Distributed Data Management Laboratory Report

Submitted in partial fulfillment of the requirements for the evaluation of

CS750 – Distributed Data Management

by

Himanshu Pandey - 232CS011
Piyus Prabhanjans - 232CS024

Department of Computer Science & Engineering.

National Institute of Technology Karnataka, Surathkal.

Contents

List of Figures

1 Lab 1: Facebook Sentiment Analysis with data streaming Using Kafka

1.1 Abstract

This project report outlines the creation of a real-time data streaming system using Apache Kafka and Python. With today's emphasis on instant data processing, the project aims to provide a practical solution for handling continuous streams of information. The system is divided into two main parts: producers, which generate the data, and consumers, which process it. Apache Kafka acts as the intermediary, offering reliability and efficiency in message transmission. Python was selected for its ease of use and extensive support libraries. The report details the setup of Kafka clusters and the development of producer and consumer applications using Python. Additionally, it explores key features of Kafka, such as partitioning and fault tolerance, and discusses challenges faced during implementation. By sharing insights and future considerations, this report aims to assist developers and data engineers in creating effective real-time data streaming applications using Apache Kafka and Python.

1.2 Introduction

In today's data-driven landscape, the demand for real-time data processing systems has surged dramatically. Businesses across various industries rely on the ability to ingest, analyze, and respond to continuous streams of data in milliseconds. To address this need, the combination of Apache Kafka and Python has emerged as a powerful solution, offering scalability, fault tolerance, and ease of development. This report documents the design, implementation, and evaluation of a real-time data streaming system leveraging Apache Kafka and Python, aimed at providing a comprehensive understanding of building such systems in practical scenarios.

The foundation of our project lies in Apache Kafka, an open-source distributed event streaming platform. Kafka's architecture is built to handle high volumes of data with low latency, making it an ideal choice for real-time data processing applications. By utilizing Kafka's distributed messaging system, data can be seamlessly transmitted and replicated across multiple nodes, ensuring fault tolerance and high availability. Additionally, Kafka's partitioning mechanism allows for parallel processing of data streams, enabling horizontal scalability to meet the demands of growing data volumes.

Complementing Kafka, Python serves as the primary programming language for developing producer and consumer applications within the system. Python's simplicity, readability, and extensive library support make it well-suited for rapid prototyping and deployment of real-time data processing pipelines. Leveraging Python's Kafka libraries, developers can easily integrate Kafka functionalities into their applications, facilitating seamless communication with Kafka clusters. Through this report, we aim to provide a detailed walkthrough of setting up Kafka clusters, developing producer and consumer applications in

Python, and exploring advanced Kafka features to build scalable and resilient real-time data streaming systems.

1.3 Background

The exponential growth of data in recent years has propelled organizations to seek innovative solutions for real-time data processing and analysis. Traditional batch processing systems struggle to keep pace with the increasing volume, velocity, and variety of data generated from various sources such as IoT devices, social media platforms, and online transactions. In response to this challenge, real-time data streaming systems have gained prominence for their ability to handle data streams continuously and provide timely insights for decision-making and action.

Apache Kafka has emerged as a leading technology for building real-time data streaming architectures due to its unique combination of features, including scalability, fault tolerance, and durability. By decoupling producers from consumers and providing a distributed messaging system, Kafka enables seamless communication between components, ensuring efficient data transmission and processing at scale. This scalability and fault tolerance make Kafka well-suited for mission-critical applications where data integrity and reliability are paramount.

The motivation behind this project stems from the need to explore and harness the capabilities of Apache Kafka in conjunction with the simplicity and versatility of Python programming. Python's popularity among developers, coupled with its extensive ecosystem of libraries and frameworks, makes it an attractive choice for building real-time data streaming applications. By leveraging Python's Kafka libraries, developers can streamline the development process and focus on implementing business logic rather than low-level infrastructure concerns. This project aims to empower developers and data engineers with the knowledge and tools necessary to architect robust and scalable real-time data streaming solutions using Apache Kafka and Python.

1.4 Technical Requirements

- **Apache Kafka Cluster**:
  - The project requires the setup and configuration of an Apache Kafka cluster comprising multiple broker nodes.
  - Each broker node must be properly configured to ensure fault tolerance, data replication, and high availability.
  - Additionally, Kafka topics should be appropriately partitioned based on data volume and processing requirements.
- **Python Environment**:
  - A Python development environment is necessary for implementing producer and consumer applications.

  - The environment should include Python interpreter, package manager (e.g., pip), and necessary dependencies such as Kafka client libraries (e.g., kafka-python).

- **Producer Application**:
  - The producer application must be capable of generating continuous data streams to simulate real-world data sources.
  - It should integrate with the Kafka cluster to publish messages to designated Kafka topics efficiently.
  - Configuration options such as message serialization, batching, and error handling should be considered to optimize performance and reliability.

- **Consumer Application**:
  - The consumer application should be able to subscribe to Kafka topics and process incoming messages in real-time.
  - It should support parallelism to handle multiple data streams concurrently and implement error handling mechanisms to ensure message delivery and processing reliability.
  - Additionally, the consumer application may include functionalities such as data transformation, aggregation, or storage based on specific use cases.

- **Kafka Monitoring and Management**:
  - Proper monitoring and management tools should be employed to monitor the health and performance of the Kafka cluster.
  - This may include tools like Kafka Manager, Confluent Control Center, or custom monitoring solutions to track key metrics such as throughput, latency, and partition lag.

1.5 System Design

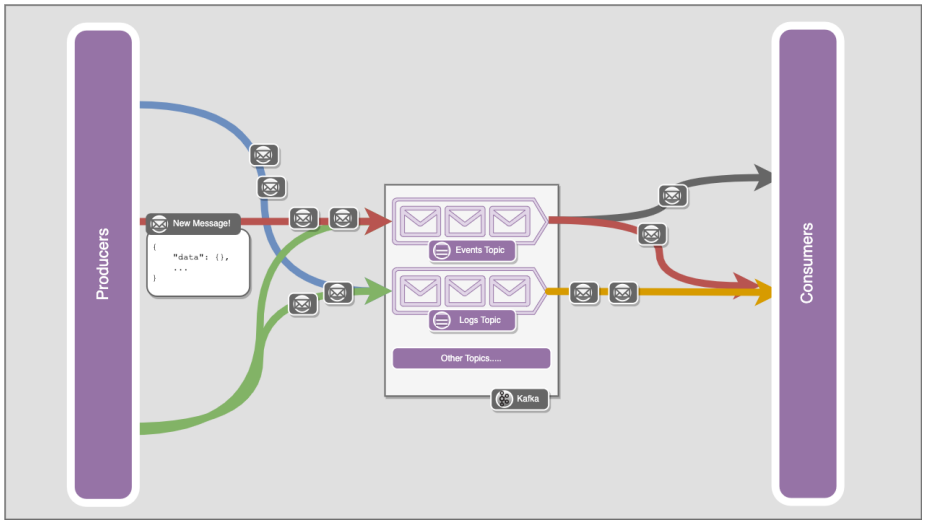

Figure 1: Running Kafka Server.

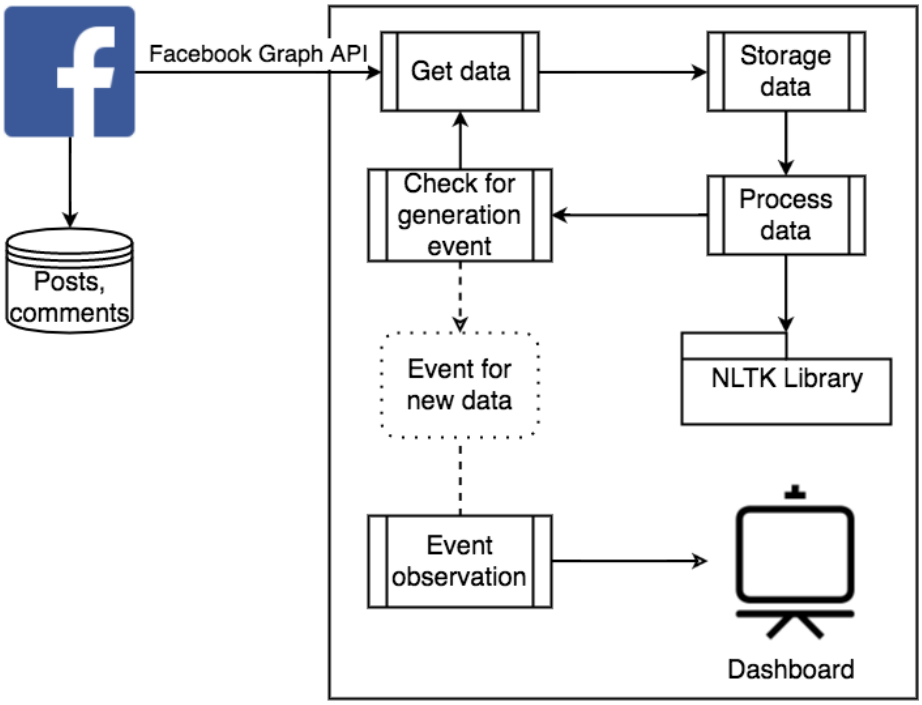

Figure 2: Real time Data Processing.

1.6 Architecture

The basic architectural design for a Kafka-based producer-consumer system, along with Zookeeper, typically involves the following components:

- Producers publish messages to Kafka topics, which are then stored in Kafka brokers.
- Consumers subscribe to specific topics or partitions and consume messages at their own pace.
- Zookeeper maintains the overall state of the Kafka cluster, including metadata about topics, partitions, brokers, and consumer groups.
- Kafka brokers handle the storage, replication, and distribution of messages across partitions and nodes in the cluster.
- The interaction between producers, consumers, Kafka brokers, and Zookeeper enables fault-tolerant, distributed, and scalable message processing in real-time and batch scenarios.

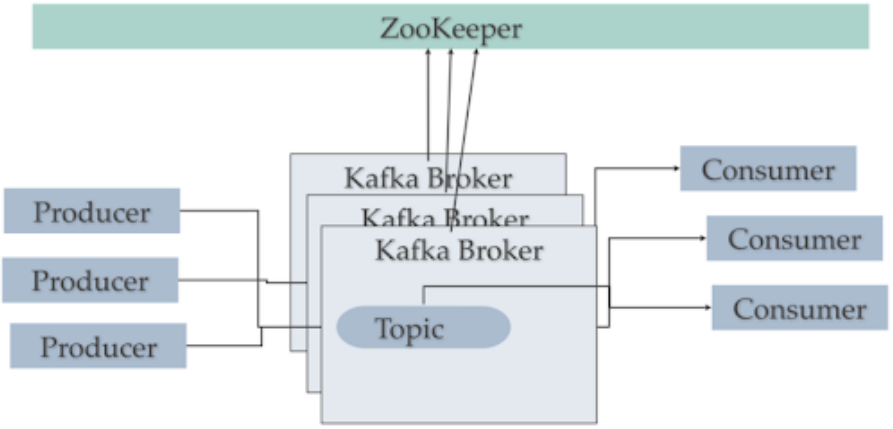

Figure 3: Running Kafka Server.

1.7 Implementation

- **Apache Kafka Cluster Setup**:
  - The project begins with the setup and configuration of an Apache Kafka cluster.
  - Using Docker containers or standalone installations, multiple Kafka broker nodes are deployed across different machines to ensure fault tolerance and high availability.
  - Configuration files such as `server.properties` are customized to specify broker IDs, listeners, replication factors, and other parameters according to the desired cluster architecture.
- **Producer Application Development**:
  - Producer applications are developed in Python using the `kafka-python` library.
  - These applications simulate data sources or sensors, generating continuous streams of data to be ingested by the Kafka cluster.
  - Each producer instance is configured to connect to the Kafka cluster and publish messages to specific Kafka topics using the `KafkaProducer` class.
  - Serialization options such as JSON or Avro may be employed to serialize message payloads efficiently.
- **Consumer Application Development**:
  - Consumer applications are also implemented in Python using the `kafka-python` library.
  - These applications subscribe to Kafka topics and consume messages in real-time.
  - Consumer instances may operate in consumer groups to achieve parallel processing of data streams, ensuring scalability and fault tolerance.
  - Depending on the use case, consumers may perform various operations such as data transformation, aggregation, or storage using the `KafkaConsumer` class.
- **Monitoring and Management**:
  - Proper monitoring and management of the Kafka cluster are crucial for maintaining system health and performance.
  - Monitoring tools such as Kafka Manager or Confluent Control Center may be utilized to track key metrics such as throughput, latency, and partition lag.
  - Additionally, custom monitoring solutions may be implemented using Kafka's built-in metrics and JMX metrics exporters for more granular insights into cluster performance.

1.8 Results and Analysis

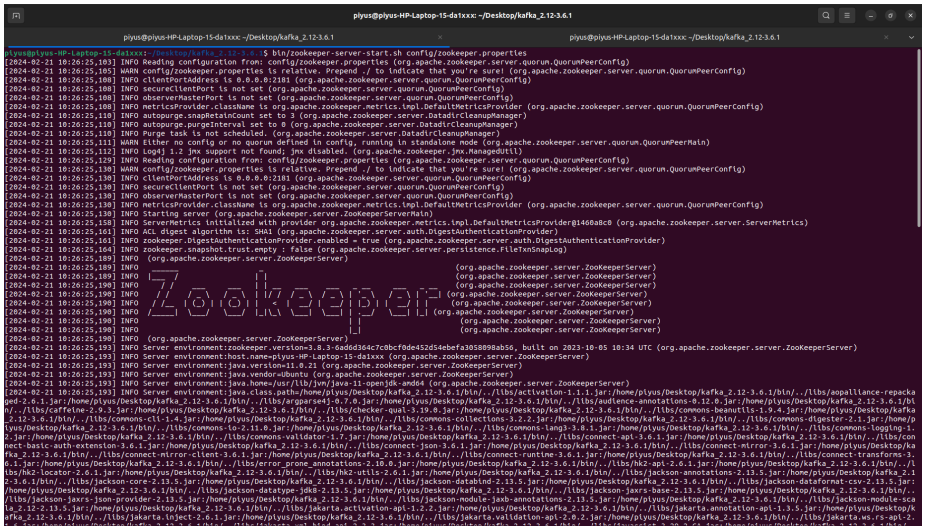

Figure 4: Running Zookeeper Server.

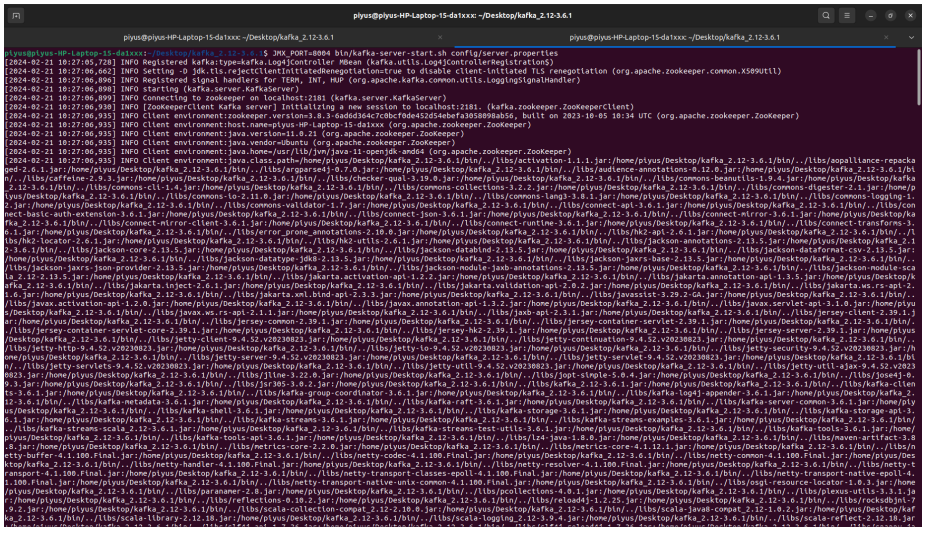

Figure 5: Running Kafka Server.

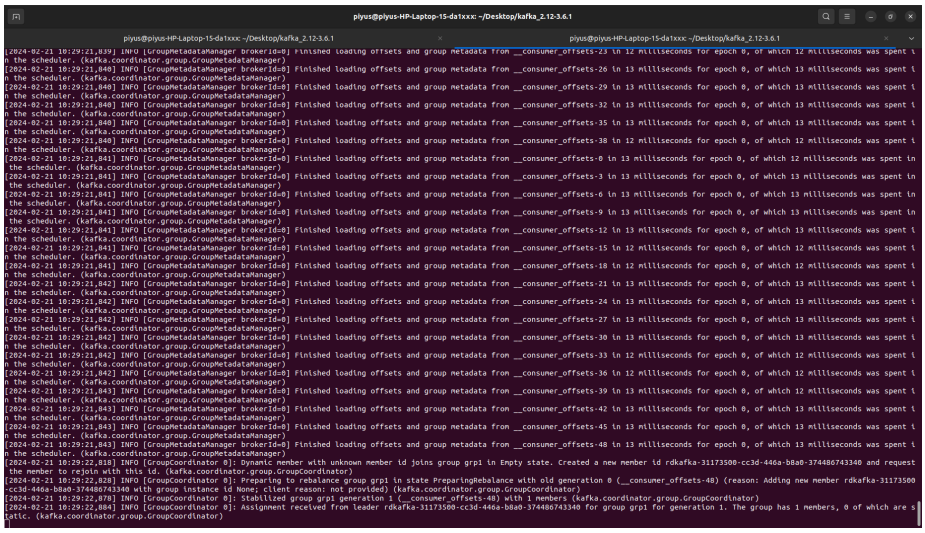

Figure 6: Setting up Kafka Server.

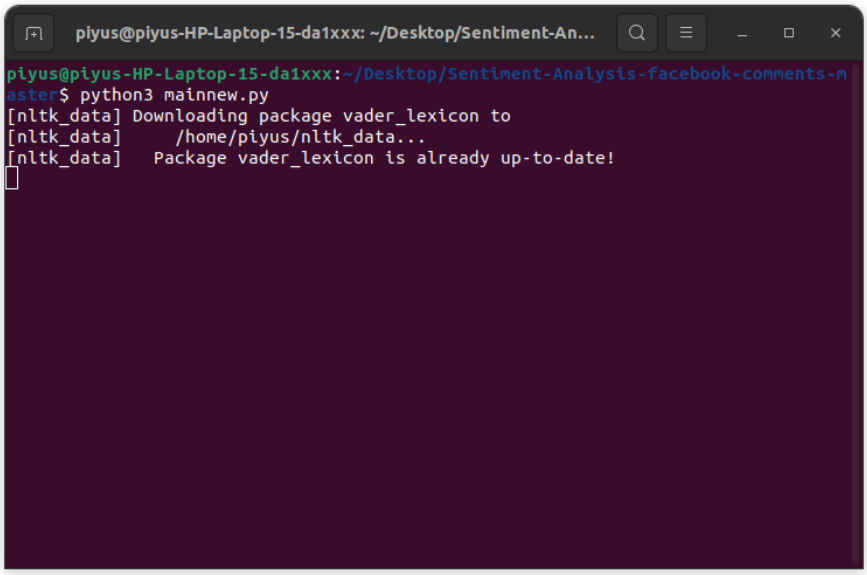

Figure 7: Running Sentiment Analyser Application.

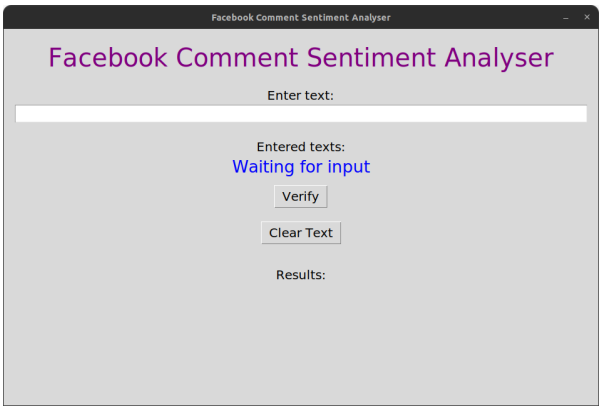

Figure 8: Tkinter Application for Sentiment Analysis.

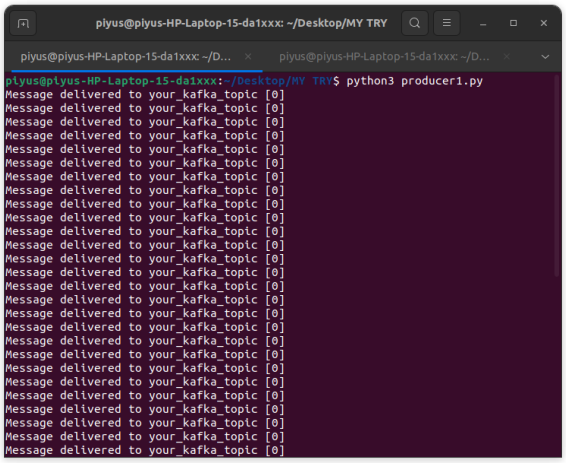

Figure 9: Running Producer Program.

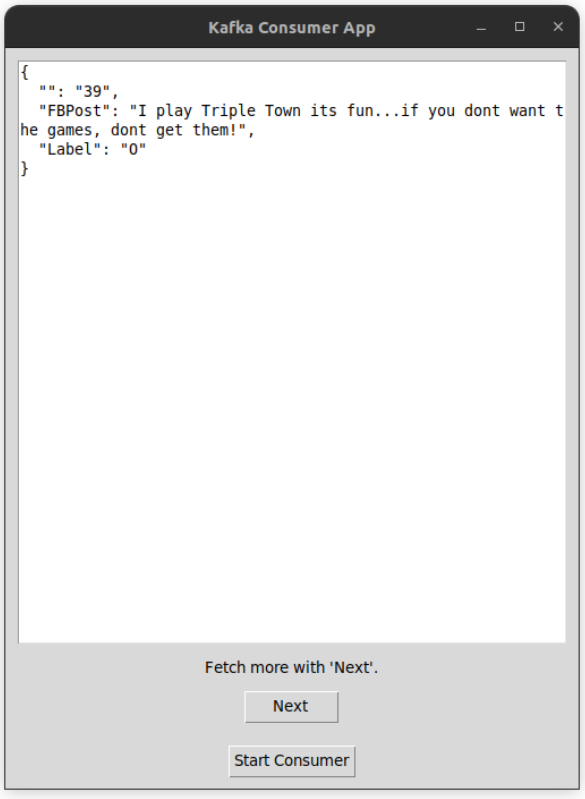

Figure 10: Kafka Consumer Application for Streamed Data.

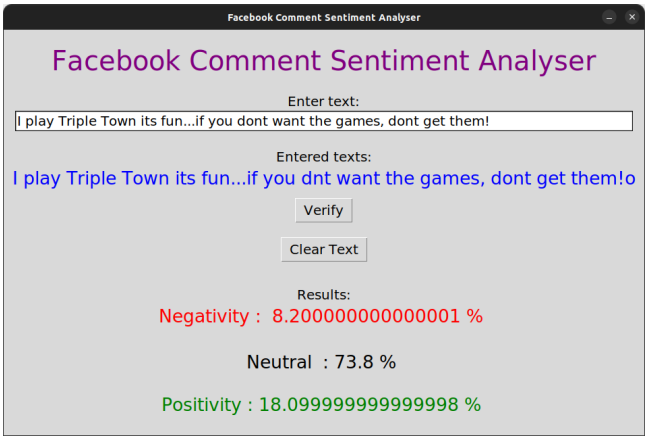

Figure 11: Sentiment Analyzer results for an input.