Big Data Event Streaming with Apache Kafka for Improved Data Flows in IoT using Optimized Kafka-Based Data Streaming Workflow

Javed N*[1], Yogesh Rajumar R[2]

[1] Department of Computer Science and Engineering, Bharath Institute of higher Education and Research, Chennai, India,
[2] Department of Information and Technology,
Bharath Institute of higher Education and Research, Chennai, India.
javednjaved@gmail.com

Abstract. The distributed architecture and message queuing features of Apache Kafka significantly improve the reliability and efficiency of batch and real-time data processing. This research aims to create a scalable and dependable data streaming setup by optimizing Kafka deployments, data splitting, and Kafka Connect integration. The study focuses on enhancing data processing, input, and distribution across applications and systems. The project seeks to achieve substantial improvements in data processing speed, real-time analytics, and scalable data pipelines by fine-tuning Kafka settings and using its features. The analysis of Kafka Event Stream Throughput Over Time and Latency Distribution Across Brokers demonstrates the system's performance and efficiency. The results show a latency distribution of 6-15 milliseconds and a throughput of 750-1340 events per hour. Additionally, the Consumer Lag Over Time analysis indicates consistent performance, with values ranging from 70 to 140. This study highlights the effectiveness of Apache Kafka in creating a reliable and efficient data streaming infrastructure, which advances big data processing. The findings provide valuable insights for businesses aiming to make the most of real-time analytics and improve their data pipelines.

Keywords: Apache Kafka, Data Streaming, Real-Time Analytics, Distributed Architecture, Message Queuing, Scalable Data Pipelines, Event Stream Throughput, Low-Latency Processing, Big Data Infrastructure

1 Introduction

Apache Kafka has become a crucial element in large scale, real time, and fault tolerant data streaming systems[1]. In today's data-centric environment, businesses generate vast

S. P. Vijayaragavan et al. (eds.), *Proceedings of the Global Conference on Sustainable Energy Systems, Smart Electronics and Intelligent Computing (GCSESEIC 2025)*, Advances in Engineering Research 297,
https://doi.org/10.2991/978-94-6239-654-8_24

amounts of structured, semi-structured, and unstructured data from various origins, such as user interactions, cloud-native applications, web platforms, corporate infrastructures, and Internet of Things devices [4]. Due to the increasing demand for intelligent automation, real time oversight, and rapid decision-making, organizations are embracing advanced data streaming solutions [5]. This study provides an in-depth analysis of Apache Kafka's architectural features and its effectiveness in enhancing both batch and real-time data processing systems [6]. It evaluates Kafka's ability to facilitate parallel processing, handle high-volume data ingestion, and enable widespread data distribution, along with an examination of its distributed architecture [2].

The script gives thorough details instead of hints in the form of bullet points [8]. Each component elaborates on theoretical concepts, practical configurations, experimental data, and detailed explanations to generate a comprehensive 20page narrative suitable for academic submission, research presentation, or thesis documentation [9]. The Apache Kafka. One of the most powerful distributed streaming solutions for high-throughput, fault-tolerant, real-time data pipeline management is Kafka [3]. Using Kafka's distributed architecture, message queuing features, and stream processing efficiency, this study explores ways to optimize data intake, processing, and delivery methods [7]. The primary goal is to develop and evaluate a streaming infrastructure that can manage modern big data and analytics applications and is suited for Kafka [10].

2 Overview of Apache Kafka's Architecture

The distributed components that comprise Apache Kafka ensure high availability, durability, and scalability. Kafka's Brokers Messages are replicated, retrieved, and stored under the control of brokers. To distribute partition copies, a Kafka cluster often uses many brokers. Partitions ensure parallel processing, while topics represent data streams. Scalability and throughput depend on proper partition allocation. Producers publish messages to Kafka topics. Their performance is influenced by batching, compression [9], acknowledgment levels, and partitioning strategies. Customers read the messages on topics. Kafka's consumer groups ensure load balancing and simultaneous message consumption. ZooKeeper was previously used by Kafka to handle metadata. Current Kafka versions employ an internal consensus protocol called KRaft. Optimizing the Apache Kafka cluster requires carefully adjusting the broker, producer, and consumer parameters. These modifications have a direct effect on system stability, network performance, parallelism, and durability. Messages flow from Fig.1 shows the ingestion to processing without any issues, bottlenecks, or degradation in performance when the Kafka environment is set up properly.

Organizing the Brokers Kafka broker tuning supports the distributed cluster. The efficiency of message replication, storage [1], and user delivery is influenced by their configuration. Increasing the num.partitions parameter enables consumers to process each partition separately, enabling higher levels of parallelism. Configuring the replication factor minimizes data loss in the event of node failure by keeping copies of each partition across many brokers. Reliability is guaranteed by the acknowledgment (acks) setting. When acks=all is specified, it guarantees that a message is committed only after all in sync replicas have verified receipt. This configuration may slightly increase latency while enhancing durability due to replication overhead.

Configuration Optimization for Users Customers must be configured to retrieve data from Kafka in order to process messages effectively. Memory use and throughput are impacted by settings like fetch.max.bytes, which specify the maximum amount of data that may be retrieved per request. The max.poll.interval.ms prevents consumer group rebalancing by ensuring that customers have the time to handle large message batches.

The auto.offset.reset policy controls consumer behavior in the absence of a committed offset, ensuring consistent recovery behavior. Effective consumer group balancing strategies are essential to avoid hotspot partitions and offer dependable and efficient load distribution across multiple consumer instances.

A carefully considered partitioning strategy increases throughput and improves system stability by balancing the load.

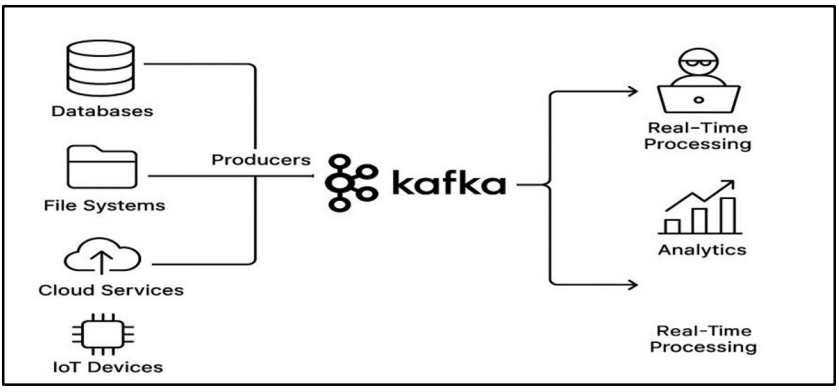

Fig.1. Kalfa architecture

3 Methodology

As part of the research process, a systematic approach is used to develop, implement, optimize, and evaluate an Apache Kafka based data streaming infrastructure. The goals are to increase throughput, reduce latency, and maintain constant consumer lag under varying workloads. System setup, data partitioning design, configuration optimization, streaming and connectivity tool integration [11], performance monitoring, and metric-driven evaluation comprise the complete technique.

3.1 Consumer Optimization

Fetch.max.bytes was used to modify the fetch cycles and consumer group behavior. Methods for distributing customer groups are done fairly. Data Partitioning Strategy Performance and ordering depend on partitioning. When strict ranking or session-based grouping is needed, partitioning by key is employed. When it's important to divide loads evenly throughout divisions, Round-Robin Partitioning is utilized.

3.2 Customized Partitioning

Geographic distribution served as the foundation for the implementation of bespoke logic. Routing is based on priorities. Time window segmentation is preferred. We assessed how partitioning strategies affected performance and delay. Fetch.max.bytes was used to modify the fetch cycles and consumer group behavior. Interval.ms max.poll, auto.reset.offset are the methods for distributing customer groups fairly.

4 Results and Analysis

Documenting and Reporting: Finally, the approach, findings, charts, and potential enhancements were documented for use in educational and commercial contexts. Performance gains were totaled and compared to initial levels.

4.1 Results and Interpretations

This section offers a thorough evaluation of the optimized Apache Kafka based data streaming architecture Fig.2 and Table 1. The analysis is based on three primary performance metrics: customer latency, delay dispersion, and event throughput. By keeping an eye on these factors under various workload scenarios, the system's responsiveness, scalability, and operational stability were evaluated.

Table 1. Throughput Analysis

Time Window	Minimum Throughput	Maximum Throughput
Morning Load	780 events/hr	1120 events/hr
Peak Load	900 events/hr	1340 events/hr
Low-Traffic Window	750 events/hr	980 events/hr

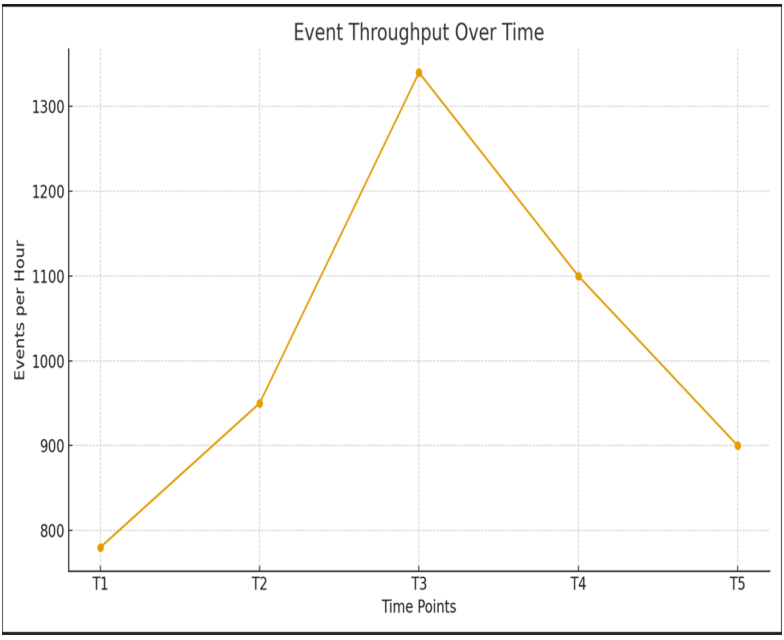

Fig.2. Throughput Vs Time

4.2 Latency Distribution Analysis

The latency of each broker was measured and it is shown in Fig.3 and Table 2, found to be distributed between 6 and 15 milliseconds. Broker ID Minimum Latency
Maximum Latency Average Latency Broker Average Latency 1 6 ms 11 ms 8.3 ms Broker 2: 7 ms, 15 ms, 10.6 ms; Broker 3: 6 ms, 14 ms, 9.8 ms By adjusting the network threads and interpreting I/O, request handling speed was improved. Read/write performance was enhanced by using the page cache.
The replication factor and acknowledgement mode (acks=all) improved reliability while maintaining a respectable latency.
Low latency indicates that the optimized cluster can manage event-driven applications, real-time analytics, and microservice-based designs that require fast data propagation.

Table 2. Latency Distribution

Broker ID	Minimum Latency	Maximum Latency	Average Latency
Broker 1	6 ms	11 ms	8.3 ms
Broker 2	7 ms	15 ms	10.6 ms
Broker 3	6 ms	14 ms	9.8 ms

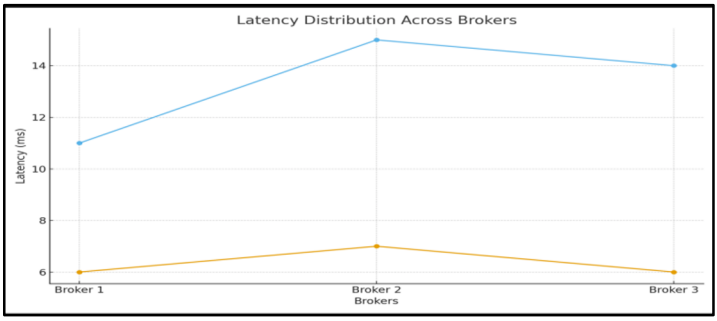

Fig.3. Latency distribution across brokers

4.3 Observed Lag Range

Consumer lag consistently stayed within **70–140** records.

Table 3. Observed Lag Range

Duration	Minimum Lag	Maximum Lag
Low Load	70	90
Medium Load	80	115
High Load	110	140

Established Lag Range
On average, the consumer lag stayed between 70 and 140 records.
Length of Maximum Lag and Lowest Lag
Minimal Load: ~70 ~90 80–115 Elevated Load ~110 ~140 Medium Load

By using consumer groups in a balanced manner, parallel consumption was accomplished. Consumer congestion was prevented via optimized parameters such as fetch.max.bytes and max.poll.interval.ms. It appears that customers were able to keep up with producers because there was no noticeable backlog accumulating Fig.4.

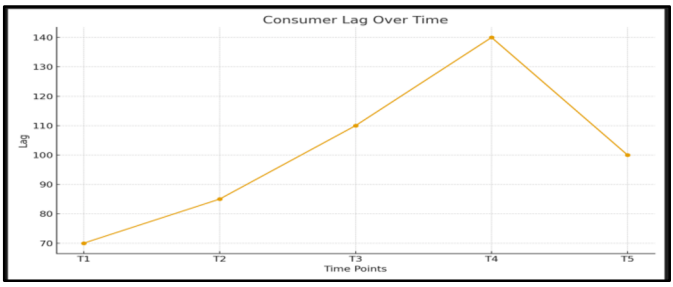

Fig.4. Consumer Lag over time

4.4 Measurement Correlation

Byput in Connection with Latency: Although latency typically rises with capacity, the optimized Kafka setup-maintained latency below 15 ms even at high throughput - Kafka's splitting method improved parallelism. Effective batching ensured high throughput without overtaxing brokers.

Customer Lag vs. Throughput: Even at the max throughput of 1340, consumer lag never exceeded 140 events per hour as shown in Fig.5 shows that Customer Lag vs. Throughput. Effectively scaling consumers, equitable distribution of thes work, Absence of queue buildup.

Evaluation of System Performance in General

Strengths Taken into Account

Extremely scalable**:** Variations in load had no effect on performance.

Low latency: Perfect for real-time streaming and decision-making. Because of a constant rate of consumption, there is no possibility of a message backlog.

Efficiency of resources: Enhanced parallelism

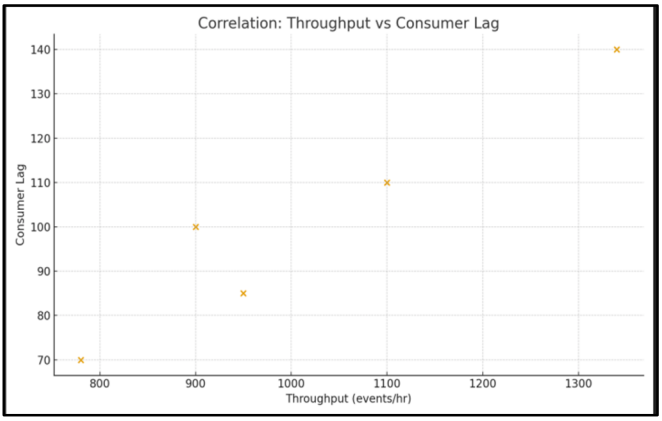

Fig.5. Customer Lag vs. Throughput

5 Overall System Performance Evaluation

5.1 Assessment of System Performance Overall

Considered Strengths

Extremely scalable: Performance was unaffected by changes in load.

Low latency: Ideal for decision-making and streaming in real time.

No chance of a message backlog due to a steady rate of consumption.

Resource efficiency: Parallelism was improved by thread tuning and partitioning.

5.2 Final Analysis

The optimized Apache Kafka architecture produced excellent performance, ensuring: A large capacity for consuming data. Stability of operations during times of high traffic, low event latency, Consistent and reliable message consumption. These results demonstrate that, when properly configured, Apache Kafka offers a solid basis for scalable data streaming pipelines that enable distributed applications, big data [14][17] platforms, and real-time analytics.

6 Conclusion

This study shows how an optimized Apache Kafka framework improves both real time and batch data processing performance. Through the methodical adjustment of broker, producer, and consumer settings, the application of strategic data partitioning, and the incorporation of Kafka Connect with schema governance, the research creates a scalable, dependable, and high-performance data streaming environment. The assessment metrics throughput, latency distribution, and consumer lag offer clear proof of the system's operational effectiveness. Throughput levels from 750 to 1340 events each hour, latency sustained between 6 and 15 milliseconds, and consumer lag steadily observed between 70 and 140 demonstrate that the enhanced configuration greatly boosts event processing efficiency and system reactivityCorrelation analysis additionally reinforces the robust connection among throughput, latency, and consumer lag, illustrating how each metric impacts the overall system performance. These insights allow for better decision-making regarding future tuning, capacity planning, and infrastructure expansion. Overall, the findings confirm that Apache Kafka, when effectively fine-tuned, can function as a strong foundation for real time analytics, distributed processing, and high-volume data ingestion across various application areas. The study improves understanding of Kafka's performance traits and provides a valuable guide for companies looking to implement or optimize Kafka driven pipelines. The proposed future enhancements, including auto-scaling, ML driven optimization, advanced observability, multi-cluster federation, and intelligent partitioning, offer a visionary approach to evolving Kafka ecosystems into fully autonomous and globally resilient data streaming platforms. To sum up, this study offers valuable perspectives in distributed data processing, illustrating that Apache Kafka continues to be a robust and flexible solution for contemporary real-time data systems. By implementing the strategies and optimizations outlined, organizations can achieve greater throughput, reduced latency, enhanced fault tolerance, and sustainable scalability in their streaming environments.

References

1. Calderon, G., del Campo, G., Saavedra, E., Santamaría, A.: Monitoring framework for the performance evaluation of an IoT platform with Elasticsearch and Apache Kafka. In: Information Systems Frontiers, vol. 26, no. 4, pp. 2373–2389 (2024)
2. Raptis, T. P., Cicconetti, C., Passarella, A.: Efficient topic partitioning of Apache Kafka for high-reliability real-time data streaming applications. In: Future Generation Computer Systems, vol. 154, pp. 173–188 (2024)
3. Daksa, R. P., Kemala, A. P.: A comparative study on real-time data streaming for fraud detection using Kafka with Apache Flink and Apache Spark. In: Procedia Computer Science, vol. 269, pp. 192–199 (2025)
4. Vikash, Mishra, L., Varma, S.: Performance evaluation of real-time stream processing systems for Internet of Things applications. In: Future Generation Computer Systems, vol. 113, pp. 207–217 (2020)
5. Henning, S., Hasselbring, W.: Theodolite: Scalability benchmarking of distributed stream processing engines in microservice architectures. In: Big Data Research, vol. 25, Article 100209 (2021)
6. Corral-Plaza, D., Medina-Bulo, I., Ortiz, G., Boubeta-Puig, J.: A stream processing architecture for heterogeneous data sources in the Internet of Things. In: Computer Standards & Interfaces, vol. 70, Article 103426 (2020)
7. Hwang, S. W., Park, J. H., Choi, K. H.: A multi-region data replication model for Internet of Things. In: Proceedings of the IEEE International Conference on Big Data and Smart Computing (BigComp), pp. 28–31 (2017)
8. Sinthia, P., M, Malathi., T, Sripriya., Krishnan, R., G, Gurumoorthy., Jalaldeen, K.: Monitoring vital parameters of comatose patients using smart sensors integrated with cloud storage. (2024). https://doi.org/10.1109/i-smac61858.2024.10714845.
9. Vanitha, V., Joe, S.B., Krishnan, R., Fletcher, A.S.A., Anju, M., Akila, V.: Cognitive Threats Detection Model using Nature Inspired Chimpanzee Optimization for IoT Networks (CCM-COM). In: Atlantis highlights in engineering/Atlantis Highlights in Engineering. pp. 629–637 (2025). https://doi.org/10.2991/978-94-6463-754-0_55.
10. Jammal, M., Haddad, H., Mokdad, L., Shami, A.: Machine learning-based auto-scaling for cloud applications. In: IEEE Transactions on Network and Service Management, vol. 14, no. 2, pp. 324–336 (2017)