Quest Journals
Journal of Research in Business and Management
Volume 9 ~ Issue 8 (2021) pp: 01-07
ISSN(Online):2347-3002
www.questjournals.org

Research Paper

Kafka – Idiomatic Messaging System for all Enterprises

Manjula N[1], Mahesh Kumar[2]
Department of Information Science
Global Academy of Technology, Bengaluru
Genpact, Bengaluru

ABSTRACT:
Streaming is a bigdata technology, the data produced by sources such as industrial sensors, clickstream servers and user application activity across multiple domains. Streaming services made it possible for us to consume content continuously without uploading the whole file. During data retrieval process, this creates challenges around data freshness, missing records and data synchronization. Kafka data processing engine is designed for the high speed, real time information processing which makes AI and big data possible for all group of enterprises.
Kafka, a distributed messaging system is used for collecting and delivering high volumes of log data with low latency. The paper provides information about Kafka messaging systems, the use cases, features to facilitate users to make an informed decision and also pave way for future research and development. An in-depth study about Kafka is provided to decide which features of a messaging system meet the needs of the enterprise applications.

Received 06 August, 2021; Revised: 18 August, 2021; Accepted 20 August, 2021 © The author(s) 2021. Published with open access at www.questjournals.org

I. INTRODUCTION

The big data is generated constantly and delivered in a continuous stream of small files, capturing event based interactions without any breaks. Kafka stream processorintegrates applications and data streams via an API.To resolve the log data processing problem, which is a critical component of the data pipeline for consumer internet companies, kafka incorporates ideas from existing log aggregators and messaging systems, and is suitable for both offline and online message consumption. Kafka is an efficient distributed messaging system providing built-in data redundancy and resiliency while retaining both high-throughput and scalability.Ideally, none of the Big Data related projects gets completed without Kafka because a major part of data ingestion into HDFS taken care of by Kafka through distributed publish-subscribe model.

II. RELATED STUDIES

Apache kafka has been compared with traditional messaging systems kike RabbitMQ. The results shows that when handling large amounts of messages, kafka performs better.The presented framework solution is based on open-source Apache Kafka—a messaging system

with robust data integration libraries (Kafka Connect) and stream processing API (Kafka Streams)to meet the need of real-time data/message processing in Confluent Platform [21].

Apache Kafka aims to unify offline and online processing by providing a mechanism for the integrationof heterogeneous dataset as well as the ability to analyse and process streaming data over a clusterof machines [22].At LinkedIn we are currently running over 60 billion unique message writes through Kafka per day

The main purpose of Kafka's usage in business enterprise is based on producer consumer problem. The data processing with respect to producer and consumer is done on same application using apache kafka. Even though the producer, consumer data is isolated from each other, it is difficult manage interactions created by a large number of applications because every application has their own settings with respect producers, consumers and queues. So resolve the problem related to this, a dataflow adapter is used for centralized data management and to integrate data among all local applications for distributed data processing of data streams from Apache Kafka.This data flow adapter measures the speed and the latency of data processing, the durationof data transfer between the application and the streamingplatform and other performance metrics, which transmitsthrough Apache Kafka for the further display and analysis of data[2].

The performance of data streaming process[3] is analysed based upon message size, batch size, message replication and hardware used. If the number of nodes vary then the performance may drop suddenly due to internal synchronizations and the network. [4] the queueing theory is used to evaluate the end-to-end latency of packets

III. KEY FEATURES OF KAFKA STREAMING

a) Kafka pub/sub messaging system is fault tolerant:The Kafka pipeline architecture[6] discusses about the combination of a component-based framework, redundancy patterns, and a runtime manager in Kafka enables fault tolerance, detects host failures, and trigger a reconfiguration of the system at runtime. It maintains system operation in case a fault occurs and automatically restores fault tolerance. This can be used for validating process in industrial distributed automation system. The validation shows how the system quickly restores fault tolerance without the need for operator intervention or immediate hardware replacement while limiting the impact on other applications.

The component based automation system [5]the fault tolerance act as the most important factor to improve the reliability of the real-time messaging system. Apache Kafka will catch up with the lost messages in the failurecase using the checkpoint intervals in message recovery process. With this the experimental results outperform the message recovery and save time approximately 50% than the original system.

Kafka is a potential messaging and integration platform for Spark streaming. Kafka act as the central hub for real-time streams of data and are processed using complex algorithms in Spark Streaming. Once the data is processed, Spark Streaming could be publishing results into yet another kafka topic or store in HDFS, databases or dashboards. The following diagram depicts the conceptual flow.

b) The data streaming pipeline:

Kafka data streaming pipeline [4] is capable of processing data in real-time, and it eliminates the need to maintain a database for unprocessed records. The 3 things which makes kafka easier are i) Automate the cluster creation -The cluster creation is automated to simplify the disasters by defining tenant models.In shared cluster one slice of the cluster is given to the tenant. ii) Self serve Tenant, Topics and Connector management- While defining the tenant model, single or multi tenants, both clusters and tenants can be expanded separately.Either in single or multitenant model, the tenants cannot use the entire space. Theconnector management is used to export data from kafka to other external systems. iii) Centralised metrics and logs- Kafka produces over 50 counters and metrics with centralized metrics. The metrics helps to manage data and find error.

Topic view:
For each topic, the incoming and outgoing bytes and partition detailsare recorded.
Broker view:
It has higher number of partitions.If number of unreplicated partitions, pre cursors are more.

c) Performance optimization:

Kafka's performance optimization is measured by latency and throughput.A fine-tuned Kafka system has just enough brokers to handle topic throughput, given the latency required to process information as it is received. By refining the producers, brokers, and consumers to send, process, and receive the largest possible batches within a manageable amount of time will results in the best balance of latency and throughput for the Kafka cluster. To handle large messages kafka producers i) compresses the messages into smaller size using gzip, lz4 and snapy. ii) uses file location to send messages that are stored in shared storage iii)use splitter to split large messages into 1 KB segments with the producing client, using partition keys to ensure that all segments are sent to the same Kafka partition in the correct order. The consuming client can then reconstruct the original large message.

The size of the cluster based on the amount of disk-space required, which can be computed from the estimated rate at which you get data, times the required data retention period**.**A Kafka cluster has exactly one broker that acts as the Controller.

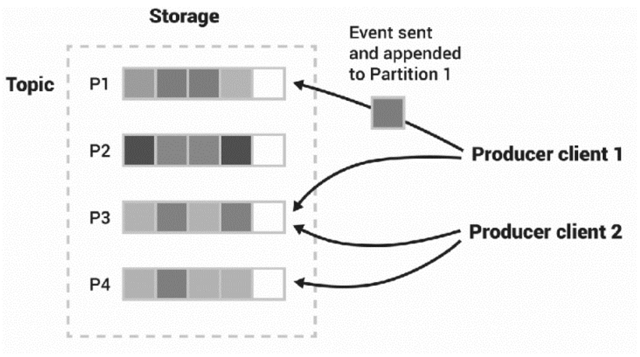
d) Retention period:

Retention period retains all the published records within the Kafka cluster. It doesn't check whether they have been consumed or not. Moreover, the records can be discarded by using a configuration setting for the retention period. And, it results as it can free up some space**.**Default retention period for Segments is 7 days.

Infinite retention using tiered storage, it extends Kafka's storage beyond the local storage available on the Kafka cluster by retaining the older data in an external store, such as HDFS or S3 with minimal impact on the internals of Kafka.-

e) Kafka streams Resiliency-having multiple data centres:
High availability, resilience and scalability have become the standard when building user-facing applications. Resiliency is in need to have nice sleep without disturbance- for profit.
Today, having a disaster recovery plan requires less effort and no up-front investment, by making use of cloud computing. As a result, many consumer-facing applications are now set up in a geographically dispersed fashion. In case of major failure, users are not affected, and latency is reduced for users across the globe.
The recovery time objective and recovery point objective can be specified as per the severity of the server down. In these scenarios zoo keeper comes to rescue. A typical ZooKeeper ensemble has 5 or 7 servers, which tolerates 2 and 3 servers down.
The infrastructure options can be of
i) Replication and Kafka streams- Active/Passive: the benefit of this type is, the clusters are independent here and is potential for less produce latency, the disadvantage is internal topics can't be replicated, because change logs and output topics may be out of sync with each other since they are replicated asynchronously. In addition upstream change logs may lag behind downstream, resulting in an unexpected and altered application state
ii) Stretch clusters and Kafka- it preserves offsets, recovery can be automatic, exactly one semantics are possible. The disadvantage is, there is no answer for recovery automation in a 2.5 DC setup

f) Kafka Event streaming[1]
The event streaming is the practice of capturing data in real-time from event sources like databases, sensors, mobile devices, cloud services, and software applications in the form of streams of events. The event sourcing, or design patterns concepts such as Enterprise Integration Patterns (EIPs), are based on event-driven architecture. Kafka's event based data flow feature is flexible and provides agile services to act on data which is relevant. It scalable central nervous system is built by design for zero downtime to handle the failure of nodes and networks and rolling upgrades. It connects anything, i.e. integrating any kind of application and system. The distributed storage feature for decoupling applications allows you to store state of a micro service instead of requiring a separate database. The event sourcing patterns and command query responsibility segregation (CQRS) pattern helps to implement stateless service and stateful business processes.Event streaming ensures a continuous data flow and interpretation of data so that the right information is at the right place, at the right time.



Event streaming is applied to a wide variety of use cases across a surplus of industries and organizations.The emergence of the Internet of Things (IoT) has enabled the adoption and development ofseveral real-time monitoring systems for diverse spheres of life such as energy management, health,smart environment, manufacturing, and security. As a result, the global IoT market is expected to hitover $10 trillion in 2025 [1].

The Kafka streaming platform in the enterprise architecturebenefits are,i) accommodation of larger loads and its elastic scalability fit the needed resources to cope with loads dynamically usually in relation to scale out capability, ii) the architecture is flexible enough to build small services, big services, sometimes still even monoliths, iii) its openness without commitment to a unique technology or data format, iv)provides independent and decoupled business services, v)its multi-tenancy ensures that only the right user can create, write to and read from different data streams in a single cluster, vi) easy deployment either in public cloud or in a hybrid environment.

g) Building a Real-Time Streaming ETL Pipeline

From developerspoint of view an ETL paradigm with distributed systems and event-driven applications is increasingly in need, where businesses process data in real time and at scale. There is still a need to “extract”, “transform”, and “load,” but the difference now is about how data is treated. Businesses no longer want to relegate data to batch processing, which often is limited to being done offline, once a day. They have many more data sources and of differing types, and want to do away with messy point-to-point connections. To resolve this problem, embedding of stream processing directly into each service can be done, and core business applications can rely on a streaming platform to distribute and act on events.

h) In-Sync Replicas

In-Sync Replicas are the replicated partitions that are in sync with its leader, i.e. those followers that have the same messages (or in sync) as the leader. It’s not mandatory to have ISR equal to the number of replicas.

Followers replicate data from the leader to themselves by sending Fetch Requests periodically, by default every 500ms.

If a follower fails, then it will cease sending fetch requests and after the default, 10 seconds will be removed from the ISR. Likewise, if a follower slows down, perhaps a network related issue or constrained server resources, then as soon as it has been lagging behind the leader for more than 10 seconds it is removed from the ISR.

Benefit

If by any chance one of the brokers goes down or due to network issues, is unreachable, one of the replicas will become the leader.

Only the replicas that are part of the in-sync replica list are eligible for becoming the leader and the list of in-sync replicas is persisted to zookeeper whenever any changes are made to it. Also, Kafka’s guarantee of no data loss is applicable only if there exists at least one in-sync replica**.**

i) Consistency and Availability

One of the fundamental challenges in distributed systems is achieving a delicate balance between consistency and availability in any recommendation systems. Kafka and Kafka Streams both provide many configurations to tune applications for such a balance. For instance, in Kafka, tuning the number of replicas, in-sync replicas, and acknowledgment gives you a wider range of availability and consistency guarantees

j)Reliability

The distributed commit log technologies of kafkaconcentrates on scalability andthroughput apart from flexibility and delivery. It resolves starvation problem by implementation of load shedding engine. But the impact of network quality will affect the Kafka’s reliability, which can be resolved by configuring the parameters properly

k) Data transmission

In real-time streaming, Kafka is used to transmit the data from controller to the data store. Therefore, in this case, the public key is in the Kafka producer for encrypting all the messages that will be transmitted to the Hadoop server. Through encryption, it assures that all messages are secured, even the messages that could be attacked by hackers. Even the size of the message will be reduced after performing this encryption method. Similarly, all of the secured Kafka messages are stored in the temporary data store of the Kafka broker. These messages will be consumed by the Kafka consumer that contains the private key for decryption.

l)Kafka Streaming Using Partition, Single and Multi-Threading model

Kafka partition number can significantly improve its throughput while increasing the producer/consumer channel count.

i) Single threaded model:

KafkaConsumer is not thread-safe, so using single thread fits in well. But, this approach is limited to single thread for processing messages. When working with KafkaConsumer, we usually employ single thread both for reading and processing of messages. In this consumer model, single thread is used in getting records, processing records, committing offsets, handling consumer group rebalance. If Kafka subscription model is used, all these functions will happens inside poll method except processing. While implementing threading model, by default offset committing and group rebalancing are handled automatically, default value for commit interval is 5000ms or it can be executed manually by calling commitSync/commitAsync method.

In group rebalancing, each consumer group is coordinated by one of the broker and one broker can be coordinator for many groups, and group leader (one of the consumer) is responsible for partition assignment. Two versions of protocols are used here, i) eager rebalancing and ii) incremental cooperative rebalancing protocol.

Group rebalancing is triggered by group coordinator when, new consumer joins or existing consumer leaves the group, when new partitions are added to a topic, or when consumer subscription changes. The group coordinator notifies consumers about rebalance through heart beat responses. The consumers within the group should finish processing for revoked partitions before they can be reassigned. And the consumers confirms the partitions release by sending group join request on next poll.

Limitation with single threaded model

By adding more consumers to this model, more TCP connections to be set to the cluster, means more requests being sent to the server which causes slightly less batching of data which in turn can cause some dropouts in I/O throughput.

The time taken for record processing will be long. This long record processing can cause group rebalance due to poll interval timeout and also cause long rebalancing periods, which makes waiting consumers to re-join group again and again.

ii) Multithreaded model:

Method 1:

Fork join threading model

In this model multiple threads are used for processing but it waits for all the threads processes to finish before the next poll call. To guarantee the processing order, process the record from the same partition and by same thread. The offset committing and group rebalancing is handled by poll method, same as single threaded model.

Steps to implement "Fork join" method

Step 1: Get records collection from poll method
Step 2: Group records by partition which results in multiple collection of records
Step 3: Create task, the callable implementation for each collection of records from the partition from step 2.
Step 4: Use Executor service function to submit all created tasks from step 3.
In this multithreading method, the poll method cannot be called after completion of every task. It is called once all the tasks of the partitions assigned to the consumers is done. This limitations is overcome by method 2: Fully decoupled multithreaded model

Method 2:

Fully decoupled multithreaded model

Record Processing

This model uses multiple threads for processing.To guarantee the processing order, process the record from the same partition and by same thread. For preserving order guarantees, records from an individual partition must be processed by only one thread at a time, which can be achieved by pausing the partition after submitting its records for processing, by resuming the partition after records processing is finished, by implementing thread synchronization when resuming threads, as KafkaConsumer is not thread safe.

Here, the completed threads doesn't wait for other processing threads to finish, instead they continue to poll i.e., will be busy in getting new records and the offset is committed manually.

m) Group rebalance

For group rebalancing, ConsumerRebalanceListener is used, so that the revoked partitions are finished first to overcome the long rebalancing periods and consumer re-join problem.

With respect to reacting to group rebalance, consumer joins group rebalance during poll method execution. Some of the partitions assigned to the consumer can be revoked while its records are still being processed by another thread.Registration of ConsumerRebalanceListener gives the ability to react to partition revocation. Upon receiving partition revoked event the user need not to do anything, this will avoidunnecessary duplicate record processing and resource consumption, the user just waits for processing task(current record) to finish and stops processing task,

With respect to stopping tasks for revoked partitions in group rebalance, send stop signal to a task, so that it can break out its processing loop. Wait for task completion and commit offset for revoked partitions by using synchronization.

n) Offset committing

There are two approaches for offset committing, First method - Committing offset from task processing thread, this approach requires thread synchronization and can cause too frequent commits. Second method-Committing offsets from main thread, it checks task for current offset on each main loop iteration and store them to a buffer. It commit offset from buffer periodically.

IV. LAMBDA ARCHITECTURE WITH KAFKA

When it comes to processing transactions in real-time, it is always insisted to use Lambda Architecture. The Lambda Architecture provides a useful pattern for combining multiple big data technologies to achieve multiple enterprise objectives.It enables us to score transactions or other business events in real-time and still consider the most recent events as well as the whole transaction history in its scoring model.

By using Kafka at the beginning of the pipeline to accept inputs, it can be guaranteed that messages will be delivered as long as they enter the system, regardless of hardware or network failure.

The lambda architecture is aimed at applications build around complex asynchronous transformations that need to run with low latency.

Example: the recommendation systems like, Music, News etc. Here, the system needs to crawl various new services, process and normalize all the input and then index, rank and store it for serving. A common error when publishing records is setting the same key or null key for all records, which results in all records ending up in the same partition or will end up with unbalanced topic.

With respect to customization of personal messages, at low level consumers, where topics and partitions are specified as is the offset from which to read, either fixed position, at the beginning or at the end.Its cumbersome to keep track of which offsets are consumed so the same records aren’t read more than once. The issue with low level consumer is solved by using high level consumer. Here, the consumer group is created by adding the property ‘group.id’ to a consumer group. The broker will distribute according to which consumer should read from which partition and it also keeps track of which offset the group is at for each partition.The broker tracks this by having all consumers committing which offset they have handled.

By using consumer groups, consumers can be parallelized, so that multiple consumers can read from multiple partitions on a topic, allowing a very high message processing throughput. The number of partitions impact the maximum parallelism of consumers as there cannot be more consumers than partitions. The broker is the backbone of kafka.

The demand for real-time analytics has led to demand for workflows that can effectively balance latency, throughput, scaling and fault tolerance.In order to fullfill the demand for real-time analytics, a system is designed that balances between the concept of "single version of truth" and "real-time analytics". Lambda Architecture is one such method.

V. CONCLUSION

Kafka is a valuable tool in scenarios requiring real-time data processing and application activity tracking, as well as for monitoring purposes.Kafka has a significantly higher throughput compared to other messaging systems and message ordering is done at each partition. Kafka guarantees, message delivery, it provides at-most once, exactly once and at-least once delivery as per the application need.The message retention period is set while configuring or for longer retention period tiered storage is used. Kafka was built from ground up as a horizontal scaling system. With the co-ordination by the zookeeper, adding and deleting brokers makes Kafka easy to scale

REFERENCES:

[1]. BunrongLeang, SokchomrernEan, Ga-Ae Ryu, and Kwan-HeeYoo, Improvement of Kafka Streaming Using Partition and Multi-Threading in Big Data Environment, Sensors (Basel). Jan 2019.

[2]. Dataflow Adapter: A Tool for Integrating Legacy Applications into Distributed Stream Processing

[3]. Paul Le Noac’h INSA Rennes et al,A performance evaluation of Apache Kafka in support of big data streaming applications, IEEE International Conference on Big Data, 2017

[4]. Han Wu, Zhihao Shang, KatinkaWolter,Performance Prediction for the Apache Kafka, IEEE 21st International Conference on High Performance Computing and Communications, 2019

[5]. Enhancement of Fault Tolerance in Kafka Pipeline Architecture

[6]. Fault-tolerant fault tolerance for component-based automation systems

[7]. B. R. Hiraman, C. V. M and C. KarveAbhijeet, "A Study of Apache Kafka in Big Data Stream Processing," *2018 International Conference on Information , Communication, Engineering and Technology (ICICET)*, Pune, 2018, pp. 1-3.

[8]. Jay Kreps, Neha Narkhede, Jun Rao, “Kafka: A Distributed Messaging System for Log Processing”,atNetDB workshop,2011

[9]. P. Bellavista, A. Corradi and A. Reale, "Quality of Service in Wide Scale Publish—Subscribe Systems," in IEEE Communications Surveys & Tutorials, vol. 16, no. 3, pp. 1591-1616, 2014.

[10]. Kamburugamuve, Supun& Fox, Geoffrey. (2016). “Survey of Distributed Stream Processing.” 10.13140/RG.2.1.3856.2968.

[11]. P. Le Noac'h, A. Costan and L. Bougé, "A performance evaluation of Apache Kafka in support of big data streaming applications," 2017 IEEE International Conference on Big Data (Big Data), Boston, MA, 2017, pp. 4803-4806.

[12]. https://www.confluent.io/resources/kafka-summit-2020/kafkaconsumer-decoupling-consumption-and-processing-for-better-resource-utilization/
[13]. https://kafka.apache.org/intro
[14]. https://www.tigeranalytics.com/blog/building-nrt-data-pipeline-debezium-kafka-snowflake/
[15]. https://www.confluent.io/blog/apache-flink-apache-kafka-streams-comparison-guideline-users/
[16]. https://www.datanami.com/2019/05/30/understanding-your-options-for-stream-processing-frameworks/
[17]. Spark Structured Streaming: Customizing Kafka Stream Processing
[18]. Performance Evaluation of Intrusion Detection Streaming Transactions Using Apache Kafka and Spark Streaming
[19]. KAFKA: The modern platform for data management and analysis in big data domain
[20]. TRAK: A Testing Tool for Studying the Reliability of Data Delivery in Apache Kafka
[21]. Confluent. Available online: https://www.confluent.io.
[22]. Garg, N. Apache Kafka; Packt Publishing Ltd.: Mumbai, India, 2013.
[23]. Hsu, C.L.; Lin, J.C.C. An empirical examination of consumer adoption of Internet of Things services:Network externalities and concern for information privacy perspectives. Comput. Hum. Behav. 2016,62, 516–527
[24]. Kafka- data Processing using data flow adapter
[25]. https://www.confluent.io/blog/apache-kafka-vs-enterprise-service-bus-esb-friends-enemies-or-frenemies/