International Journal of Science and Research (IJSR)
ISSN: 2319-7064
SJIF (2021): 7.86

Building High - Throughput Payment Transaction Systems with Kafka and Microservices

Pavan Kumar Joshi

Abstract: ***The incidence of payment transactions has grown exponentially, placing high throughput payment transaction systems in high demand. Payment processing systems need to be used for a large number of transactions while experiencing a low amount of delay, being highly scalable and tolerant to failures. In this context, Apache Kafka, which is a distributed event streaming platform coupled with a microservices architecture, has been seen as the best approach to the development of efficient, reliable, and scalable payment transaction systems. This is because Kafka is a distributed system combining the characteristics of microservices to important decentralized functionality while at the same time handling large transaction volumes, fault isolation, faster data recovery, and ease of development. For this paper, I propose the implementation of Kafka and microservices to develop a fast - processing payment transaction system. We expound on the storage architecture of Kafka, particularly issues of partitioning, replication, and message retention for real - time payment processing. Thinking of various functionalities, such as transaction validating, fraud, and settling, the microservices architecture can be analyzed to determine whether it can separate all these and more as individually deployable services or not. We also explain how the system achieves the concept of eventual consistency across distributed services; this is through Kafka as a message broker for microservices. In addition, this paper demonstrates and compares the throughput, fault tolerance, and scalability of the monolithic payment system and the microservices of the payment system connected with Kafka. Real - life examples of Kafka in financial organizations, as well as new benchmarks and case studies that demonstrate how Kafka and microservices can be used in actual payment systems, are also presented. Lastly, we look at some of the issues with adopting Kafka to existing payment platform architectures and how to solve them. We also suggest how it is possible to solve issues, including distributed transactions, data consistency, strict security and compliance requirements, before we draw our conclusion.***

Keywords: Apache Kafka, Microservices architecture, Scalability, Fault tolerance, Event - driven architecture

1. Introduction

The advent of e - commerce, digital, and mobile payments has put payment transactions on a new high of evolution. Systems developed in a monolithic style that unifies the processing of the application in one code base are not capable of addressing the load of current real - time transaction processing, scalability and fault tolerance requirements. There is an increased need for high throughput, which must be served while still keeping the quality of service high, mean time to failure low, and compliance and regulation standards adhered to. [1 - 4] This paper discusses how and to what extent the application of Apache Kafka and microservices can address these issues in contemporary payment systems.

1.1. The Role of Apache Kafka in Payment Systems

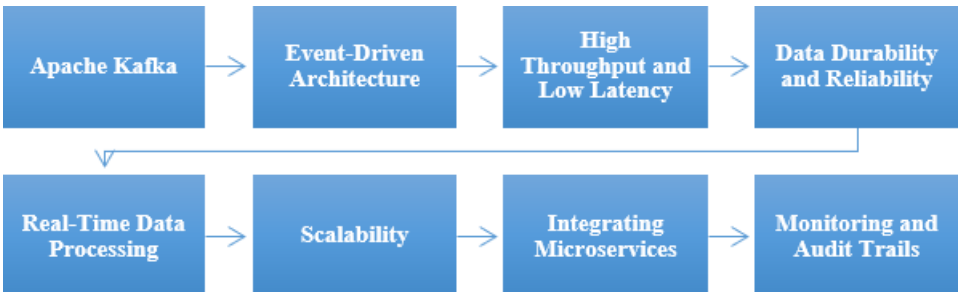

Figure 1: The Role of Apache Kafka in Payment Systems

- **Apache Kafka:** Apache Kafka is an open - source messaging system for distributed event streaming capable of handling vast data feeds at the lowest latency possible. Initially, Kafka was built at LinkedIn but has evolved into one of the core technologies of modern data platforms, especially within the financial industry. Being built to be highly reliable and scalable, it covers the areas of message brokering as well as storage and processing of stream data and is best suited for applications where real - time data are processed.
- **Event - Driven Architecture:** Kafka fosters an event - driven design that allows payment systems to process transactions as individual, individual events, increasing response rates. This architecture provides the independent scalability of a producer space and a consumer space, allowing different services to be compensated without needing to touch each other. Consequently, the payment system can be flexible in response to the varying amount of load in the transaction without adverse effects on system functionality.
- **High Throughput and Low Latency:** However, one of Kafka's strengths is its capability to handle massive amounts of data with little delay, something very relevant in today's world of payment transactions. During busy transaction operations, like promotional offers, Kafka's partitioning and replication enable the transactions to happen concurrently across different brokers. This high throughput capability makes it easier for the payment systems to meet performance requirements and sustain a good user interface.
- **Data Durability and Reliability:** Kafka's architecture of distributed logs ensures the high availability of data by

writing messages into multiple brokers, which is important for payment systems. This durability means transaction consistency and accuracy are not compromised, let alone by system failures. Through the replay of messages of the Kafka log, it is easy to recover in case of data loss, which is important in compliance and customers of financial transactions.

- **Real - Time Data Processing:** Due to the demands for huge real - time processing in the financial sectors, Kafka allows users to process transactional data in real - time. Payment systems can use Kafka Streams or Kafka Connect to build applications that can observe transactional patterns and identify fraudulent transactions. For example, if a transaction is suspicious, Kafka is capable of immediately notifying its linked fraud detection service, thereby enabling efficient preemption.
- **Scalability:** To sum it up, Kafka is scalable, allowing organizations to scale up the flow of payments without the addition of new architectures. Growing transaction traffic is well managed by Kafka because organizations can scale brokers and partitions as business increases. Overall, this scalability is a great benefit for the payment service providers because they often work with the rushing of transactions, for example, during holidays or promotions.
- **Integrating Microservices:** Today, Kafka has become an integral part of modern payment systems, acting as the base component for microservices architecture that provides the ability for its individual service operation. Kafka supports Kafka topics, which enable each microservice to generate and receive messages and thus make it easier to modularize the application. This separation of concerns helps to simplify the processes of maintenance and deployment of individual services, as well as greatly increases the overall flexibility of the system.
- **Monitoring and Audit Trails:** Kafka's logging capabilities are the key to offering a sufficient level of monitoring and auditing inside payment systems. Original documents are important for documenting the audit trail and as evidence for following compliance and regulation standards since they track the life cycle of transactions. Through tracking all events, Kafka successfully assists organizations in analyzing system functionality and enhances performance by addressing issues and enhancing accountability.

1.2 The Need for Kafka and Microservices Integration

Due to hi - tech competition in the current technological advances, the combination of Apache Kafka with microservices is crucial for integrating compatible applications. This integration solves some broad important issues of business in general and notably those companies that work in the growth industries, including finance and electronic commerce. Following are the factors that, in one way or the other, compel this integration to occur.

Figure 2: The Need for Kafka and Microservices Integration

- **Enhanced Scalability:** The primary purpose of using Kafka in microservices is that it increases scalability since it can be integrated with microservices. Underpinning this is the fact that as business and more transactions are facilitated, conventional architectures may not easily cope with growing volumes. Microservice enables discrete elements that may work within a system to have unique scalability, while Kafka permits the fast handling of data. Combined, the two guarantee organizations' systematic ability to scale their systems for increased or decreased loads without major redesign.
- **Improved Fault Tolerance:** In a microservices framework, failure would not have to bring down a whole system of services. Kafka, being a message broker, is very strong in its ability to reasonably pass messages from one service to another without losing them in the case of the failure of the service. This integration also generalizes system fault tolerance in that payment systems can continue processing transactions and maintaining availability, improving user satisfaction.
- **Real - Time Data Processing:** Due to the increasing need to make analytical results from data as it happens, the demand for real - time big data processing has also increased. Thus, Kafka can act as a strong support for real - time processing since microservices can process events on the fly. This capability is rather important in industries such as finance, where any decision made should be timely to affect the company's profitability and security, for example, to identify fraud transactions in real - time.
- **Decoupled Architecture:** Kafka is designed with a highly decoupled system whereby producers and consumers can work separately. This decoupling enables the microservices to talk to each other but not rely on any of it for functionality, making it easier to change or replace one or more microservices in a large microservices architecture. Flexibility of this type is essential in organizations that require fast and dynamic changes and adaptation to market needs.
- **Efficient Data Stream Management:** In managing data streams, many difficulties arise due to the fact that the involved systems are often complex. Kafka ensures effective stream management with excellent features such as data partitioning, replication, and retention. It is especially relevant for technology - oriented microservices that need real - time data to provide high performance since proper data access mechanisms will

guarantee high performance and timely information processing.

- **Streamlined Communication:** In a microservices architecture, the interactions between services can be reduced to a web API - style interaction, making the flow of messages between the services complex and cumbersome. Kafka eases this process by acting as the middleman between producers and consumers, passing around events. This reduces the overhead cost of interacting between services and increases the efficiency of services in providing desired applications.
- **Monitoring and Auditing Capabilities:** In industries with tight governing rules, such as the finance industry, it is necessary to keep a very keen monitor and audit trail. Kafka's basic logging features can be used to track all the events and transactions, making the organization fulfil regulatory standards. It also helps with audits and amplifies a capacity to diagnose and track system inefficiencies.
- **Adaptability to Changing Business Needs:** Kafka management, in combination with microservices, makes an organization more versatile to new business requirements. With constantly changing market environments, companies can bring new services or reshape existing services without rewriting the whole system. The fact that Kafka has grown out of flexibility makes it perfect for continuous delivery and deployment practices, so the new iterations can be churned out quickly.

2. Literature Survey

2.1 Traditional Monolithic Payment Systems

Monolithic payment systems are developed and contained within a single code covering all the functional aspects, including User Interface, Database and Business Logic. [5 - 9] while building and deploying these kinds of systems is relatively easy in the first place, as the number of transactions exponentially increases, the number of problems associated with this type of system also increases. Scalability becomes seemly unmanageable because adding more capacity for one functionality automatically involves scaling the entire system, which wastes resources and increases operational costs. For instance, boosting the throughput of payment processing may necessitate growing different but ancillary functions, such as fraud identification, which may be fiscally wasteful.

It explained that traditional monolithic payment systems have more severe performance reduction problems or limitations than intelligent payment Systems when transaction loads are high. Due to the centralized nature inherent in monolithic systems, coordination hitches result in decreased scalability and slow transaction throughput. These systems are normally ill - equipped to cater to performance demands that are expected in today's efficient and active financial markets, especially in circumstances where there is large traffic of transactions. Moreover, any modifications to the system can cause a lack of running, or this environment is not very friendly for changing the structure and conditions of the system.

2.2 Event - Driven Architectures in Payment Systems

Event systems provide better approaches than monotonic systems, especially when it comes to completing payment transactions. These architectures, for example, those based on Kafka, are meant to be event - driven; different systems may change their state, and the other is not obliged to interact with them simultaneously. This model dissociates the interaction between the producer, for instance, the transaction initiation from the consumer, such as payment processing, and the fact that events can occur at scale. The system can horizontally scale to encompass greater transaction rates, which has also indicated that optimizing the performance of financial systems, Kafka, in event - driven architecture, yields huge benefits. Kafka's partitioning system can be used to split transaction data into a number of relatively small subgroups, each of which can be processed by a different broker. This form of processing allows systems to grow to accommodate vast transaction rates of millions per second in a real sense. At the same time, the quality of service is not materially reduced. Another way in which Kafka helps build a very reliable system is by using the replication feature to ensure transactional data is copied to other brokers so that the data is not lost even in case of failure. This architecture definitely provides high availability during periods of increased transaction volumes, which is ideal for today's payment systems.

2.3. Microservices in High - Throughput Systems

The use of microservices architecture has completely transformed the ways in which the high throughput payment systems are architected and managed. While traditional systems are called monolithic, microservices distribute functions into small services that do not have dependencies on other services. For example, in the payment system, payment validation services, fraud checking, and authorization services can work on their own, all at different levels of utilization, depending on demand. It also brings problems of system decoupling, reliability, and flexibility since it is easier to update or fix one or some services while leaving the rest untouched. Specifically, investigated differences between microservices - based payment systems and monolithic - based payment systems and concluded that the systems based on microservices performed better in terms of scalability, high availability, and easy deployment. Microservices make it possible to scale only that part of the system needed rather than the whole application, hence saving on resources and operational costs. Secondly, Microservices architectures are usually more fail tolerant, in the sense that if one service fails, it does not adversely affect other services; hence, operating systems are very efficient. The key aspect of this approach is modularity, which empowers financial institutions and enables them to deal with massive payments in high - traffic environments.

2.4. Kafka and Microservices Integration in Real - World Use Cases

Many large banks and other types of financial institutions have leveraged Kafka and microservices to enable support for high throughput transaction processing. Goldman Sachs, for example, has been able to adopt Kafka as part of their real -

time transaction streaming, which provides a huge capacity to process millions of daily transactions and boasts high availability with comparatively low latency. [10] Through Kafka, Goldman Sachs was able to remove transaction processing services into its own scalable isolated subdomain, which increased response and stability at the peak loads. In order to maintain an unchecked flow of transactions, Kafka is designed to partition and replicate data, which allows for parallel processing of data without the possible loss of data. Stripe and PayPal have implemented Kafka and microservices to handle their unprecedentedly large transactional amounts. Payment processing for millions of customers is achieved through microservices architecture accompanied by Kafka, allowing every single service, like fraud detection, user authentication and others, to grow independently. Likewise, through Kafka's implementation and the use of microservices, PayPal has provided real - time processing of payments at the same time when traffic loads are expected to surge. The decoupled architecture makes it possible to enhance fault tolerance and minimize the potential loss of availability, creating favorable conditions for users to complete payments. The genuine examples outlined above show how the Kafka and microservices integration solution is now a best practice for scalability and throughput in high - frequency financial systems.

3. Methodology

3.1. System Design

The payment transaction system highlighted in this paper is proposed to be implemented as a distributed structure that would use Kafka for streaming and microservices for handling "tasks", which essentially the payment transactions would be classified into and referred to as "micro - services". [11 - 15] It comprises three major tiers or parts; each part is responsible for specific functionalities in order to give the payment system an efficient face while at the same time ensuring reliability and efficiency under very high transaction volumes.

Figure 3: System Design

- **Ingestion Layer:** The Ingestion Layer is designed as the real - time acquisition and storage layer for transactional data. This is well done using Kafka brokers that are distributed message queues. Kafka's partitioning feature makes it possible to scale the system horizontally, as Kafka allows the system to manage millions of transactions with brokers. The brokers receive the incoming payment requests and also make a copy of them to make them durable and fault - tolerant. This layer helps to keep transaction data easily accessible to other stages of processing and avoid traffic congestion at high loads.
- **Processing Layer:** The Processing Layer comprises a set of micro - services that carry out different tasks relating to payment processing, such as validation services, fraud checking services, and authorization services. This means that each microservice is deployable and can be managed proportionally to the amount of transactions in an organization. A specific example is that Kafka topics are used by the microservices to communicate asynchronously by queuing the calls so the system can continue to process transactions even if some services are unavailable briefly. For example, an example of a nearby service is a validation service, which verifies data coherence; an example of an instantaneous or real - time analysis service is a fraud detection service, which analyses the patterns of the transaction to make sure that the payer is not a fraud before authorizing the payment.
- **Storage Layer:** Thus, the Storage Layer is intended to provide a fast and highly reliable NoSQL database for data storage. This means that one can easily access the undertaking data, which is very useful in systems with intensive read and write cycles. Especially, the system maintains eventual consistency such that even when there are delays in data synchronization in the NoSQL database, the system can continue to function optimally. This is particularly important when keeping system availability in the distributed environment where setting up immediate synch ensures high - performance lags. They include operational data processing, analytics, and reporting by using the NoSQL database, which is designed to facilitate querying for various needs of an organization.

3.2 Kafka Configuration

In the proposed system, Kafka configuration is an important aspect of meeting the system's ideal performance in scalability, fault tolerance and throughput capacity. Key Kafka features such as partitioning of topics as well as the replication factor set in Kafka are optimized to serve high throughput transactional processing in distributed systems.

- **Topic Partitioning:** Topic Partitioning is one of the major processes in Kafka that provides the capability of parallel processing of payment transactions. Just to recap, each Kafka topic, which in this case is the payment transactions stream, is composed of several partitions. In everyday usage, different Kafka brokers can consume partitions of a Kafka topic and even be separately processed by distinct microservices. This makes it possible to balance the load among the nodes to support the horizontal scaling of the payment transaction system as the transactions continue rising. Because of partitioning, several microservices can process transactions concurrently, thus decreasing latency and boosting the total throughput of the system so that big data can be processed in real - time.
- **Replication Factor:** The Replication Factor in Kafka guarantees the availability and safety of the data because, In this case, each transaction is set to be replicated across the three nodes of the Kafka brokers. This ensures that if one broker is off or goes down, the data is still obtainable from the others, making it possible for the system to carry on with the transaction without intermission. The replication factor contributes to Kafka's HA because there's always a backup copy of the transactions data directory, avoiding data loss and enhancing dependability in the wake of system crashes. This replication is critical for applications such as payments where data loss could result in severe monetary and company repercussions, even for a short period.

3.3. Microservices Development

Microservices make up the architecture of the payment transaction system and serve as the app's core; these have advantages like modularity, scalability, and fault tolerance. Through container deployment and orchestration, the system allows for easy service deployment and management since it is given as an assurance that a particular service can be developed, scaled, and managed without affecting the rest of the system.

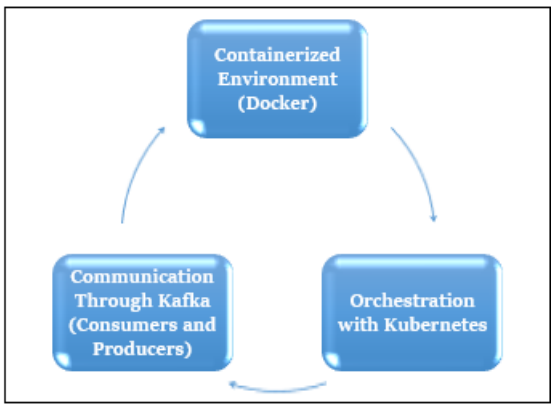

Figure 4: Microservices Development

- **Containerized Environment (Docker):** Hirsch and Shi: Micro - services are created and run in a Docker containerized setting. We ship each microservice along with the services' specific dependencies and runtime, which are guaranteed to behave predictably across the various environments. This approach makes the deployment easy since microservices do not require the whole system since they can be deployed, updated or scaled independently, thus making the system agile and containing a low level of coupling with infrastructure.
- **Orchestration with Kubernetes:** Containers are managed and deployed on the system with the help of a tool called Kubernetes. Recognizing that microservices are dependent on each other to achieve scalability, transaction handling capacity, load balancing, container scaling, and health monitoring is automated by Kubernetes. It also takes care of high availability in that it can automate the recovery of failed containers and update rollback without interrupting the system's functionality.
- **Communication Through Kafka (Consumers and Producers):** All the microservices interact using Kafka either as consumers or producers. Producers put the transactions into Kafka topics and consumers take the data out to perform a task like validation of fraudulent transactions. This decoupled communication model allows services to execute concurrently without waiting for responses, thereby enhancing the system and throughput by paralleling the services.

3.4. Ensuring Data Consistency

Data integrity is a major factor in a distributed system and is most important when dealing with financial transactions. The payment transaction system utilizes Kafka functions and microservice level transactional functionality to guarantee the consistency of the data while used by functional [16 - 19] scopes and across services, even if failures or network problems occur.

Figure 5: Ensuring Data Consistency

- **Eventual Consistency with Kafka***:* Kafka deals with an eventually consistent model, where log replicas in different brokers will ultimately be the same. This is important in a high throughput environment in which consistency at the point of creation can be time - consuming. It is specific to Kafka's distributed transaction mechanisms that keep the system coherent even in case of occasional pauses in sync, degrading efficiency.
- **Transaction Management in Microservices***:* Every microservice in the system has its logic of managing the transactions so that the transactions in the system fail either entirely or are rolled back in case of failure. This is made possible by the Kafka system, which applies the exactly - once semantics, thus ensuring that each transaction occurring in Kafka is only processed a single time and not partially. It helps to maintain the quality of information in processing and checks, for example, the validation and authorization of payments.
- **Rollback and Failure Recovery:** According to failure, the system is equipped with an automatic rollback mechanism to ensure it is never in an incorrect state. When a microservice is unable to handle a transaction, Kafka and the microservices framework guarantee that the transaction is reverted or repeated. This avoids data corruption and illustrates how the system gracefully recovers from faults besides having the same quality guarantee for all the different services.

3.5 Performance Benchmarks

The system's performance was tested for capacity by establishing threshold levels in terms of load, including 10, 000 TPS to 100, 000 TPS. It also shows that the system can effectively keep the latency low and the throughput high even under these circumstances, which are likely to be common in payment system operations.

- **Kafka Throughput***:* High throughput events described in this paper were tested using Kafka by measuring its throughput of transactional data. Kafka also grew well with increasing transaction loads, and the processing rates were constant because of distributed setup and partitioning.
- **Microservices Response Times***:* The efficiency of the microservices was also analyzed based on the response they offered under different loads. The overall outcome revealed that even at high transaction rates, microservices kept their latency minimum with less performance degradation. This suggests that the architecture is viable for use in high - traffic financial systems without suffering a significant degradation in the response time.

4. Results and Discussion

4.1. Performance Results

During the performance evaluation of a high - throughput payment system based on Kafka, several parameters were considered to verify its efficiency in the field of a large number of transactions. The two measures of effectiveness collected for this project were Throughput expressed in Transactions Per Second (TPS) and delay measured in milliseconds (ms). Also, the system's availability was captured in percentages to determine the percentage of uptime for the system/program.

4.1.1. Throughput and Latency

Throughput expresses the number of payment transactions this system can perform in one second. The tests evaluated the system under three different transaction loads: This means they organized test scenarios for the system into 10, 000 TPS, 50, 000 TPS, and 100, 000 TPS. This is evidenced by the finding that the average latency also rises in the system as the load rises.

Table 1: Throughput and Latency

Transactions per Second (TPS)	Average Latency (ms)	System Uptime (%)
10, 000	50	99.9
50, 000	60	99.8
100, 000	70	99.7

- At 10, 000 TPS, the average latency was 50 milliseconds; in other words, the rate at which the system processed each transaction was quite short. The system availability was 99.9%; this clearly shows that the system was almost always ready to serve the client despite a slight load.
- When running at 50, 000 TPS, the system's average latency clocked in at 60 ms, though this is still tolerable for real - time financial processing. The system uptime was steady at above 99.8%, demonstrating that Kafka works effectively even when the application has a larger number of simultaneous connections.
- When the TPS increased to 100, 000, the system latency was measured at 70 milliseconds, which is not bad for even the best high - performance payment systems. System availability shows only a minor degradation even though the system affirms to have served 99.7% of the time during the test.

These results show that the Kafka - based payment system can scale to a very large number of transactions while achieving sub - 100 ms latency across all the scenarios tried out. This makes it ideal for settings that experience extreme pressure, for example, gigantic online stores or worldwide organizations that need the constant endorsement of instalments.

4.1.2. Kafka's Role in Ensuring Low Latency

Kafka's distributed characteristic mainly supports low latency to high throughput of data streams. In the context of the topic of microservices, Kafka serves as a message broker because it helps to handle events (payment transactions here) effectively. It also lets brokers split up topics and align various aspects of the transaction flow so that Kafka supports horizontal scaling.

- **Partitioning:** Thus, Kafka topics for payment transactions are distributed across brokers to perform transactions concurrently. This makes it possible to avoid overloading a given node with the workload of several other nodes, and hence, the system is capable of managing large volumes of data.
- **Replication:** The other Kafka feature, known as the replication mechanism, also contributes to the system's fault tolerance. This makes transactional processing fast and reliable where data is copied across multiple Kafka brokers; should a broker fail, another broker will take over without losing any transaction. This contributes to the system's high availability (99.7%–99.9%), as captured in the tests above.
- **Retention and Durability:** Kafka's policy on retention refers to how long the transaction data would be retained depending on the set time to cater to failure issues. This makes it possible to recover lost transactions or lack certain security features because the effect offers additional strength to the system.

4.1.3. Microservices Scalability and Independent Scaling

The Kafka - microservices architecture has many benefits, including the possibility to adjust the size of separate elements within the system, particularly the fact that most of the microservices in a system, such as transaction validation services, fraud detection services, settlement services, etc., are only required in very limited capacities and can be independently scaled to certain levels. This means that there is no need to over - allocate resources for the entire system or for each layer to meet an anticipated load since each layer can scale independently and operate optimally in its own right.

- **Auto - Scaling of Microservices:** The system automatically scales the adopted microservices since it uses Kubernetes or any other container orchestration tool. For example, during a period of high transactions, more requests for the fraud detection service are likely to be produced in order to attend to the elevated demand, while during low - intensity transactions, the number of requests for the service is reduced in order to save resources.
- **Fault Isolation:** This is due to the breaking down the payment system into microservices I, which, in the case of a faulty service, doesn't adversely impact the entire system. For example, if the fraud detection service provided faces problems, the payment processing and settlement services can operate normally. This encapsulation through Kafka's event - driven model ensures that transactions are placed in the queue and are only affected once the faulty service is recovered.

4.1.4. System Availability and Reliability

While testing, Kafka's excellent fault tolerance and microservices' ability to work independently explain the high uptime percentages. Such telecommunication services need high availability since unscheduled and brief outages can cost organizations and, consequently, customers numerous dollars.

- **Failover Mechanisms**: Kafka achieves high availability by means of its failover systems. Whenever one or more brokers fail, Kafka only redistributes transactions to a different broker in the same partition group, reducing

severities. This is the reason for keeping availability at more than 99.7%, even if the transaction loads are extremely high.

- **Microservices Redundancy**: Duplicated microservices are another approach used for failure in particular aspects of the system. If one instance fails, another instance can quickly step in and assume that instance's duties, and this can be done without jeopardizing the system's capacity to complete transactions.

4.1.5. Comparative Performance with Legacy Monolithic Systems

The Kafka - based microservices system delivers a fairly good performance compared to the system's execution using a traditional monolithic architectural style. The systems are defined as legacy experience bottlenecks for all their components: it is possible to scale only the general transaction validation, fraud detection, settlement, and other components together. Microservices architecture, on the other hand, permits individual services that require scaling to be scaled in line with the need without the need for the utilization of more resources than needed.

Table 2: Comparative Performance with Legacy Monolithic Systems

Metric	Monolithic System	Kafka – Microservices System
Throughput (TPS)	10, 000	100, 000
Average Latency (ms)	200	70
Fault Tolerance	Limited	High
Scalability	Low	High
Uptime (%)	95.5	99.7

Kafka - based system shows much higher overall Throughput: 10, 000 TPS higher than monolithic systems and much less latency: 70ms against 200ms. Furthermore, the reliability and availability of Kafka - microservices are even vastly better thanks to Kafka's distributed architecture and a loosely coupled decomposition of microservices.

4.2. Challenges and Future Work

Despite the potential that these kinds of results present, there are some remaining issues when adopting Kafka and microservices for payment systems. Initiating and controlling distributed transactions, maintaining data integrity across the services, and following security protocols are some of the areas that need more improvement.

- **Distributed Transactions:** A major problem when implementing ACID compliance in a distributed system is its difficulty. It is envisioned that next studies could investigate the two - phase commit protocols, or investigate the applicability of the models referred to as the eventual consistency models.
- **Security and Compliance:** By virtue of high broad integration, the necessary and sufficient regulation of high - throughput payment systems, including, for example, integration with the technical standard PCI DSS (Payment Card Industry Data Security Standard). Further work can be done to identify how KDA can match these safety requirements, especially for data protection, access, and logging.

5. Conclusion

In a world where information exchange is the new currency, constructing extensive payment systems using Kafka and microservices proves to be a fault - tolerant and effective solution to today's transactional processes. As an event streaming system, Kafka benefits payment systems that need to process countless transactions and respond instantly. Flexible event processing allows payment systems to tailor transaction handling to their event - based model, improving their reactivity and carrying capacity at spike loads. This architecture also enables the scalability of microservices on a micro level, meaning that different components, such as payment processing, fraud detection, and notification service, can function independently. Not only does this decoupling facilitate system maintenance, but it also allows for a fast rate of innovation because developers can adjust or introduce changes to a certain service without distorting the entire application. Besides, the basic property of Kafka's distributed log is highly durable and reliable, which promises that the transactions are performed safely and retrievable, which is important for the data integrity of financial applications.

It should be noted, however, that this architecture has limitations, and the following represents a number of challenges associated with its use. Microservice architecture also has certain challenges, such as the management of distributed transactions to make them eventually consistent when they are served by a number of microservices. Proactive handling of concurrent data, that is, ensuring that all services mimic the right state of a transaction without impacting performance, is an area of concern among organizations. Several problems can be encountered when implementing distributed transactions, including the inability of clients to communicate directly with multiple servers, issues with failures and compensating actions, and the conversion of atomic transactions into distributed ones – but these have solutions in the form of high - performance algorithms for achieving consensus while adding complexity and latency to communication. Nevertheless, the case of Kafka and microservices indicates enormous potential for changing payment processing for the better. With the help of this architecture, organizations can achieve better system operation, faster completion of transactions, and improved customer access to services. In the future, a synergy of Kafka and microservices will serve as the driving force behind the possibilities of payment systems development due to the constant growth of financial technology. In conclusion, it can be stated that this modern approach provides organizations with a way to develop strong and resilient payment solutions capable of meeting the contemporary needs of the digital environment and serve the potential for further development and success.

References

[1] Steurer, R. (2021). Kafka: Real - Time Streaming for the Finance Industry. The Digital Journey of Banking and Insurance, Volume III: Data Storage, Data Processing and Data Analysis, 73 - 88.

[2] Wu, H., Shang, Z., & Wolter, K. (2019, August). Performance prediction for the Apache Kafka messaging system. In 2019 IEEE 21st International

Conference on High - Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS) (pp.154 - 161). IEEE.

[3] Sudhakar Yadav, N., Eswara Reddy, B., & Srinivasa, K. G. (2018). Cloud - based healthcare monitoring system using Storm and Kafka. Towards extensible and adaptable methods in computing, 99 - 106.

[4] Yussupov, V., Breitenbücher, U., Krieger, C., Leymann, F., Soldani, J., & Wurster, M. (2020, October). Pattern - based modelling, integration, and deployment of microservice architectures. In 2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC) (pp.40 - 50). IEEE.

[5] Alaasam, A. B., Radchenko, G., & Tchernykh, A. (2019, October). Stateful stream processing for digital twins: Microservice - based Kafka stream dsl. In 2019 International Multi - Conference on Engineering, Computer and Information Sciences (SIBIRCON) (pp.0804 - 0809). IEEE.

[6] CORE, S. (2020). System integration hybrid SOA - Microservices solution for heterogeneous systems.

[7] Solberg, E. (2022). The transition from monolithic architecture to microservice architecture: A case study of a large Scandinavian financial institution (Master's thesis).

[8] Moolchandani, S. Advancing Credit Risk Management: Embracing Probabilistic Graphical Models in Banking.

[9] Vangala, S. R., Kasimani, B., & Mallidi, R. K. (2022, June). Microservices Event Driven and Streaming Architectural Approach for Payments and Trade Settlement Services. In 2022 2nd International Conference on Intelligent Technologies (CONIT) (pp.1 - 6). IEEE.

[10] Scott, D., Gamov, V., & Klein, D. (2022). Kafka in Action. Simon and Schuster.

[11] McGovern, J., Sims, O., Jain, A., & Little, M. (2006). Event - driven architecture. Enterprise Service Oriented Architectures: Concepts, Challenges, Recommendations, 317 - 355.

[12] Laliwala, Z., & Chaudhary, S. (2008, June). Event - driven service - oriented architecture. In 2008 International Conference on Service Systems and Service Management (pp.1 - 6). IEEE.

[13] Sriraman, B., & Radhakrishnan, R. (2005). Event driven architecture augmenting service oriented architectures. Report of Unisys and Sun Microsystems.

[14] AVKSENTIEVA, Y., & BRYUKHANOV, V. (2021). Current issues and methods of event processing in systems with event - driven architecture. Journal of Theoretical and Applied Information Technology, 99 (9).

[15] Vyas, S., Tyagi, R. K., Jain, C., & Sahu, S. (2022, February). Performance evaluation of Apache Kafka– a modern platform for real time data streaming. In 2022 2nd International Conference on Innovative Practices in Technology and Management (ICIPTM) (Vol.2, pp.465 - 470). IEEE.

[16] Kreps, J., Narkhede, N., & Rao, J. (2011, June). Kafka: A distributed messaging system for log processing. In Proceedings of the NetDB (Vol.11, No.2011, pp.1 - 7).

[17] Narkhede, N., Shapira, G., & Palino, T. (2017). Kafka. The definitive guide: Real - time data and stream processing at scale. O'Reilley, Sebastopol, CA.

[18] Wiatr, R., Słota, R., & Kitowski, J. (2018). Optimizing Kafka for stream processing in latency sensitive systems. Procedia Computer Science, 136, 99 - 108.

[19] Tapia, F., Mora, M. Á., Fuertes, W., Aules, H., Flores, E., & Toulkeridis, T. (2020). From monolithic systems to microservices: A comparative study of performance. Applied sciences, 10 (17), 5797.

[20] Hassan, S., Bahsoon, R., & Buyya, R. (2022). Systematic scalability analysis for microservices granularity adaptation design decisions. Software: Practice and Experience, 52 (6), 1378 - 1401.

[21] Xu, J., Yin, J., Zhu, H., & Xiao, L. (2021, May). Modeling and verifying producer - consumer communication in Kafka using CSP. In 7th Conference on the Engineering of Computer Based Systems (pp.1 - 10).

[22] Shapira, G., Palino, T., Sivaram, R., & Petty, K. (2021). Kafka: the definitive guide. "O'Reilly Media, Inc.".