A NEW PAYMENT SYSTEM DESIGN WITH APACHE KAFKA

[1]NESLIHAN YAYLICIOĞLU, [2]MURAT DERMAN

[1]Collection Solutions Manager, [2]Associate Principal Developer
E-mail: [1]neslihan.kukut@turkcell.com.tr, [2]murat.derman@turkcell.com.tr

Abstract - Apache Kafka is a highly scalable messaging platform, which provides high throughput with low latency and useful for real-time data processing. This article is about replacing a Messaging Queue technology with Apache Kafka for a Payment system. The legacy Payment system, which was being used for Payment processing and payment distribution to clients, was using a Queue based infrastructure and had various problems due to the limitations of Payment processing rules and increased number of transactions over years. In this article, we present the elucidations offered by Kafka regarding the complications of the legacy system. Moreover, the limitations of the old system with queue messaging technology was discussed in detail.

Keywords - Payment Distribution, Payment Processing, Apache Kafka, Queue Messaging System

I. INTRODUCTION

A payment system with Queue Messaging Technology was being used in a Telecom Company with over 30 million subscribers and manages 15 million transactions on daily basis. The Payment system with queue messaging system was failing to handle the load on invoice due dates as a result of bulk payment notifications. There was a great need a change on the architectural design of the system, as it caused several different malfunctions.The utmost importance for payments systems is that the transactions have to be processed consecutively. A design with a master server was needed as to keep the order of payment transactions, hence the design with queue messaging system was structured on a single server being the master and of its distribution of transactions to other four servers. Noting that the queue messaging system brokers use Java Remote Method Invocation (RMI)connection in between during transmissions, where the master server had high latency and timeout errors while trying to reach to slave servers due to usage of RMI connection under heavy load on specific days which were expected of a larger volume of transactions. Furthermore, high latency led to disk space problem as the queue length grow under the high load while queue messaging servers would try to process previous transactions.If the daily transaction numbers were under a certain threshold, the system worked without a problem. However, on invoice due dates, when the number of the expected payments were over millions, the processing time for a transaction starts taking longer time under load, as a result the file store would take a larger disk space, even reaching a size of over 1 TB weekly. Master server could not manage a file of a such a large size and would become unresponsive, thus, it needs maintenance regularly. Another problem of the file maintenance was data loss. After server became unresponsive under the load, the queue storage file would become a big file of size over 1 TB, which almost took all the free space on server. To maintain the server, the file contents would be deleted after shutting down the system. However, deleting the content of the payment queue file means losing the sequence of transactions for the payment system. As it is mentioned before, for the payment system, order of the transactions is very important, since the transaction order decides the status of an invoice whether it is a paid or unpaid.In consequence, an additional maintenance is needed after deleting file contents.

II. THE METHOD AND BENEFITS OF USING APACHE KAFKA

A new approach with Apache Kafka seemed promising to solve the problems at hand. The problems associated with the payment system were scalability, data processing performance, resource management, data loss and fault tolerant. Kafka is a distributed streaming platform which is highly scalable and useful for a high volume of data processing for online and offline messages. Kafka also prevents data loss by replicating messages on the disk within the clusters and manages messages with the Zookeeper synchronization service. Kafka can handle failures with databases and servers/client systems with its clusters. In aspect of scalability, the design of queue messaging structure was based on a master server, which was distributing the load to the four other servers that be seen as in Figure 1.

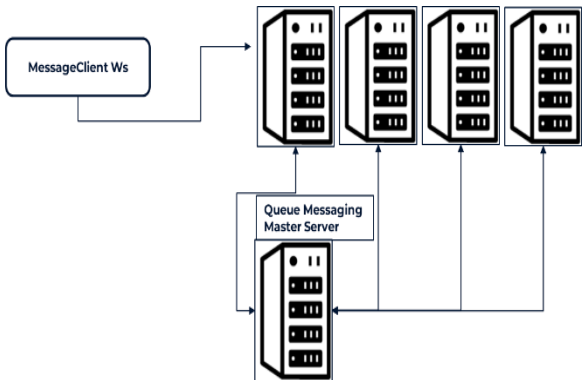


Figure 1: Queue Message System Architecture.

Adding a new server would not change the equation as all transactions had to pass through a single server. Kafka is horizontally scalable, since it replicates the data through its partitions. Adding or removing new servers to balance the load is easy with Kafka as it can be done without any downtime.

In this new design, five brokers are used more efficiently, without the need for a master server, the overall performance of the system is improved. In the previous design, one master server and four slave servers were being used where all load had to be distributed by the master server to slave servers. In comparison to the old design, all five servers can be used simultaneously, thus the load is divided to five servers.

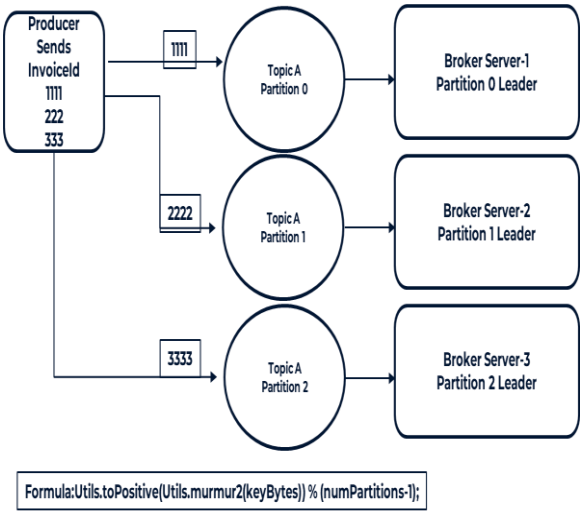


Figure 2: Kafka Architecture.

In addition to boosted performance with the scalability feature, data processing time decreased by 93% compared to the old system. Kafka brokers can write messages at a higher rate, which enables Kafka to process them in high throughput with low latency. As mentioned before, high latency in a queue structure would lead to a big file store and unresponsive server problem. Even if terabytes of messages are stored, Kafka maintains stability. Kafka also guarantees on no data loss.

Messages is being persisted on the disk and replicated within each cluster. Even if a broker is down, messages are safe on the disk. Kafka behaves like a log and for each topic (data stream), it maintains a partitioned log. The records in each partition have a unique sequential id, the offset, which helps the consumer to keep the order of records while processing. In case of a downtime for both consumer and producer, the consumer can continue pulling messages from where it was left off, previously. In case of an unexcepted shutdown, partition offset helps to keep the order for each client, which is critically important for a payment system.Further more, Kafka is inherently fault tolerant.

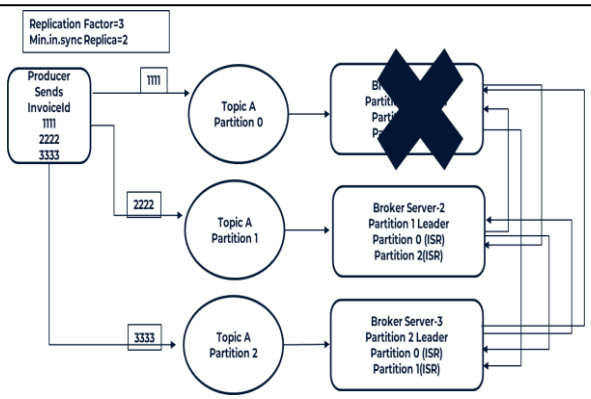


Figure 3: Broker 1 failed.

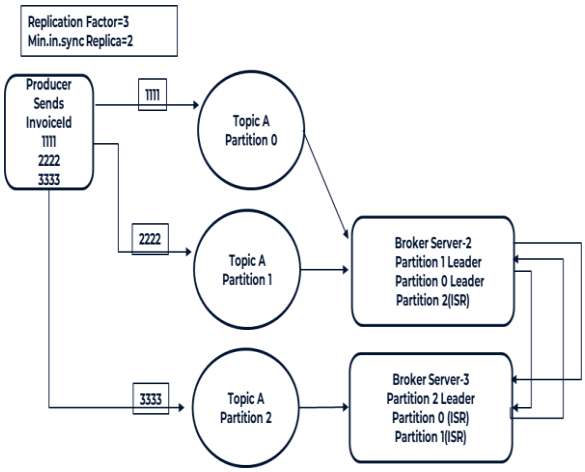


Figure 4: Broker 1 is replicated to Broker 2 and Broker 3.

If a problem occurs on a leader broker, it replicates its data to the other brokers, and it continues to inform data from another broker, which enables continuation of data stream without any interruptions. In case of the system with queue messaging technology when the master server has a problem, system would not able to function anymore so that it would need a maintenance to be up and running again.

Kafka has another advantage for disk and CPU usage as well. In our payment system, payment transactions are being informed to twenty different systems/clients. Data had to be informed according to business rules, defined by the clients. In this case, for each client, a different queue has to be defined, which means that payment data has to be replicated on the disk by the number of the clients. On the other hand, Kafka can manage all clients within a single topic. Hence resource management on the disk is handled better for the systems with multiple clients. Moreover, Kafka uses lesser CPU than the system with queue messaging. Old system needed a physical machine to run which is not cost effective, while Kafka can run on a virtual machine without a glitch.

III. CONCLUSION

In conclusion it can be said that Kafka answered all the requirements for the payment system such as high

performance, fault tolerance, durability. The design with queue messaging infrastructure could not meet the demands of today's conditions.

Millions of payment transactions are processed and distributed to twenty different systems daily and performance of the system has become a serious issue as the old system was not capable of handling the load. Sustainability is a prerequisite of a healthy payment system. Our purpose for redesigning the system was to build a more scalable, and stable system in terms of performance. It is shown that, Kafka is highly scalable, fault tolerant and durable which facilitated a reliable system.

REFERENCES

[1] R. Shree, T. Choudhury, S. C. Gupta and P. Kumar, "KAFKA: The modern platform for data management and analysis in big data domain," 2017 2nd International Conference on Telecommunication and Networks (TEL-NET), 2017, pp. 1-5, doi: 10.1109/TEL-NET.2017.8343593.

[2] Wu, Han &Zhihao, Shang & Wolter, Katinka. (2019). Performance Prediction for the Apache Kafka Messaging System.154-161. 0.1109/HPCC/SmartCity/DSS.2019.00036.

[3] Goodhope, Ken, Joel Koshy, Jay Kreps, Neha Narkhede, Richard Park, Jun Rao and Victor Yang Ye. "Building LinkedIn's Real-time Activity Data Pipeline." IEEE Data Eng. Bull. 35 (2012): 33-45.

[4] Kleppmann, Martin & Kreps, Jay. (2015). Kafka, Samza and the Unix Philosophy of Distributed Data.

★★★