Conestoga College

School of Applied Computer Science & Information Technology

SENG8080 – Case Studies Big Data

Apache Kafka

Gurjyot Anand (25%)

Divya Chandwani (25%)

Santhra Margarat Alex (25%)

Foram Bhavsar (25%)

June 21st, 2019

Abstract

Being able to gather continuous, real-time data in a timely manner is of great importance in any industry today. We need a reliable, robust and fast messaging system like Kafka, to transfer data originating from constant data producing sources to data consuming recipients. Being a highly scalable, and responsive distributed computing platform makes it a great area of interest for any big data enthusiast. Kafka is used by business organizations as a distributed streaming platform for creating real-time data feeds. It is being used by the big industry giants like Twitter, LinkedIn, Uber, Mozilla and many more. In this report, we will take a technical dive into Kafka and understand how it is used in different applications.

Table of Contents

Introduction

In the world of big data, a huge amount of data is required for the analysis. Collecting this huge amount of data and then analyzing this data to get some valuable information are the challenges we face in big data. To overcome these challenges, we require a system to transfer data from one application to another.

Apache Kafka is a messaging system that provides a platform to send real-time data, over a continuous stream. The data transfer can be visualized as messages sent on the stream by the data source(s), and voluntarily fetched by the data consumer(s). This solves the problem of accessing and analyzing the data generated at a high velocity.

What is a Messaging System?

A messaging system provides reliable communication between two or more applications, allowing them to focus on the intended jobs instead of sending or retrieving the data. There are basically 2 types of messaging systems;

Point to Point Messaging System

In this type of messaging system, messages are placed in a line. Receivers can retrieve any number of messages they want. However, a message can only be retrieved by only one receiver. As shown in the picture below, the sender sends the message, the message is placed in the messaging queue and later-on it is sent to the retrieve

Figure 1- https://www.tutorialspoint.com/apache_kafka/apache_kafka_introduction.htm

Publish-Subscribe Messaging System

In this type of messaging system, the messages are stored in topic(s). A receiver can retrieve messages in a particular topic. Each topic can have one or more messages stored in it. Once a receiver retrieves messages from a topic, all the messages placed in the topic are sent to it.

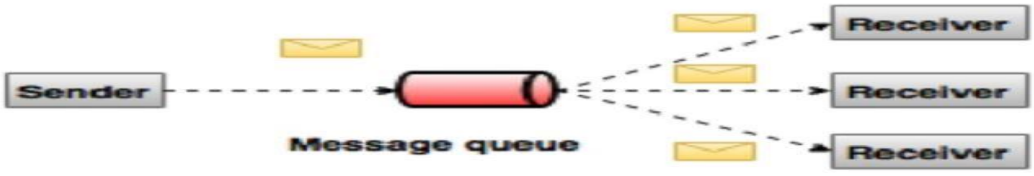

Figure 2- https://www.tutorialspoint.com/apache_kafka/apache_kafka_introduction.htm

Overview

Kafka was initially developed by LinkedIn in order to overcome the growing complexity along with its increasing number of users during the year 2010. It was kept under the Apache incubator until 23rd October 2012, even after being open sourced in 2011. Although Kafka clients are currently provided in multiple programming languages, the original source code was written in Scala and Java.

Apache Kafka uses the publish-subscribe messaging system. It can be used for both offline message consumption as well as online message consumption. It is built on the top of technology called Zookeeper Synchronization services.

Kafka majorly uses 2 APIs, the Kafka Connect and Kafka Steam API's. Kafka Connect API is used to import or export data from one computer to another and vice- versa. Kafka Stream API library is used for stream processing which is written in java. It allows the development of stream processing applications that are scalable, elastic and fault-tolerant.

Kafka integrates very well with technologies such as Apache Storm and Apache Spark. It is responsible for making the data available for all the users and applications by retrieving data from an existing topic. The data retrieved is analysed and then this data is passed to a new topic. Kafka is also used to retrieve logs from multiple applications and services, then convert those logs into a standardized format, which further can be accessed by other applications.

We can say that Kafka is the perfect substitution for a normal messaging broker because it is fast, scalable, fault tolerant, designed for high-performance systems. Also, its replication system ensures that the data is safe. Kafka can be used as a topic as well as a queue. Furthermore, Kafka has a built-in partitioning, hence it's an advantage as compared to other messaging system.

Apache Kafka is currently used by the following organizations:

Twitter - Twitter is the most famous online social networking service that uses Apache Kafka. It is basically a platform that sends and receives user tweets. Twitter demands the user to sign up or register to post tweets. But it does not restrict an unregistered user from reading tweets. Storm-Kafka is utilized for its stream processing infrastructure.

Netflix - Netflix is an American media-service provider who does real-time monitoring. They make use of Kafka for this and event processing as well. Netflix uses the Keystone pipeline that asynchronously generates messages without blocking any applications.

LinkedIn - LinkedIn is undoubtedly, one of the best applications of Apache Kafka. A variety of LinkedIn products such as Newsfeed, LinkedIn Today perform online message consumption and activity stream data with the help of Kafka. Apart from that. Hadoop which is an offline analytics system is also a consumer of Kafka. Kafka plays a major role in operational metrics functions of the LinkedIn. Strong durability offered by Kafka is a prominent reason for LinkedIn to rely on this service.

Mozilla - Mozilla is a libre software community developed by Netscape. A replacement of a part of its production system by Kafka is expected to come soon for collecting user data and also performance from end-user browsers for architectures like Test Pilot and Telemetry.

Architecture

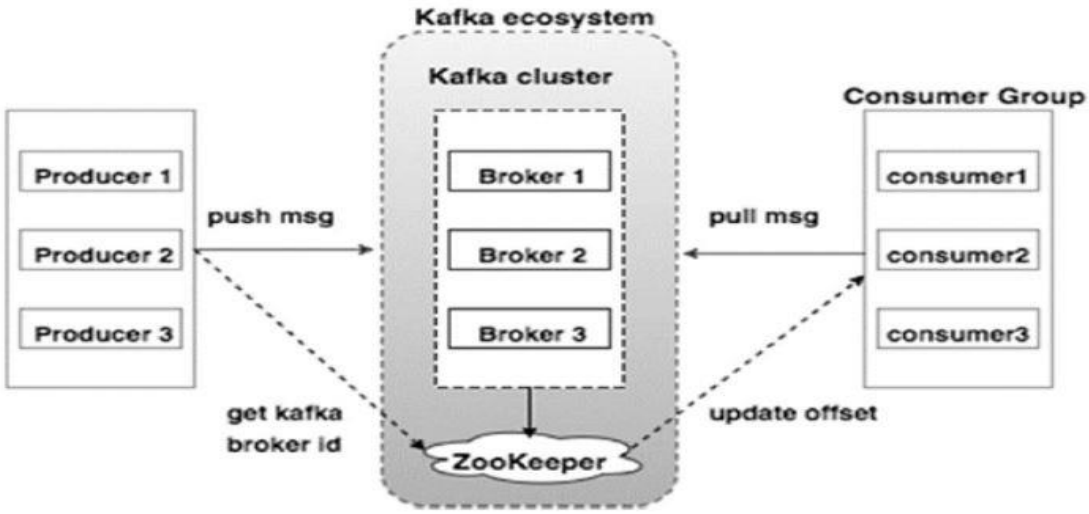

Figure 3- https://www.tutorialspoint.com/apache_kafka/apache_kafka_cluster_architecture.htm

There are some terms that helps in understanding the basic schema of Kafka;

Messages

An array of bytes that are being transferred in Kafka are called Message. It is a main element of Kafka. It can also be referred as a key/value pair. This is similar to a row in any database. A message may also have a key in it. Key is a bit of metadata. When we have to write messages to partitions in an efficient way, keys are used.

Batches

A collection of messages that are being generated to the same topic/partition are called Batch. Sending individual round-trip of every message in a network would result in overhead, but sending them in a batch will reduce this overhead. Batches also provides efficient data-transfer.

Schemas

Kafka requires a consistent data format, so it's easy to read and write messages. Schema provides a structure to the messages, so that they can be understood easily. There are various types of schemas that we can use such as JavaScript Object Notation (JSON), Extensible Markup Language (XML) and Apache Avro. JSON and XML are easy to implement and can be understood easily by humans, but are not compatible between different schemas. Many of the Kafka developers use Apache Avro, as Apache Avro provides features such as compact serialization format.

Topics

A topic is a category in which messages are published. Messages in a topic belong to a same category. The process of sending a message in Kafka is you need to send it to a certain topic. Similarly, if one wants to read messages, they can read it from the same specific topic. A topic can be consumed my many consumers, there is no limit to it. Topics have a unique name. The message remains in the cluster till the TTL (time to live) is reached.

Partitions

Each Kafka topic is further divided into partitions, there can be one or many partitions in a topic. Messages are stored in partitions in a sequence which cannot be changed and the sequence is FIFO (first in first out). A unique location is assigned to every message in a partition, this location is known as it's offset. Increasing the number of partitions will increase the parallelism. Furthermore, each partition can be hosted on different servers, that helps the topic to be scaled horizontally among various servers and provide better performance than a single server. Partitions helps Kafka to provide redundancy as well as scalability.

Producers and Consumers

Kafka users are basically of two types; producers and consumers. Moreover, there is also Kafka Connect API and Kafka stream, which are known as advance clients of Kafka. These advance clients give higher level functionality at the top of consumers and producers.

Producers produce messages. These are also known as publishers and writers. Mostly the producers only choose the topic where they want to publish the message, but if they also want to choose the partition, they can also do that using the message key. Producers can assign a same value to all the keys and they will be redirected to a same partition. Choosing the partition will not balance the topic with messages evenly.

While producers produce messages, the consumer reads them. Consumers are also known as subscribers and readers. The consumer can subscribe to many topics and can read the messages in them. The messages are read in the order they were produced and with the help of offset the consumer knows which messages are read and which are not. Consumers can also be organized into consumer groups. Each consumer in the consumer group reads from a unique partition, doing that the whole group reads every message from the entire topic. The mapping of a consumer to a unique partition is called ownership. In some cases, you have a greater number of consumers than the partitions in the topic, so some consumers remain idle because they have no partitions to read from.

Brokers and Clusters

Kafka broker is also known as Kafka server and Kafka node. Kafka broker receives messages from Kafka producers, assigns them with a unique offset and store them on the disk. Brokers also allows Kafka consumers to retrieve the messages that are stored in specific topics, partitions or offsets. A single broker is capable of handling thousands of partitions/messages per second.

Kafka brokers are capable of creating a Kafka cluster by sharing data and information amongst them. In the cluster of brokers, one of the brokers will function as a controller, i.e. responsible for operations such as monitoring brokers, assigning unique partitions to the brokers. Each partition is controlled by a broker and that broker is the leader of that partition. A partition can be also assigned to many brokers, resulting that the partition gets replicated many times.

Kafka has a feature retention, which means the data is stored for a period of time. So, Kafka brokers have a default retention period for topics, either it is for some amount of time or the topic reaches a size limit. Once it reaches these limits, they get deleted. We can also set retention period, according to us, we can keep the messages, till they are useful.

Multiple Clusters

Having multiple clusters in a growing Kafka network is very useful, because of the following reasons; Segregation of types of data, isolation for security requirements, multiple data-centres. There are three types of multiple cluster deployment models in Kafka and they are stretched cluster, active-active cluster and active-passive cluster.

In stretched cluster, there is only one logical cluster, which consists of many physical clusters. This model has a strong consistency because of the co-existing data duplication between the clusters. Moreover, Clusters are used to the full extent and the clusters are not dependent on each other, so if a cluster fails other tend to run normally without any downtime. But this model has high latency due to data duplication between the clusters so it’s recommended to use this model only when clusters have very high network bandwidth.

In active-active cluster, there are two clusters in it, both of them have bi-directional mirroring between them. The data is transmitted in both directions within the clusters. In this model, the client’s application is aware of many clusters, so if a cluster fails, they know which one to move on to. The benefits of this model are, resources are utilized to the fullest extent in both the clusters, there is no downtime if a cluster fails and there is no relation between network bandwidth and performance. But this model lacks consistency as there is mirroring of data between the clusters.

Why choose Kafka over other messaging systems?

Kafka has an upper hand when compared to other messaging systems, lets look at a few points that develop this idea-

Multiple Producers can write data onto a topic(s) through the Kafka broker.

Kafka smoothly handles multiple producers writing data into the same topic, saving the consumers time and resources for aggregating data coming from multiple streams. For example, consider that a page-views analysis needs to be conducted, for pages on multiple websites. Each website (acting as a producer) can write data into one topic for page views. The consumers can then easily read this data, eliminating the need for processing data coming from multiple topics.

Multiple Consumers can read data from a single stream/topic.

Kafka allows more than one consumer to read the same message from a stream. This is advantageous when the same data is being used differently, it is possible for it to be read by two different applications. This solution is not possible in conventional messaging systems where a previously read message is not made available to other consumers. We can also visualize this as, and each message (piece of data) is processed only once, by a consumer in a consumer group having multiple consumers, again saving time and resources.

Flexible & durable disk-based retention of the continuous data stream.

Disk based data retention means that the continuous data sent on the stream is not lost even when a consumer(s) are down. The policy used for specifying data retention is flexible for every topic, it allows different streams to have different data retention rules depending on the business requirements. Such a data retention policy is advantageous when the consumers are face difficulty in reading data that is published in occasional bursts, or even while performing maintenance on the consumers. When a consumer(s) is down the data can be retained on disk by the Kafka broker until the consumer comes back up and starts from where it left. This prevents loss in precious near real-time data.

High scalability to accommodate changing requirements of organizations.

Kafka makes it very easy for organizations to scale up when the amount of data being produced is multiplying each day. This can be achieved by simply expanding a single Kafka broker to a cluster of brokers, as and when the data grows. Another upside is that, these expansions do not affect the current operations, because they can be carried out even when the broker(s) is online. Scalability also comes with fault tolerance, the data is replicated over the cluster, and the failure of a single cluster can be recovers by other nodes in the cluster. One can always set a higher replication factor in clusters where more failures need to be configured, or if the data handled by the cluster is more critical.

Kafka provides a high performance, and highly available, fault tolerant system even in high data load.

All the above discussed features of Kafka, make it possible to send high volumes of data within sub-seconds, in a reliable manner. Each of the it's components (producer, brokers and consumers) can be scaled up in order to manage high data volumes as and when required, without compromising much on the performance.

Apache Kafka provides a backbone to the data ecosystem, by integrating different applications.

Kafka eliminates the need for having tightly coupled producers and consumers (no direct communication needed between these), as it acts as a communication channel between them. This is of great advantage to dynamically changing organizations since it is easy to add or remove components from the system without having to worry about the data transfer and standardization of data.

Kafka Use Cases

Activity tracking

Kafka was developed with the purpose of achieving this use-case. Activity tracking is nothing but recording the actions performed by the users on a particular application. This is used for the purpose of monitoring tasks performed by users, and drawing some profitable business insights from the collected data. Information related to user activity like page views, information added by users, click tracking etc. can be published by the front-end application into one or more topics, which is then consumed by the backend applications. The data sent through these streams can help in near-real-time analysis for generating reports related to important metrics, or even help in providing insights for taking necessary actions in order to improve the user experience.

Messaging

Applications sending push notifications to users can also use Kafka for sending these messages. This will ensure that a single application collects the messages from the senders, formats them, and routes them to the receivers. Kafka can not only format the messages sent, but also Aggregates multiple messages into a single notification. It also helps in applying the user's preferences related to how often they would recover the notifications. Using Kafka in such applications eliminates the needs of having duplicate functionality in multiple front-end applications.

Metrics and Logging

System logs and metrics are the most important in today's technology world. These are not only used to track activities, but there is a lot of analysis can be performed on this kind of data. Kafka can be used for collecting system logs and metrics generated by applications, this is done by having the applications publish these logs into a Kafka topic(s). These logs and metrics can be routed to security applications, or applications like Elasticsearch that are used for log search.

Commit log

As we know that Kafka is used for real time communication in distributed systems, these bring along their own challenges related to node failures, data re-syncs in failed nodes, data replication across multiple nodes and so on. To improve these issues, Kafka can be used in maintaining an eternal commit log that supports in data replication, and resyncing data when node failures occur. This is done by creating a stream where all database changes/node failures are published. In this way all the nodes of the distributed system are informed of the changes and can take actions accordingly. Solution architects also design this with a durable retention policy where a buffer stores the changelog. This data stored in the buffers can be re-visited in case the consumers fail to read the data. Another way of optimising this application is that the topic storing the changelog only stores the latest operation for every key.

Streaming Processing

Kafka is widely known as an application that supports stream processing. This is because almost all of its operations are used for transferring messages in real-time over a stream. Stream processing applications are the ones that allow operations on real time data. Operation performed on this data depends on the business requirements. However, in such stream processing applications, Kafka only holds the role of transforming the data over the streams, the transformations and data processing are by other applications like Apache Storm, Apache Flink, and Apache Spark. The key here is all these stream processing applications rely on Kafka for reliable real-time data delivery

Kafka's GIT Repository

Apache Kafka source code is available in GitHub repository in Apache/Kafka repository. There are a lot of GitHub repositories available for Confluent supports for Kafka. This include KSQL, schema-registry, Kafka-rest, Kafka-streams example. The main contributors of Apache-Dev/Kafka repository are Jay Kreps, Rui Wang, Jun Rao, Neha Narkhede, Fatih Emekci, Lin Guo, Shirshanks Das, Roshan Sumbaly, Sam Shah and Chris Burroughs. There are 554 contributors to Apache/Kafka and to give any contribution to Apache/Kafka, it is mandatory that it is original work and license the work in open source. On submission of any copy righted material, it is believed to be licensed under project's open source license. Becoming a contributor to Apache/Kafka is possible even if the candidate does not know Java or Scala. Documentation supporters are also welcomed to be a contributor of Apache/Kafka. For contributing a code change, it is expected to have some unit tests for the newly introduced functionalities. There may be many simple tickets in JIRA which can be used by a beginner to start working with the project. By solving these issues, one can enter contributing to Apache/Kafka.

Apache Kafka uses JIRA for tracking bugs. To file a log, we need to visit Apache JIRA and set the issue type, title of the issue, set component field and log the issue. If it is a developer who wants to solve a bug, he/she must use JIRA ID in the Assignee field. But for that they must be in the contributors list which can be made through contacting Kafka team.

While logging incident, it is important to ensure that the project is set to Kafka and identify issue type. There are five types of issues;

- Addition of New Feature
- Improvement of a New Feature
- Test
- Wish
- Task

But if the motive is to make changes onto an existing ticket, a new ticket should not be made. Additions can be made to the existing discussions and work. And it is important to assign the ticket to avoid any clashes. Also, it is advisable to check pull requests.

Kafka's Licenses

As already discussed in the introduction, Kafka is developed by LinkedIn and the developers of Kafka later found Confluent. Apache Kafka is licensed under Apache License 2.0. To have a better understanding of the license, it is necessary to know the permissions granted by the license for its users. It allows the user to use it for any purpose or make necessary changes. Even the user has rights to distribute the real software itself of the one which he/she modified. Therefore, it can be said that Kafka is free, and the real source code of the software is publicly available.

Things that are allowed under Apache 2.0

- Private and Commercial uses
- Modification and redistribution
- Sublicense and Patent claims
- Warranty service

Things that are unallowed:

- Usage of Trademark
- Legally responsible for mis happenings

However, Kafka receives support from Confluent and Cloudera. Confluent, who are the creators themselves, extents support in maintaining and improving the experience of Kafka users. The Confluent Kafka version contains certain features which are licensed under Confluent Community License. Though their platform fully supports the open source Kafka version, these enhanced features are valuable additions for a Kafka user. These services include Confluent Schema Registry, Confluent REST Proxy, Confluent KSQL and some of the Confluent connectors. But according to the announcement Confluent group made on 2014, they changed the license of components of Confluent platform from Apache 2.0 to Confluent Community License. Both the licenses share some similarities like allowing the user to download the source code, make changes and reorganizing it. But the main difference between them is that the latter does not support the usage of software as a SaaS service. In most of the cases, this license allows user to make a SaaS product out of the code. To cite an example, user can use KSQL as a product or service for his/her own needs but opposes the building of a KSQL-as-a-service offering. Altogether, developing competent services or products to Confluent like software-as-a-service, platform-as-a-service, infrastructure-as-a-service which have the software is forbidden under the license.

There are many benefits in getting this license from Confluent as they provide a lot of support to Apache Kafka. Their Confluent Platform Enterprise version has the Control Center monitoring application and many similar supports for Kafka. Confluent KSQL is a substitute of Python and Java to interact with streaming SQL engine. There are a lot of benefits from KSQL such as real-time monitoring and analysis, detection of anomalies, understanding and delivering sensor data, gaining customer 360 – view. Another product of Confluent schema-registry helps in storage and retrieval of Apache Avro schema. They provide serializers to manage Kafka messages. Kafka REST proxy is an interface to the clusters. It has got various applications such as managing messages

(producing and consuming), check the status of the cluster and perform some of the administerial actions. They maintain metadata about partitions, clusters, brokers etc. It also supports Producers, Consumers, Data formats, simple consumers, REST Proxy clusters and Load Balancing. Kafka Streams is a library for client and is mainly used for developing applications and other services. Here, the Kafka cluster is used to store data. There are several features that attracts customers like elasticity, scalability, fault-tolerance, viability, security features and so on.

Another support Kafka receives is from Cloudera. They maintain engineering relationships and contributed to security integrations. They also provide support to large-scale production customers. They involve in deployment and bringing in improvements for Kafka. Cloudera provides enormous support for Kafka Broker and uses it with Hadoop components. They officially give support for Kafka Streams. Kafka is being integrated to Cloudera's platform and therefore it works smoothly with components like Apache Flume, HBase and so on. Simple deployment and debugging and reviewing of Kafka is provided by Cloudera.
Though Apache Kafka can be availed freely, for an enterprise to use this smoothly they might need license of vendors to ensure support and get other value adding services.