Global Ordering of Messages in a Distributed Kafka Cluster

Sachin Sharma Nisha Murarka Shubham Rathi

***Abstract*—Current distributed systems generates terabytes of logs in a very short span of time. These logs contains essential data like system metric, user activities and events. Ordering of these logs are of pivotal importance to determine the correct state of the system and avoid ambiguities and inconsistencies. Apache Kafka is one of most widely used distributed message queue that addresses several key challenges in distributed log processing. However, it has its own shortcoming when it comes to ordering of messages. Even though Kafka ensures that the messages from single partition is delivered in order to the consumer, there is no such guarantee across different partitions. As part of this research we are exploring different ways to achieve global ordering of messages for a Kafka Topic.**

***Index Terms*—Messaging Queue, Distributed Messaging, Log Processing, Kafka**

I. Introduction

With the onset of Web 2.0 and the advancement towards Web 3.0 we have seen a tremendous growth in decentralization of computer systems. These systems generates large amount of data or "log". The data can be generated several sources like (1) user activity events- login, content fetch, user actions, user transactions (2) System metrics like service calls, network, heap memory, disk utilization, CPU on each node in a cluster of nodes.

This kind of data analysis was done by earlier systems in an offline manner by scraping log files from production servers. But the current systems relies highly on the feedback from analysis of data in real-time to execute future actions. A very general example can be observed in trading platforms executing high volumes in stocks. These systems relies on the real time processing of transaction logs from ledgers. The performance of such systems relies heavily on accuracy and high availability of data.

Apache kafka is distributed and scalable, and offers high throughput and also provides applications the capability to consume logs events in real time. Kafka consists of *consumer*, *producer*, and *broker*. Kafka defines *topic* as a stream of messages of a particular type. A producer publishes messages to a topic, which is then stored in a set of nodes called brokers. Consumers can subscribe to one or more topics from the brokers, and consumes messages by pulling data from brokers.

Since there are multiple consumers consuming messages from a topic, ordering of messages cannot be guaranteed at the client end. Kafka overcomes this by allowing to create partitions against a given key, which can be consumed a single consumer. This ensures the ordering of the message within the partitions but fails to provide ordering across partitions. The constraint of a partition being available to be consumed by a single consumer is a *tradeoff* on the distributed nature of the system.

II. Motivation

The motivation for this projects came from a practical application of Kafka as message pipeline for transaction data of users in a trading platform. The users events like buying and selling of stocks is considered as an event and pushed to Kafka for processing. The ordering of such events holds great significance, as we don't want a consumer to process the sell request of a stock 'X' by user A before processing the buy request of the stock 'X' by user A. The solution that Kafka provides for this problem is by creating partitions based on unique user identifiers. This will ensure that messages for a user can only be consumed by single a consumer and hence will never be processed out of order. This design fails for use cases where there is no clear definition for partitioning of topics using a key like logs from a service. So it is only apt to look for alternative designs for overcoming global ordering in Kafka topics.

III. Project Proposal

In this research project we aim to explore various ways to maintain global order of messages for a given topic of messages. Creating partitions requires keys which might not be logical for uses case where logs are not logically tied to any defined entity. Partition also restrict the number of consumers to a single consumer node. We aim to ensure global ordering irrespective of the consumer count. We also aim to explore data recovery in case any broker goes down and unconsumed messages gets lost. Adding built-in replication in kafka to redundantly store messages on multiple brokers can achieve that.

References

[1] https://engineering.linkedin.com/distributed-systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying
[2] https://cs.uwaterloo.ca/ ssalihog/courses/papers/netdb11-final12.pdf
[3] https://www.dataversity.net/how-to-overcome-data-order-issues-in-apache-kafka/
[4] https://dl.acm.org/doi/10.1145/3448016.3457556