Big Data Streaming Architecture for Edge Computing Using Kafka and Rockset

Ajay Bandi
School of Computer Science and Information Systems
Northwest Missouri State University
Maryville, Missouri, USA
ajay@nwmissouri.edu

Julio Ariel Hurtado
IDIS Research Group
Universidad del Cauca
Popayán, Colombia
ahurtado@unicauca.edu.co

***Abstract*—Due to the ever-increasing growth of continuously generated data streaming applications, there is a need for adopting new streaming data architectures to support low-latency between the source and destination. This work proposes an architecture to extract and pipeline Twitter's streaming data using Kafka, complemented by Rockset, and visualize the analytics with Tableau. We developed an application to capture tweets for each trend and ingesting the data into Kafka. The data aggregation and processing occurred in Kafka before storing it in the cloud using Kafka producers and consumers. The only consumer we used is Rockset, a real-time streaming analytics tool to filter the data by writing queries. Finally, the data Rockset is connected to Tableau to visualize real-time trends using treemaps on dashboards and storyboards.**

***Index Terms*—Big Data, Multi-access Edge Computing, Fog Computing, Architecture, Twitter, Kafka, Rockset, Tableau**

I. Introduction

The rapid increase of continuous data generation by various data sources such as mobile applications [1] and IoT devices [2] brings out challenges to store, process, and compute data. The continuous emission of data is called streaming data or interchangeably called data streaming. The need for analyzing the streaming data is increasing. The following are a few examples that require processing streaming data. Real-time systems must detect distributed denial of service attacks for security, analyze social media posts for intelligence purposes, predict trends in stock markets [3], monitor the sensory data to find out the wear-out parts in a manufacturing unit, and identify the fraud transactions in banking. The International Data Corporation (IDC) predicts that there will be 42 billion IoT devices, and the expected data produced from these devices would be 80 zettabytes ($8x10^{22}$) [4]. It is difficult and time-consuming for organizations to capture, pipeline, and process the streaming data. Multi-access Edge Computing (MEC) offers decentralized storage and computing resources capabilities and reduces the latency in communication. However, in MEC, there should be an end-to-end message delivery system, like Kafka [5], and the streaming analytic platform from the source to the destination. When vast amounts of data are continuously generated from social media sites, the data is stored in cloudlets [6] or fog nodes [7] to process, compute, and then communicate back with end devices. It is challenging to store and analyze streaming data. We focus on processing and curating the data before transferring it into the cloud or the edge node instead of storing and retrieving it. This approach minimizes the latency and able to analyze the streaming data on the fly.

This work focuses on adopting the data streaming kappa architecture [8] to utilize technologies such as Apache Kafka and Rockset used in capturing and pipelining big data . This research aims to minimize the latency between the cloud and the end devices while capturing, processing, and visualizing the real-time trends of social media, such as Twitter data. A test application is implemented to extract streaming data from Twitter and ingest it to Kafka, a message streaming system. Kafka uses its producers and consumers to process, aggregate propagate the data through Rockset and then to Tableau creating dashboards and storyboards. The trends from the real-time tweets are updated every 30 seconds. Therefore, the end-to-end messaging and data transmitting time from source (Twitter) to destination (Tableau) in our application is 30 seconds. The rest of the paper is organized as follows: Section II presents the related work in big data streaming architecture. Section III explains our proposed architecture for developing an end-to-end messaging system. Section IV details the implementation of the case study. Section V highlights the results of our study. Section VI presents the conclusion.

II. Related Work

A. Big Data in Edge Computing

Cloud computing plays an essential role in processing a massive amount of data. With the increase in IoT devices' growth, a middle layer called fog node is introduced between the end devices and the cloud [6], [9]. These fog nodes can store and compute data, which takes data processing in one hop, reducing the latency and network traffic. However, there are still challenges in processing the streaming data as it still has to deal with storing and retrieving it. Big data technologies address this issue and provide capturing and processing the data before storing it to the fog node, thus reducing communication latency. Apache Kafka, Flink, Spark play a vital role in processing streaming data [7].

Wang et al. [10] created a transcoding strategy for adaptive wireless video streaming to deploy transcoding servers closer to the base stations. Garg et al. [11] used an edge computing

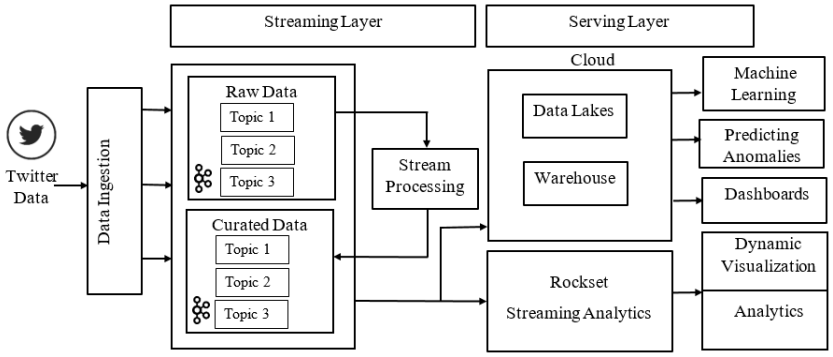

Fig. 1. Proposed architecture to visualize Twitter streaming data

security-based framework to process vast volumes of vehicular ad-hoc networks (VANETs), including Vehicle to Vehicle (V2V) and Vehicle to Everything (V2E), for providing data analytics. In addition to these, several studies focus on edge computing to minimize latency. Minimal research is concentrated on utilizing and implementing big data technologies to create an end-to-end messaging system reducing latency. Due to the exponential growth of data, there is a need to design big data streaming frameworks that use big data technologies to process it and communicate with low latency.

B. Data Streaming Architectures

Researchers worked on big data streaming architectures to minimize latency. Scolati et al [12] proposed a container-based approach that process and filter data before communicating to the edge cloud. They developed a test system using a batch of Raspberry Pis relying on Docker to containerize and deploy a Hadoop and Spark cluster. Carreira et al. [13] analyzed the Spark streaming performance. They identified the reasons for performance bottlenecks. The authors delineate and implement solutions and report learned lessons related to using caching and more quantity and resources for consumers threads.

Lohrmann et al. [14] show a flexible solution for achieving low latencies in scalable stream processing engines of general-purpose while minimizing resource consumption. They propose a queue-based latency model; when applied, data parallelism is minimized, improving the latency expectations. Applying this process iteratively adapts the system to the current load. They processed network social data, with a latency under 20 ms over 90% of the window time. Cooper et al. [15] followed the idea of analyzing stream processing models based on queue theory. They used a server with an external and an internal queue. Considering instantaneous or non-instantaneous data transfers, they identified some trade-offs for balancing holding costs and transfer costs. Thus, if the optimal batch is underestimated, transference could be very expensive, but overestimating it could affect data safety. We proposed kappa-based data streaming architecture where the raw data is processed and curated using Kafka before sending it to the cloud, and this happens on the end-user side. Thus, adjusting the latency compared to other related works.

III. Proposed Architecture

This section compares the lambda and kappa data streaming architectures and presents our proposed architecture to extract the streaming data from Twitter and visualize the real-time data in Tableau complemented by Rockset.

The rapid growth of data streaming applications created a demand for processing data online. The lambda architecture [8], [16] is a standard streaming deployment architecture pattern used to feed data in batch and streaming layers. The streaming layer uses the previous insights derived in the batch layer for processing new incoming data. The data from multiple sources is ingested and applies the appending or aggregating techniques to the data and delivered to the service layer. The service layer consumes data for descriptive and predictive analytics. It is important to note that the lambda architecture has a separate batch layer to provide insights to the streaming layer.

Kappa architecture [8], [16] is not a substitute for lambda architecture; instead, it provides an alternate approach to process data online exclusively and communicates synchronously with the voluminous data streams. Unlike lambda architecture to process streaming data, kappa architecture has no batch layer and all the complex computations happening in the streaming layer. The data ingested to the streaming layer is real-time streaming data and batch data such as data from mass databases and files, and consolidate streaming data with change data capture. The data from the streaming layer is disseminated to the service layer. The service layer comprises cloud-based data lakes and data warehouses. It also includes several other business intelligence tools to perform analytics, create dashboards, storyboards, and predict anomalies using machine learning algorithms.

The kappa architecture [8], [16] offers several advantages by loosely coupling the streaming and service layers with tools like Kafka, Spark, and Flink. This helps businesses independently build and maintain the origin and destination systems to maximize resilience and minimize latency and downtime. Companies can also store the original streaming data for longer intervals in the messaging systems that allow reprocessing before delivery to the service layer.

The proposed architecture uses kappa data streaming architecture and consists of streaming and serving layers as shown in Figure 1. The data is extracted from Twitter ingested into the streaming layer. Kafka processes the ingested raw data and converts it to the curated data in an organized way, ready to consume by the consumers. The only consumer we used is Rockset, and the data is connected to Tableau for real-time visualizations.

IV. Case Study

This section explains the end-to-end messaging system from Twitter to Tableau to create dashboards and storyboards to visualize the real-time streaming analytical data using Kafka and Rockset.

A. Tools Used

1) Twitter App as Data Source: Twitter is a social networking application with approximately 100 million daily active users and 500 million tweets sent daily. In implementing the research goal, we used the Twitter API, which allows retrieving the trending topics worldwide and country-wise along with tweet_volume of the respective trending topic for the last 24 hours. A developer account is needed to access Twitter topics.
2) Apache Kafka: Kafka is an open-source publish-subscribe messaging system used to send and receive messages from different data sources. This project uses Kafka as a messaging system to deliver the Twitter real-time data retrieved from the Twitter API to Tableau to analyze and visualize the trending topics and their respective tweets-volume. In this research, Kafka's primary purpose is to deliver the data from Twitter to Tableau using the connector called Rockset.
3) Rockset: Rockset automatically indexes your data- structured, semi-structured, geo, and time-series data- for real-time search and analytics at scale. Create personalized user experiences, build real-time decision systems or serve IoT applications with a real-time indexing database that can power sub-second queries at a massive scale.
4) Tableau: Tableau is one of the most commonly used data visualization tools, which helps us to visualize and analyze the data. In our project, we are making use of Tableau to visualize the trending topics and tweets volume.

B. Data and its source

The current trends are the popular hashtags or the most repeated themes in the tweets that determine by Twitter. We can collect country-wise or worldwide trends using the where on earth identifier (WOEID), a 32-bit identifier used to identify each country. We collect the following data features for every 30 seconds.

1) Trending names: The names of top 50 current trends
2) Tweet volume: Number of tweets on a particular trend
3) Trend rank: Individual rank of top 50 trends

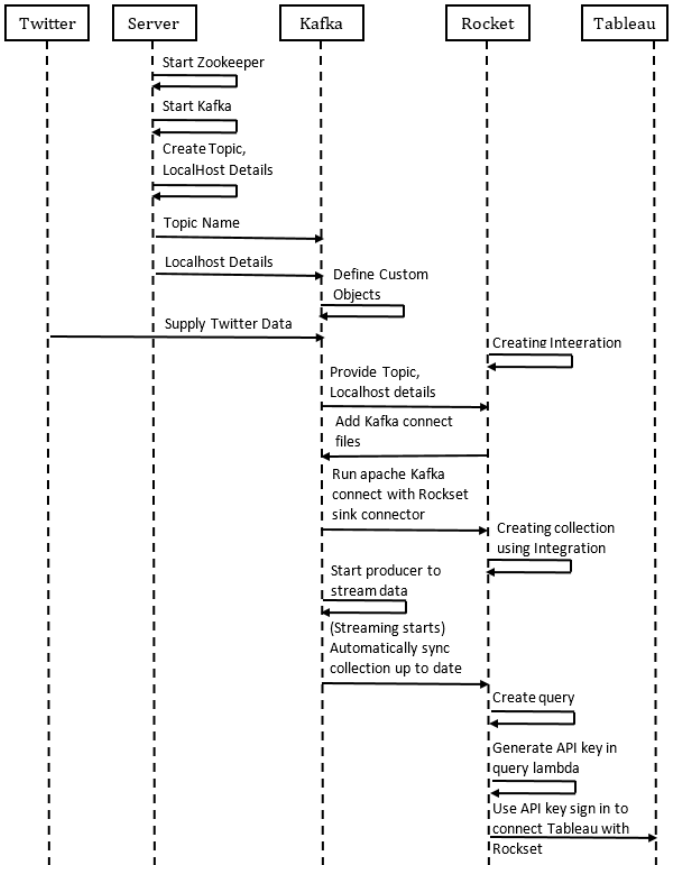

Fig. 2. Interaction of Big Data technologies represented in a sequence diagram

One of the social media platform, Twitter is the data source to collect real-time trend names, rank, and the tweet volume.

C. Data Extraction

In this research, the real-time tweets from Twitter is used as the streaming data. There is a need to create a developer account on the Twitter website[1] to access and extract the streaming data. Then, the user must specify the role that describes the intention to use the Twitter developer platform. Some of the available options are professional, hobbyist, student, academic researcher, etc. There are limitations of approving Twitter developer's accounts, so it is essential to specify its proper use case. If the account gets approved, a developer can begin developing apps, improving skills, and showcasing the completed projects using the Twitter API. After the developer's account approval, login into the dashboard to create a new app under the projects and apps category. After naming the app, it creates API keys (API key and API secret key) and authentication keys (access token and secret access token) that will be used further used in the project to access the data from Twitter.

This research used the `twitter4j` package and `twitter4j.properties` file to create an instance for Twitter and retrieve the required data from that instance. The

[1]https://developer.twitter.com/en

twitter4j is a library in Java language that can easily integrate Java application with the Twitter service. This library provides a convenient API for accessing the Twitter APIs with built-in OAuth support and no additional dependencies. The Maven build tool is used to create the Twitter instance. The complete source code for data extraction and processing is given in the GitHub repository[2]. Create a Java Maven project and a dependency for twitter4j in the pom.xml file.

```
<dependency>
  <groupId>org.twitter4j</groupId>
  <artifactId>
        twitter4j-stream
  </artifactId>
  <version>4.0.6</version>
</dependency>
```

Then, create the twitter4j.properties file and fill the below OAuth information from the Twitter developer's account. The values of these attributes can be obtained from the Twitter developer console after creating a new app.

```
oauth.consumerKey =          // key
oauth.consumerSecret =       // secret
oauth.accessToken =          // token
oauth.accessTokenSecret =    // token secret
```

After setting the keys and tokens in the twitter4j.properties file, the developer needs to create an instance for Twitter API. The classes FileInputStream, Properties, ConfigurationBuilder, and TwitterFactory, are used to create a Twitter instance. The following is the pseudocode for witter instance.

```
FileInputStream twitterStream = null;
Properties twitterProperties =
          new Properties();
twitterStream = new
      FileInputStream(twitterFile);
twitterProperties.load(twitterStream);
ConfigurationBuilder cb = new
      ConfigurationBuilder();
TwitterFactory tf = new
      TwitterFactory(cb.build());
Twitter twitter = tf.getInstance();
```

Initialize the LinkedHashMap and put the key and value pairs. The key value is the woeID, and the value is the country name.

```
countries.put(1, ``Worldwide");
countries.put(23424848, ``India");
countries.put(23424975, ``United Kingdom");
countries.put(23424977, ``United States");
```

Iterate over the LinkedHashMap, obtain the trends from the Twitter instance object based on the woeID for the respective country. Then, iterate over the trends and get the trend name and tweet volume. Next, initialize the CustomObject with the following values: country, trend name, the rank of the trend, and tweet volume of the respective trend.

[2]https://github.com/bandiajay/kafka-twitterStream

D. Data Pipelining through Kafka

The streaming data (trend name and number of tweets) is retrieved based on the country and worldwide for every 30 seconds in ascending order of trend at that moment. This data is then pipelined through Kafka, a distributed event streaming platform for high-performance and throughput with a latency rate as low as two milliseconds. Kafka also helped to capture, pipeline, integrate the streaming data, and derive its analytics. The sequence diagram depicts in Figure 2 how the data is processed through Kafka, Rockset, and connected to a visualization tool, Tableau.

To setup Kafka, we need to install the Kafka cluster, start the Zookeeper, and run Kafka sever. Zookeeper keeps track of the Kafka cluster nodes' status and keeps track of Kafka topics, partitions, etc. A topic is a unique feed name given to a message stream or a data stream. Kafka producer sends message stream to the topic, and then the broker stores the message with that topic. Multiple producers can send message streams to a single topic. A topic can have one or more partitions. Kafka can handle voluminous data, and data can originate from multiple producers to a single topic; it is challenging for the broker to store data in a single machine. Due to the distributed nature of Kafka, the data is distributed to various partitions on different machines.

Kafka has an inbuilt KafkaProducer<K, V> class that stores the streaming data in a user-defined format (CustomObject) using the serialization process. It is the process of converting the given data type into byte format. In this research, we defined the CustomObjectSerializer class to perform the serialization to send data to the consumer. In this research, the CustomObject has a country, trend rank, trend name, and tweet volume as attributes.

```
CustomObject co = new CustomObject
 (country.getValue(),
 rank, trend.getName(),
 trend.getTweetVolume());
```

The Kafka producer is created with the configuration properties file. The key-value pair is the topic name and CustomObject.

```
Producer<String,CustomObject> producer=new
KafkaProducer<String,CustomObject>
(configProperties);
```

```
BOOTSTRAP_SERVERS_CONFIG=localhost:9092
KEY_SERIALIZER_CLASS_CONFIG=
   org.apache.kafka.common.
   serialization.ByteArraySerializer
VALUE_SERIALIZER_CLASS_CONFIG=
   KafkaCustomObjectSerializer
TOPIC=test
```

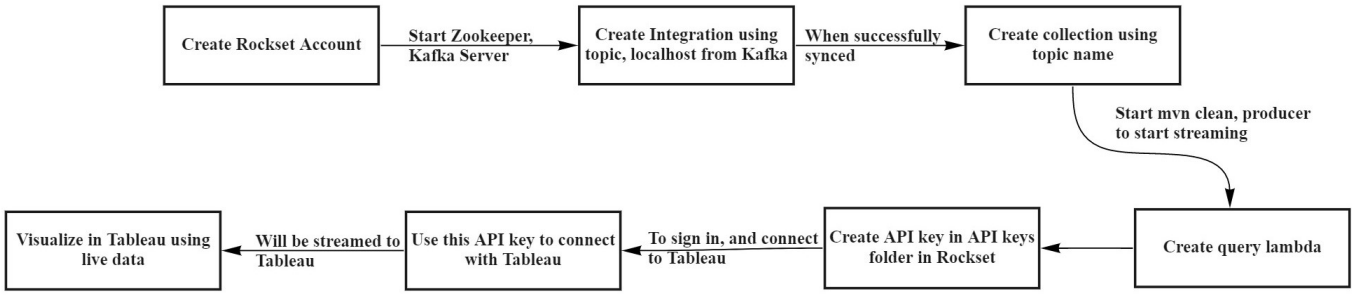

Fig. 3. Data streaming from Rockset to Tableau

Apache Kafka has a built-in ProducerRecord class to create a record with no key using the record contents and the topic name where the record to be sent. Next, Kafka producer asynchronously sends this record to a topic. Now, the data is ready to use by the consumer.

```
ProducerRecord<String, CustomObject> record
 = new ProducerRecord<>(topicName, co);
producer.send(record);
```

Kafka also has an inbuilt `KafkaConsumer<K, V>` class that reads the streaming data from Kafka producer and deserializes it. Desearlization is the process of converting the byte format into the required format. In this research, we defined the `CustomObjectDeserializer` class to perform the deserialization. This case study used only one consumer, Rockset. However, multiple consumers can consume the data using the topic name.

E. Rockset as a streaming analytics tool

Rockset is designed for developers to use modern data for decision-making. It is a real-time indexing database for a low latency search (usually in milliseconds), performs aggregations, and joins any data. Rockset facilitates this consumed data and store into its inbuilt database. Rockset offers various databases, and we chose SQL to store data. Figure 3 shows the steps involved of the data transmission from Rockset to Tableau.

The developer can create collections where the data source is chosen will be used to load it into the Rockset. This real-time indexing tool will automatically synchronize the collections with the external data sources. To integrate Kafka with Rockset, the developer must choose one or more Kafka topics that they need to synchronize. Also, the developer needs to provide the integration name, description, data format. The data format we used in this research is JSON object.

The next steps are to configure Rockset-Kafka[3] integration through Kafka connect. It is used to send the data from multiple Kafka topics to the Rockset collections, and it is different from the Kafka brokers. The Kafka connect can be configured using two plugins – source or standalone connector and sink connector plugins. The source connectors are used to write from external resources into a Kafka topic. In contrast, the sink connector is used to write from a Kafka topic to the external data resources. The downloaded Kafka connect properties file and the Rockset-sink connector files are placed in a directory as shown below.

```
   / connect-standalone.properties
connect-rockset-sink.properties
kafka-connect-rockset-1.2.0-jar-with
-dependencies.jar
kafka_2.11-2.3.0
```

Rockset can filter the data by writing a query lambda. Then, the API keys must be generated in the collection to connect the streaming data to Tableau. The following query extracts the latest 200 trends from four countries, 50 for each country.

```
SELECT   countryName, trendName,
         trendRank, tweetCount
FROM StreamingDataTwitter.Trends
ORDER BY _event_time DESC LIMIT (200)
```

F. Tableau for dynamic visualizations

The Kafka-connect-rockset-1.2.0-jar-with-dependencies.jar generated .jar file is placed under Tableau drivers under program files. The following configuration steps are involved in connecting between Rockset and Tableau. In the Tableau, click to connect other databases (JDBC), in the pop-up window, set the follwoing feilds. Use `jdbc:rockset://api.rs2.usw2.rockset.com` for URL, Select "PostgreSQL" as dialect, the login credentials to connect the server: Username – connection and password: API Key generated in the Rockset. Once the connection is successful, select the Rockset as the database, select the collection name, and automatically update the option. The streaming data is updated to Tableau for a given interval. In this case, the data is updated every 30 seconds.

V. Results

After successfully capturing and pipelining streaming data through Kafka, Rockset consumes Kafka's data to aggregate, join, and filter the data using database queries. Then, the streaming data from Rockset is connected to Tableau for dynamic visualizations, including dashboards and storyboards.

[3]https://docs.rockset.com/apache-kafka/

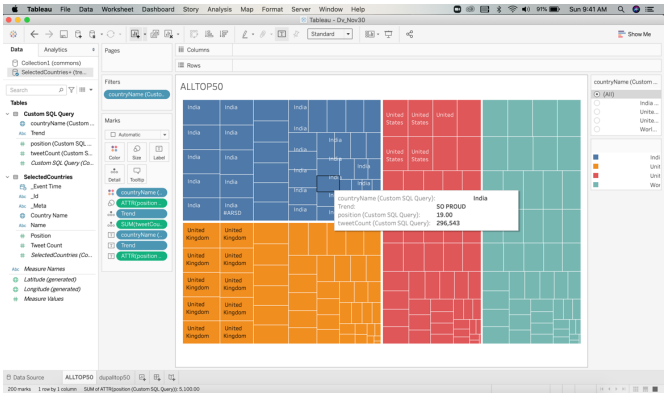

Fig. 4. Treemap showing top trends of four countries

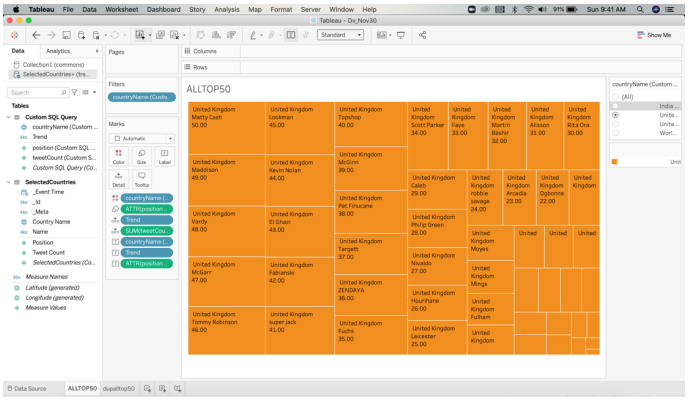

Fig. 5. Treemap presents the UK trends on Twitter using filters

These dynamic visualizations are updated every 30 seconds and are recorded and shown in a video[4].

One of the goals is to visualize the top 50 trends of different countries. The appropriate chart is a treemap to represent the data in nested rectangles. The outer rectangles represent the country, and the inner rectangles represent the trend — the size of the rectangles trend rank based on the tweet count. The treemap of the four countries is shown in Figure 4. To visualize this, drag and drop the trend and its rank in rows and the country name in Tableau's columns.

The second goal is to filter the data based on the country name to visualize the treemap with the particular country's details. For this, drag and drop the county name under the filters. The result of this goal is shown in Figure 5. The final goal is to create the dashboard and storyboard to present the results of the real-time Twitter trends while processing them through Kafka and Rockset.

Thus, we created dashboards and storyboards to visualize streaming data. It is difficult to analyze and make inferences on the continuously generated data from social media. The major challenge is in storing and retrieving the streaming data for later analysis. The storing and retrieving from databases increases the delay in communication from the origin to the end devices. With the proposed architecture, we implemented a test system minimizing its latency. The latency is the amount of time it takes to reflect the data changes between a request and its response. The usage of 5G network applications on mobile devices requires ultra-low latency for effective communication between the cloud and the end devices. In this study, the delay is 30 seconds. Even though 30 seconds is higher, it is due to the limitation of the full access of the tools used in the study.

[4]https://www.youtube.com/watch?v=JHjw0kb8cKE

VI. Conclusion

This research proposed the Big Data streaming architecture that uses existing kappa architecture to process real-time trends and tweets on social media. These tweets are based on the location on earth, a unique identifier for a given country or worldwide. The streaming data is extracted from Twitter and pipelined through Kafka using Kafka producers. Then, the curated data from Kafka is consumed to the streaming analytical tool, Rockset. The Rockset is then connected to a business intelligence tool, Tableau, for creating dynamic dashboards, storyboards, and visualizations. In the results, the treemaps represent the trends on Twitter and tweet volume of four different countries. The streaming data is dynamically updating every half a minute. The overall end-to-end messaging system's latency is 30 seconds. This higher latency is due to the limited access to the tools used. The accessibility of the full version of the software gives desirable results. In the future, we extend this research in introspecting the tweets by predicting the sentiments using machine learning algorithms.

Acknowledgements

We thank Venkatasandeep Katrevula, Kavya Reddy Mylapurapu, and Rohan Goud Bhandari for helping us implement the test system.

References

[1] A. Bandi and A. Fellah, "Design issues for converting websites to mobile sites and apps: A case study," in *2017 International Conference on Computing Methodologies and Communication (ICCMC)*, 2017, pp. 652–656.

[2] A. Kumari, S. Tanwar, S. Tyagi, and N. Kumar, "Verification and validation techniques for streaming big data analytics in internet of things environment," *IET Networks*, vol. 8, no. 3, pp. 155–163, 2018.

[3] A. Bandi, "Data streaming architecture for visualizing cryptocurrency temporal data," in *Proceedings of 3rd International Conference on Computer Networks, Big Data and IoT (ICCBI 2020). Springer*, 2020.

[4] M. Shirer and C. MacGillivray. The growth in connected IoT devices is expected to generate 79.4zb of data in 2025, according to a new idc forecast. [Online]. Available: https://www.idc.com/getdoc.jsp?containerId=prUS45213219

[5] B. R. Hiraman *et al.*, "A study of apache kafka in big data stream processing," in *2018 International Conference on Information, Communication, Engineering and Technology (ICICET)*. IEEE, 2018, pp. 1–3.

[6] S. Mahadev, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[7] S. P. Singh, A. Nayyar, R. Kumar, and A. Sharma, "Fog computing: from architecture to edge computing and big data processing," *The Journal of Supercomputing*, vol. 75, no. 4, pp. 2070–2105, 2019.

[8] J. Lin, "The lambda and the kappa," *IEEE Internet Computing*, vol. 21, no. 5, pp. 60–66, 2017.

[9] A. Bandi and J. A. Hurtado, "Edge computing as an architectural solution: An umbrella review," in *26th Annual International Conference on Advanced Computing and Communications*. Springer, 2020.

[10] D. Wang, Y. Peng, X. Ma, W. Ding, H. Jiang, F. Chen, and J. Liu, "Adaptive wireless video streaming based on edge computing: Opportunities and approaches," *IEEE Transactions on Services Computing*, vol. 12, no. 5, pp. 685–697, 2019.

[11] S. Garg, A. Singh, K. Kaur, G. S. Aujla, S. Batra, N. Kumar, and M. S. Obaidat, "Edge computing-based security framework for big data analytics in vanets," *IEEE Network*, vol. 33, no. 2, pp. 72–81, 2019.

[12] R. Scolati, I. Fronza, N. El Ioini, A. Samir, and C. Pahl, "A containerized big data streaming architecture for edge cloud computing on clustered single-board devices." in *Closer*, 2019, pp. 68–80.

[13] J. Carreira and J. Li, "Optimizing latency and throughput trade-offs in a stream processing system," *University of California at Berkeley, Computer Science Division*, 2014.

[14] B. Lohrmann, P. Janacik, and O. Kao, "Elastic stream processing with latency guarantees," in *2015 IEEE 35th International Conference on Distributed Computing Systems*, 2015, pp. 399–410.

[15] T. Cooper, P. Ezhilchelvan, and I. Mitrani, "A queuing model of a stream-processing server," in *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2019, pp. 27–35.

[16] S. Henning and W. Hasselbring, "Scalable and reliable multi-dimensional sensor data aggregation in data streaming architectures," *Data-Enabled Discovery and Applications*, vol. 4, no. 1, pp. 1–12, 2020.