CSP-554-BIG DATA TECHNOLOGIES

PROJECT DRAFT

BIG DATA PROCESSING PIPELINE

APRIL 19, 2021

YASH PATEL	A20451170
HARSH VORA	A20445400
VISHNU BHARATH	A20465596
VARUN VEERLA	A20458191

ILLINOIS INSTITUTE OF TECHNOLOGY
PROF. JOSEPH ROSEN

SECTION-I: INTRODUCTION

TOPIC: Big Data Processing Pipeline of Twitter Stream API

PROBLEM STATEMENT:

The stream API on Twitter allows you to receive approximately 50 tweets per second. However, this figure must be even higher. Handling, processing, and analyzing this massive volume of real-time data upon its arrival in order to gain information without exceeding the time allotted for decision making or an analytical procedure.

PROPOSED SOLUTION:

A big data processing pipeline is proposed as a workaround. To collect real-time data, also known as event streaming, we will use Apache Kafka as the first portion of the pipeline, which offers a coherent, high-throughput, and low-latency solution. The performance of Apache Kafka will be absorbed as the middle portion of the pipeline for real-time stream data processing into the Apache Spark distributed processing system, which provides data parallelism and fault tolerance. To store vast volumes of processed real-time data, we can use Google Firebase Realtime Database as the last portion of the pipeline, which is a NoSQL database that allows you to store, sync, and query data between users in real-time. We will stream this real-time data to the HTML web-page client for visualization using the firebase kit in the Node.js server.

PROJECT GOALS:

- Ingest data using Twitter's streaming API.
- Capture data using Apache Kafka.
- Process streaming data using Apache Spark.
- Store these processed data using Google Firebase.
- Visualize these processed data using Node.js server and HTML web-page client.

BIG DATA TECHNOLOGIES: Kafka, Spark, Firebase Realtime Database
OTHER TECHNOLOGIES: Node.js, HTML

The main objective of this project is to use big data technologies to build a data processing pipeline that collects data from a source, transforms it, and maintains it in an optimized format such that implementable insights can be extracted from real-time data.

SECTION-II: LITERATURE REVIEW

II.I.I: TWITTER:

Twitter is a social media and news platform that allows users to send and receive short messages known as tweets. Tweeting is the method of posting short tweets to people who follow you on Twitter in the hopes that your comments will be useful and interesting to someone in your circle.

The most significant benefit of Twitter is the ease with which it can be searched. You'll be able to follow hundreds of fascinating Twitter users and read their content easily, which is beneficial in our attention-deficit society.

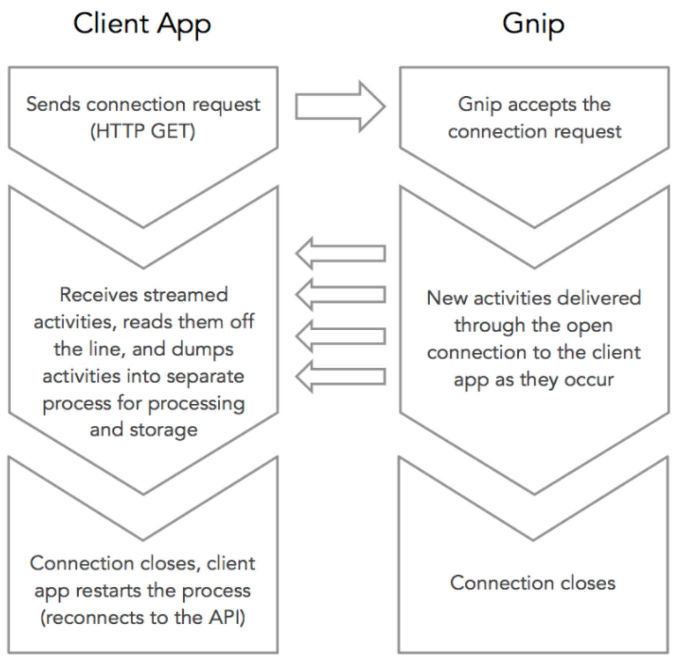
Twitter is used by thousands of people to advertise their staffing agencies, consulting firms, and shopping stores, and it works. The high influence of Twitter on businesses motivates the analysis of Twitter's real-time data to extract actionable insights.

II.I.II: FILTERED STREAM API:

Developers may use the filtered stream endpoint community to filter the real-time stream of public Tweets. The functionality of this endpoint category involves several endpoints that allow you to build and maintain rules, as well as apply certain rules to filter a stream of real-time Tweets and return matching public Tweets. This endpoint community enables users to listen in real-time for relevant issues and activities, track the discussion surrounding competitions, learn how patterns evolve in real-time, and even more.

The Streaming HTTP protocol is used by PowerTrack, Volume (e.g. Decahose, Firehose), and Replay streams to deliver data over an open, streaming API connection. Instead of delivering data in batches through repetitive requests from your client app, as a REST API would, a single connection is established between your app and the API, and new results are transmitted via that connection anytime new matches occur. As a consequence, a low-latency delivery mechanism with high throughput is developed.

For these streams, the ultimate relationship between your app and Twitter's API is as follows:

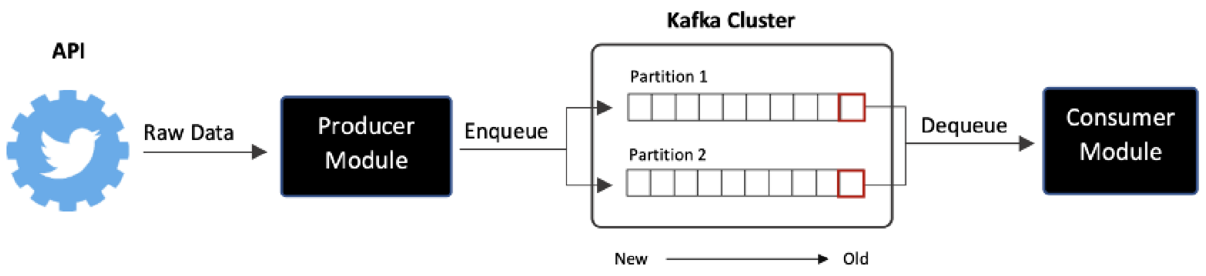

Client App
Gnip
Sends connection request (HTTP GET)
Gnip accepts the connection request
Receives streamed activities, reads them off the line, and dumps activities into separate process for processing and storage
New activities delivered through the open connection to the client app as they occur
Connection closes, client app restarts the process (reconnects to the API)
Connection closes

II.II: Apache Kafka:

Apache Kafka is a real-time data management platform that uses event streaming. It offers a unified, high-throughput, low-latency interface for dealing with massive volumes of real-time data streams. Zookeeper serves as the foundation for Apache Kafka. It monitors the health of brokers and serves as a coordinator for Apache brokers and customers.

Initially, Kafka producers deliver the message to a topic, and Kafka brokers assist in storing the messages in partitions of that topic. When the Apache User subscribes to a topic, Kafka offers the current offset of that topic to the consumer and saves the offset into Zookeeper, which aids in monitoring the previous offset. Messages from the topic are sent to the User before it ceases.

Fault resistance is the most important benefit of using Apache Kafka. It comes with two modules, the first of which is the Producer, which collects data from Twitter. The data is then saved as logs in the queue without being processed, and a module called Consumer reads and processes the logs.



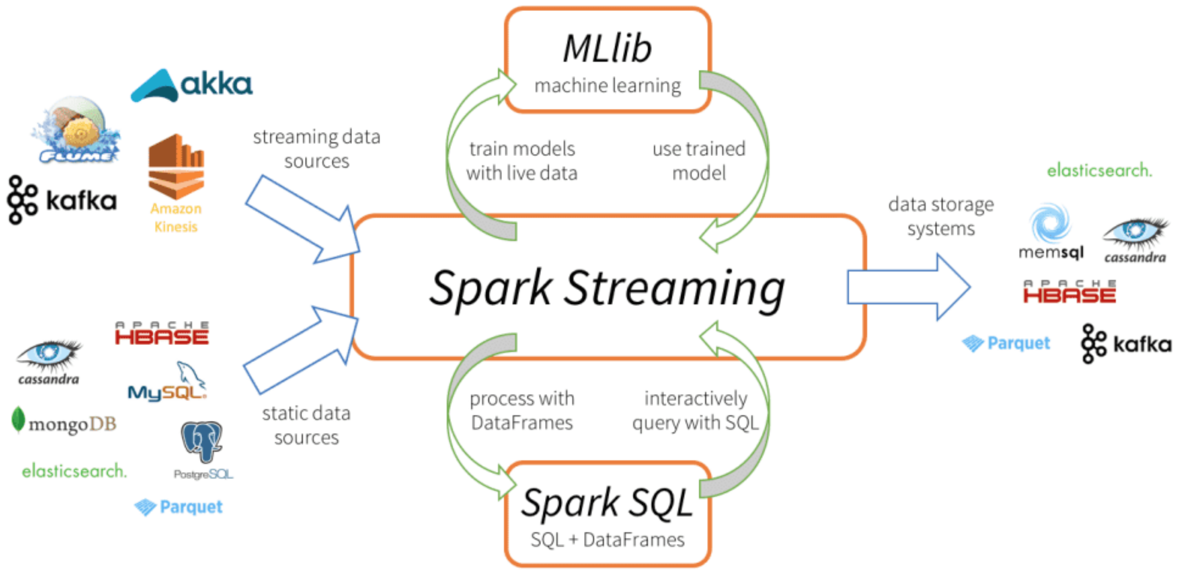
II.III: Apache Spark:

Apache Spark is a Scalable fault-tolerant stream processing that supports both batch and streaming workload. Spark Streaming is an extension of the core Spark API that allows processing real-time data from various sources like Apache Kafka. This transferred data is pushed into file systems, databases, and live dashboards. A Discretized Stream, or DStream, is the primary abstraction, which defines a stream of data separated into small batches. RDDs, Spark's central data abstraction, are the foundation for DStreams.This enables Spark streaming to work in conjunction with other spark elements such as MLlib and Spark SQL effectively.

Since the data is flowing in as a stream, it makes sense to choose a streaming product like Apache Spark Streaming to process it. If you want to write data to disk, this holds data in memory. Streaming info, on the other hand, has meaning when it is life, that is when it is streaming.

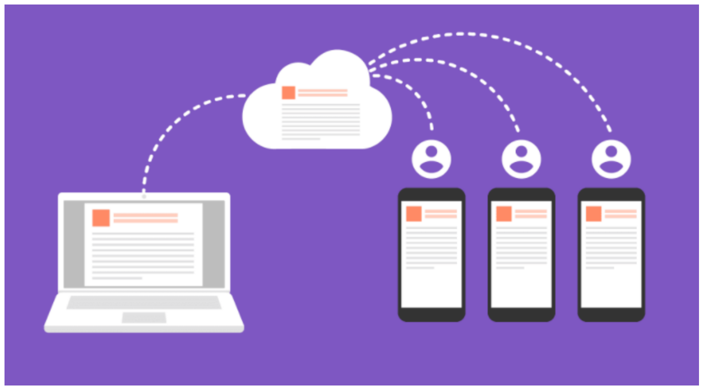
The main benefits of using Apache Spark is:

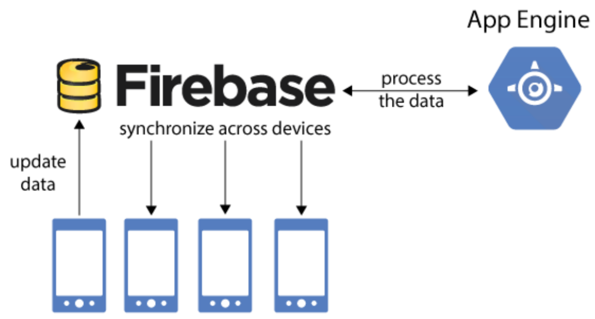
- Fast recovery from failures and stragglers
- Better load balancing and resource usage



II.IV: Firebase Realtime Database:

The Firebase Realtime Database is a NoSQL cloud database that allows you to store, sync and query data in real-time at global scale. The Firebase Realtime Platform is a cloud-based database system. Data is stored in JSON format. When we use the iOS, Android, and JavaScript SDKs to build cross-platform apps, all of our clients share a single Realtime Database instance and receive automated updates on the most recent results.

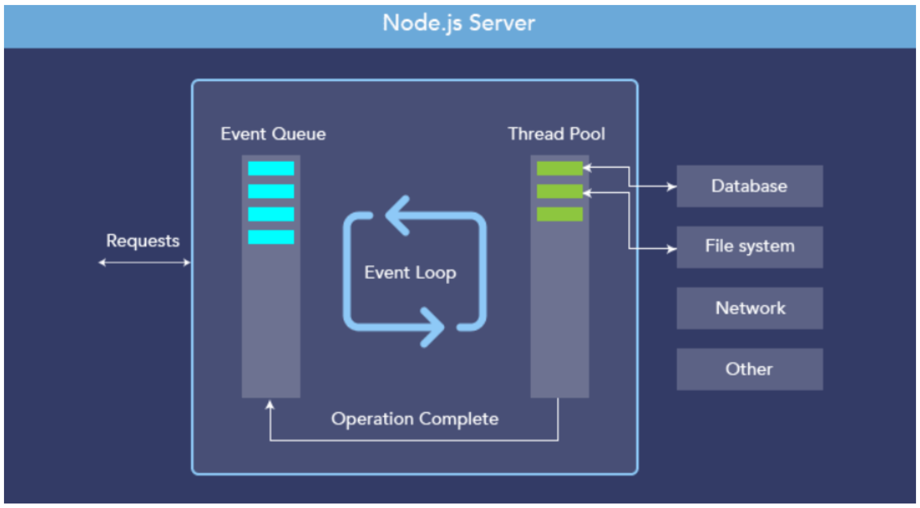




Data is synced in real-time through all connected clients. Realtime events continue to fire, giving the end user a responsive experience.

II.V.I: Node.js:

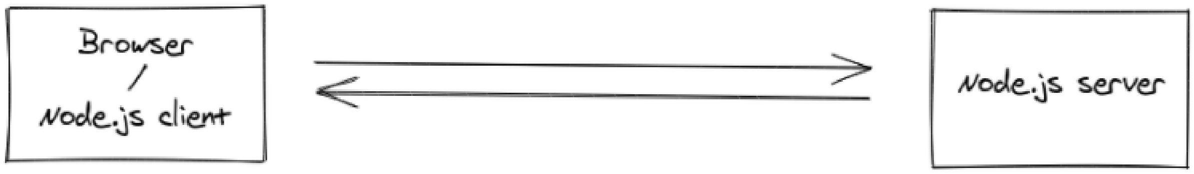
Node.js is a back-end JavaScript runtime environment that is open-source, cross-platform, and runs on the V8 engine. It executes JavaScript code outside of a web browser. Node.js allows developers to use JavaScript to create command-line tools and server-side scripting, which involves running scripts on the server to generate complex web page content before the page is submitted to the user's web browser. As a result, Node.js reflects a "JavaScript everywhere" paradigm, bringing web application development together around a single programming language rather than separate languages for server-side and client-side scripts.



Node.js is a scalable network server runtime that is configured as an asynchronous event-driven JavaScript framework. Many connections can be addressed at the same time. Thread-based networking is inefficient and complex to implement. Furthermore, since there are no locks, Node.js users are not concerned about deadlocking the operation.

II.V.II: Socket.IO:

Socket.IO is a Javascript Library that enables real-time, bidirectional communication between web clients and servers. It consists of 2 things: a Node.js server and a javascript client library for the browser.



In this the client will try to create a WebSocket connection if possible, and will fall back on HTTP long polling if not. So Socket.IO is not a WebSocket Implementation, Although it uses WebSocket as a transport when possible. It also adds additional metadata to each packet. You can consider Socket.IO as a “slight” wrapper around the WebSocket API.

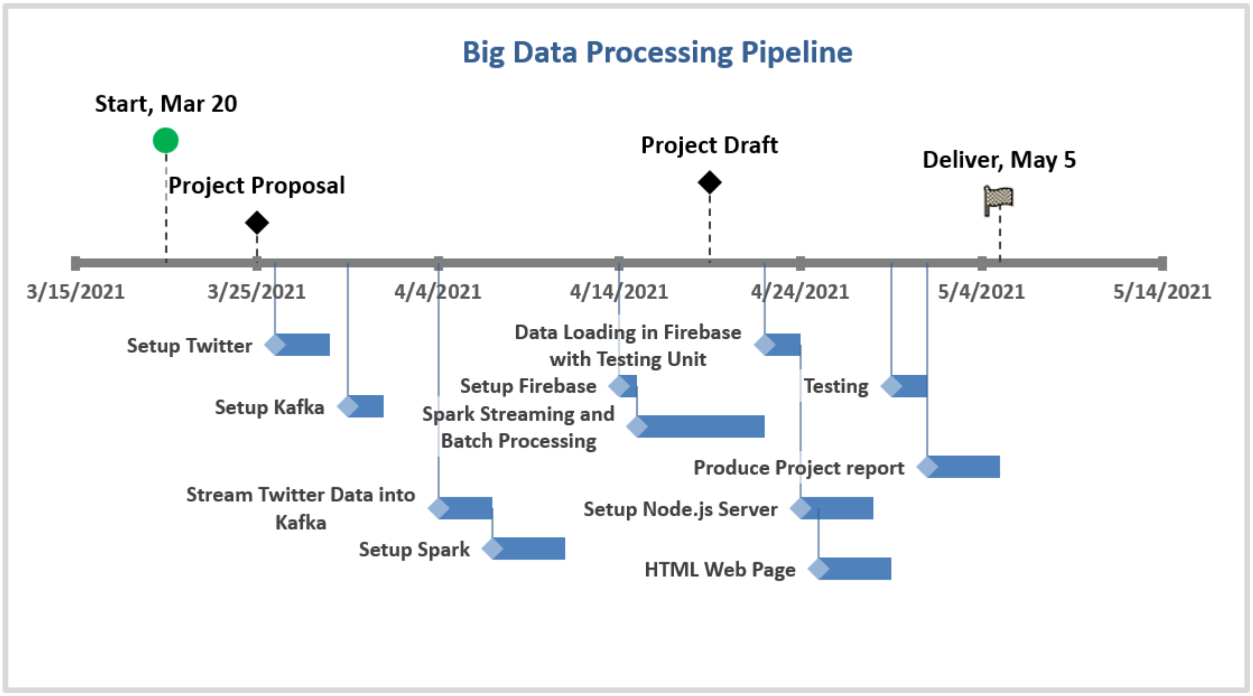
Apart from that Socket.IO provides different features such as Reliability, Automatic Reconnection, Packet Buffering, Multiplexing, Binary Support and a Simple and Convenient API.

SECTION-III: PROJECT PLAN & MILESTONES

Task Name	**End Date**	**Resources Names**	**Notes**	**Status**
Setup Twitter	03-28-2021	Vishnu, Yash	Create Developer Account, Project and App	Completed
Setup Kafka	03-31-2021	Varun, Vishnu	Installing Kafka on local machine	Completed
Stream Twitter Data into Kafka	04-06-2021	Harsh, Varun	Create Producer and Consumer in Kafka	Completed
Setup Spark	04-10-2021	Yash, Harsh	Installed Spark on local machine	Completed
Setup Firebase	04-14-2021	Harsh, Vishnu	Create Project and Setup Database	Completed
Spark Streaming and Batch Processing	04-21-2021	Yash, Varun	Process Twitter Data i.e. Deleting unnecessary columns, NULL Handling, Convert Data in Data model format	In-progress
Data Loading in Firebase with Testing Unit	04-23-2021	Harsh, Vishnu	Load processed data into Firebase	In-Progress
Setup Node.js Server	04-27-2021	Yash, Varun	Installing, Setup and Write server script	Future Task
HTML Web Page	04-28-2021	Harsh, Vishnu	Visualization and Build Reports	Future Task
Testing	04-30-2021	Yash, Harsh, Vishnu, Varun	Check the full data pipeline from data	Future Task

			intake to visualization	
Future Implementation	05-01-2021	Harsh, Yash	Insights originate from	Future Task
Big Data Processing Pipeline of Twitter Stream API	05-02-2021	Yash, Varun, Vishnu, Harsh		Future Task
Produce a Project Report	05-03-2021	Yash, harsh, vishnu, varun		Future Task

PROJECT TIMELINE



SECTION-IV: REFERENCES

The list of sources below is a recommended reading list. The knowledge collection mechanism may determine whether or not a reference is eventually relevant to this program, and to what capacity/extent. As a result, all references listed below may or may not be referenced. Additional sources may also be applied during the review process.

[1] https://developer.twitter.com/en/docs/twitter-api/tweets/filtered-stream/introductio
n

[2] https://developer.twitter.com/en/docs/tutorials/consuming-streaming-data

[3] https://dzone.com/articles/running-apache-kafka-on-windows-os

[4] http://kafka.apache.org/documentation/

[5] https://data-flair.training/blogs/kafka-workflow/

[6] https://phoenixnap.com/kb/install-spark-on-windows-10

[7] https://databricks.com/glossary/what-is-spark-streaming

[8] https://spark.apache.org/streaming/

[9] https://firebase.google.com/docs/database

[10] https://pypi.org/project/firebase-admin/

[11] https://firebase.google.com/docs/database

[12] https://firebase.google.com/docs/database/web/start

[13] https://nodejs.org/en/about/

[14] https://www.npmjs.com/package/socket.io

[15] https://socket.io/