Project Report
Course: CSE511 Data Processing at Scale

Aditya Umesh Upadhyay, ASUID - 1237523263

Abstract

This project presents a complete end-to-end data processing and analytics pipeline constructed across two phases of the CSE 511 Data Processing at Scale course. Phase 1 focused on batch-oriented data ingestion and graph construction using Neo4j inside a Dockerized environment. The NYC Yellow Taxi dataset (March 2022) was filtered, cleaned, transformed into a graph schema, and loaded into Neo4j as Location nodes and TRIP relationships. The GDS library was used to compute PageRank for identifying influential locations and Breadth-First Search (BFS) for determining shortest hop-based paths.Phase 2 expanded the pipeline into a real-time, scalable streaming architecture using Kubernetes, Kafka, Zookeeper, Kafka Connect, and Neo4j deployed via Helm. Streaming data produced by a Kafka producer was ingested through a Neo4j sink connector, dynamically updating the graph with new records. The same GDS algorithms like PageRank and BFS were executed again on this live-updating graph to validate data flow, consistency, and analytical correctness. Together they form a unified system capable of handling batch and streaming workloads, supporting scalable ingestion, near real-time updates, and continuous graph analytics.

Introduction

This project implements an end-to-end graph-based data processing pipeline, integrating multiple data-engineering and distributed-systems technologies to process, transform, and analyze real-world NYC Taxi trip data at scale. The entire workflow comprising of Phase 1 and Phase 2 forms one unified system that moves from batch data ingestion to streaming analytics. The goals of Phase 1 were:

- Set up a Neo4j environment in Docker
- Load the March 2022 NYC Taxi dataset
- Transform the dataset into a graph structure (Location nodes, TRIP edges)
- Install and configure the Neo4j GDS plugin
- Implement two GDS algorithms externally via interface.py
    - PageRank
    - Breadth-First Search (BFS)

This produced a complete graph database representing Bronx taxi movements, enabling analytical algorithms such as influential-node detection and shortest-path traversal.

Phase 2 extends this system into a scalable, highly available streaming pipeline. Goals included:

- Deploying Kafka and Zookeeper in Kubernetes (minikube)
- Deploying Neo4j in Kubernetes with Helm
- Creating a Kafka → Neo4j streaming connector
- Using a producer to stream document events into Kafka

Together, both phases create a modern data pipeline that combines batch ingestion, graph modelling, real-time streaming, and analytics in a unified architecture.

Methodology

Phase 1 Methodology: Docker-Based Graph Construction

1. Dataset Preparation & Filtering

Using *data_loader.py*, the workflow read the March 2022 NYC Yellow Taxi Parquet dataset using PyArrow and Pandas. Only essential fields were extracted:

- pickup and drop-off timestamps
- pickup and dropoff location IDs
- trip distance
- fare

To reduce dimensionality and improve graph performance, the dataset was filtered to Bronx-only trips, as recommended in the official assignment.

2. Transformation Into Neo4j Import Format

After filtering, the data was written to Neo4j's /var/lib/neo4j/import/ directory in CSV format, enabling bulk import.

3. Graph Schema Construction

Following the schema defined in the Phase 1 instructions

- Node Type: Location
    - Property: name (integer location ID)
- Relationship Type: TRIP

    - Properties: distance, fare, pickup_dt, dropoff_dt

4. The Dockerfile installed:

- Neo4j Community Edition
- OpenJDK 21
- Required Python packages (neo4j, pandas, pyarrow)
- GDS plugin v2.21.0

Neo4j was configured to allow remote access, whitelist APOC and GDS procedures and set default password to "*graphprocessing*". This produced a working Neo4j container ready for dataset ingestion and querying.

5. Graph Algorithms Implementation

The *interface.py* script executed PageRank and BFS externally via the Bolt driver on port 7687.PageRank used a weighted graph projection (distance) and returned the max-rank and min-rank locations. BFS used an unweighted projection and returned the shortest hop-based path between start and end nodes.

Phase 2 Methodology: Kafka–Kubernetes Streaming Pipeline

Phase 2 extends Phase 1 by introducing distributed data ingestion and real-time updates.

1. Kubernetes Environment Setup (Step 1: Order in Chaos)

Using minikube:

- Zookeeper deployed using *zookeeper-setup.yaml*
- Kafka deployed using *kafka-setup.yaml*
- Required configurations applied:
    - KAFKA_BROKER_ID=1
    - Zookeeper connection string
    - Listener security protocols
    - Advertised listeners (internal + localhost)

2. Deploying Neo4j in Kubernetes (Step 2)

Using Helm charts with *neo4j-values.yaml*, the cluster deployed a standalone Neo4j instance with

- GDS plugin installed
- Password set to *processingpipeline*
- Service exposed through neo4j-service:7474

This moved Neo4j from a local Docker container into an orchestrated distributed environment.

3. Kafka → Neo4j Streaming Pipeline (Step 3)

Using Kafka Connect with the *kafka-neo4j-connector.yaml*:

- Events produced into a Kafka topic
- Kafka Connect transformed them into Cypher-compatible payloads
- Neo4j ingested the records in real time

This allowed document streams to become graph nodes and relationships.

4. Running Analytics on the Live Graph (Step 4)

The same algorithms from Phase 1 were reused. Page-Rank to measure node importance in the evolving graph. BFS to compute shortest paths in near real-time.This validated end-to-end data flow:

Producer → Kafka → Connector → Neo4j → GDS algorithms

Result

Phase 1 Findings

After ingestion:

- 42 Bronx Location nodes created as expected.
- 1530 TRIP edges loaded matching expected values.
- Graph schema validated using:
    - CALL db.schema.visualization()
- PageRank results successfully returned correct max and min ranked nodes across varying iteration counts
- BFS returned correct paths for all grading test cases.

These results show that the graph was correctly constructed and traversable.

Phase 2 Findings

After Kubernetes deployment:

- Kafka, Zookeeper, and Neo4j responded correctly inside minikube
- Events published by the producer streamed into Kafka
- The Neo4j connector successfully inserted real-time updates
- Running PageRank on live-updated data produced dynamic rank changes based on stream order
- BFS remained valid and stable across a continuously updating graph
- The full architecture mirrored the data-flow diagram on p.7–8 of the Phase 2 instructions

Overall, the transition from batch to streaming analytics succeeded.

Discussion

The combined project demonstrates the power of building a modern data-processing ecosystem that spans:

- Batch computation (Phase 1)

- Real-time streaming analytics (Phase 2)
- Graph-based storage and processing
- Kubernetes orchestration
- Distributed messaging (Kafka)

Key Observations

1. Graph models outperform relational schemas for mobility data due to natural representation of routes and connectivity.
2. Filtering to the Bronx region improved both scalability and algorithmic tractability.
3. The GDS library provided high-performance algorithms, which are significantly faster than manual implementations.

Limitations

- No temporal segmentation of trips.
- Streaming pipeline processes document-style events and not true taxi trip streams.

Conclusions

This project brought together two phases to build a practical and scalable graph-based data pipeline. Phase 1 handled the heavy lifting including cleaning the NYC taxi data and turning it into a meaningful Neo4j graph. Phase 2 built on that work by adding a real-time streaming setup with Kubernetes and Kafka, allowing new data to flow in and be analyzed instantly.Together, the phases showed how raw mobility data can be transformed, updated continuously, and analyzed using powerful graph algorithms like PageRank and BFS. The final system is flexible, scalable, and similar to what real-world data platforms use today. It provides a solid starting point for future work in real-time analytics, distributed systems, and large-scale graph processing.

References

1. Neo4j Documentation - https://neo4j.com/docs/
2. Neo4j GDS Library - https://neo4j.com/docs/graph-data-science/
3. Apache Kafka Documentation - https://kafka.apache.org/
4. Kubernetes Documentation - https://kubernetes.io/
5. NYC Taxi Data - https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page