Uncovering YouTube Insights: Crowd Source Systems for Effective Big Data Analysis

Anusha Yaramala[1], Jashia Mitayeegiri[1], Divya Sree Sandineni[1], Pavani Pabboju[1], and Bojja Hemanth[1]

[1]University of North Texas, Denton, TX 76205, USA

I. Problem Statement

The difficulty in managing streaming data efficiently emerges in the big data space, where massive volumes of data are created at unprecedented rates. The streaming platform paradigm presents challenges for traditional data storage solutions, which are frequently dispersed, because of the high cost of data input and querying. Streaming Analytics, a crucial instrument for deriving insights from dynamic data features, has challenges when utilizing current methods and technologies. This study's main goal is to design and build an analytical platform on top of streaming data to find hidden patterns and glean insightful information from the ever-changing streaming landscape.

II. Existing solutions

The creation of big data systems that can effectively handle "big data" and "fast data" for analytical purposes has drawn more attention in recent years. "Fast data" includes high-velocity, real-time or almost real-time data streams, such as system logs, impressions, click streams, search queries, and Twitter feeds. Services like breaking news reporting systems, which identify trending topics and breaking news in a matter of seconds by quickly analyzing Twitter feeds, are prime examples of this need. Another example of the need for fast within seconds identification of unusual search queries is Google's Zeitgeist pipeline. This section of the report mentions the existing solutions for the streaming data problems.

The research provides uniform framework for data processing that can effectively handle large and quick data for analytical uses. The framework should enable large-scale data and be scalable across several computers while maintaining user-specified latency limitations. The work addresses high latency in current Hadoop systems and presents an expanded architecture with processing and shuffling done by mini-batches. Workload studies in the real world demonstrate a 2x–5x increase in throughput and a decrease in average delay. When it comes to latency and throughput, the system performs better than top distributed stream systems like Storm and Sparking Streaming [1].

Stream-based applications rely on the Aurora runtime engine, which is built to manage continuous data streams. A QoS monitor keeps an eye on system performance, and the architecture consists of a router, storage manager, and scheduler. The catalog is essential as it contains statistics and network information. The paper examines scheduling algorithms with an emphasis on application-aware techniques for stream data management [2]. In addition to improving algorithms to suit application specific QoS expectations and offering an approximation strategy for balancing scheduling quality with overhead concerns, it stresses rigorous decision-making in parameters such as train size and super box traversal techniques.

The study investigates the necessity of modifying architectural designs to use the benefits of MapReduce for incremental one-pass analytics. It points out drawbacks in the traditional sort-merge technique for parallel processing and partitioning. To expand processing to workloads requiring a huge key-state space, the study presents a unique data analysis platform that uses hash algorithms for quick in-memory processing and a frequent key-based approach [3]. The Hadoop-based prototype decreases internal data leaks, synchronizes reduce work with map progress, makes significant progress on map jobs, and makes it easier to provide results continuously while a job is being executed. This platform will hopefully be expanded in the future to accommodate a wider variety of incremental computing workloads.

Big data is seeing a growth in the need for streaming computing models, especially for low latency uses like machine learning and real-time log monitoring. This is addressed by D-Streams, a revolutionary architecture for distributed streaming computing, which achieves fast recovery from errors and stragglers in sub-second timeframes without requiring resource-intensive replication. By using a novel technique of batching data into tiny time increments, it makes effective recovery techniques possible [4]. D-Streams provides fault recovery, sub-second latency, linear scaling to 100 nodes, high throughput per node, and support for a wide range of operators. It offers a flexible computing environment by integrating with batch and interactive queries with ease. Spark Streaming is used to illustrate this interaction.

Batch data analysis dominated the scene in the past, and data-stream and static data processing were handled as independent entities. On the other hand, a lot of use cases for large-scale data processing include continually generated data streams from sources such as application logs, web logs, and sensors. Existing configurations frequently process data without considering the timely and continuous flow of data, artificially batching these records into static datasets. The study presents Apache Flink, a flexible dataflow engine that can manage batch and stream analytics, to overcome these

constraints. Operator state and logical intermediate outcomes are treated as fundamental components in Flink, and the streaming API makes it easier to manage recoverable state and divide, transform, and aggregate data stream windows in an effective manner [5].

III. Comparative analysis

Presented a smart grid big data eco-system based on the Lambda architecture, managing huge smart grid data through batch and real-time processing(Munshi and Mohamed, 2018), after comparing this technique with previous research works. A Hadoop-based data lake for managing batch, real-time, or hybrid ingestion with technologies like NiFi and Kafka, processing data using Spark, and visualizing outcomes was covered by [6]. In [7] introduced an online system for real-time TCP performance monitoring that uses collectors, a stream processor (Spark Streaming), and a messaging system (Kafka).

Every strategy shows a different way to combine tools to meet certain data processing requirements. For real-time analytics, the YouTube technique combines PySpark, MangoDB, Kafka Streams, YouTube APIs, and visualization tools. On the other hand, [6] employ Hadoop, NiFi, Kafka, Spark, and visualization tools for general big data processing, whereas Munshi and Mohamed's Lambda architecture concentrates on smart grid data. For real-time TCP performance monitoring, Baojun Zhou's online monitoring system makes use of collectors, Kafka, and Spark Streaming. Every method adjusts its toolkit to meet the unique needs of the data source and processing objectives.

In contrast to the proposed strategy, this approach makes advantage of the MapReduce architecture to process and retrieve data in real time. By including specialized tools for a more thorough and successful study of YouTube data, the suggested approach provides a more advanced and effective analytical procedure [8].

The designs of the Dynamic Adaptive Streaming over HTTP (DASH) technique for YouTube video streaming and the YouTube data analysis methodology are dissimilar. Using tools like PySpark, MangoDB, Kafka Streams, and a real-time analytics dashboard, the YouTube data analysis technique offers comprehensive insights into top-performing videos, rivals, and channels. In contrast, DASH prioritizes adaptive video streaming according to connection bandwidth with the goal of minimizing bandwidth usage, particularly for partially seen videos. Both strategies seek to maximize the user experience [9]. DASH concentrates on adaptive video streaming, while YouTube concentrates on analytics and insights.

IV. Methodology Used

Five stages are included in the approach that is suggested in the paper to analyze data from YouTube to extract meaningful insights from the massive volume of video that is submitted to the platform. The first step concentrates on gathering metadata—such as title, channel name, and date published from YouTube APIs. This helps to comprehend the content and possible success indicators. To handle streaming data more efficiently, the second stage uses Kafka Streams to subscribe video objects into a topic. Because of its scalability and fault-tolerant nature, Kafka is a good option for managing high message volumes.

The third module uses PySpark, which is an open-source framework for large-scale distributed computations, to handle the Kafka Stream. The approach tackles the problem of ingesting streaming events from numerous producers to multiple consumers by merging Spark and Apache Kafka. The emphasis of the fourth stage is on keeping processed video metadata in Apache Cassandra, a NoSQL database in the research paper, however we have used Mango database to store the data due to its ease of use and adaptability to the dynamic nature of streaming data. High availability and the ability to subscribe to real-time changes in the database are ensured by MangoDB's scalable architecture, which is essential for managing the dynamic nature of streaming data.

The last module focuses on creating a live-streaming analytics dashboard using various python modules and Semantic-UI for both front-end and back-end tasks. The comprehension of different insights derived from the processed data is improved by the efficient real-time presentation of metrics and data. The methodology's complete approach, which can be applied to analyze YouTube video objects, incorporates many open-source big data technologies to manage high data velocity and scale data processing.

The suggested method includes gathering video metadata from YouTube's API, processing it using Spark, storing it in MangoDB or any other NoSQL database, using Kafka for effective streaming, and displaying the outcomes via a dashboard. The methodical integration of these elements shows a comprehensive approach to real-time analytics of YouTube data, offering insightful information to both academics and content makers.

V. Simulation and results

We will have to build a specific environment in order to have the right dependencies installed. Before we get the videos uploaded in a specific duration, we will have to get the categories in which YouTube puts the videos. We can get the category and the associated id by using the YouTube API video Categories () fucntion which takes the part of the video we are interested in also the region Code in our case it is US. The results are as follows:

Now we need have Kafka which is used to send the data to the consumer this can be done by initializing the zookeeper which is our resource manager. Though the command prompt we can use the '.\bin\windows\zookeeper-server-start.bat .\config\zookeeper.properties' command to start the zookeeper and '.\bin\windows\kafka-server-start.bat .\config\server.properties' to start the Kafka server. Firstly, the zookeeper should be running before the Kafka server is initialized. The initialization of both the servers is as shown.

We will now use the search function of the YouTube API to retrieve the videos of the specific window of time and the videos fucntion takes the video unique id and retrieves the

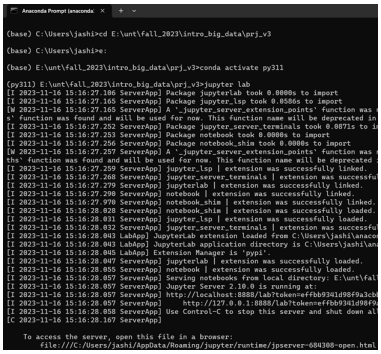


Fig. 1: Environment Activation

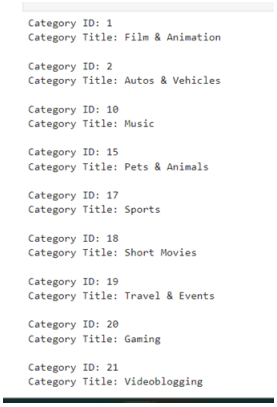


Fig. 2: Youtube Categories

likes, views and the duration of the specific videos. we create a method which called as 'get_videos_info()' which takes the id, category and the API key used to access the YouTube data. The ouput is as shown.

When we are retrieving the meta data related to the video, the youtube API v3 provides a dashboard which tells us the request generated and the quotas that user has exceeded. The following graphs are related to the traffic, latency and errors. From the graphs we can see that between November 14 and

Fig. 3: Zookeper Server Intialization

Fig. 4: Kafka Server Intialization

Fig. 5: Producer Sending Code

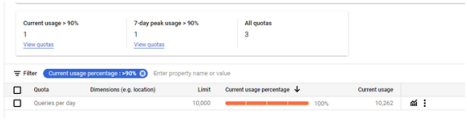


Fig. 6: Youtube Api Quota

November 15 we see that there are much more errors and the latency is also high. Each user is located with 10,000 requests per day, we can also monitor the quotas as shown, we see that it has exceeded 10,000 and an error occurs as the our quotas has ended.

After the Kafka takes the data from the YouTube API and the consumer which is our mongoDB takes the data using a consumer which subscribes to the topics of the producer. The consumer polls for 1 minutes if the producer doesn't send any data the consumer closes the connection. The data can be seen the mongoDB as shown.

When we perform the analytics with pypark a spark connection is create which takes performs the necessary analytucs much faster. The UI of the pyspark when it performed the analytics is as shown. From the pic we can see that the session has be created. The dataframe from the pyspark is generate by the schema.

A. *comparison*

We have collected data for 10 days related to the topics such as Gaming, Travel and Events, Sports, Film and Animation, People and Blogs, Pets and Animals, Autos and Vehicles, Music, entertainment and Comedy. And we see that the music category and film and animations have been the top most in

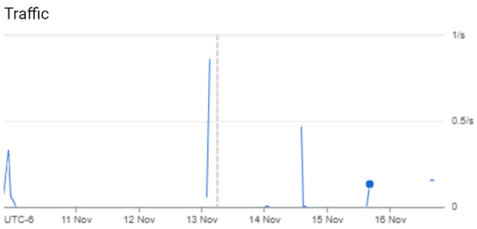


Fig. 7: Traffic

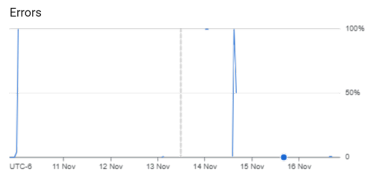


Fig. 8: Erros

Fig. 9: Latency

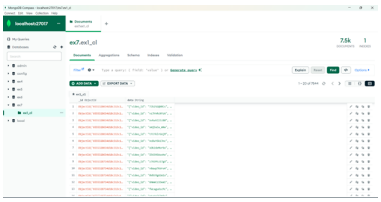


Fig. 10: Mongodb

terms of number videos uploaded in a specific time frame of 30 mins. Whereas in the research paper entertainment and music take the top two places.

The next graph focuses on the number of videos in each category and the duration of video in association with the category. From our results we see that the over 14 videos related to music have a duration of 30 minutes duration and the film and animations are of 40 minutes duration with 10 videos. We see that the top 5 of both the graphs there is entertainment and also the film and animation we make us to ensure that the trend is continuous with respect to music and film animation even after 1year since the paper was published. From the graph we extracted we can observe that there are many categories that fall in the region between 0 minutes to 150 and then later from 150 minutes to 300 minutes. This pattern may indicate that there is a concentration of videos with shorter durations, and as the duration increases beyond 150, the number of videos decreases significantly. It could imply that there is a common duration limit or preference for most videos within the dataset.

In the research paper analysis we see that the top 5 categories, both the entertainment and film & animations category

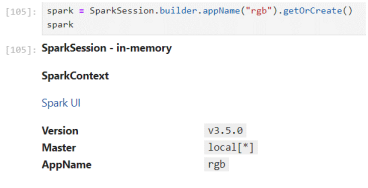


Fig. 11: Spark Session

Fig. 12: Apacha Spark UI

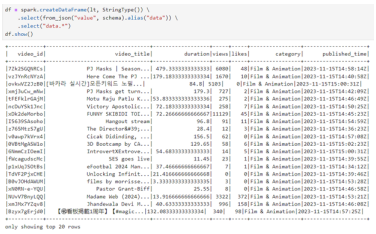


Fig. 13: Pyspark Dataframe

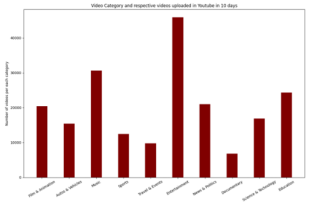


Fig. 14: Reference Paper Number Videos Per Category

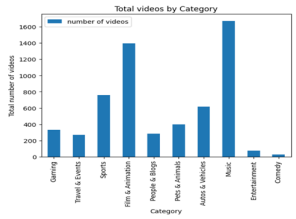


Fig. 15: Number Videos Per Category

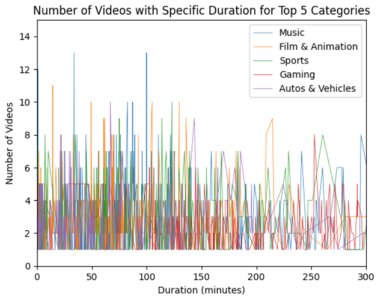


Fig. 16: Videos Per Duration

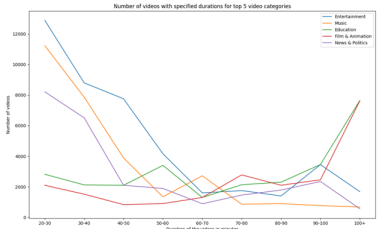


Fig. 17: Reference Paper Videos Per Duration

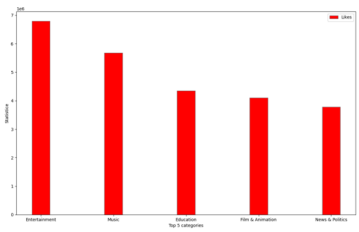


Fig. 4. Number of likes for the top 5 video categories

Fig. 18: Reference Paper Likes Per Category

category	total_likes
Gaming	404658.0
Travel & Events	12617.0
Sports	270221.0
Film & Animation	73923.0
People & Blogs	283448.0
Pets & Animals	47141.0
Autos & Vehicles	24096.0
Music	277702.0
Entertainment	22595.0
Comedy	962.0

Fig. 19: Likes Per Category

has the highest number videos in terms of duration and also the likes, whereas in our study we saw that even must number of videos in terms of duration have been the music and films & animation, but the likes are much more to the gaming and people and blogs which hints us that the duration has little relevance to the number likes in a category.

From the research paper we see that there are many more views to music and entertainment, but in our study we fond that the music has the highest number views but is not liked the most, as people may want to listen to music often so the views might go really high, but the likes are relatively low. The next most viewed is the sport category which has the almost same likes as the music. But when we see that the gaming category has the m0st like and the total views are almost half of the music category.

During out study we went ahead took the duration as one of the dimension across which we can quantify the YouTube videos, we found that the music category has been seeing the much uploaded videos content so the duration is also high

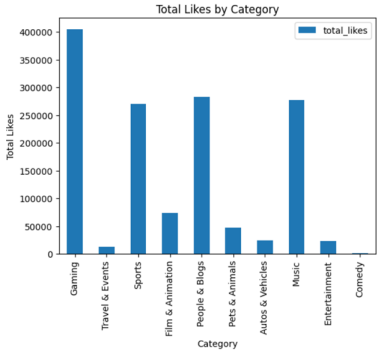
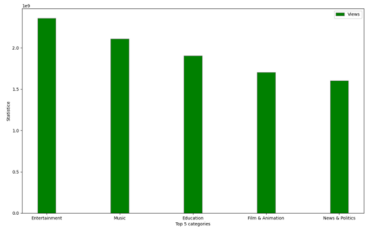


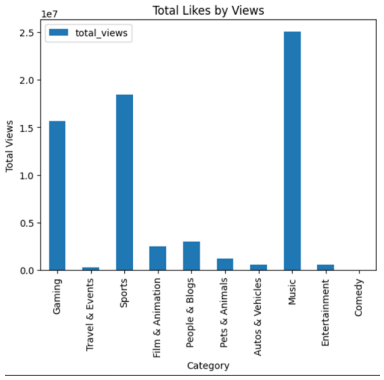
Fig. 20: Likes Per Category Graph



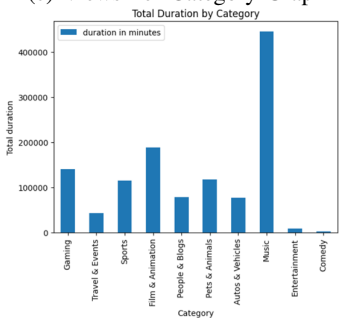
(a) Reference Paper Number Of Views Per Category

category	total_views
Gaming	1.5685554E7
Travel & Events	299286.0
Sports	1.8412784E7
Film & Animation	2466235.0
People & Blogs	2988548.0
Pets & Animals	1232465.0
Autos & Vehicles	540787.0
Music	2.5048487E7
Entertainment	596617.0
Comedy	27223.0

(b) Views Per Category



(c) Views Per Category Graph



(d) Duration Per Category

and is in the same way for the film & animation. The comedy category has relatives low number videos uploaded as well as the duration which is very relevant.

VI. Potential Applications, Pros, and Cons of the Method

There are several possible uses for the suggested stream-based meta-data analytics system for YouTube videos, along with several benefits and drawbacks. The field of YouTube video generation is one important application. Content makers may obtain useful insights into viewer preferences and create

videos that are more likely to engage and draw in viewers by examining metadata, including views, likes, and other metrics. In the highly competitive world of YouTube, where video creators want to improve their content strategy by understanding trends and attitudes, this knowledge is very important.

The system's adaptability in processing massive amounts of streaming data effectively is demonstrated by its capacity to crowdsource calls to YouTube's search and data APIs, process data using Kafka Stream and PySpark, and store processed video information in MangoDB. The system can handle enormous volumes of data and discover valuable patterns thanks to the usage of big data technologies like Hadoop and Spark. The real-time analytics tool gives content producers a way to monitor how their work performs over time and discover which categories and videos are most popular.

The suggested method's ability to benefit several industries outside of content development is one of its main advantages. According to the research, the architecture may be expanded for data processing applications in logistics, health, and space science. For instance, governments and businesses might get important insights by using YouTube data to analyze public opinion and responses to certain events. The framework's versatility presents chances for a wide range of applications, demonstrating its scalability and usefulness across several fields. But there are other issues and problems to consider. The cost of ingesting data and doing queries on it is mentioned in the study, highlighting the necessity of effective streaming analytics methods and tools. Additionally, there is a daily cap of 100,000 API queries because the system depends on YouTube's API [10]. Although experimenting with multiple time periods is mentioned in the study to overcome this issue, careful monitoring of the API requests would be necessary to guarantee ongoing and uninterrupted data collecting.

In conclusion, a thorough method for utilizing big data technologies for in-the-moment insights is provided by the suggested system for stream-based meta-data analytics of YouTube videos. Its versatility makes it a useful tool for identifying trends, attitudes, and patterns in streaming data. Its potential uses go beyond content production to a variety of sectors. However, while implementing and developing the system in the future, issues like API limitations and effective streaming analytics should be considered.

REFERENCES

[1] B. Li, Y. Diao, and P. Shenoy, "Supporting scalable analytics with latency constraints," *Proceedings of the VLDB Endowment*, vol. 8, no. 11, pp. 1166–1177, 2015.

[2] B. Babcock, S. Babu, M. Datar, R. Motwani, and D. Thomas, "Operator scheduling in data stream systems," *The VLDB journal*, vol. 13, pp. 333–353, 2004.

[3] B. Li, E. Mazur, Y. Diao, A. McGregor, and P. Shenoy, "A platform for scalable one-pass analytics using mapreduce," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, 2011, pp. 985–996.

[4] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica, "Discretized streams: Fault-tolerant streaming computation at scale," in *Proceedings of the twenty-fourth ACM symposium on operating systems principles*, 2013, pp. 423–438.

[5] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache flink: Stream and batch processing in a single engine," *The Bulletin of the Technical Committee on Data Engineering*, vol. 38, no. 4, 2015.

[6] R. Liu, H. Isah, and F. Zulkernine, "A big data lake for multilevel streaming analytics," in *2020 1st International Conference on Big Data Analytics and Practices (IBDAP)*. IEEE, 2020, pp. 1–6.

[7] B. Zhou, J. Li, X. Wang, Y. Gu, L. Xu, Y. Hu, and L. Zhu, "Online internet traffic monitoring system using spark streaming," *Big Data Mining and Analytics*, vol. 1, no. 1, pp. 47–56, 2018.

[8] T. Ashwini, L. Sahana, E. Mahalakshmi, and S. P. Shweta, "Youtube data analysis using hadoop framework."

[9] D. K. Krishnappa, D. Bhat, and M. Zink, "Dashing youtube: An analysis of using dash in youtube video service," in *38th Annual IEEE Conference on Local Computer Networks*. IEEE, 2013, pp. 407–415.

[10] J. M. Reddy, A. Attuluri, A. Kolli, N. Sakib, H. Shahriar, and A. Cuzzocrea, "A crowd source system for youtube big data analytics: Unpacking values from data sprawl," in *2022 IEEE International Conference on Big Data (Big Data)*. IEEE, 2022, pp. 5451–5457.