CSE 511 Project Report: Data Processing with Neo4j, Kafka and Kubernetes

Jainil Trivedi
Arizona State University
Tempe, Arizona
jtrived7@asu.edu

***Abstract*—This project investigates the integration and implementation of graph-based data processing systems using Docker, Neo4j, and Kubernetes to explore scalability and real-time analytics capabilities. Phase 1 focused on establishing a Neo4j environment, developing Docker containers, and executing graph algorithms using the NYC Yellow Cab dataset. Phase 2 expanded to a distributed data pipeline incorporating Kafka and Kubernetes for scalable real-time processing. The outcomes demonstrate practical deployment of graph analytics, showcasing efficiency and flexibility in data management and analysis.**

I. Introduction

Graphs serve as a powerful tool for representing complex, interconnected data and are widely used in various applications, from social networks to urban transportation systems. They provide unique insights by capturing relationships and structures that are often challenging to represent with traditional tabular data. This project focused on utilizing graph-based methods to process and analyze urban mobility data using advanced computational tools.

In the initial phase, we developed a graph-based analysis system using Neo4j, a leading graph database, to model and query data from the NYC Yellow Cab Trip dataset. Each trip was depicted as a relationship linking pickup and drop-off locations, with key attributes like distance, fare, and timestamps incorporated into these relationships. To extract meaningful insights, we implemented fundamental algorithms such as PageRank and Breadth-First Search (BFS) to assess node centrality and traversal efficiency. Docker was employed to containerize this environment, ensuring consistency and portability during development.

Building on this foundation, the second phase expanded the system to tackle scalability and real-time processing challenges. We integrated Kubernetes, a prominent container orchestration platform, and Apache Kafka, a distributed event streaming system, to create a pipeline capable of dynamically processing data streams. This pipeline ingested trip data, processed it through Kafka, and updated the Neo4j graph database in near real-time. By deploying Neo4j within a Kubernetes cluster, we enhanced availability and scalability to meet the demands of growing datasets and continuous data ingestion.

The integration of these technologies demonstrates the viability of a distributed, graph-based data processing system for urban analytics. This approach is especially relevant for smart city initiatives, where the ability to process, analyze, and visualize data dynamically is crucial for effective decision-making. Through this project, we explored the intersection of graph analytics, containerization, and distributed systems, highlighting their combined potential for addressing complex data challenges.

II. Methodology

The project was executed in two phases, each focusing on different aspects of graph-based data processing and distributed systems. The first phase aimed to establish a foundational graph processing environment using Docker and Neo4j. A Dockerfile was created to set up a containerized Neo4j instance, ensuring consistency and portability across various environments. The NYC Yellow Cab Trip dataset from March 2022 was chosen as the main data source to represent urban mobility data. The graph schema included nodes for locations (labeled "Location") and relationships for trips (labeled "TRIP"), with each trip relationship enriched with properties like distance, fare, and timestamps for pickup and drop-off events. A Python script named

data_loader.py was developed to load this data into Neo4j using Cypher queries to construct the graph structure. Two graph algorithms, PageRank and Breadth-First Search (BFS), were implemented using GDS Library: PageRank assessed the relative importance of locations based on trip frequencies, while BFS enabled traversal from specific pickup locations to target drop-off locations, simulating use cases like proximity-based searches. The system underwent rigorous testing using Neo4j's query tools to ensure data integrity and algorithm accuracy.

$$PR(A) = (1-d) + d(\frac{PR(T_1)}{C(T_1)} + ... + \frac{PR(T_n)}{C(T_n)})$$

Figure 1: Page Rank Algorithm

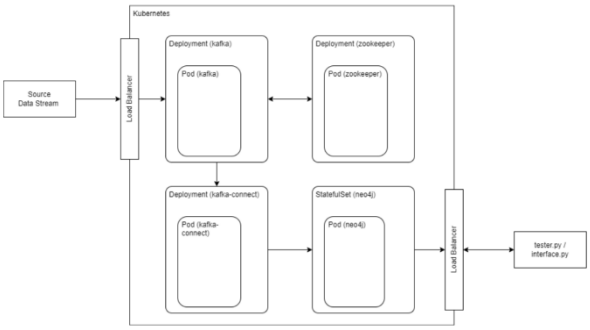

Figure 2: Project Architecture/Pipeline

In the second phase, the project expanded by introducing a distributed and scalable data processing pipeline using Kubernetes and Apache Kafka. Minikube, a local Kubernetes environment, managed the deployment of various system components, including Kafka and Neo4j. Kafka facilitated real-time data ingestion with topics configured for incoming trip data streams. A Kafka-Neo4j connector was deployed to transform and stream data into the Neo4j database in near real-time. Neo4j was integrated into the Kubernetes cluster as a standalone instance equipped with the Graph Data Science (GDS) plugin for advanced analytics. YAML configuration files were created for each deployment to ensure seamless communication between components, detailing resource allocation, service definitions, and connectivity settings. The graph algorithms from Phase 1 were reapplied in this distributed setup to validate its performance and scalability. Data flowed from a Python-based producer into Kafka, was processed and streamed into Neo4j, and analyzed using Cypher queries and GDS procedures. The pipeline was tested end-to-end, demonstrating its capability to handle dynamic data updates efficiently while performing graph analytics.

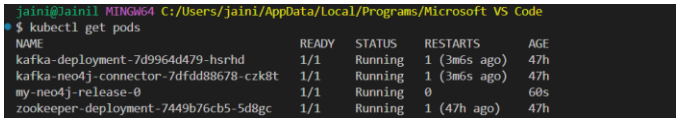

Figure 4: Pods

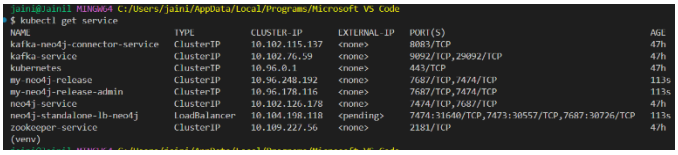

Figure 3: Services

By integrating Docker, Neo4j, Kubernetes, and Kafka, the project demonstrated a robust approach for scalable and real-time graph-based data processing, effectively addressing challenges in system setup, data modeling, and distributed architecture implementation.

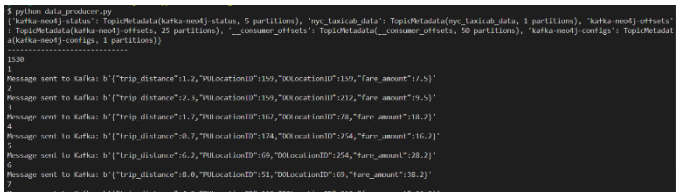

Figure 5: Data Stream from Kafka to Neo4j

III. RESULTS

In Phase 1, the standalone Neo4j instance effectively processed the NYC Yellow Cab Trip dataset. Nodes and relationships were accurately created based on the schema, and the PageRank algorithm successfully identified locations with the highest trip inflows and connectivity. The BFS algorithm efficiently traversed the graph, confirming its utility for proximity-based queries. These outcomes demonstrated the effectiveness of graph databases in analyzing urban mobility data.

Phase 2 resulted in a fully operational distributed pipeline. Kafka managed real-time trip data ingestion, while the Kafka-Neo4j connector facilitated seamless data integration into the Neo4j database. The graph algorithms were successfully reapplied, yielding similar results in the distributed setup as in the standalone environment. Furthermore, the system proved scalable, handling increased data

volumes without significant performance loss. This validated the feasibility of using a distributed architecture for real-time graph processing.

IV. Discussion

The project illustrated the practical integration of graph analytics with distributed systems. Phase 1's standalone setup highlighted Neo4j's ease of use for modeling and querying interconnected data. However, the need for scalability and real-time processing led to implementing a distributed pipeline in Phase 2. Kafka's ability to manage streaming data and Kubernetes' orchestration capabilities provided a solid foundation for scaling the system. The successful implementation of PageRank and BFS in both phases demonstrated the system's adaptability and reliability.

Challenges included configuring Kubernetes resources and managing Kafka's dependencies, which were addressed through extensive testing and documentation. The project also identified areas for further exploration, such as integrating additional graph algorithms, optimizing resource allocation in Kubernetes, and exploring Neo4j's clustering capabilities for enhanced scalability. Overall, the project highlighted the potential of combining graph databases with distributed systems to tackle complex data processing challenges in real-time scenarios.

V. Conclusion

This project showcased a comprehensive approach to graph-based data processing, transitioning from a standalone environment to a distributed system. By leveraging Docker, Neo4j, Kubernetes, and Kafka, I demonstrated the practical application of graph analytics to real-world datasets. The two-phase methodology emphasized the importance of scalability and real-time capabilities in modern data processing systems. The successful implementation of PageRank and BFS confirmed the system's ability to provide actionable insights, paving the way for future advancements in distributed graph processing.

References

1. https://neo4j.com/docs/operations-manual/current/kubernetes/
2. https://neo4j.com/docs/operations-manual/current/kubernetes/quickstart-standalone/
3. https://neo4j.com/docs/kafka/current/
4. https://neo4j.com/product/auradb/
5. https://www.nyc.gov/assets/tlc/downloads/pdf/data_dictionary_trip_records_yellow.pdf
6. https://www.digitalocean.com/community/tutorials/how-to-install-and-configure-neo4j-on-ubuntu-22-04#step-3-optional-configuring-neo4j-for-remote-access
7. http://infolab.stanford.edu/~backrub/google.html