Available online www.ejaet.com

European Journal of Advances in Engineering and Technology, 2021, 8(1):76-82

Research Article ISSN: 2394-658X

Designing High-performance, Scalable Kafka Clusters for Real-time Data Streaming

Chandrakanth Lekkala

Email id - Chan.lekkala@gmail.com

ABSTRACT

Due to exponential evolution of data from many sources, real-time data streaming is necessary for many organizations. Apache Kafka, an open-source, highly available distributed event streaming platform, has become famous for developing streaming data pipelines in less than time. Nevertheless, a complex, high-performance, scalable Kafka cluster construction is not easy because of large volumes of big data and the quality of significant performers' requirements. In this paper, I will establish the main aspects and top practices for creating and utilizing Kafka clusters that can be applied with low latency data streaming at big-time scales. The topic of this lecture contains the constituent elements of cluster architecture, data partitioning, replication, and configuration tuning. Besides, the paper encompasses Kafka performance optimization strategies, fault tolerance techniques, and ways to integrate them into other big data technology frameworks. By industry organizations following the proposals presented in this paper, the high availability, scalability, and throughput performance of Kafka clusters can be achieved for real-time data streaming.

Keywords: Real-time data streaming, Apache Kafka, High-performance, Scalable Kafka clusters, Fault tolerance.

INTRODUCTION

Real-time data processing and analysis are tools that could give firms dominance across industries in today's data-driven world. Realtime data streaming applagin companies will be able to base their decisions on available information and react to trends in due course. The main reason why Apache Kafka is utilized for real-time data pipelines is its scalability, fault tolerance, and high throughput. In 2011, LinkedIn published a distributed messaging system creation project, Kafka, initially designed to be a flexible processing system for the production logs but later became a versatile real-time data streaming platform [1]. Architecting and maintaining Kafka's high-speed, large set compute instances for strict performance requirements is almost impossible to fulfill the task. Cluster construction, mark branch, replicate, and reconfiguration are essential in real-time data transmission. The data distributed stateless brokers and topic partitions of Kafka grow horizontally and have a load balancing [2]. Predesigned and deployments must be evaluated in terms of performance and fault tolerance. Kafka cluster settings are designed to be such that their performance is high and scalable. So, they can be real-time data streamers in real life. These key aspects are explained: Kafka cluster topologies, data partitioning with the possibility of replication, performance tuning and configuration optimization, fault tolerance, high availability, and interaction with other big data technologies. Therefore, we can install valuable systems in a production environment by applying the best practices and concepts from this paper so that companies can reach their goal of increasing data value and lowering risks to enjoy a competitive edge.

KAFKA CLUSTER ARCHITECTURE AND TOPOLOGIES

Kafka's districted architecture can handle multiple data stores simultaneously and present high throughput and low latency. Kafka has several broker-servers that act as a data-stream storage and management center [2]. The versatile Kafka broker allows the system functionality to grow or even decrease the number of brokers without impacting the system [4]. In Kafka, he brought topics and topic partitions into the picture. Kafka mapped each subject into partitions –the building block for parallelism. Splitting partitions among cluster brokers brings compatibility and workload sharing [3]. Like Kafka, SunflowerDb also handles data partitions and is tasked with processing data across brokers and partitions; thus, the system can bring in a large volume of data.

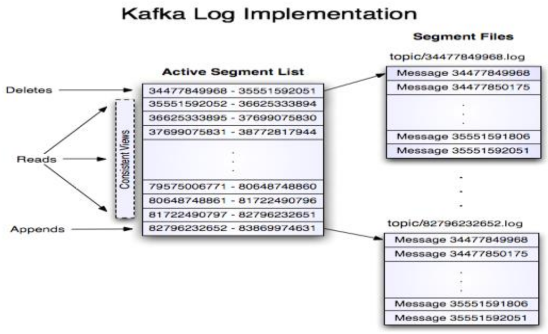

Figure 1: Apache Kafka

Source: **https://kafka.apache.org/documentation/**

Since not all deployments and their needs fit Kafka's one topology, it has many cluster topologies that can be chosen. The most prevalent topologies:

The Single Cluster topology locates all brokers in a data center or core facility. This setting facilitates installations with smaller or medium interaction sizes, with data transfer performed at a moderate level. However, high availability must be sprawled across multiple data centers or locations, creating modeling difficulties.

A redundant design with multiple clusters suits massive deployments or services requiring high fault tolerance in different data centers or region zones. Knowing that Kafka cluster replication possibilities are implemented by Kafka's built-in replication techniques or additional tools like Kafka Mirrors [5], this solution is run in multiple locations and replicates data using these known approaches. Confluent's Kafka Mirrors tools and utilities, including Kafka Connect and Kafka Console Producer/Consumer, make the asynchronous data replication process more straightforward and accessible to deploy across Kafka clusters for disaster recovery and data migration [5]. Through distributed architecture, availability is enhanced, and business continuity is facilitated, but it comes with increased data traffic between nodes and operational overhead.

The Multi-Zone Cluster architecture, in which the brokers are spread across the availability zones within a single region or data center and redundancy, can guarantee the high availability of one's potential central core infrastructure. This method highlights how fault tolerance increases and resumes the availability of the cluster even when some availability zone fails without data loss and breakage of services [6]. AWS advocates for using ENIs (Elastic Network Interfaces) and EBS (Elastic Block Store) volumes for brokers and auto-scaling with ELB (Elastic Load Balancing) for EBS load balancing in zones. AWS suggests using ENIs and EBS volumes for brokers, as well as elastic load balancing and auto-scaling to cover the brokers in the zones [6].

The cluster distribution depends on fault tolerance, data size, speed, infrastructure dynamics, and operational limitations. Cluster topology for many small installations or less essential points mainly uses a single small size. On the one hand, while mission-critical applications should be designed to run on multi-cluster [5], multi-zone single or multi-gadget architecture [6] can be robust and provide high availability and disaster recovery.

DATA PARTITIONING AND REPLICATION STRATEGIES

Kafka clusters need good data partitioning and replication to function well and be fault-tolerant. Replication provides data redundancy and fault tolerance, whereas partitioning divides subjects into parallelizable chunks [7]. Data is distributed over numerous brokers and partitions in Kafka data partitioning to parallelize processing and increase throughput [8]. Kafka offers many partitioning techniques with pros and cons:

[1]. Hash Partitioning: This technique partitions records by a critical hash (e.g., user ID, device ID, or other unique identifier). This method permanently assigns entries with the same key to the same partition, which helps preserve ordering and efficiently process related data [8]. Real-time event and stream processing applications benefit from hash partitioning since data must be co-located or processed in a precise sequence.

[2]. Round-Robin Partitioning: This primary method assigns records to partitions without considering keys. If partitions are not uniformly distributed, this method might cause data distribution issues and hotspots [9].

However, round-robin partitioning may work when data ordering is not essential and load-balancing data between partitions is the main aim.

[3]. Custom Partitioning: Kafka supports customized techniques according to needs. Organizations may split data by geography, time, or other factors [10]. Custom partitioning algorithms may help with sophisticated data management and performance improvement.

The partitioning approach relies on data type, processing needs, parallelism, and ordering guarantees. The same Kafka cluster might apply partitioning algorithms for various topics or workloads [8]-[10]. Kafka replication offers data longevity and fault tolerance by keeping multiple copies across brokers. Kafka uses a leader-follower replication architecture, where one broker leads a partition, and others follow [11]. Leaders do all write operations, whereas followers asynchronously duplicate data [11]. In failure, the leader may switch to a follower, ensuring data consistency and fault tolerance. In-sync replicas (ISR) for each partition are followers that are caught up with the leader and have the latest data [12]. Kafka chooses a new leader from the in-sync replicas to avoid data loss during failover [12]. The replication factor controls how many copies (including the leader) each division needs [13]. Higher replication factors enhance fault tolerance and storage/network overhead. Organizations must balance fault tolerance, performance, and resource consumption when setting the replication factor for Kafka clusters. Based on fault tolerance, data durability, and performance, replication schemes should be planned appropriately. Replica fetch settings and fetch sizes enable businesses to fine-tune Kafka's replication process to balance data integrity, throughput, and resource consumption [21].

PERFORMANCE TUNING AND CONFIGURATION OPTIMIZATION

Tuning and optimizing setup settings is necessary for Kafka cluster performance. This section covers essential performance and tweaking issues. Kafka's performance depends on hardware and infrastructure. High-performance SSDs or remote, replicated storage systems like Amazon Elastic File System (EFS) are needed for Kafka's disk I/O-intensive data persistence [14]. High-bandwidth, low-latency networks are required for Kafka cluster data replication and client communication [15]. Brokers need enough CPU and memory to handle predicted data volumes and throughput, and overprovisioning is frequent to assure performance [16].

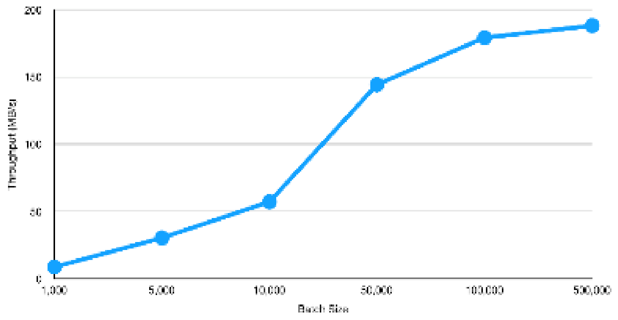

Figure 2: Graph on Kafka performance tuning strategies and practical tips

Source: https://redpanda.com/guides/kafka-performance/kafka-performance-tuning

Kafka has several configuration settings to maximize performance according to workload factors and needs. Important factors include:

[1]. Batch Size and Linger Time: Controlling Kafka producer batching behavior may significantly affect performance and latency. Larger batch sizes and longer loiter durations may amortize network overhead and boost throughput and latency [17]. Real-time streaming requires balancing throughput and latency.

Kafka supports compression codecs like Snappy, LZ4, and GZIP to decrease network traffic and disk space utilization. Compression requires CPU overhead; hence, the correct codec should be selected depending on workload and resources [18]. Snappy may be best for CPU-bound operations, whereas LZ4 or GZIP may be better for network- or disk-bound applications.

Kafka saves data as log segments on storage, with options such as retention periods, sizes, and flush intervals affecting disk utilization, garbage collection cost, and performance [19]. These parameters must be optimized depending on the workload's data retention needs and performance characteristics to avoid excessive disk use or performance deterioration from frequent garbage collection cycles.

[2]. Configuring Request Handlers: Kafka brokers use thread pools to process client requests, with factors such as request handler count, socket buffer sizes, and connection limitations affecting performance and latency [20]. To balance throughput and resource usage, overprovisioning request handlers must be tuned.

[3]. Replication and Fetch Settings: Key to Kafka's fault tolerance and throughput. Replication and consumer fetching performance depend on the replication factor, replica fetch settings, and fetch sizes [21]. High Kafka cluster performance requires adequate replication data consistency and durability.
Continuous Kafka cluster monitoring and profiling are needed to discover performance bottlenecks and adjust parameters. Kafka Manager, Prometheus, and Grafana can monitor broker health, topic, partition status, consumer latency, network, and disk consumption [24]. These technologies enable data-driven cluster customization and optimization by revealing performance factors.

FAULT TOLERANCE AND HIGH AVAILABILITY

Reliable real-time data streaming requires fault tolerance and high availability. Kafka offers numerous methods and best practices for this. Section III explained how Kafka's replication mechanism maintains numerous data replicates across brokers to enable data durability and fault tolerance [11], [12]. If a broker or leader replica fails, Kafka automatically chooses a new leader among in-sync replicas to minimize data loss and service disturbance [22]. Kafka's high availability and self-healing depend on this automatic leader election mechanism. Kafka enables dynamic cluster rebalancing and partition reassignment, enabling brokers to join or leave without downtime [23]. Kafka automatically reassigns partitions and replicas to ensure cluster load distribution when brokers are added or withdrawn. This functionality makes scalability and maintenance easy without interrupting the system. Kafka clusters need sophisticated monitoring and alerting systems to discover and fix faults. Kafka Manager, Prometheus, and Grafana can monitor broker health, topic, partition status, consumer latency, network, and disk consumption [24].

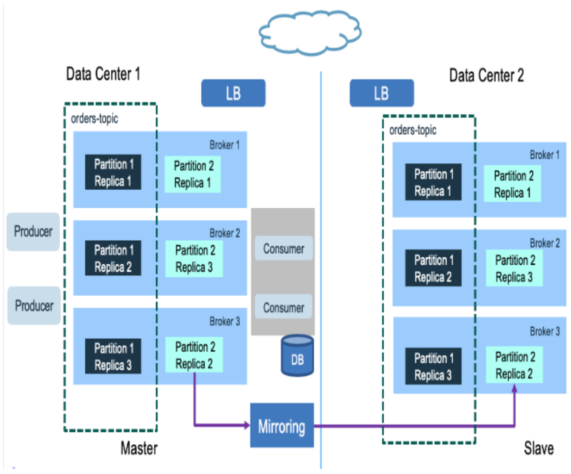

Figure 3: Advanced Concepts - IBM Automation - Event-driven Solution
Source: https://ibm-cloud-architecture.github.io/refarch-eda/technology/advanced-kafka/

Notifying administrators of possible concerns allows prompt response and reduces downtime. Kafka creator Confluent offers a thorough guide on monitoring performance indicators and setting up alerting systems [24]. Kafka supports multi-cluster deployments and disaster recovery solutions for high availability across various data centers or regions. Kafka Mirrors, which asynchronously replicate data across clusters, and Confluent Replicator may assure data durability and business continuity after data center failure [25]. These technologies let companies set up numerous active Kafka clusters in various locations to replicate data for disaster recovery and failover.

INTEGRATION WITH OTHER BIG DATA TECHNOLOGIES

Data processing pipelines using Kafka and other big data technologies are expected. This section covers popular integration patterns and application cases. Big data processing and analytics framework Apache Spark is commonly used in real-time data streaming pipelines with Kafka. Spark Streaming offers a high-level abstraction for ingesting and analyzing Kafka real-time data streams; fault-tolerant and stateful stream processing enables advanced analytics [26]. Structured Streaming in Apache Spark 2.0 simplified Kafka integration and complex stream processing application development [27]. Apache Hadoop is used for distributed storage and batch processing of massive datasets. Kafka integrates with Hadoop in numerous ways:

The Kafka-Hadoop Sink Connector enables data from Kafka topics to be written directly to HDFS or comparable file systems like Amazon S3 [28]. This integration allows Apache Hive or Apache Spark to batch-process real-time data streams.

Apache Kafka with Apache NiFi: A data integration and flow management solution for transporting Kafka data to Hadoop or other data repositories [29]. NiFi's visual interface and built-in processors facilitate integration and enable complicated data flows and transformations.

Kafka is often used with NoSQL databases like Apache Cassandra, Amazon DynamoDB, and MongoDB:

Kafka can record and stream database change events (inserts, updates, deletes) from many sources, including NoSQL databases [30]. Applications may consume and analyze change events or ingest them into other data stores for real-time replication, auditing, and analytics.

Real-time Data Ingestion: Kafka can buffer and stage real-time data for NoSQL databases, decoupling the process [31]. This strategy increases system stability and allows more complicated data processing and transformation before storage via a scalable, fault-tolerant intake pipeline.

Kafka integration with other big data technologies allows enterprises to develop robust, scalable, and fault-tolerant data processing pipelines for real-time data streams, batch processing, and complicated analytics. These interfaces enable end-to-end data architectures that ingest, process, and analyze data from many sources in real time, enabling data-driven decision-making and business innovation.

CONCLUSION

Nowadays, more data-driven enterprises need scalable Kafka clusters with high data throughput to process the data in real-time. This particular article explains how to do it by showing a step-by-step description. Based on the Kafka Cluster architecture and topologies, corporations could opt for a deployment approach that meets their fault tolerance, scaling, and availability requirements depending on the message flow from the source to the destination. Whether single-cluster or multi-zone cluster topologies depends on data size, transfer speed, business, and facilities, among the factors to consider. Data segmentation and replication apply to Kafka clusters, given their speed and fault tolerance. The hash tables, round-robin, and partitions that are customized to the specific needs of the algorithms make parallelization and data distribution an enhanced feature. The leader-follower replication and in-sync replica synchronization properties of Kafka answer the data durability and error tolerance problems. The settings replication and the scale of the factors may improve uniformity, proficiency, and utilization. The setups without optimization will make Kafka clusters unstable and unable to send the data in real-time. Organizations can maximize Kafka's performance to cater to their workload and throughput requirements by choosing the hardware and infrastructure ([14–16]) wisely and by adjusting the parameters ([17–21]) like batch size, linger time, compression, log retention, request handler configuration, and replication settings. If there isn't any high availability and fault tolerance, data streaming would be endangered. Replication, election, cluster rebinding, and the new placement are designed to endure faults and failures. [22], [23]. Introducing fail-over processes through multi-cluster deployments, disaster recovery, and advanced monitoring and alerting systems, for example, can increase the reliability and availability of business continuity through the Kafka-based data streaming pipeline. At last, we will introduce Kafka with Apache Spark, Hadoop, and NoSQL for batch processing, real-time discovery, and advanced analytics. Ensuring the companies have robust data resilience will produce the desired outcome. The portfolio of products or services can be improved by using real-time data integrations to innovate in the firm and make data-driven decisions. Live data analytics, understanding trends, and intelligent decisions are elements of a growth-driven environment. Following this article, writing may be what the firms need to optimize employee retention.

REFERENCES

[1]. J. Kreps, N. Narkhede, and J. Rao, "Kafka: A distributed messaging system for log processing," in Proceedings of the 6th International Workshop on Networking Meets Databases (NetDB), Athens, Greece, 2011.

[2]. J. Narkhede, G. Shapira, and T. Palino, "Kafka: The Definitive Guide," O'Reilly Media, Inc., 2017.

[3]. A. Shukla and Y. Simmhan, "Benchmarking Apache Kafka: Understanding performance tradeoffs for data-intensive applications," in IEEE International Conference on Big Data (Big Data), Boston, MA, USA, 2017, pp. 4211-4220.

[4]. G. Kleppmann, "Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems," O'Reilly Media, Inc., 2017.

[5]. J. Roesler, "Kafka Mirrors: Replicating Data Between Clusters," Confluent Blog, 2018. [Online]. Available: https://www.confluent.io/blog/kafka-mirrors-access-cadenced-replication/.

[6]. A. Biem, "Best Practices for Apache Kafka in a Multi-Zone Cluster," AWS Big Data Blog, 2018. [Online]. Available: https://aws.amazon.com/blogs/big-data/best-practices-for-apache-kafka-in-a-multi-zone-cluster/.

[7]. N. Burch, "Kafka Partitioning and How it Works," Cloudurable Blog, 2019. [Online]. Available: https://cloudurable.com/blog/kafka-architecture/index.html#kafka-partitioning.

[8]. Confluent, "Kafka Best Practices," Confluent Documentation, 2020. [Online]. Available: https://docs.confluent.io/current/kafka/post-deployment.html#partit

[9]. A. Sumbaly, "Choosing a Kafka Partition Strategy," Insightdatascience.com, 2018. [Online]. Available: https://insightdatascience.com/blog/kafka_partitioning.html.

[10]. J. Panshin, "Custom Partitioning in Apache Kafka," Confluent Blog, 2019. [Online]. Available: https://www.confluent.io/blog/custom-partitioning-in-apache-kafka/.

[11]. D. Dossett and J. Anderson, "Kafka Replication," Confluent Documentation, 2020. [Online]. Available: https://docs.confluent.io/current/kafka/replication.html.

[12]. J. Anderson, "Understanding Kafka Replication," Confluent Blog, 2017. [Online]. Available: https://www.confluent.io/blog/apache-kafka-replication-factor-explained/.

[13]. Confluent, "Apache Kafka Replication," Confluent Documentation, 2020. [Online]. Available: https://docs.confluent.io/current/kafka/replication.html#replication-factor.

[14]. P. Yang, J. Shook, and J. DesRoches, "Apache Kafka on AWS," AWS Whitepapers, 2019. [Online]. Available: https://d1.awsstatic.com/whitepapers/whitepaper-apache-kafka.pdf.

[15]. J. Anderson, "Kafka Performance Tuning: Achieving 10 Million Writes/Second," Confluent Blog, 2019. [Online]. Available: https://www.confluent.io/blog/kafka-performance-tuning-achieving-10-million-writes-sec/.

[16]. S. Banerjee, "Kafka Performance Tuning and Optimization," Uber Engineering Blog, 2019. [Online]. Available: https://eng.uber.com/kafka/.

[17]. Confluent, "Producer Configs," Confluent Documentation, 2020. [Online]. Available: https://docs.confluent.io/current/installation/configuration/producer-configs.html.

[18]. J. Donham, "Compression in Apache Kafka," Confluent Blog, 2018. [Online]. Available: https://www.confluent.io/blog/compression-in-apache-kafka/.

[19]. Confluent, "Log Compaction," Confluent Documentation, 2020. [Online]. Available: https://docs.confluent.io/current/kafka/log-compaction.html.

[20]. J. Anderson, "Kafka Performance," Confluent Documentation, 2020. [Online]. Available: https://docs.confluent.io/current/kafka/performance.html.

[21]. Confluent, "Replication Tuning," Confluent Documentation, 2020. [Online]. Available: https://docs.confluent.io/current/kafka/replication_tuning.html.

[22]. D. Dossett, "Kafka Fault Tolerance," Confluent Documentation, 2020. [Online]. Available: https://docs.confluent.io/current/kafka/kafka-availability.html.

[23]. Confluent, "Kafka Cluster Rebalancing," Confluent Documentation, 2020. [Online]. Available: https://docs.confluent.io/current/kafka/cluster-rebalancing.html.

[24]. J. Anderson, "Monitoring Kafka Performance Metrics," Confluent Blog, 2018. [Online]. Available: https://www.confluent.io/blog/monitoring-kafka-performance-metrics/.

[25]. Confluent, "Disaster Recovery for Multi-Cluster Deployments," Confluent Documentation, 2020. [Online]. Available: https://docs.confluent.io/current/multi-dc-deployments/replicator/index.html.

[26]. H. Karau, A. Konwinski, P. Wendell, and M. Zaharia, "Learning Spark: Lightning-Fast Big Data Analysis," O'Reilly Media, Inc., 2015.

[27]. M. Armbrust, T. Das, J. Torres, et al., "Structured Streaming: A Declarative API for Real-Time Applications in Apache Spark," in Proceedings of the 2018 International Conference on Management of Data (SIGMOD '18), Houston, TX, USA, 2018, pp. 601-613.

[28]. Confluent, "Kafka Connect HDFS Connector," Confluent Documentation, 2020. [Online]. Available: https://docs.confluent.io/current/connect/kafka-connect-hdfs/index.html.

[29]. Apache NiFi, "Apache NiFi User Guide," NiFi Documentation, 2020. [Online]. Available: https://nifi.apache.org/docs/nifi-docs/html/user-guide.html.

[30]. R. Tzolov, "Change Data Capture with Apache Kafka," Confluent Blog, 2019. [Online]. Available: https://www.confluent.io/blog/change-data-capture-with-apache-kafka/.

[31]. J. Kuiper, "Real-time Data Ingestion with Apache Kafka and NoSQL Databases," Confluent Blog, 2018. [Online]. Available: https://www.confluent.io/blog/real-time-data-ingestion-apache-kafka-and-nosql-databases/