**Neo4j vs. GraphDB**

**Unraveling the Key Differences in Graph Databases**

Aravinda Vijayaram Kumar DATS 6102 Section 13 Group 10

aravinda.avk@gmail.com

Aaron Yang

DATS 6102 Section 13 Group 10

aaron.young@gwu.edu

Luhuan Wang

DATS 6102 Section 13 Group 10  
luhuan.wang@gwu.edu

**ABSTRACT**

This paper explores the significance of innovative data management systems in handling the challenges posed by the increasing volume and complexity of data in data science. It focuses on the rise of graph databases as solutions for storing and retrieving interconnected data structures, particularly in fields like social network analysis, recommendation systems, network security, and bioinformatics.

The study delves into Ontotext GraphDB and Neo4j, two prominent graph database systems. Ontotext GraphDB specializes in semantic graph data management with optimized RDF storage, while Neo4j is known for its versatile query language and performance.

A comprehensive comparative analysis assesses Ontotext GraphDB and Neo4j's effectiveness in managing RDF data, including data import, queries, and performance metrics. Results show that Neo4j remains stable with increasing data size, while GraphDB excels in various query types, especially with smaller datasets. The study provides insights for data scientists seeking the ideal graph database solution, considering specific use cases, data size, and performance requirements.

In summary, this research offers valuable insights into the characteristics, advantages, and limitations of Ontotext GraphDB and Neo4j in RDF data management.

**CCS Concepts**

**RDF**: It stands for Resource Description Framework, is a standard model for data interchange on the web. RDF has features that facilitate data merging even if the underlying schemas differ, and it specifically supports the evolution of schemas over time without requiring all the data consumers to be changed.

**TTL**: It stands for "Turtle Terse RDF Triple Language." It is a widely used syntax for representing RDF (Resource Description Framework) data in a human-readable and compact form.

**NT**: It stands for "N-Triples." N-Triples is a simple and straightforward syntax for representing RDF (Resource Description Framework) data in a plain text format.

1. **Introduction**

As the field of data management in data science evolves, the explosion in the amount, variety, and velocity of information has led to an ongoing pursuit of innovative storage and retrieval systems. Rapid growth in data volume, complexity and interconnectivity highlights the critical role of innovative data management systems. At the same time, Resource Description Framework (RDF) serves as a powerful standard for representing information on the web in a graph-like format, paving the way for semantic web applications. In the field of data management, the emergence of graph databases has revolutionized the storage and retrieval of highly interconnected data structures. Graph databases are different from traditional relational databases. Graph databases focus more on the relationships and connectivity between data. It stores data in the form of graphs. This structure makes graph databases advantageous in processing highly interconnected and complex relationship data, such as social network analysis, recommendation systems, network security, bioinformatics and other fields.

Ontotext GraphDB is a graph database system focusing on semantic graph data management. It has highly optimized RDF storage and reasoning functions.

In the field of graph database systems, Ontotext GraphDB and Neo4j are both backbones, each providing unique methods and capabilities for managing and querying graph-structured data. Known for its RDF storage and reasoning capabilities, GraphDB specializes in semantic graph data management. On the contrary, Neo4j is known for its versatile query language and powerful performance, occupying an important position in various industries and applications. As one of the major representatives in the database field, Neo4j has extensive applications and community support. Its flexible data model and powerful query language make it ideal for working with graph data.

We chose to compare Ontotext GraphDB with Neo4j because of its unique advantages in processing RDF data. GraphDB's focus is on the storage and reasoning of RDF data, which is in sharp contrast to Neo4j's common application in graph data management. Comparing the two can help us gain a deeper understanding of their respective characteristics, advantages, and limitations, and provide clearer guidance for graph database selection in different application scenarios.

Our team embarked on a comprehensive comparative analysis of Ontotext GraphDB and Neo4j to explore their effectiveness in managing RDF data.

Our group will describe the basic characteristics of graph DB and conduct a systematic comparison with Neo4j in terms of versatility and complexity of processing RDF data. A comprehensive assessment of their capabilities, limitations, and their symbiotic potential to harmoniously combine to facilitate advanced data management systems. We also selected suitable database imports, aiming to highlight their unique features, query languages, Performance characteristics and application areas. By illuminating the strengths and limitations of each system, this study aims to provide valuable insights for data scientists when choosing the best graph database solution for a specific use case.

1. **Methodology**

**2.1 Select and Process Data**

A suitable database system significantly affects data management and analysis in various fields. In this study, we use data from "*https://data.world/nlm/medical-subject-headings-mesh*", specifically, we aim to compare Ontotext GraphDB and Neo4j when processing the same dataset. Functionality and performance. This study strives to reveal the unique properties and potential advantages of these two database software solutions in processing data.

The data retrieved from "*https://data.world/nlm/medical-subject-headings-mesh*" was processed and loaded into Ontotext GraphDB and Neo4j and comparative analysis was performed in Python. The comparison focuses on several key aspects, including but not limited to data retrieval efficiency, query performance, scalability, and ease of use. Standardized query and performance metrics are used to evaluate and compare the capabilities and limitations of each database software.

**2.2 Comparison Process**

Our team experimented with loading and querying databases on Neo4j and GraphDB. These queries are then executed in the Python environment and logged in the corresponding two databases.

In tables 1 and 2, we first briefly compare the loading data and basic Queries of graphDB and Neo4j.

*2.2.1 Importing*

|  |  |
| --- | --- |
| **Data base** | **Data Importing Process** |
| **Neo4j** | 1. Create a unique constraint for resources: CREATE CONSTRAINT n10s\_unique\_uri FOR (r:Resource) REQUIRE r.uri IS UNIQUE  2. Initialize graph configuration: CALL n10s.graphconfig.init  3. Import RDF data from a Turtle file: CALL n10s.rdf.import.fetch; |
| **GraphDB** | Use the GraphDB interface's 'Import' button for straightforward RDF data importation. |

*2.2.2 Queries*

|  |  |  |
| --- | --- | --- |
| **Operation** | **Neo4j (Cypher) Query** | **GraphDB (SPARQL) Query** |
| **Checking the Number of Properties (Edges)** | CALL db.propertyKeys() YIELD propertyKey RETURN count(propertyKey) AS NumberOfPropertyKeys | SELECT (COUNT(DISTINCT ?p) AS ?NumberOfProperties) WHERE { ?s ?p ?o . } |
| **Checking the Number of Labels (Node Types)** | MATCH (n) RETURN count(DISTINCT labels(n)) AS NumberOfLabels | SELECT (COUNT(DISTINCT ?type) AS ?NumberOfTypes) WHERE { ?s a ?type . } |
| **Checking the Number of Relationships (Triplets)** | MATCH (a)-[r]->(b) RETURN count(r) AS NumberOfTriplets | SELECT (COUNT(\*) AS ?NumberOfTriplets) WHERE { ?s ?p ?o . } |
| **Checking the Total Amount of Data (Nodes + Relationships)** | MATCH (n) WITH count(n) AS totalNodes MATCH ()-[r]->() RETURN totalNodes, count(r) AS totalRelationships, totalNodes + count(r) AS totalRecords | SELECT (COUNT(DISTINCT ?s) AS ?TotalSubjects) (COUNT(DISTINCT ?p) AS ?TotalPredicates) WHERE { ?s ?p ?o . } |

**2.3 Comparative Execution in Python**

Throughout the evaluation process, a comprehensive comparison of the capabilities of Ontotext GraphDB and Neo4j in processing RDF (Resource Description Framework) data and executing SPARQL queries in a Python environment was conducted.

Initially, our team imported data into GraphDB, loading various TTL (Turtle) files. Different TTL files may contain various data types and structures. Loading different TTL files can help evaluate the system's adaptability to diverse data. This helps in testing and validating real-world data from a variety of sources. By loading various TTL files, the data diversity in real scenarios can also be better simulated, which helps evaluate the system's performance when facing real data sets. The most important thing is to evaluate the performance of the two databases under different loads, such as data loading speed, query execution efficiency, etc. To ensure that the system can effectively process and utilize various types of data.

The process involves measuring the time required to load datasets of different sizes, thereby recording the duration of loading different numbers of triples (an RDF triple consists of a subject, a predicate, and an object). At the same time, each triplet represents a single RDF statement.

The process was then extended to loading the data into Neo4j, focusing on the same TTL files. The main focus is on loading different numbers of triples into a Neo4j database and evaluating the load execution time.

This was followed by an exploration of basic queries in GraphDB, executing SPARQL queries with different limits (100, 1000, 10000, 100000, and 1000000). The goal is to evaluate the time required for data retrieval under these different constraints.

Similarly, Cypher queries were executed in Neo4j, fetching different numbers of records (100 and 1000) to measure the execution time for different query sizes. The purpose of this exploration is to understand how efficient and performance query processing is at different data sizes. By executing SPARQL queries with different constraints in GraphDB and Cypher queries with different numbers of records in Neo4j, you can achieve the following goals: Understanding how the system performs under different data loads can help optimize query performance and data retrieval efficiency. Determine whether the response time of queries is affected as the size of the data grows, so that the system can be adjusted and optimized according to the needs.

Decision support: Detailed understanding of database performance can be used as the basis for decision support. If performance drops significantly as the amount of data increases, you may need to consider a different database management strategy or data partitioning scheme.

User experience improvement: In practical applications, query response time is one of the important factors of user experience. Understanding how queries perform at different data sizes can help improve user experience and ensure system response is fast enough.

Additionally, a visualization component has been integrated to facilitate comparative analysis of load durations for different ternary numbers in GraphDB and Neo4j using scatter plots.

In summary, the research effort focuses on a thorough examination of Ontotext GraphDB and Neo4j, emphasizing the efficiency of their data loading and querying capabilities in the context of RDF data. Subsequently, a detailed step-by-step analysis of the query execution in each environment was performed, with detailed logging of the time spent on each operation.

This detailed approach is designed to collect comprehensive data on the performance and efficacy of each application in query processing.

By comparing the data loading, the code loads different numbers of nodes (10000, 100000, 1000000) into GraphDB and Neo4j, and records the time required for loading. The purpose of this is to observe the loading efficiency of the two databases under different data amounts.

After comparing loading efficiency, execution efficiency is still an important comparison.

We used SPARQL queries. This part uses the SPARQL language to perform various graph traversal and search operations in GraphDB, including depth-first search (DFT), breadth-first search (BFT), best path search (Best path), and pattern matching. (Pattern matching) and Full text search. Record the time required to execute each query operation. This helps compare GraphDB's performance under different query types. Cypher queries can also help us perform various graph traversal and search operations in Neo4j, including depth-first search (DFT), breadth-first search (BFT), shortest path search (shortest path), and simple pattern matching (Simple matching). ) and full-text search. Likewise, record the execution time of each query operation. This helps evaluate Neo4j's performance under different query operations.

Finally we compare data deletion, which is an important and sensitive operation in data science. Although more emphasis is usually placed on data collection, storage, and analysis, data deletion is equally important. We perform node deletion operations on different numbers of nodes (100, 1000, 10000, 100000, 1000000) from GraphDB and Neo4j. These operations also record the time required to delete the node. This helps to understand how efficiently node deletion is performed in the two databases with different data volumes.

After completing these evaluations, a comparative analysis of the performance metrics of Neo4j Desktop and GraphDB Desktop will be performed. This comparative review will prioritize aspects such as import efficiency, query processing speed, and overall operational efficiency, providing insight into each software's comparative strengths and weaknesses in processing RDF data.

1. **Results**

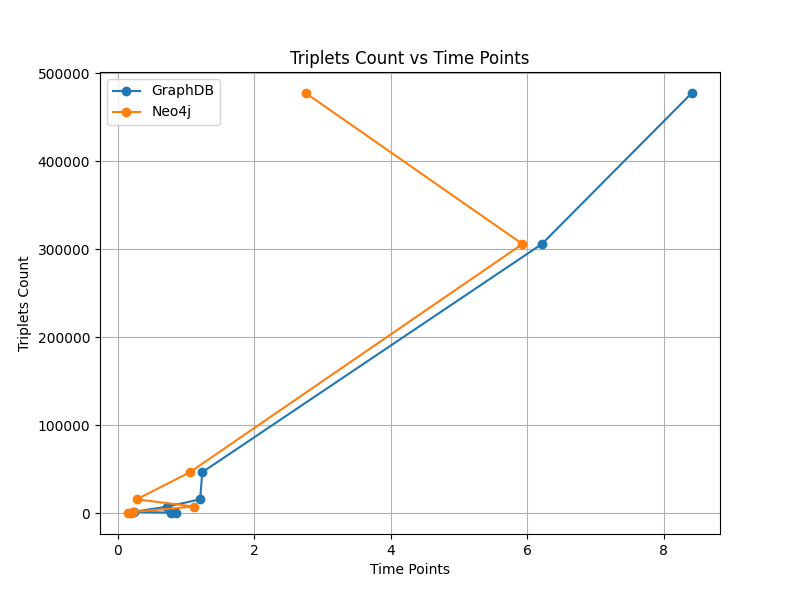
After analyzing and comparing RDF data files with different triple counts (ranging from 157 to 15,439,946 triples), it is important to note that the execution time can be affected by several factors. Well for the results, comparing the execution times between GraphDB and Neo4j, it is clear that there is a difference in load times between the two systems. Interestingly, the results show that as the number of triples increases, the load time increases for both GraphDB and Neo4j. Neo4j tends to outperform GraphDB on datasets, with GraphDB showing competitive performance (0.73 seconds) on 7481 datasets. When the data is 1433 triples, the time taken by the two databases is consistent (0.23 seconds). The choice between the two databases should consider the specific use cases, data size and performance requirements.

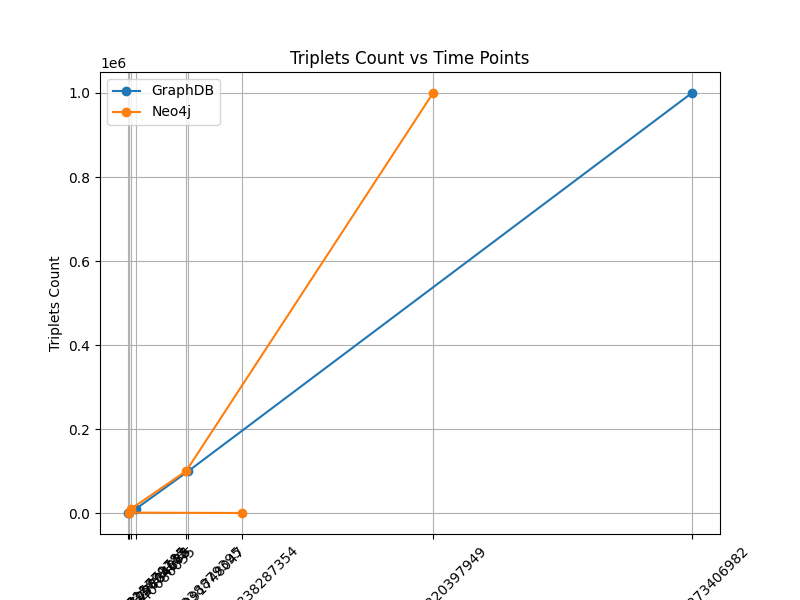
Then we also briefly summarize and compare the execution time of select queries with different constraints in GraphDB and Neo4j. These results highlight the different query performance between GraphDB and Neo4j. GraphDB generally exhibits faster execution times on small to medium result sets, while Neo4j performs better for LIMIT 1000 and LIMIT 10000 queries (0.035 seconds and 0.074 seconds).

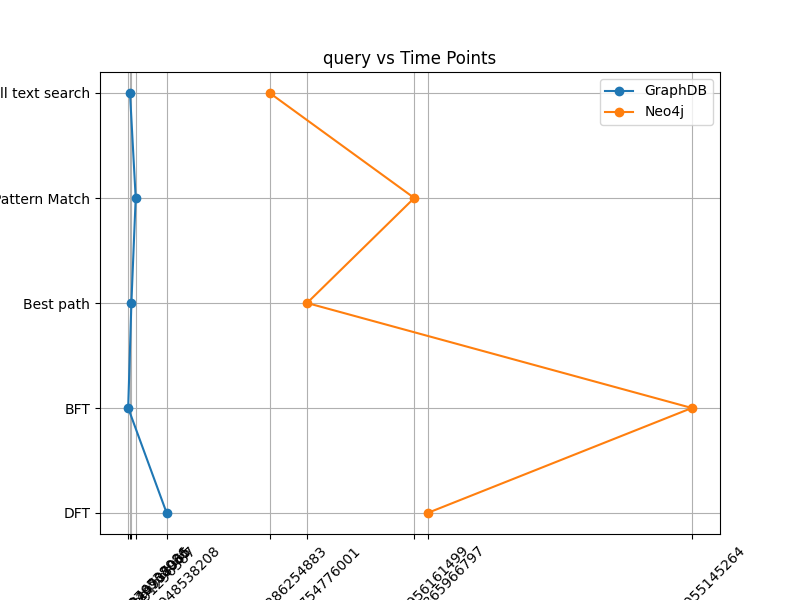
We also performed various graph traversal and search modes in the Neo4j graph database. Each query type has a different purpose, such as exploring paths, finding shortest paths, pattern matching, and performing full-text searches. Comparing the performance of GraphDB and Neo4j based on the provided query, Depth First Search (DFT): GraphDB performs significantly better than Neo4j on DFT and has much lower execution time. In terms of breadth-first search (BFT), GraphDB demonstrates excellent performance in BFT, executing queries much faster compared to Neo4j. Best path search in GraphDB is faster than Neo4j. In pattern matching (CONSTRUCT) queries, GraphDB is better than Neo4j in pattern matching and has shorter execution time. Not surprisingly, GraphDB also shows faster execution speed in full-text search compared to Neo4j. So in this comparison, GraphDB generally shows better performance across different types of queries compared to Neo4j. For queries against RDF data, let's dive into the performance comparison with additional context where the data set is in RDF format. GraphDB demonstrates excellent performance across various query types over RDF data in the provided examples. Its RDF-centric design may help handle RDF triple patterns and graph traversals more efficiently. When working with RDF datasets, GraphDB may be a good choice based on observed performance.

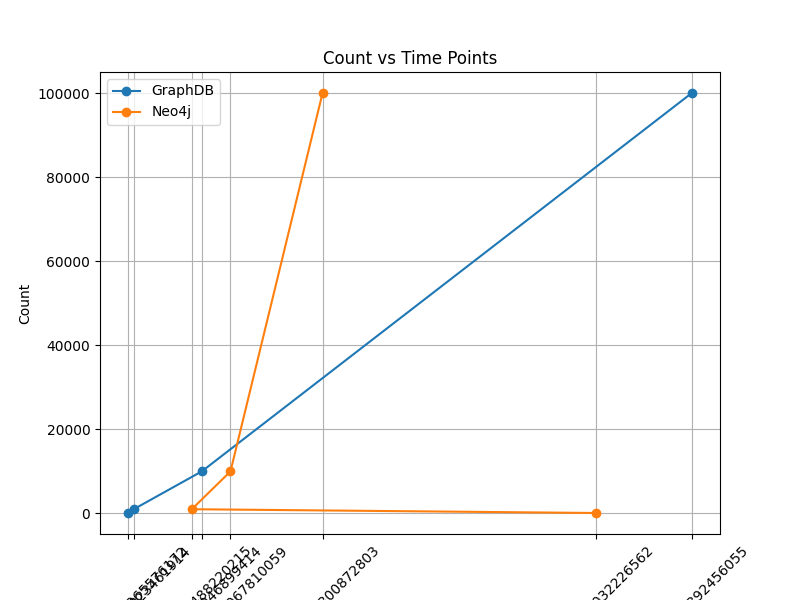
Finally, we compared data deletion. GraphDB showed lower execution time below 100,000 Nodes when deleting triples. Although Neo4j's deletion time is generally fast, the execution time of deleting 100 nodes to 100,000 nodes is relatively long compared to GraphDB. But as the data set increases, the deletion time of GraphDB increases more significantly. As the data increases, Neo4j shows more consistent and relatively lower execution times.

So overall, comparative analysis of data for GraphDB and Neo4j can see the advantages and disadvantages of these two different databases from different aspects. Neo4j is relatively more stable as the data increases, while GraphDB's performance is better than Neo4j in various query types, and it is also faster in processing small databases.





[](http://doi.acm.org/10.1145/161468.16147)



1. **Reference**
2. Bowman, M., Debray, S. K., and Peterson, L. L. 1993. Reasoning about naming systems. *ACM Trans. Program. Lang. Syst.* 15, 5 (Nov. 1993), 795-825. DOI= <http://doi.acm.org/10.1145/161468.16147>.
3. Shad. (2017). Data World. https://data.world/ddw/ontology-v0.
4. Neo4j, Inc. (2023). Neo4j Download. Retrieved from https://neo4j.com/download/.
5. Ontotext. (2023). GraphDB Download. Retrieved from https://graphdb.ontotext.com/.
6. Abdel-Fattah, M.A., Mohamed, W., & Abdelgaber, S. (2022). A Comprehensive Spark-Based Layer for Converting Relational Databases to NoSQL. Big Data Cogn. Comput., 6(3), 71. https://doi.org/10.3390/bdcc6030071
7. Bause, F., Schubert, E., & Kriege, N.M. (2022). EmbAssi: embedding assignment costs for similarity search in large graph databases. Data Mining and Knowledge Discovery, 36, 1728–1755. https://doi.org/10.1007/s10618-022-00850-3
8. Nan, Z., & Bai, X. (2019). The study on data migration from relational database to graph database. Journal of Physics: Conference Series, 1345(2), 022061. https://doi.org/10.1088/1742-6596/1345/2/022061
9. Alm, R., & Imeri, L. (2021). A Performance Comparison between Graph Databases: Degree Project about the Comparison between Neo4j, GraphDB, and OrientDB on Different Operations. Independent Project (Bachelor of Science Degree), Faculty of Natural Sciences, Kristianstad University Sweden,Spring Semester 2021.
10. Patil, N. S., Kiran, P., Kavya, N. P., & Patel, K. M. N. (2018). A Survey on Graph Database Management Techniques for Huge Unstructured Data. International Journal of Electrical and Computer Engineering (IJECE), 8(2), 1140–1149. http://iaescore.com/journals/index.php/IJECE. DOI: 10.11591/ijece.v8i2.pp1140-1149
11. Monteiro, J., Sá, F., & Bernardino, J. (2023). Experimental Evaluation of Graph Databases: JanusGraph, Nebula Graph, Neo4j, and TigerGraph. Applied Sciences, 13, 5770. https://doi.org/10.3390/app13095770.
12. Zhu, J., Chong, H.-Y., Zhao, H., Wu, J., Tan, Y., & Xu, H. (2022). The Application of Graph in BIM/GIS Integration. Buildings, 12, 2162. https://doi.org/10.3390/buildings12122162.
13. Kovács, T., Simon, G., & Mezei, G. (2019). Benchmarking Graph Database Backends: What Works Well with Wikidata? Acta Cybernetica, 24, 43-60. DOI: 10.14232/actacyb.24.1.2019.5.
14. Hribernik, M., Tomažič, S., Umek, A., & Kos, A. (2023). Unified platform for storing, retrieving, and analysing biomechanical applications data using graph database. Journal of Big Data, 10, 68.https://doi.org/10.1186/s40537-023-00747-y.