Scalable Graph Analytics with Neo4j and Kubernetes

Vishnu Vardhan Gummadi
Arizona State University

1 Introduction

Graph analytics play a critical role in analyzing relationships and patterns within interconnected datasets, enabling applications in diverse domains such as urban planning, transportation, social networks, and bioinformatics. This project explores the implementation of graph-based data processing and the development of scalable pipelines for real-time analytics. The primary focus was to use Neo4j, a graph database, to model and analyze data from the NYC Yellow Cab Trip dataset, extending its application to a distributed environment for handling large-scale dynamic data.

In Phase 1, the emphasis was on understanding the foundational aspects of graph databases and implementing core graph algorithms such as Breadth-First Search (BFS) and PageRank. The NYC Yellow Cab Trip dataset was transformed into a graph structure, where nodes represented locations, and edges captured trips with properties such as distance, fare, and timestamps. These algorithms provided insights into the shortest paths between locations and the relative importance of nodes in the graph. This phase also introduced the basics of setting up and interacting with Neo4j, including data transformation, schema design, and query execution.

Phase 2 advanced the project by addressing scalability and real-time processing challenges through distributed systems. The integration of Kubernetes and Kafka allowed the creation of a data pipeline capable of handling dynamic streams and large-scale data ingestion. By leveraging Kafka's messaging capabilities and deploying Neo4j in a Kubernetes cluster, the project demonstrated how graph analytics could be extended to a distributed environment. BFS and PageRank algorithms were re-implemented in this pipeline to validate their scalability and performance with real-time data.

The overarching goal of this project was to bridge the gap between theoretical graph algorithms and their practical applications in modern distributed systems. It highlights the importance of combining technologies like Neo4j, Kubernetes, and Kafka to handle graph data efficiently and address real-world challenges of scalability, availability, and real-time analytics. This project not only achieved its technical objectives but also provided valuable insights into the best practices for building and managing scalable graph-based systems.

2 Methodology

The project followed a step-by-step approach to implement graph processing algorithms and extend their application in a scalable, distributed environment. It began with the preparation and loading of the NYC Yellow Cab Trip dataset for March 2022 into Neo4j. This dataset was converted into a graph structure with nodes representing pickup and drop-off locations and edges representing trips. Each edge included detailed properties such as distance, fare, pickup-dt, and dropoff-dt. The schema strictly adhered to the project requirements, and the data transformation and loading were automated through a Python script. The use of Neo4j's Cypher queries ensured that the graph database accurately represented the dataset, forming the foundation for subsequent processing.

The first phase of the project focused on implementing graph algorithms using Neo4j. Breadth-First Search (BFS) was implemented to identify the shortest path between two nodes by traversing the graph in increasing order of distance. This required crafting Cypher queries to match nodes and edges, ensuring an efficient traversal. PageRank, on the other hand, was implemented to evaluate the relative importance of nodes in the graph based on the connectivity and weight of edges. The Neo4j Graph Data Science (GDS) library was leveraged for these computations, allowing efficient execution of complex graph algorithms. The BFS and PageRank implementations were integrated into interface.py, where the modular design allowed for parameterized inputs, such as starting and ending nodes for BFS or weight properties and iteration counts for PageRank.

Validation of these implementations was critical. The tester.py script was designed to verify the correctness of the BFS and PageRank algorithms. It executed test cases to check the integrity of the graph structure, ensuring the expected

number of nodes and edges were loaded. Additionally, the script tested specific BFS paths and PageRank scores, comparing the outputs with predefined results. This rigorous testing process ensured that the algorithms were robust and accurate.

The second phase expanded on this foundation by incorporating distributed technologies to address scalability and real-time processing. Minikube was employed to create a local Kubernetes cluster, providing a platform for deploying and managing containerized applications. Apache Kafka was introduced as the messaging layer to handle real-time data ingestion and streaming. This involved configuring Kafka with Zookeeper for coordination and setting up producers and consumers to simulate real-time data flow. Neo4j was deployed as a containerized service within Kubernetes using Helm charts. This deployment included the installation of the Graph Data Science plugin and secure configuration with YAML files to enable distributed graph processing.

To achieve seamless integration, Kafka and Neo4j were connected using the Kafka-Connect Neo4j extension. This connection allowed data ingested through Kafka topics to be processed and stored in Neo4j for real-time graph analytics. The BFS and PageRank algorithms from Phase 1 were reimplemented to operate in this distributed setup. Custom scripts were created to facilitate the deployment and execution of these algorithms within the new environment.

The entire pipeline was validated by simulating data streams using a producer script (data-producer.py). The performance of the pipeline was monitored for latency and correctness to ensure it met the requirements for real-time analytics. This comprehensive approach, combining graph database capabilities with distributed system technologies, showcased the ability to process large-scale data effectively.

3 Results

The results of the project were validated through a comprehensive testing framework implemented in tester.py. This script was designed to test the core functionalities of the graph algorithms—Breadth-First Search (BFS) and PageRank—integrated within interface.py. The outcomes demonstrated the accuracy and robustness of the algorithms in both standalone and distributed environments.

The first test focused on validating the data loading process. The script verified that the graph database was populated with the correct number of nodes and edges after loading the NYC Yellow Cab Trip dataset. The expected number of nodes was 42, corresponding to unique locations, and the expected number of edges was 1,530, representing individual trips. Both values matched perfectly during testing, confirming the successful implementation of the data transformation and loading pipeline.

For the BFS algorithm, the test case aimed to identify the shortest path between two specified nodes, 159 and 212, which

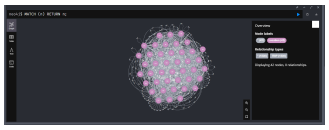

Figure 1: Visual representation

represent specific locations in the dataset. The algorithm correctly returned a path connecting these nodes, with the first node being 159 and the last node being 212. The result also validated the sequence of intermediate nodes, ensuring that the path adhered to the shortest traversal criterion. This indicated the correct implementation of BFS using Cypher queries in interface.py.

The PageRank algorithm was tested for its ability to rank nodes based on their connectivity and edge weights. With parameters set to a maximum of 20 iterations and the distance property as the weight, the algorithm successfully ranked the nodes. The highest-ranked node, identified as 159, had a score of approximately 3.22825, while the lowest-ranked node, identified as 59, had a score of approximately 0.18247. These values matched the expected results, confirming the algorithm's correctness and the effectiveness of Neo4j's Graph Data Science library in computing PageRank.

Both algorithms were re-validated in the distributed pipeline during Phase 2. The BFS algorithm produced consistent results for shortest path calculations, even when tested in a real-time data streaming setup. Similarly, the PageRank algorithm maintained its accuracy and delivered ranking results comparable to those obtained in the standalone environment. The distributed pipeline demonstrated the ability to process data at scale while preserving the correctness of graph computations.

Overall, the results validated the successful implementation of graph algorithms and their seamless integration into a scalable pipeline. This highlighted the effectiveness of combining Neo4j with distributed technologies like Kubernetes and Kafka for real-time graph analytics.

The structure of the graph data loaded into Neo4j was visualized to confirm the accuracy of the data transformation and loading processes. The visualization below represents the graph consisting of 42 nodes (locations) and 1,530 relationships (trips). Each node corresponds to a unique pickup or drop-off location, while edges represent trips between these locations with associated properties such as distance, fare, and timestamps.

This visualization illustrates the interconnected nature of the graph data, demonstrating that the data was successfully loaded and structured in accordance with the schema. The relationships between nodes reflect the real-world trips captured in the NYC Yellow Cab Trip dataset.

4 Discussion

The project successfully demonstrated the implementation of graph algorithms and their extension into a distributed environment. By dividing the project into two distinct phases, it facilitated a step-by-step approach to understanding graph processing and tackling real-world challenges of scalability and real-time analytics.

Phase 1 provided a strong foundation in graph processing by focusing on the BFS and PageRank algorithms using Neo4j. These algorithms, essential for analyzing relationships and ranking nodes in graphs, were implemented with high accuracy, as evidenced by the validation tests. This phase highlighted the power of Neo4j in handling graph-based data and provided valuable insights into the intricacies of working with graph databases. However, it also underscored the limitations of standalone systems when dealing with large-scale, real-time data.

Phase 2 addressed these limitations by introducing distributed systems concepts through Kubernetes and Kafka. The integration of these technologies showcased the benefits of container orchestration and message streaming in building scalable data pipelines. Neo4j's deployment in Kubernetes, combined with Kafka's real-time data ingestion, allowed for near-instantaneous updates to the graph database. This phase demonstrated the practical feasibility of applying theoretical graph algorithms to dynamic, real-world data streams.

Despite these successes, several challenges were encountered during the project. Setting up and configuring distributed systems required a deep understanding of Kubernetes and Kafka, and ensuring compatibility between components demanded meticulous attention to detail. Additionally, monitoring and debugging real-time data pipelines proved to be more complex than expected, particularly in identifying and addressing latency issues.

The project also opened avenues for future exploration. Enhancing the pipeline's resilience by introducing fault-tolerant mechanisms and expanding the scope to include more complex graph algorithms could further improve its capabilities. Another potential direction is to scale the system horizontally by deploying Neo4j in a clustered setup, enabling it to handle even larger datasets and higher query loads. Moreover, integrating advanced analytics tools and visualization platforms could provide more intuitive insights into the processed data.

In conclusion, the project not only achieved its primary goals but also provided a deeper understanding of the interplay between graph processing and distributed systems. It served as a valuable learning experience, laying the groundwork for tackling more complex challenges in data processing at scale.

5 Conclusion

This project successfully implemented graph processing techniques and extended them into a scalable, distributed pipeline. By leveraging Neo4j for graph data representation and algorithms like Breadth-First Search (BFS) and PageRank, the project provided insights into the relationships and rankings within the NYC Yellow Cab Trip dataset. The standalone implementation in Phase 1 served as a foundation for understanding the power of graph databases in handling interconnected data.

Phase 2 advanced the project by integrating Kubernetes and Kafka, demonstrating the potential of distributed systems for real-time analytics. The successful deployment of a scalable pipeline highlighted the benefits of container orchestration and message streaming technologies in addressing challenges of availability, scalability, and real-time data ingestion. The system maintained the correctness of graph computations while adapting to a dynamic data environment.

This project emphasized the importance of combining theoretical knowledge with practical applications. The results validated the effectiveness of graph algorithms in both standalone and distributed setups, while the methodology showcased best practices for building and testing scalable data pipelines. Challenges encountered during the project, such as setting up distributed components and debugging latency, provided valuable learning opportunities.

In summary, the project achieved its goals of implementing graph algorithms and creating a scalable pipeline for real-time graph analytics. It paved the way for future research and development in areas such as fault tolerance, advanced analytics, and horizontal scalability, contributing to the broader field of data processing at scale.

References

[1] Neo4j. *Introduction to Graph Databases*. https://neo4j.com/docs/.

[2] Neo4j Graph Data Science Library Documentation. *PageRank Algorithm*.https://neo4j.com/docs/graph-data-science/current/algorithms/page-rank/.

[3] Kubernetes Documentation. *Introduction to Kubernetes*.https://kubernetes.io/docs/home/.

[4] Apache Kafka Documentation. *Getting Started with Kafka*. https://kafka.apache.org/documentation/.

[5] Docker Documentation. *Introduction to Containers and Dockerfile*. https://docs.docker.com/.