Analyzing Covid-19 Tweets Dataset Using Apache Spark and Kafka

Adriana Danko
Concordia University
40029681

Himanshu Rathod
Concordia University
40234325

Khush Hardikkumar Jani
Concordia University
40230516

Darshansinh Dilipsinh Devda
Concordia University
40261713

Pranaykumar Chauhan
Concordia University
40266722

ABSTRACT

This project focuses on the analysis of a COVID-19 related tweet data set collected in April 2020 (12). Employing distributed technologies, specifically Apache Spark and Apache Kafka within the Docker environment, a reliable and scalable framework is established. The primary objectives include exploring the data set and understanding public sentiments. Apache Spark, with its layered architecture, and Apache Kafka, a resilient data handling system, form a dynamic platform allowing parallel analysis (5,8,10). The chosen technologies, along with Docker's support for dynamic scaling, contribute to efficient data processing and scalability (5,8,10,11). The study utilizes the "Coronavirus (covid19) Tweets - early April" dataset from Kaggle, emphasizing key features such as dynamic scaling, fault tolerance, reliability, and resource sharing (12). Results from exploratory analysis reveal insights into tweet length distribution, sentiment polarity, and visualizations through t-SNE, showcasing the effectiveness of the distributed system in handling large datasets (12,13). In conclusion, the project demonstrates the value of distributed technologies in real-world applications, particularly in the context of analyzing COVID-19-related tweets.

1 INTRODUCTION

In today's day, social media plays an important role in distributive systems. This is due to the increasing traffic all over the world where individuals communicate with one another through various different social media platforms and applications. In this project, the primary objective focuses on the comprehensive analysis of a data set which is composed of COVID-19 related tweets which was initially collected during the first few weeks of April 2020. In this study two main components are being looked at. These are 1) exploring the data and 2) understanding people's feelings. In order to achieve this, a strong and scalable framework must be created. In order to do this, different distributive technologies, specifically Apache Spark and Apache Kafka. Thus by using together Apache Spark which is a powerful stream processing framework, along with Apache Kafka a distributed event streaming platform, together this creates a dynamic system that will allow us to analyze the data set in parallel. The goal is to gain a deeper understanding into the data set and showcase the effectiveness of distributive technologies in managing large data analysis tasks.

2 SYSTEM ARCHITECTURE

Obtaining a deeper understanding of the framework requires first understanding the architecture of the tools used and how they are integrated within distributive systems.

2.1 Components of Apache Spark Architecture

Apache Spark is a robust and versatile data processing framework known for its efficacy in handling diverse datasets. Illustrated in Figure 1, Spark delineates into four layers: APIs and libraries, execution engine, deployment, and storage components (1,2). A comprehensive exploration of each layer will be undertaken to unravel the framework's intricacies within the context of this project.

Kernel

Central to the architecture of Apache Spark is its core execution engine, akin to the heart of the framework (3). This runtime engine serves as an efficient conductor orchestrating the flow of data through various defined operations (3). Users interact with Apache Spark by crafting code or queries to delineate data processing tasks and stipulate the specific operations to be carried out on datasets or data streams (3). Subsequently, the runtime engine assumes these tasks, orchestrating their execution across the distributed Spark cluster in an efficient manner, thereby managing tasks such as parallel execution, fault tolerance, and resource optimization (3). Operating as an integral part of the Spark framework, the runtime engine diligently processes data in accordance with the users' specified operations (3). Its inclusion within

the framework significantly enhances the framework's capability to seamlessly execute diverse data processing tasks, ensuring efficient utilization of resources and optimal task management.

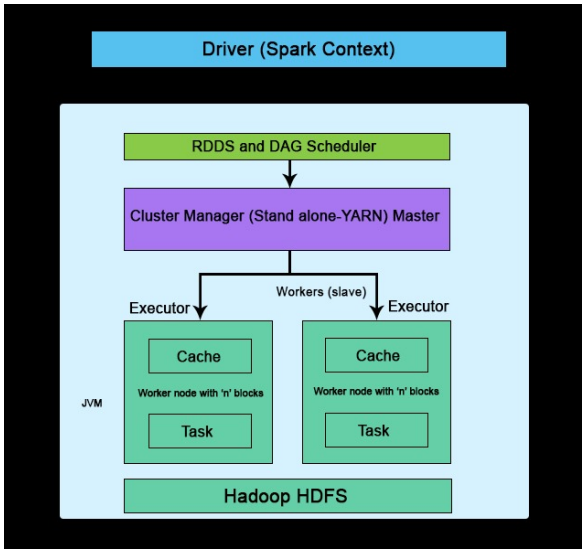

Figure 1: Overview of Apache Spark Architecture (1)

Deploy

The deployment layer in Apache Spark encompasses diverse deployment options, including local, cluster, and cloud configurations (3). This layer offers remarkable flexibility, enabling the execution of data processing tasks in various environments. It facilitates running tasks on a local cluster, which is often preferred for testing and development purposes, as well as in the cloud for large-scale operations (3). This versatile deployment capability allows seamless adaptation to different computing infrastructures, ensuring that Apache Spark can efficiently operate in diverse environments, whether it's for smaller-scale developmental tasks or handling extensive processing workloads in cloud-based environments (3)

The amalgamation of distinct abstractions, the robust runtime engine, flexible deployment options, and adaptable storage solutions collectively position Apache Spark as a highly flexible and adaptable framework capable of accommodating different data handling paradigms. The inclusion of versatile storage options enriches Spark's ecosystem, enhancing its capacity to manage diverse data sources and formats efficiently, thus contributing to its adaptability across various data processing scenarios.

2.2 Apache Kafka

Apache Kafka provides a fast and reliable way of handling large amounts of information (5). There are three main components in Apache Kafka, these are producers, brokers and consumers respectively (5). As shown in figure 2, a brief overview of Apache Kafka architecture is shown.

Brokers

Brokers are important for storing and managing messages and organizing them into topics which as a result helps with fault tolerance through replication with leader and follower replicas (5). This plays an important role to ensure data durability, parallel processing and reliable communication between the producers and consumers in a Kafka cluster (5).

Producers

Lastly, in Kafka the producers are responsible for publishing messages to specific topics (5). They do so by initiating a flow of data within the Kafka system by creating and sending messages to their designated topics (5). These messages are in forms of "key-value" pairs, which are stored and managed by Kafka (5). Thus in other words, the producers constantly create and share data and filling up Kafka with details from places and systems (5).

Consumers

Consumers on the other note subscribe to specific topics and processes messages from the producers which facilitate the real time event reactions or data stream analysis (5). It maintains offset to track its position in a partition which ensure smooth resumption in case of a failure (5). Due to this, the consumers portion contribute to the distributive processing by allowing parallel and scalable consumption, which also helps with fault tolerance (5) .

3 DATA SET

The data set used for this project is Coronavirus (covid19) Tweets - early April from Kaggle (12). This data set contains tweets with related hashtags to the pandemic, such as coronavirus and covid19 which were collected from a time starting around April 11 (12). Other hashtags like StayHomeStaySafe and TestTraceIsolate were included later. The data set is the second in a series, organized to manage the large volume of data, and it contains variables associated with Twitter, including tweet text, account information, hashtags, and locations (12). The absence of retweets is notable, and the data set aims to explore the relationship between the number of covid-19 cases and the volume of related tweets (12).

4 FEATURES

4.1 Dynamic Scaling

Dynamic scaling in distributive system refers to the ability to adjust the capacity of the system with respect to the resource demands (6).

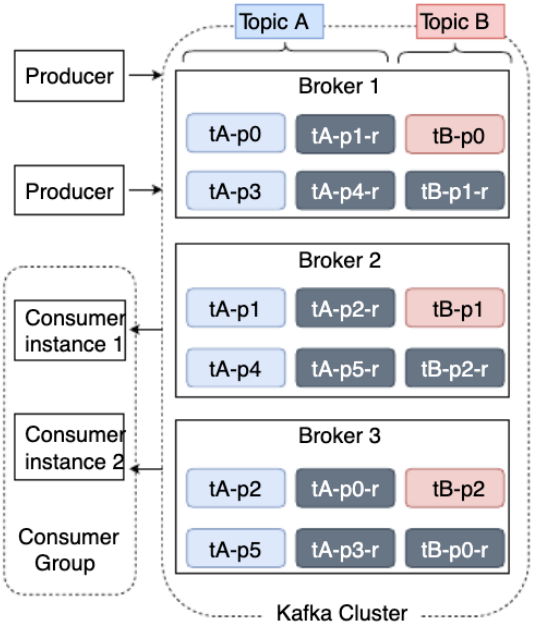

Figure 2: Overview of Apache Kafka Architecture (4)

By using orchestration tools such as Docker Compose, our system can dynamically change the number of Kafka broker instances in response to workload fluctuations. By spreading incoming data streams across the available broker containers in an effective manner, this flexibility guarantees the Kafka cluster's continued resilience.

Our Spark application is able to adapt to the processing needs of distributed jobs by simply scaling up or down thanks to Apache Spark's integrated dynamic resource allocation functionality.

4.2 Fault Tolerance and Reliability

Fault tolerance in distributive systems indicates its ability to continue functioning and providing services when some of its components experience failure (7). Reliability plays a key role in distributed systems when it comes to ensuring accurate data and service processing (8).

In the case of Apache Kafka, fault tolerance is a core feature achieved through data replication as mentioned in the previous paragraph. Kafka has the ability to partition data into multiple replicas which are then distributed across different broker nodes (5,7). Thus, in terms of fault tolerance, if one replication fails, another replica can take over which helps maintain both availability (5,7). Kafka-Docker containerization facilitates fault tolerance mechanisms such as data replication and replica takeover, making it easier to manage.

Apache Spark exploits lineage information and resilient distributed datasets (RDDs) to provide fault tolerance. In the event of a node failure, RDDs' partitioned and immutable nature makes it easier to recover missing partitions, while lineage information allows Spark to recalculate transformations on the original data source.

4.3 Resource Sharing

Resource sharing in distributive system involves the effective allocation and utilization of the computing resources with the different components (9). Merging Kafka topics into a single consumer group allows for parallelized message processing, where multiple consumers within the group can independently handle different partitions concurrently. Within a cluster, Apache Spark automatically facilitates resource sharing by dividing workloads among executor instances so that tasks may be completed concurrently.

5 DEMO

Our demo showcases the seamless integration of Apache Spark and Apache Kafka for efficient data analysis. Using Docker containers, Kafka and Zookeeper are hosted, with Zookeeper managing Kafka brokers for distributed messaging. Multiple Kafka topics are established to segregate data from CSV files, maintaining structural consistency. Spark is intricately integrated, enabling efficient data consumption, diverse data analysis tasks, and extraction of meaningful insights. The system demonstrates adaptability through dynamic scaling and fault tolerance, ensuring continuous processing and mitigating data loss risks. Kafka ensures reliable data ingestion and streaming, maintaining ordered logs within topics. Resource sharing in the Spark cluster optimizes CPU and memory utilization, improving overall data processing efficiency. The insights derived from analysis are visually represented, emphasizing the successful integration of Spark and Kafka with attributes like dynamic scaling, fault tolerance, reliable data ingestion, and resource optimization.

6 RESULTS

In the initial exploratory analysis, we have used an online source "Covid tweets v3" by Mallik Sruti (13). We have merged all the CSV files obtained from Kaggle and any account names were removed. In Figure 3, we've generated a distinctive word cloud using unique hashtags, employing the stylecloud library. This visual representation offers an engaging snapshot of prevalent themes within the dataset.

The polarity metric, analyzed using the TextBlob library,

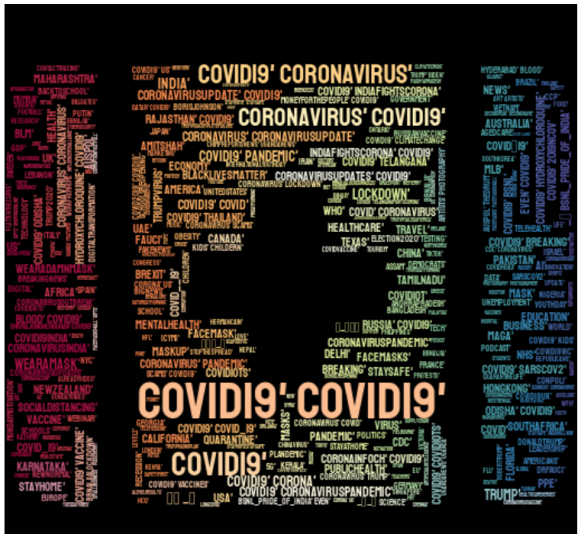

Figure 3: Stylecloud of unique hashtags

assigns numerical values representing the sentiment polarity of text. (13) Figure 4 illustrates that the dataset exhibits a notable inclination towards positive sentiments.

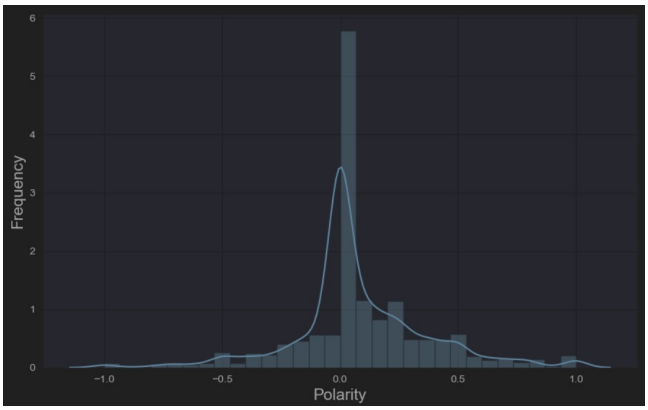

Figure 4: Polarity of the data

Topic modeling of COVID-19 tweets in figure 4 involves applying dimensionality reduction, such as t-SNE, to represent the tweets in a 3D space. This enables visualization of distinct clusters or topics within the tweets, providing insights into prevalent themes and discussions related to COVID-19 on social media.(14)

7 CONCLUSION

In conclusion, this project employs Apache Spark, Apache Kafka, and Docker to analyze a COVID-19 tweets dataset for sentiment, length, and similarity. These distributed technologies, through dynamic scaling, fault tolerance, and resource

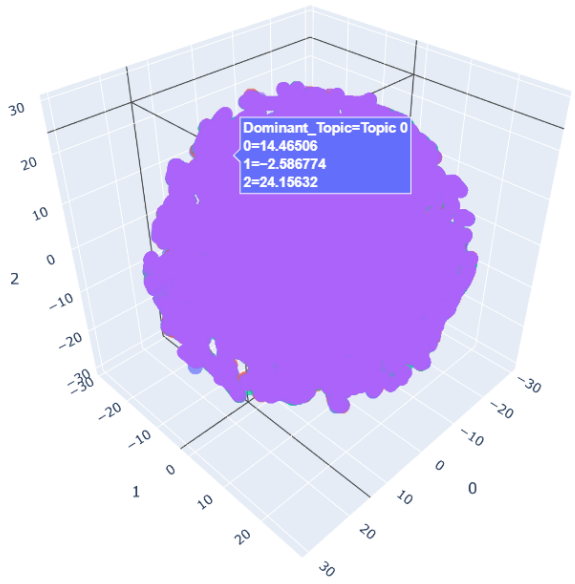

Figure 5: Topic Modeling through t-SNE

sharing, create a robust and scalable framework. Apache Spark's architecture facilitates flexible data processing, while Apache Kafka ensures reliable streaming. Docker enables seamless deployment of distributed components. Preliminary results show positive sentiments dominate, and visualizations reveal data relationships and frequent hashtags, demonstrating the effectiveness of distributed technologies in real-time data processing.

8 REFERENCES

(1) APedamkar.P. Apache Spark Architecture. (2023). https://www.educba.com/apache-spark-architecture/

(2) Aktas. M and Dhaouadi.J. On the Data Stream Processing Frameworks: A Case Study. (2018). https://www.researchgate.net/publication/329621212_On_the_Data_Stream_Processing_Frameworks_A_Case_Study

(3) Petrakis.E, Tsakos.K and Lazidis.A. Publish-Subscribe approaches for the IoT and the cloud: Functional and performance evaluation of open-source systems. (2022) https://www.researchgate.net/publication/360518587_Publish-Subscribe_approaches_for_the_IoT_and_the_cloud_Functional_and_performance_evaluation_of_open-source_systems

(4) Wu.H, Zhihao.S and Wolter.K. Performance Prediction for the Apache Kafka Messaging System. (2019). https://www.researchgate.net/publication/336254369_Performance_Prediction_for_the_Apache_Kafka_Messaging_System

(5) Haji.L, Zeebaree.S.R.M., Ahmed.O and Bibo Sallow.A. Dynamic Resource Allocation for Distributed Systems and Cloud Computing. (2020) https://www.researchgate.net/publication/342317991_Dynamic_Resource_Allocation_for_Distributed_Systems_and_Cloud_Computing

(6) Dagur.A, Yadav.R.S and Ranvijay.A.J. Fault Tolerance in Real Time Distributed System. (2011) https://www.researchgate.net/publication/50247535_Fault_Tolerance_in_Real_Time_Distributed_System

(7) El Ghor. H, Hage.R and Fliti.T. Reliability Analysis in Parallel and Distributed Systems with Network Contention. (2013). https://www.researchgate.net/publication/316788043_Reliability_Analysis_in_Parallel_and_Distributed_Systems_with_Network_Contention

(8) Weerasinghe.S, Kathriarachchi.R, Hettige. B and Karunananda.A. Resource Sharing in Distributed Environment using Multi-agent Technology. (2017). https://www.researchgate.net/publication/317608226_Resource_Sharing_in_Distributed_Environment_using_Multi-agent_Technology

(9) Rad.B.B, Bhatti. H and Ahmadi.M. An Introduction to Docker and Analysis of its Performance. (2017). https://www.researchgate.net/publication/318816158_An_Introduction_to_Docker_and_Analysis_of_its_Performance

(10) Smith. S. Data Set - Coronavirus (covid19) Tweets - early April. (2020) https://www.kaggle.com/datasets/smid80/coronavirus-covid19-tweets-early-april?select=2020-03-29+Coronavirus+Tweets.CSV

(11) Mallik.S. Covid tweets v3. (2021) https://www.kaggle.com/code/srutimallik/covid-tweets-v3