graph database

For graph queries, the objective was to find the shortest path between nodes with varying depths as we wanted to emphasize the power of traversals. Our results in Figure 2 show that it is advisable to use Neo4j up to a depth of three. With higher levels of depths, OrientDB outperforms Neo4j; the difference is most significant in the case of GQ5. However, when the objective in graph data is to traverse different nodes up to the depth of three, Neo4j appears to be a more suitable choice.

B. Results of comparison with a document database

Moving to document queries, we wanted our queries to emphasize the difference between querying upon an indexed and non-indexed field.

Figure 3 shows that the execution times of OrientDB were many times larger than those of MongoDB. Our benchmarks display how good these databases handle non-indexed data, which means that MongoDB provides better management of document data than OrientDB.

Fig. 2: Average times of GQ queries (in seconds)

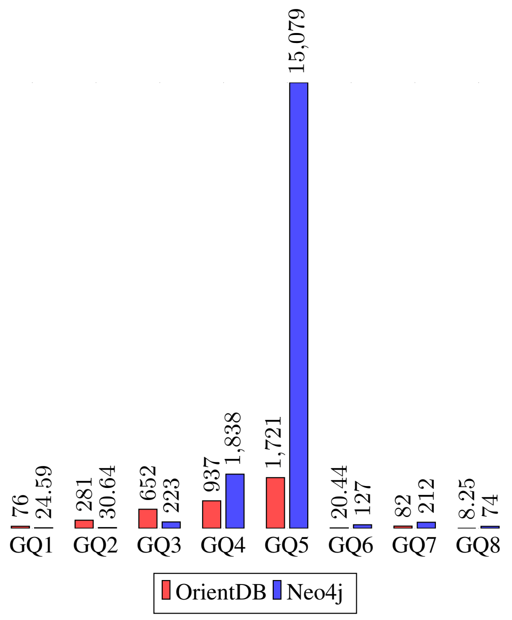
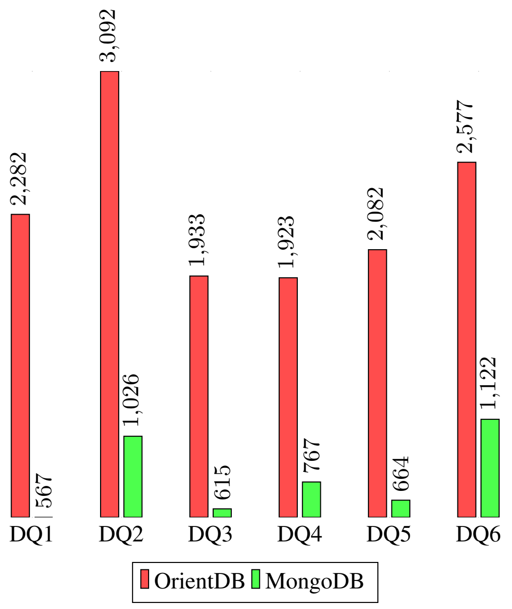
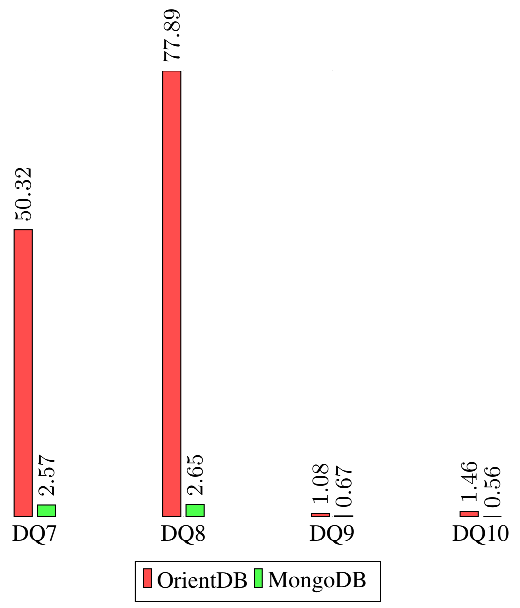


Fig. 3: Average times of DQ queries (in seconds)



Indexing fields resulted in a significant improvement of both MongoDB and OrientDB performances, as shown in Figure 4. In our test case, an increasing amount of RAM had a minor effect on MongoDB, but rather a significant impact on query latencies of OrientDB. For OrientDB, the number of results that our query produces is also very important, as we can see in queries DQ7 and DQ9. In both queries, we used the same data, but the difference between execution times of MongoDB and OrientDB was remarkably smaller. In the case of DQ9 and DQ10 queries, indexing data made the performance of OrientDB comparable to the MongoDB.

Fig. 4: Average times of DQ queries (in seconds)



C. Key observations

In this section, we summarize a list of key observations from our experiments. They also serve as recommendations for future work, which could explore these observations further.

- **KO1**: OrientDB is beneficial when the practitioner is unsure whether more data models will be needed in the future.
- **KO2**: In the case of using only document data, MongoDB is a more suitable choice than OrientDB.
- **KO3**: In the case of queries containing a significant level of depth, OrientDB is a better choice than Neo4j. On the other hand, Neo4j performed better for queries with a smaller level of depth.

From our experiments, it was visible that OrientDB is comparable to (or even better than) Neo4j for graph data and MongoDB for document data. We have to take into consideration that using two different database management systems would have an impact on overhead time. Therefore we assume OrientDB is more beneficial to use when more than one data model is needed. Furthermore, as stated in **KO1**, it is also beneficial to consider OrientDB when the practitioner is unsure whether the support of more data models will be needed in the future.

On the contrary, if the practitioner is aware that only the document model will be used, as mentioned in **KO2**, MongoDB is a more suitable choice because its performance was significantly better in both indexed and non-indexed data.

Moreover, if the practitioner knows that only graph data will be used, we can recommend the proper database based on the level of depth of queries, as stated in **KO3**. When the objective is to traverse different nodes up to the depth of three, Neo4j performed better. However, if the queries use a higher level of depth, OrientDB is a more suitable choice.

VII. Threats to validity

In this study, we needed to narrow our scope to keep focus. This section discusses the limitations we opted for, together with our reasoning behind it.

A. Construct validity threats

We are aware that the fact that in our comparisons, we used only one metric, the query response time, which might be limiting in the decision guidance. Measuring other metrics, like throughput, memory usage, or processor usage, shall be considered in future work to provide more detailed results, although it is out of the scope of this paper.

Another construct validity is the fact that the queries we used in our tests are not complete. There might be many more queries that could be run in a cluster setup to determine the suitability of multi-model database OrientDB over Neo4j and MongoDB. In the future, we would suggest adding queries with complicated aggregate functions. Using queries with even greater depth could result in the possibility of showing another threshold where Neo4j may outperform OrientDB. However, we believe that our tests sufficiently contribute to state of the art.

B. Internal validity threats

We are aware that the configuration of each benchmarked database can affect the results of experiments. We have tried several configurations of each one, and we believe that the chosen configurations are designed for the best efficiency. An exhaustive search of all configurations would be infeasible.

However, it is worth considering that experimenting with multiple configurations of the nodes in a cluster may provide different results. Also, the results may vary when the database cluster contains a different number of nodes.

C. External validity threats

In order to get generalized results, there is a need to perform more tests on different datasets. We have chosen a sufficiently large graph and document dataset. However, this selection might have an impact on the results. Despite this, we believe that our work provides a step towards this goal.

D. Conclusion validity threats

Datasets we used are available online. Therefore they can be downloaded for further investigation and replication. We provided the configuration of our cluster so the tests might be performed again on the same configuration. Using the same approach of testing, we should obtain the same results. However, as the development of chosen databases is fast, it does not make sense to perform these tests on the older versions.

VIII. Conclusion

In this paper, we compared the OrientDB multi-model database with the Neo4j graph database and MongoDB document database. We describe these databases, together with a brief description of used datasets. We compared the performance of these databases on several different queries that were focused on various properties. Our work was aimed at the cluster setup, precisely three nodes.

These queries are split into two different categories. The first category is focused on queries related to graph data, and the second is focused on document data. In each group, we try to aim at queries that are possible for real-world scenarios. For graph data, we focus on traversal between nodes. On the other hand, for document data, we focus on filtering.

Based on the experiments, we provide a set of key observations. We believe that they are proper candidates for future examination in this area.

Acknowledgment

The work was supported by the European Regional Development Fund Project *CERIT Scientific Cloud* (No. CZ.02.1.01/0.0/0.0/16_013/0001802). Access to the CERIT-SC computing and storage facilities provided by the CERIT-SC Center, under the "Projects of Large Research, Development, and Innovations Infrastructures" programme (CERIT Scientific Cloud LM2015085), is greatly appreciated.

References

[1] M. Macak, H. Bangui, B. Buhnova, A. J. Molnár, and C. I. Sidló, “Big data processing tools navigation diagram.” in *IoTBDS*, 2020, pp. 304–312.

[2] F. Gessert, W. Wingerath, S. Friedrich, and N. Ritter, “Nosql database systems: a survey and decision guidance,” *Computer Science-Research and Development*, vol. 32, no. 3-4, pp. 353–365, 2017.

[3] S. Kaisler, F. Armour, J. Espinosa, and W. Money, “Big data: Issues and challenges moving forward,” 01 2013. doi: 10.1109/HICSS.2013.645. ISBN 978-1-4673-5933-7 pp. 995–1004.

[4] P. J. Sadalage and M. Fowler, *NoSQL distilled: a brief guide to the emerging world of polyglot persistence*. Pearson Education, 2013.

[5] E. Raguseo, “Big data technologies: An empirical investigation on their adoption, benefits and risks for companies,” *International Journal of Information Management*, vol. 38, no. 1, pp. 187 – 195, 2018. doi: https://doi.org/10.1016/j.ijinfomgt.2017.07.008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0268401217300063

[6] M. Macak, M. Stovcik, and B. Buhnova, “The suitability of graph databases for big data analysis: A benchmark.” in *IoTBDS*, 2020, pp. 213–220.

[7] A. Messina, P. Storniolo, and A. Urso, “Keep it simple, fast and scalable: A multi-model nosql dbms as an (eb) xml-over-soap service,” in *2016 30th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, 2016, pp. 220–225.

[8] T. P. Hong and P. Do, “Combining apache spark orientdb to find the influence of a scientific paper in a citation network,” in *2018 10th International Conference on Knowledge and Systems Engineering (KSE)*, 2018, pp. 113–117.

[9] W. Schultz, T. Avitabile, and A. Cabral, “Tunable consistency in mongodb,” *Proc. VLDB Endow.*, vol. 12, no. 12, p. 2071–2081, Aug. 2019. doi: 10.14778/3352063.3352125. [Online]. Available: https://doi.org/10.14778/3352063.3352125

[10] T. T. Aung and T. T. S. Nyunt, “Community detection in scientific co-authorship networks using neo4j,” in *2020 IEEE Conference on Computer Applications(ICCA)*, 2020, pp. 1–6.

[11] S. Ataky T. M, L. Ferreira, M. Ribeiro, and M. Prado Santos, “Evaluation of graph databases performance through indexing techniques,” *International Journal of Artificial Intelligence & Applications (IJAIA)*, vol. 06, pp. 87–98, 09 2015. doi: 10.5121/ijaia.2015.6506

[12] C. Messaoudi, M. Amrou, R. Fissoune, and B. Hassan, “A performance study of nosql stores for biomedical data,” 11 2017.

[13] D. Jayathilake, C. Sooriaarachchi, T. Gunawardena, B. Kulasuriya, and T. Dayaratne, “A study into the capabilities of nosql databases in handling a highly heterogeneous tree,” in *2012 IEEE 6th International Conference on Information and Automation for Sustainability*, 2012, pp. 106–111.

[14] C. Messaoudi, R. Fissoune, and B. Hassan, "A performance evaluation of nosql databases to manage proteomics data," *International Journal of Data Mining and Bioinformatics*, vol. 21, pp. 70–89, 09 2018. doi: 10.1504/IJDMB.2018.10016724

[15] F. R. Oliveira and L. del Val Cura, "Performance evaluation of nosql multi-model data stores in polyglot persistence applications," in *Proceedings of the 20th International Database Engineering & Applications Symposium*, ser. IDEAS '16. New York, NY, USA: Association for Computing Machinery, 2016. doi: 10.1145/2938503.2938518. ISBN 9781450341189 p. 230–235. [Online]. Available: https://doi.org/10.1145/2938503.2938518

[16] D. Fernandes and J. Bernardino, "Graph databases comparison: Allegrograph, arangodb, infinitegraph, neo4j, and orientdb," in *Proceedings of the 7th International Conference on Data Science, Technology and Applications - Volume 1: DATA,*, INSTICC. SciTePress, 2018. doi: 10.5220/0006910203730380. ISBN 978-989-758-318-6 pp. 373–380.

[17] G. Bathla, R. Rani, and H. Aggarwal, "Comparative study of nosql databases for big data storage," *International Journal of Engineering & Technology*, vol. 7, no. 2.6, pp. 83–87, 2018. doi: 10.14419/ijet.v7i2.6.10072. [Online]. Available: https://www.sciencepubco.com/index.php/ijet/article/view/10072

[18] S. Mazumdar, D. Seybold, K. Kritikos, and Y. Verginadis, "A survey on data storage and placement methodologies for cloud-big data ecosystem," *Journal of Big Data*, vol. 6, no. 1, p. 15, Feb 2019. doi: 10.1186/s40537-019-0178-3. [Online]. Available: https://doi.org/10.1186/s40537-019-0178-3

[19] F. Holzschuher and R. Peinl, "Performance of graph query languages: comparison of cypher, gremlin and native access in neo4j," in *Proceedings of the Joint EDBT/ICDT 2013 Workshops*. ACM, 2013, pp. 195–204.

[20] D. Dominguez-Sal, P. Urbón-Bayes, A. Giménez-Vañó, S. Gómez-Villamor, N. Martínez-Bazán, and J. L. Larriba-Pey, "Survey of graph database performance on the hpc scalable graph analysis benchmark," in *Web-Age Information Management*, H. T. Shen, J. Pei, M. T. Özsu, L. Zou, J. Lu, T.-W. Ling, G. Yu, Y. Zhuang, and J. Shao, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. ISBN 978-3-642-16720-1 pp. 37–48.

[21] S. Jouili and V. Vansteenberghe, "An empirical comparison of graph databases," in *2013 International Conference on Social Computing*, Sep. 2013. doi: 10.1109/SocialCom.2013.106 pp. 708–715.

[22] M. Ciglan, A. Averbuch, and L. Hluchy, "Benchmarking traversal operations over graph databases," in *2012 IEEE 28th International Conference on Data Engineering Workshops*, April 2012. doi: 10.1109/ICDEW.2012.47 pp. 186–189.

[23] A. S. Mondal, M. Sanyal, S. Chattopadhyay, and K. C. Mondal, "Comparative analysis of structured and un-structured databases," in *Computational Intelligence, Communications, and Business Analytics*, J. K. Mandal, P. Dutta, and S. Mukhopadhyay, Eds. Singapore: Springer Singapore, 2017. ISBN 978-981-10-6430-2 pp. 226–241.

[24] R. A. Rossi and N. K. Ahmed, "The network data repository with interactive graph analytics and visualization," in *AAAI*, 2015. [Online]. Available: http://networkrepository.com