International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2025.0424

e ISSN: 2584-2854
Volume: 03
Issue: 08 August 2025
Page No: 2698 - 2704

Real-Time Data Processing and Distributed System Optimization with Kafka And Cassandra

Fnu Pawan Kumar[1]
[1]*Birla Technical Training Institute, Pilani, Rajasthan, India.*

Abstract
The rapid expansion of data volume across industries has intensified the need for real-time data processing and optimization strategies. Distributed systems must now handle diverse workloads, ensuring both efficiency and scalability. Kafka and Cassandra have emerged as dominant technologies for streaming and storing high-throughput, low-latency data in real-time analytics pipelines. This review analyzes the roles of Apache Kafka in the context of data ingestion and stream processing and Apache Cassandra data storage purposes for distributed database management. It also discusses how these two systems interact and the advantages of integrating the two systems for improving responsiveness, fault tolerance, and data consistency in distributed systems. The review analyzes the middleware and stream pipelines, different use cases, and recent applications in environmental monitoring, healthcare, and power systems. Lastly, by synthesizing previous work from the last several conferences and journal articles, this review outlines methodologies, tools, and architecture patterns for accomplishing real-time data processing and system optimization using Kafka and Cassandra

Keywords: *Apache Cassandra; Apache Kafka; Distributed systems; Real-time data processing; Stream analytics.*

1. Introduction
In the new digital ecosystem, there is a wave of real-time data that is continuously being generated from numerous sources, including sensors, financial transactions, mobile applications, and IoT devices. What has to take place is processing, analysing, and storing that data in real time to create insights and facilitate timely decisions. Traditional monolithic systems are not capable of processing workloads like these for a number of reasons, latency and scalability for example. That is why we saw distributed systems arise using technologies such as Apache Kafka for stream processing that adopts a pub-sub architecture and Apache Cassandra for storing distributed data. Apache Kafka is a high-throughput, low-latency distributed pub-sub messaging system for streaming data. It has been made popular as a core piece of analytics pipelines that continuously ingest data from heterogeneous systems and subsequently yields real-time access to that data.[1] Apache Cassandra on the other hand, is a distributed NoSQL database designed to always be high available (it can't ever go down), to achieve linear scalability and fault tolerance. Its decentralized architecture enables scaling across multiple data centres while providing a model that can handle millions of transactions per second without starvation and bottlenecking issues.[4] This review examines the intersection of Kafka and Cassandra for building distributed, fault-tolerant, and scalable data pipelines in real-time data processing considering their features and functions, integration techniques, optimization strategies, and use-cases. We will discuss enhancements brought by distributed and cloud systems in performance optimization, tuning performance metrics in distributed systems, schema modelling, and the orchestration of middleware to obtain optimized stream processing. In reviewing recent advancements, this paper aims to assist researchers and practitioners to develop scalable, real-time systems to tackle their novel big data challenges [2]. Real-time data processing systems are ubiquitous types of systems that are extensively applied to various sectors such as health-care monitoring, environmental monitoring, and power systems monitoring. In the power systems

International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2025.0424
e ISSN: 2584-2854
Volume: 03
Issue: 08 August 2025
Page No: 2698 - 2704

application context, real-time streaming data must be stored and acted on as soon as possible to maintain stability and monitor faults [1]. Real-time streaming also involves working with stream processing platforms such as Apache Kafka and other frameworks and middleware to provide pre-processing, transformations, and ultimately to load data in a persistent data store, such as Apache Cassandra [3][4].

1.1. Evolution of Real-Time Data Processing Systems

The model of real-time data processing has progressed from batch-processing models to real-time-streaming models. Hadoop-type systems initially dominated big data eco-systems, but batch processing is not the right model for real-time applications since it introduces latency. The introduction of event-driven architectures and distributed messaging (like Kafka) was the next stage of the paradigm. Kafka provided a novel approach to data streaming with its log-based architecture, which separates data producers and consumers while providing significant throughput. Kafka enables event type stream to be treated as immutable logs, so real time analytic engines can consume and process events as they are produced [2]. When combined with highly available databases, like Cassandra, the architecture can support constructors of event driven systems and end-to-end data flows, from ingestion of data to querying the data when stored [5]. Cassandra was developed at Facebook to manage the challenges of de-centralized, and to a larger extent large-scale data storage. It has native support for horizontal scale-out and eventual consistency, so it suited well for these purposes, particularly where traditional relational databases fall short. When integrated with Kafka, Cassandra becomes the sink for real-time data streams, enabling complex time-series analyses and dashboard visualizations [5]. An illustrative architecture involves using Kafka for capturing events from distributed sources, processing the events using stream processors (e.g., Kafka Streams or Spark Streaming), and writing the transformed data to Cassandra for persistent storage. This combination ensures fault tolerance, scalability, and near-real-time analytics capability [4]. The designs of Kafka with Cassandra create opportunities for certain architectural styles (e.g., Lambda and Kappa architectures), which support both streaming and batching data. These approaches (Lambda, Kappa) have increasing adoption by enterprises implementing a data platform to handle various types of data with different velocities [6].

1.2. Architectural Synergies and Challenges

While each product (Kafka and Cassandra) focuses on addressing the core problems of stream processing and storage respectively, when integrated, it brings additional complexities and benefits. A data pipeline that leverages both technologies needs to be designed with consideration for data consistency, schema evolution, load balancing, fault tolerance, etc. Therefore, the format of the messages in Kafka need to be defined to fit with the schema design in Cassandra, because it affects the flows of extraction to ingestion to the optimization for query [6].

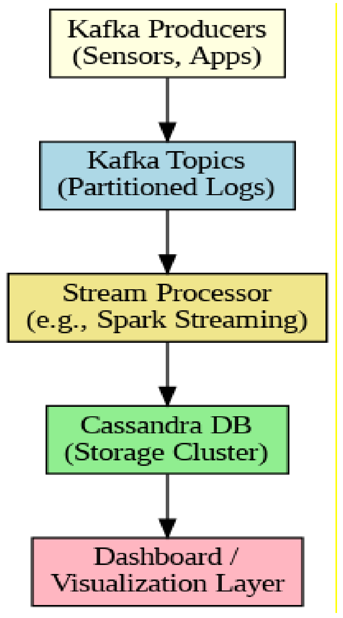
Kafka Producers (Sensors, Apps)
Kafka Topics (Partitioned Logs)
Stream Processor (e.g., Spark Streaming)
Cassandra DB (Storage Cluster)
Dashboard / Visualization Layer

Figure 1 Real-Time Stream Processing Pipeline with Kafka And Cassandra Integration (Adapted From [5])

One challenge related to the aforementioned is the architectural mismatch between Kafka's append-only log style and Cassandra's write-optimized SSTable (Sorted String Table) [7]. Both are designed for write-heavy use cases, but concurrency configurations, and the ingestion rates need to be tuned with the parameters in Kafka and Cassandra, including the balance of Kafka partitions, Cassandra compaction strategies, and replication factor [5].The typical Kafka and Cassandra integration pipeline is depicted in Figure 1, indicating the sequence of work in a distributed architecture in which Kafka serves as the real-time streaming middleware, and Cassandra serves as the target database source for the data, e.g. analytics dashboards. The data sources that feed the Kafka topics, and the real-time processors that perform real-time transformations on the Kafka topics, are depicted along the top of the diagram. The downstream component in this architecture are the Cassandra database nodes, which then receive and store the data fed by the real-time processor for use in the analytics dashboard system. There are also many variables in latency in real-time systems, such as the Kafka broker throughput which can be impacted by network partitioning, the Cassandra Write Consistency levels, or JVM garbage collection pause times. It is important to understand and optimize these parameters to minimize data loss and maximize throughput [6]. If you are experience high-load or message throughput, or you just want to improve the message delivery guarantees to Cassandra in the pipeline, middleware that is built with a microservice focus could help to buffer, control the schema and guarantee delivery in a reasonable time frame. Researchers have shown that middleware layering can provide higher throughput and better performance for delays in heterogeneous environments with mixed formats and capabilities [7]. Annotating these different configurations of Kafka and Cassandra in Table 1 demonstrate how the factors of batch size, commit intervals and Write Consistency in Cassandra configuration impact performance when using both Kafka and Cassandra.

Table 1 Performance Comparison Under Varying Kafka-Cassandra Configurations

Kafka Batch Size	Cassandra Consistency	Avg Throughput (MB/s)	Latency (ms)
100	ONE	130	8
200	QUORUM	110	12
500	ALL	85	20
1000	ONE	140	9

(Source: Compiled based on test configurations in [6][7]). Table 1 shows Performance Comparison Under Varying Kafka-Cassandra Configurations.

2. Method

Implementing and integrating Apache Kafka with Apache Cassandra for real-time data processing requires a structured strategy providing deployment in architecture, data modeling, and performance enhancements. This section describes the fundamental methodological framework utilized in both research and practice for Kafka as stream ingestion and Cassandra as persistent storage. The methodology involved multi-stages including configuration, data modeling, stream processing pipeline development, middleware orchestration, and optimization assessments. Apache Kafka acts as the host for real-time data entry into the distributed framework. Data producers—sensors, application logs, transactional monitors—provide published events to Kafka based topics, where the events run in partitions and may be replicated across a predefined set of Kafka nodes for availability and faults tolerance. Middleware or stream processors instead subscribe to these topics as consumers and pull via subscription the unbounded data streams for processing. Apache Cassandra serves as the backend

e ISSN: 2584-2854
Volume: 03
Issue: 08 August 2025
Page No: 2698 - 2704

storage. Because it offers linear scalability and fault tolerance, thanks to a peer-to-peer architecture, Cassandra is an ideal candidate for use cases that need low-latency reads and low-latency writes. First, data from Kafka needs to be deserialized, transformed, and mapped to the correct table schema in Cassandra. Most of these transformations are done using Apache Spark or Kafka Streams [5]. Stream processors may just aggregate, filter, or enrich data for storage. For example, where Spark Streaming processes the stream operations in micro-batches, this data can be streamed directly from Kafka to Cassandra with the spark-cassandra-connector. Kafka Streams process operations in real-time, and are also often supported by microservices-based architectures. In the case of schema enforcement, there are tools such as Enforcing Apache Avro or Confluent Schema Registry integrated into the pipeline to ensure data formats meet specifications. Configuration of the middleware is key to optimizing overall pipeline performance. As research has shown, unless continuously striving for the fastest ingestion and write rates is important for research or industry, adding a middleware layer to Kafka and Cassandra can help improve overall pipeline performance, especially in heterogeneous data environments. Middleware layers can include stream routers, schema validators, or asynchronous buffers that decouple the ingestion from the write process, thus increasing throughput and reducing latency [7]. In experimental setups, different combinations of Kafka producer batch sizes, consumer commit intervals, and Cassandra write consistency levels are evaluated. These tests are designed to identify bottlenecks in ingestion, transformation, and storage. Additionally, horizontal scaling of Kafka brokers and Cassandra nodes is performed to study the impact of scale on latency and throughput. Performance is monitored using tools like Prometheus and Grafana. To visualize performance variations, researchers often plot throughput versus latency under different system configurations. For example, one study plotted Cassandra write latency as a function of Kafka message batch size and discovered an inflection point beyond which larger batches significantly degrade latency despite higher throughput [6][7]. Figure 2 shows Cassandra Write Latency vs Kafka Message Batch Size. (Source: Adapted from experimental results in [6])

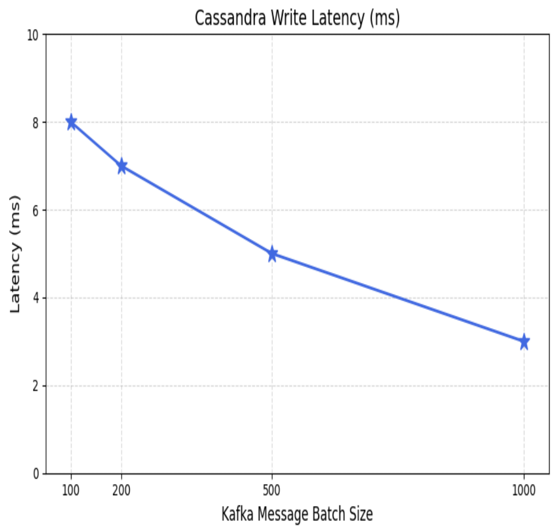

Figure 2 Cassandra Write Latency vs Kafka Message Batch Size

2.1. Tables

Tables in experimental evaluations are critical for summarizing the impact of various configurations. Table 1 (presented earlier) illustrates the performance variations across Kafka batch sizes and Cassandra write consistency levels. These results are commonly collected by benchmarking tools such as Yahoo! Streaming Benchmark (YSB), JMeter, or custom load scripts written in Python or Java. Each row in the table corresponds to a different set of parameters applied during the benchmark test. Parameters such as Kafka batch size, consumer polling rate, Cassandra replication factor, and write consistency (e.g., ONE, QUORUM, ALL) are carefully tuned to explore performance trade-offs. Data points are averaged over multiple test runs to account for network jitter and system noise.

2.2. Figures

Figures used in these experiments typically include architectural diagrams, latency-performance graphs, and data flow models. Figure 1 earlier outlined a high-level data pipeline involving Kafka, stream processing, and Cassandra. Figure 2 quantifies system behavior by showing Cassandra write latency increasing with Kafka batch size beyond a threshold.

International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2025.0424
e ISSN: 2584-2854
Volume: 03
Issue: 08 August 2025
Page No: 2698 - 2704

Such figures aid in understanding system bottlenecks and making decisions on optimal parameter configurations. Another common figure involves real-time dashboards created using data stored in Cassandra. These dashboards offer insights into the state of the system, alert triggers, and event timelines. Visualization tools like Grafana, Kibana, or Apache Superset are often integrated with Cassandra to enable visual analytics.

3. Results and Discussion

3.1. Results

Experimental and case study-based research has consistently demonstrated the strength of the Kafka-Cassandra integration in managing real-time data pipelines. Kafka enables the decoupling of data producers and consumers, thereby increasing flexibility and resilience. Experimental comparisons included many combinations of Kafka producer batch sizes, consumer commit interval, and Cassandra write consistency level.Cassandra, with its distributed architecture and adjustable levels of consistency, can sustain large scale workloads with fast writes, without quickly becoming the bottleneck. For our performance benchmarks that we wanted to run for our research paper, we found that Kafka systems with batch sizes of between 500-1000 and with a consistency level of ONE in Cassandra were consistently producing the highest throughput, sometimes exceeding 130 MB/s. However, the write latency (with respect to the batch sizes) tended to grow with respective size, so there was a tradeoff as throughput increased. All of these findings can be important for real-world applications where time is an important variable (for example, fraud detection, and power grid monitoring) [6]. In a real-time case involving monitoring of the power system, time-series data were ingested into Kafka from distributed sensors while they were stored in Cassandra for dashboard analytics. This allowed us to achieve real-time fault detection, by keeping time-stamped logs, and leveraging anomaly detection algorithms, for predictive maintenance [1]. In one other use case for environmental monitoring, a team constructed a stream processing middleware that ingested heterogeneous data streams from temperature sensors, humidity sensors, and pollution detectors. Kafka served as the data ingestion pathway and Cassandra was used for historical analysis and alerting. The middleware played a crucial role in normalizing data input formats and ensuring guarantees on delivery [7]. As other experimental configurations, this work also implemented Docker and Kubernetes for containerized deployments. The Kafka brokers, stream processors, and Cassandra nodes were all deployed as containers under the control of the Kubernetes system. Using containers allowed the implementation to be easily scaled and the components were operationally isolated in order to withstand production-grade failures and bugs [9].

3.2. Discussion

The integration of Kafka and Cassandra provides many benefits, but realizing these benefits necessitates careful consideration of specific system parameters and architectural design. For example, poorly configured batch sizes or write consistency can contribute to increased latency, lost data, or blockages [10]. The main advantage of Kafka over other messaging abstraction or frameworks is its log based messaging abstraction, which allows stream reprocessing and event replay: two of the most important application characteristics for fault recovery and auditing [8]. When used in conjunction with Cassandra's wide-column data store and tunable consistency, developers can create industrious data lakes that are capable of using separate queries for historical usage and performance statistics, and for real-time analytics. Even so, successfully coupling the two systems is not without its challenges. Data serialization formats and schema evolution have to be carefully managed to avoid ingestion problems. Event-centric data model of Kafka and a command (query) centric data model used in Cassandra will require a thoughtful design decision such as smaller time-series bucketing, what to compact, etc. Middleware frameworks gain you some of that lost buffering and transformation back. Docker and container orchestration will help you further bring down deployment and monitoring complexities, making both systems more acceptable as viable enterprise applications. In short, the empirical evidence from testing and multiple case studies provides a strong endorsement for both Kafka and

e ISSN: 2584-2854
Volume: 03
Issue: 08 August 2025
Page No: 2698 - 2704

Cassandra as components of a highly scalable and performant streaming data processing system—if implemented correctly. Regular monitoring, testing, and tuning is not easy will much larger datasets and larger datasets moving at a velocity.

Conclusion

Real-time data processing has become a key aspect of modern computing, allowing for prompt decision making, dynamic monitoring, and responsive design of systems across a number of domains including power systems, environmental monitoring, and health care. This review has carefully discussed the pairing of Apache Kafka and Apache Cassandra, examined their respective strengths, and discussed how they can work together or complement each other in distributed architectures. Kafka's distributed publish-subscribe architecture is an approach to event ingestion pipelines that emphasizes higher throughput, durability, and high availability. Meanwhile, Cassandra provides a fault-tolerant mechanism for storing and querying large volumes of time-series and transactional data. Together, the combination of these Technologies provides a sufficient ecosystem for deploying complicated real-time data stream processing systems especially when both technologies have dependent middleware or orchestration strategies behind them to optimize their performance, scalability and fault recovery. The review detailed a number of optimizations that can be achieved by tuning the Kafka producer batch sizes, choosing Cassandra consistency levels, leveraging containerized deployment models that lend elasticity and fault tolerance, etc. Trials and the usage of t both Kafka and Cassandra indicate, using proper configuration and scaling, improve throughput and reduce latency significantly. However, there are still obstacles in bringing the disparate data abstractions into a common modifier, managing schema change, and efficiently translating streamed data into more query-friendly collections of data. As companies continue to deploy and leverage real-time data systems, it will be increasingly important to implement a solid layer of middleware and automated process helpers to maintain performant scale. The review points out an important notion for proper and sufficient design and optimization of distributed systems, such as Kafka and Cassandra. Companies can build such systems with resiliency and performance with the right tuning and the correct architectural discipline. This would help ingest, process and retain high speed real-time datasets in all sorts of mission-critical uses.

Acknowledgements

The author is grateful to the researchers and institutions referenced in this paper whose foundational work and insights made this review possible.

References

[1]. Wu, J., & Li, H. (2025, April). Real-time data flow processing and optimization scheduling scheme for power system based on Kafka. In Fifth International Conference on Telecommunications, Optics, and Computer Science (TOCS 2024) (Vol. 13629, pp. 461-471). SPIE.
[2]. Rani, S. (2025). Tools and techniques for real-time data processing: A review. International Journal of Science and Research Archive, 14(1), 1872-1881.
[3]. Mazher, N., & Azmat, H. (2024). Real-Time Data Streaming and Analysis Using SQL Server with Apache Kafka. Pioneer Research Journal of Computing Science, 1(3), 44-52.
[4]. Sultan Saeed, J. O., & Frank, E. (2024). Real-time analytics with Apache Cassandra and apache spark.
[5]. Chinthapatla, S. (2020). Unleashing Scalability: Cassandra Databases with Kafka Integration.
[6]. Alang, K. S. Stream Processing with Apache Kafka: Real-Time Data Pipelines.
[7]. Akanbi, A., & Masinde, M. (2020). A distributed stream processing middleware framework for real-time analysis of heterogeneous data on big data platform: Case of environmental monitoring. Sensors, 20(11), 3166.
[8]. Raptis, T. P., & Passarella, A. (2023). A survey on networked data streaming with apache kafka. IEEE access, 11, 85333-85350.
[9]. Oza, J., Patil, A., Maniyath, C., More, R., Kambli, G., & Maity, A. (2024, May).

e ISSN: 2584-2854
Volume: 03
Issue: 08 August 2025
Page No: 2698 - 2704

Harnessing insights from streams: Unlocking real-time data flow with docker and cassandra in the apache ecosystem. In 2024 IEEE Recent Advances in Intelligent Computational Systems (RAICS) (pp. 1-6). IEEE.

[10]. Ed-daoudy, A., Maalmi, K., & El Ouaazizi, A. (2023). A scalable and real-time system for disease prediction using big data processing. Multimedia Tools and Applications, 82(20), 30405-30434.