K-CS259: Data Analysis Using Apache Kafka

Riti Gupta
Computer Science Department
San Jose State University
San Jose, CA 95192
408-924-1000
riti.gupta@sjsu.edu

ABSTRACT
In today's world, data is present everywhere in various fields which needs to be analyzed and processed to extract necessary information from it. Since the size of data might be huge, the storage and processing powers has always been a concern. With the latest development in technology, both hardware and software, we can enhance the latency and throughput of the applications processing big data. Hardware level parallelism (instruction level pipelining, etc.) have already reached its limits and it is time that we trap the software level parallelism benefits by running the application by running multiple threads or processes in parallel on different cores of the machine or on multiple machines. As a part of this project, I worked on two big-data applications. One was to count the number of Warning, Error and Info messages in the application logs. The second application was to analyze the data stream of tweets present on twitter website and count the number of messages in the relevant categories over a period of time. The parallelism has been achieved using big data techniques like Apache Kafka which has producer-consumer as the underlying mechanism. The producer reads the data stream and consumer keeps processing it as per the requirements improving the latency and throughput of the application.

Index Terms: Map-reduce, Apache Kafka, Apache Storm, Hadoop, Broker, Partition, Zookeeper, Publisher-Subscriber, Scalability, Fault-tolerant

1. INTRODUCTION
There are lot of areas where the data size is huge and should be executed in parallel to improve the performance of the application. I worked on applications related to analyzing the user logs and also analyzing the twitter data stream. Following is the brief description of both the applications.

1.1 Logs Analysis:
Producer keeps reading the logs and consumer processes them counting warning, error and info messages for each category.

Snapshot of logs:
[main] Producer - Put value: ERROR, for key: user76499
[main] Producer - Put value: INFO, for key: user76500
[main] Producer - Put value: WARNING, for key: user76501
[main] Producer - Put value: ERROR, for key: user76502
[main] Producer - Put value: ERROR, for key: user76509
[main] Producer - Put value: ERROR, for key: user76578

Expected Output:
The application counts the logs over a period of time as follows.
(WARNING,8153)
(ERROR,8018)
(INFO,8089)

1.2 Twitter Data Stream Analysis:
Twitter tweets have been analyzed as a part of this project. Tweets are short messages shared by the twitter users. These tweets are public and are available to be used for various exploratory projects related to data analytics and machine learning. The data can be obtained by registering on Twitter website for developer access and getting relevant keys and tokens for using with the application. The aim of the project was to count hashtags corresponding to specified keywords.

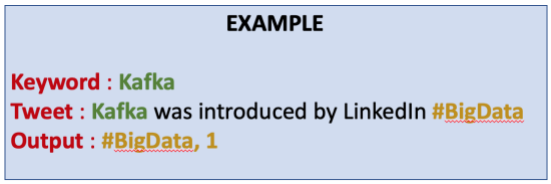

Figure 1. Example of twitter stream data analysis.

In Figure 1. above, the tweets related to keyword, "Kafka" are processed by the consumer. The hashtags in these messages are collected and counted. In the above example, there is only one hashtag, '#BigData' and hence the count is 1.

Following is the snapshot from my application to give a real example. Let us say, we want to process and analyze tweets related to the following operating system keywords.

keyWords = {"linux", "ubuntu", "windows", "unix"};

The tweets can be received for these keywords by connecting to twitter data stream. Following is the sample snapshot of the twitter data stream that would be received with above keywords.
Hashtag: ***gnu***

*Tweet:StatusJSONImpl{createdAt=Thu Nov 07 08:23:04 PST 2019, id=1192477578675638272, text='RT @schestowitz: #**Ubuntu** #Podcast from the #**UK** LoCo: S12E31 – Ikari Warriors https://t.co/si8xApIWyo #gnu #**linux**'*

Hashtag: ***linux***

Tweet:StatusJSONImpl{createdAt=Thu Nov 07 08:23:12 PST 2019, id=1192477610330247168, text='RT @MrThomasRayner: ***#MSIgnite*** *this book might change your life. Or at least help you run MUCH more secure infrastructure & systems. #JEA fo…'*

As a part of this project, the count of each hashtag received would be computed. Following is the sample output snapshot that is expected to be received.

(#101thingsthatmakemesmile,2)

(#edgecomputing?,1)

(#UK,5)

(#Ubuntu,5)

(#linux,5)

(#gnu,5)

(#Podcast,5)

2. BACKGROUND

2.1 Kafka Mechanism

For this project, Apache Kafka has been used for analysis of data for both the application. Apache Kafka is an open-source platform developed by LinkedIn to analyze the user activity. It is based on publisher-subscriber mechanism where the publisher reads the data and puts it into kafka brokers. The subscriber reads the data from the brokers and processes them as per the requirements. The subscriber can be either databases, apache spark, apache ksql, machine learning tools, etc. Kafka has many other benefits apart from being able to execute in parallel like being fault tolerant and horizontally scalable which shall be discussed in later parts of the report.

Apache Kafka is based on Producer-Consumer mechanism. Figure.1 below is the general diagram of Producer-Consumer methodology where the producers P1, P2 and P3 produce the data into the buffer and consumers C1, C2 and C3 read the data from the buffer in parallel.

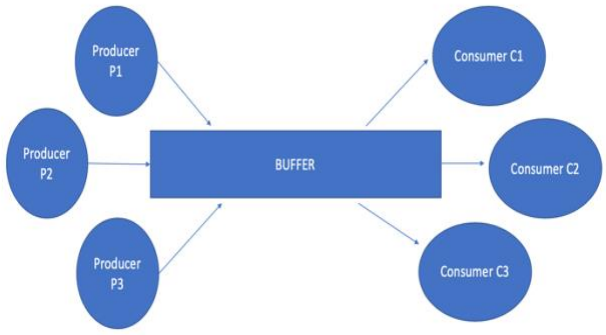

Figure 2. Producers and Consumers accessing buffer in parallel.

It is also known as Publisher-Subscriber mechanism in which the publisher publishes the data and subscriber subscribes to the data it wishes to process.

The above mechanism mentioned in Figure 2. needs to deal with the synchronization. The producer would be able to produce the data only when there is space in the buffer. Similarly, Consumers can consume only when the data is available inside the buffer and not when it is empty. It also needs to deal with mutual exclusion problem as all the producers and consumers should access the location on a buffer without interference from each other.

Figure. 3 below is a real-world scenario based on ecommerce site. Different product lines write different data to different publisher/subscriber buffers and consumers read from them.

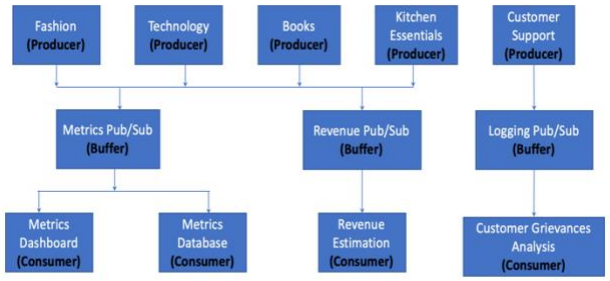

Figure 3. Real-World Example of Publisher-Subscriber.

In Figure. 3, three queueing infrastructures need to be maintained corresponding to each buffer. Apache Kafka provides a mechanism to have just one such infrastructure avoiding the overhead of maintaining multiple infrastructures.

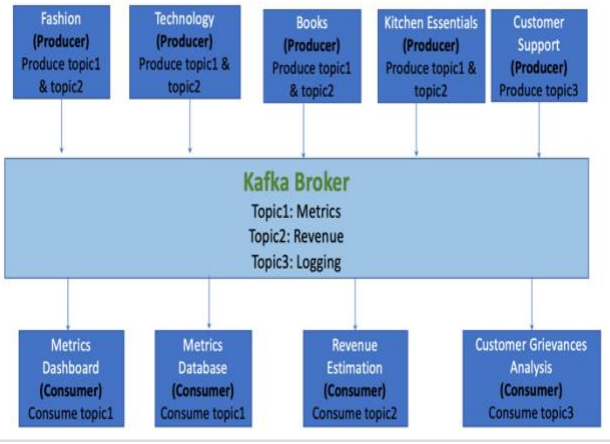

Figure 4. Publisher-Subscriber Using Kafka.

In Apache Kafka, Producers along with the data send the topic name as well to the broker. In Figure. 4, producers Fashion, Technology, Books and Kitchen Essentials generate information on topics Revenue and Metrics. The consumers as per their requirements subscribe to different topics and consumer from the broker. In Figure. 4, Metrics Dashboard and Metrics Database subscribe to Metrics topic and process the information as per their roles. One key point to note is that in Apache Kafka, producers

and consumers do not interact with each other directly. Kafka broker acts as a medium of interaction for producers and consumers to interact with each other.

2.2 Kafka Internals

We can further improve the degree of parallelization by having multiple partitions for each topic as shown in Figure 5., in which topic 'A' has five partitions. In case of heavy load or traffic, multiple producers and consumers can access the data on different partitions. The partition to which data should be sent is decided by kafka partitioner. The partition number is usually decided based on the load of the partition and the key of the data.

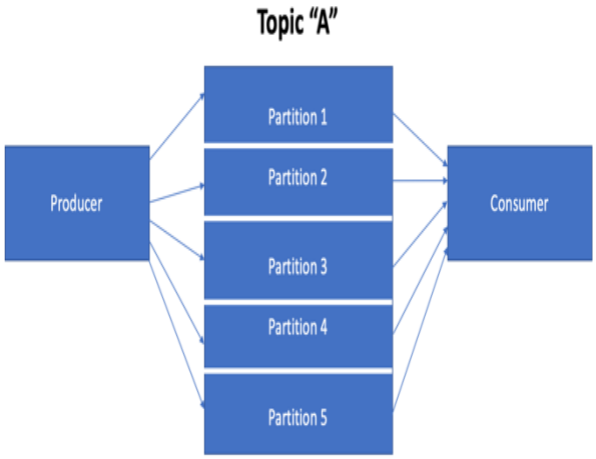

Figure 5. Parallelism improved by adding more partitions.

The data remains ordered within each partition which means that the consumer would see the data in similar order as it was sent by the producer on a particular partition but, there is no order guarantee across different partitions even for the same topic.

There can also be a group of kafka brokers functioning together as a cluster. The cluster has one broker assigned as a controller that perform administrative functions like assigning the partitions, data replication, etc. This helps in scalability when data size increases as multiple producers and consumers can function at a much higher speed by accessing different machines. This would also help in making the application fault tolerant as the data can be copied on multiple machines avoiding single point of failure.

In order to run Kafka, the application known as Zookeeper is needed. It is an administrative tool for Apache Kafka to track information related to Kafka topics, partitions, broker and clusters. It is highly available and consistent due to its criticality for Kafka.

The data that is present in the broker is not erased as soon as the consumers read the data. Kafka application developers can specify various data retention policies like storing the data for the specified time interval or when the space reaches certain limits. This has various advantages like maintaining the consumers. For example, the consumer can reboot as the data would already be present on Kafka brokers.

Kafka brokers send acknowledgement to Kafka producers when the data is received. In case of errors, producers receive an exception. If written successfully, the producers receive the location of the data which generally includes the partition and offset number.

Kafka consumers keep sending heartbeats to the broker to signal that they are functioning properly without any issues. This helps Kafka brokers in assigning the partitions to the consumers that can handle the data.

Kafka brokers need to write data in the format uniform for all the producers and consumers. Kafka brokers see's each data simply as an array of bytes. The application developers can also specify the serialization techniques to write data on Kafka brokers.

Kafka can be used in variety of applications ranging from machine learning, managing databases, message passing, dashboard application to many more. It is easy to configure and build applications on top of it in multiple languages like Java, Scala and Python. It is being widely used in the industry for various tasks.

3. IMPLEMENTATION DETAILS

For experimenting with the projects, I used two threads. One thread was for producer and other for the consumer. The number of threads can be decided based on the application being developed and the load it is expected to receive. There are various mechanisms to verify the process ids of producers and consumers being run. I verified the process ids using 'ps -eaf' command. The machine on which I performed the experiments had four physical cores and eight logical cores. I used Java for the project.

In order to run Kafka, we need to run zookeeper and also load the configuration file which contains information related to location of the logs, data retention policies, replication policies, zookeeper connection string, etc.

For both the projects, the consumer processes the data using map-reduce which is again a distributed methodology and manages the data passing between various processes along with providing scalability and fault tolerance. The producer reads the logs data or the twitter data stream logs and sends it to the consumers. The consumer filters the required words needed for map-reduce, hashtags for twitter data stream and message type (INFO, WARN, ERROR) for logs analysis. These words are then processed using map-reduce to generate the required statistics from them.

4. CONCLUSION

There are lot of big data technologies available which should be explored while developing the applications. They make the application to run faster by running them in a parallel manner as multiple processes. These technologies can be decided by the requirements of the application. The criteria might be the load of

the application or the processing times that is expected based on the criticality of the application. There are various technologies available which can be used for real time processing or batch processing. Some of them are Apache Kafka, Apache Spark, Apache Storm, Hadoop, etc. The application should be parallelized whenever it is possible to do so. One should also take care of avoiding under-utilization and over-utilization by running the application with appropriate number of threads. If too many threads are used for an application with very less data, most of the time might be spent in maintaining the threads instead of doing actual processing. There are lot of open-source platforms available where everyone should contribute and help other developers.

5. REFERENCES

[1] J. Bang, S. Soon, H. Kim, Y.S. Moon and M.Choi, "Design and Implementation of a load shedding engine for solving starvation problems in Apache Kafka," in IEEE Int. Conf. on Networks Operations and Mgmt. Symp., 2018.

[2] P.L. Noac'h, A. Coston and L. Bougé, "A performance evaluation of Apache Kafka in support of big data streaming applications," in IEEE Int. Conf. on Big Data, 2017.

[3] R. Shree, T. Choudhury, S.C. Gupta and P.Kumar, "The modern platform for data management and analysis in big data domain," in IEEE Int. Conf. on Telecommunications and Networks, 2017.

[4] B.R. Hiraman, C.V. M and C.K. Abhijeet, "A Study Of Apache Kafka in Big Data Stream Processing," in IEEE Int. Conf. on Info., Communication, Eng. And Tech., 2018.

[5] H.Wu, Z. Shang and K. Wolter, "Performance Predication for the Apache Kafka Messaging System," in IEEE Int. Conf. on High Perf. Computing and Communications, 2019.

[6] "Kafka Tutorial", Apache Kafka. [Online]. Available: https://www.oreilly.com/library/view/kafka-the-definitive. [Accessed: 21-Nov-2019]

[7] "Introduction," *Apache Kafka*. [Online]. Available: https://kafka.apache.org/intro. [Accessed: 21-Nov-2019].

[8] S. Thirumala, "Spark streaming part3: Real time twitter sentiment analysis using kafka "stdatalabs,", 04-Aug-2018. [Online]. Abailable: https://stdatalabs.com/2016/09/spark-streaming-part3-real-time/. [Accessed: 21-Nov-2019].