International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 07 July 2026
Page No: 5083-5090
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0668

Building High-Performance Streaming and Analytics Platforms

Srivardhan Jalan[1]
[1]*Purdue University, West Lafayette, Indiana.*

Abstract
The quantity of data that is created, processed, and streamed into a contemporary organisation is significant. As an engineering challenge, it is how to keep this up to date at all times and how to ingest, process, and serve this data with high availability and fault tolerance. This paper reviews architectural patterns, technology decisions, and optimization strategies of high-performance streaming and analytics systems. We cover Lambda and Kappa architecture, the messaging systems, essential stream processing engines, the available storage solutions, and best practices. Benchmark data and design trade-offs are then provided to guide practitioners on such systems, which allow millions of events per second with sub-10ms latency.
Keywords: *Real-Time Data Streaming, Apache Kafka and Stream Processing, High-Performance Analytics Platforms, Cloud-Native Data Engineering, Scalable Event-Driven Architecture*

1. Introduction
The streaming platform is a distributed system in which data is moved continuously from producers to consumers, via a persistent, ordered log [1]. The difference between a batch platform and a streaming platform is that batch processes work with a finite amount of data while streaming processes work with a set of 'unbounded' data, so that you can perform the calculations instantly, trigger alerts as they flow in, perform model inference continuously, and more [2]. Topics, partitions, offsets, and consumer groups (coordinated parallel consumption) are the foundation abstractions. The design space is further broken into two common design paradigms Lambda Architecture and Kappa Architecture. One way that Lambda systems work: They use a separate batch layer (for correctness and historical replay), a speed layer (for low-latency approximations), and they synchronise outputs in a serving layer. As strong as it is, Lambda carries the load of having to maintain two sets of code that give the same semantics. In doing so, Kappa strives to make all computation a ‘stream process' over an immutable log, where no reprocessing from retained offsets is necessary for batch recomputation [3]. The choice of paradigms is based on the level of maturity of the organization, acceptable operational complexity, or whether reprocessing historical material leads to the need for a single batch route [4]. There is also a new trend of using Kappa for greenfield systems, where the old stream processors, such as Apache Kafka and Apache Pulsar, are not used; instead, more modern stream processors like Apache Flink are chosen. To prevent code duplication and to support event-time semantics, it has introduced the DataStream API and Table API in addition to exposing APIs for standard semantics, such as batch semantics and stream semantics [5].

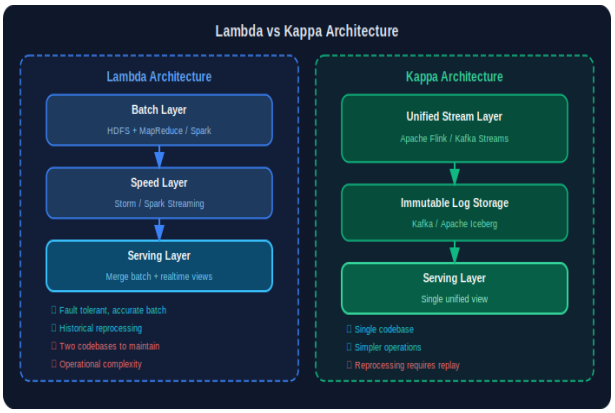

Figure 1 Lambda Architecture (left) vs. Kappa Architecture (right): design trade-offs in streaming system design.

The major properties of the architecture that all streaming platforms have to tackle include: delivery semantics (at least once vs exactly once), ordering guarantees (total order per partition), back-pressure propagation (consumer-driven flow), and state management (checkpointing, RocksDB-based state stores). The correctness and properties are

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 07 July 2026
Page No: 5083-5090
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0668

determined based on these properties with production loading [6] shown in Figure 1.

2. Core Infrastructure: Messaging Systems and Brokers

Current streaming analytics systems rely on distributed messaging systems to guarantee scalable, fault-tolerant and reliable communication between consumers and producers of varying types. They perform the following: Separate event producers from the downstream processing applications using Message Brokers, they support asynchronous communications, they ensure ordered delivery of messages in a partition, they ensure the persistence of the message store. With this architectural pattern, organisations can ingest and process millions of events per second, with little latency, high availability. With its distributed commit-log architecture and its enormous ecosystem and real-time analytics capabilities, Apache Kafka has become the industry standard among the multitude of messaging platforms. There are other platforms as well, that provide similar features and cater for other requirements when it comes to deployment such as cloud native scalability and infrastructure that is only managed [7]. In the figure 2 a distributed streaming platform architecture of the modern is illustrated. Many new data sources are able to publish their events, including IoT devices, enterprise data sources, web-based applications and APIs, application logs, change data capture (CDC) pipelines and anything in between.There are many applications that can publish events, including IoT devices, enterprise data sources, web apps and APIs or application logs and change data capture (CDC) pipelines, etc. Topics are broken up into several partitions, so that multiple brokers can process multiple partitions simultaneously. The replication policy is defined in the configuration and is used in order to achieve fault tolerance and high availability, each partition may be replicated. These events are fed into stream processing frameworks like Apache Flink, Apache Kafka Streams, and Apache Spark Structured Streaming, where they can be processed in real time to transform, aggregate, and perform analytics on them, and then they are stored in analytical databases or in a data lakehouse or cloud object storage shown in Figure 2.

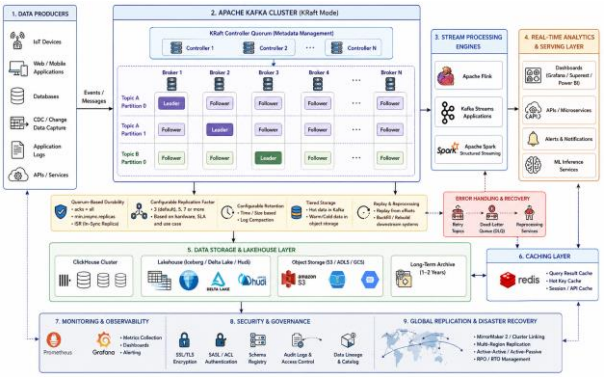

Figure 2 Enhanced Distributed Streaming Platform Architecture with Configurable Replication, Quorum-Based Durability, and Real-Time Analytics

Multiple brokers, each containing a number of partitions, are used for handling data planes, while an aggregate service, referred-to as the metadata management layer, is based on the KRaft consensus protocol, which is previously known as the Apache ZooKeeper service. Topics are split into partitions so that they can be processed in parallel but ordering guarantees are still guaranteed within each partition. As the workload increases more brokers (partitions) can be dynamically added which provides an almost linear scale up without having any impact on the availability of an application [8]. The replication factor can be fully tuned to the organization's needs and is not fixed with any architectural restrictions, depending on the hardware resources available, critical nature of the workload and the desired fault tolerance. The typical replication level for most production systems is three, however for highly-available enterprise systems with multiple failures of the brokers, a five or seven replication is typical. Thus, it is important to see the replication factor as a parameter that can be configured according to the deployment and not as mandated by the architecture [8]. To ensure message durability Kafka uses In-Sync Replica (ISR) protocol-based replication with a quorum. There is one leader, one follower replica, and one or more follower replicas that are continuously synching with the leader replica, for each partition. For producers, all will force them to wait for the number of replicas that they need (min.insync.replicas) to persist the message. This

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 07 July 2026
Page No: 5083-5090
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0668

write protocol with quorum guarantees more durability – and it won't cause loss of acknowledged messages from brokers. If the partition leader does not exist, Kafka elects a new partition leader automatically from ISR set, thus without major service disruption, continuous processing of messages from the previous partition occurs. Kafka uses a distributed append-only log that has a configurable set of aging policies for messages based on log compaction, storage space usage or retention time. A typical high-throughput production deployment strategy is to keep a few days of stream events in memory and a deployment like a financial application, a healthcare application, or a regulatory application might use a one day window on memory and then migrate the data into the object storage backend (Amazon S3, Azure Blob Storage or Google Cloud Storage) for anywhere up to a few years of offline access. It comes with a multi-tiered storage approach, identified to be able to reduce storage costs while still retaining historical data for the application of preparation, compliance, auditing, DR and machine learning [8]. Apache Pulsar's architecture, however, accounts for the separation between compute and storage layers, having it separated by Apache BookKeeper enables the separate independent scaling of brokers and persistent storage. The design provides build-in capabilities of geo-replication, multi-tenancy, tiered storage, which is ideal for cloud-native applications that are used across the globe. Compared to AWS Kinesis Data Streams is fully managed, a cloud-based service that takes the friction of learning to provision and manage infrastructure and allows event streaming to become easier to use and scale [7,9] shown in Table 1.

Table 1 Streaming Platform Comparison

Platform	Throughput	Latency (p99)	Scalability	Best Use Case
Apache Kafka	2M+ messages/s	<10 ms	Horizontal scaling	Event streaming and log aggregation
Apache Pulsar	1.5M+ messages/s	<15 ms	Geo-replicated	Multi-tenant cloud-native deployments
AWS Kinesis	1M+ messages/s	<200 ms	Fully managed	AWS-native analytics
Apache Flink	500K+ records/s	<5 ms	Task-parallel	Stateful stream processing
ClickHouse	1B rows/s	<50 ms	Distributed columnar shards	Real-time OLAP analytics

3. Stream Processing Engines and Computation Models

Continuous queries are run on an event stream by stream processing engines. The most popular streaming solution for stateful stream processing is Apache Flink, which guarantees sub-second latency, millisecond-level event-time processing, CPU scalability with RocksDB as state backend, and ensures exactly once semantics with asynchronous Chandy-Lamport checkpoints. Both the Table/SQL API for declarative analytics as well as the low-level DataStream API compile to a distributed execution graph in Flink [10][11]. The types of computation models for stream processing are: record-at-a-time processing (lowest latency, no batching semantics), micro-batch processing (Spark Structured Streaming, high throughput and higher latency), and continuous operator processing (Flink, Kafka Streams—true streaming, bounded latency). Different models mean different latency-throughput compromises and are applicable to SLA needs. Four different types of windowing strategies are crucial to aggregating unbounded streams: tumbling windows (fixed windows without overlap), sliding windows (overlapping windows with high memory consumption), session windows (activity gap-based

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 07 July 2026
Page No: 5083-5090
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0668

windows), and global windows (windows that are triggered manually by the user or feature of the platform). Event-time processing with watermarks helps properly process out-of-order events—which is crucial when events cross interdependent networks with varying latency distributions. It is also possible to configure Flink watermark generators with acceptable delay, which allows saving state space by discarding events that are beyond the watermark [12].

- **Apache Flink:** Perfect for stateful stream processing at millisecond latency and with exactly-once semantics.
- **Kafka Streams:** Thin, embedded processing into the application's JVM; perfect for microservices.
- **Apache Spark Structured Streaming:** Micro-batch model; suited when you need to do close work with the batch pipelines.
- **ksqlDB:** SQL-native streaming on Kafka that is ideal for operational analytics and transforms.

4. Analytics Storage: OLAP, Time-Series, and Data Lakehouses

Streaming application data needs to be stored in systems that are optimized for analytical workloads and low latency queries, as well as for long term historical analysis. Today's streaming platforms are therefore comprised of multiple storage technologies like columnar OLAP databases, time-series databases, distributed data lakehouses, object storage, and distributed caching systems. These layers of storage are combined to support all sorts of analytic workloads – from answering questions about the dashboard every millisecond, to denoising and handling petabyte-scale historical analytics – and support scalability, consistency, and operation efficiency. Major streaming service production workloads have latencies similar to the one depicted in figure 3. Cloud services often have higher latency-even Apache Kafka and Apache Flink have consistently poor latencies for this type of real-time event processing, which adds abstraction layers. Generation layouts are key to the development of the downstream storage technologies that are required, according to the service-level requirement of the application.

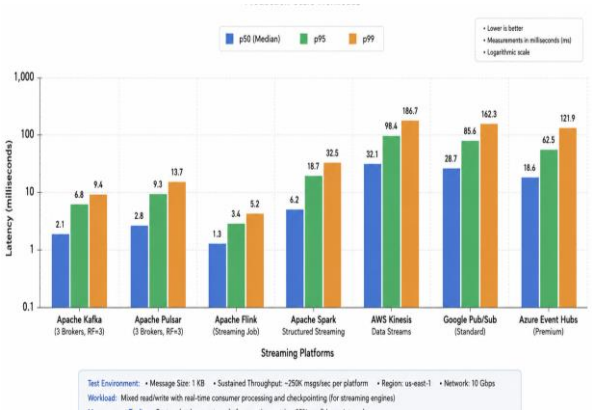

Figure 3 Latency percentile comparison (p50, p95, and p99) across major streaming platforms under production-scale workloads.

Column-oriented analytical databases like ClickHouse are created with a specific purpose in mind – to handle billions of records and deliver sub-second query speeds. ClickHouse data is stored in columnar format, processed vectorized, compressed and categorized by MergeTree to significantly reduce disk I/O and improve the performance of analytical queries. While storing the data in a row-based database is generally more efficient for handling complex queries such as joins, aggregation, filtering and scanning of large datasets over the entire table are more conducive to columnar databases for fast real-time business intelligence and operational analytics [13]. A Real-Time Online Analytical Processing (OLAP) system like Apache Druid can provide other features like approximate query processing, bitmap indexing, distributed query processing, pre-aggegation and rollup. Such optimizations facilitate applications for interactive dashboards or operational monitoring that are able to perform complex analytical queries in time – usually under a few milliseconds. Likewise, there are time-series databases that are designed to easily support time-based indexing, dynamic time-granulation down-sampling, retention policies and compression, including InfluxDB and TimescaleDB, that are great for storing and distributing an uninterrupted flood of measurements, infrastructure data, and application telemetry all along the lines of efficiency. Business data systems are getting increasingly sophisticated, and one of them is the Data Lakehouse—the use of

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 07 July 2026
Page No: 5083-5090
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0668

the scalability of object stores with those transactional guarantees typically found in data warehouses. New technologies such as Apache Iceberg, Apache Delta Lake, and Apache Hudi store data in efficient columnar format (Parquet ORC), along with intriguing additional capabilities such as time-travel query, schema evolution and time-series version control, plus Apache Hadoop's support of Hadoop's ACID transactions. The compaction mechanism runs periodically to optimize file organization for efficient analytics queries for the PDF data and processed events are continuously written to the tables in the lakehouse from the streaming engines such as Apache Flink. While this single architecture eliminates the need for multiple data repositories like a data warehouse and a data lake, saving on complexity and ease of use, on the flipside it makes it easier to have consistent data [11,14]. Distributed caching isn't restricted to storing persistent data, either though; it is also crucial in reducing request latency, particularly repeated requests for common, frequently requested information from a database. These are typical use cases of streaming analytics platforms using Redis or Memcached for caching such as dashboard metrics, machine learning inference results, session information and frequently used API responses. The advantage of query caching is that queries can return the same answers directly from the memory, without having to process a time-consuming analysis over a large storage system. Caching may reduce response time from milliseconds to hundreds of milliseconds for business intelligence applications that give interactive dashboards to end-users—while helping to lighten the load on the systems and improving overall system throughput and performance. There are various levels of architecture on which these caching mechanism exist. Application-level caches that are used for business objects that are accessed frequently, query-result caching for quickly available analytical results, and metadata-level caching for schema and catalog lookups, and state caches in stream processing systems for avoiding repeated access to persistent state stores over time. Moreover, there are analytical databases which directly leverages in-memory data, e.g., ClickHouse — enables caching of in-memory query result and skipping indexes; or Apache Spark — intermediate datasets are in-memory cached for optimized iterative analytical processing. These strategies collectively work towards creating a more responsive and scalable real-time analytics platform. This has been further optimized by a number of techniques for analytical queries. The advantages of partition pruning are that redundant data partitions are not fetched in query execution, projection pushdown filters the data at the boundary of the storage engines instead of fetching all data, predicate pushdown eliminates redundant data partitions by applying filter (predicate) at the boundary, and vectorized execution saves multiple rows in data partition by SIMD (Single Instruction Multiple Data) instructions during the steps that create a SELECT statement. Such optimizations, along with efficient compression techniques and more general distributed execution models, can provide orders-of-magnitude more analytical performance than traditional, row-based DBMS systems [13,14]. Streaming analytics platforms include columnar analytical databases, scalable lakehouse architectures that are readily available, distributed object storage, intelligent caching systems, and enable industry-leading low latency, high throughput, fault tolerance and long-term scalability. The result is that we can use the modern analytics software stack to implement the widest variety of interactive dashboards, scale it to perform large-scale historical analysis, implement pipelines with machine learning capabilities and use it in enterprise decision-support infrastructures without sacrificing performance or operational efficiency shown in Table 2.[11, 13, 14]

Table 2 Key Optimization Techniques for Streaming Analytics Platforms

Technique	Description	Performance Benefit
Batch Compression	Snappy, LZ4, or ZSTD compression on producer batches	40–70% bandwidth reduction and lower storage cost

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 07 July 2026
Page No: 5083-5090
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0668

Partition Tuning	Align partition count with consumer parallelism	Linear throughput scaling and reduced consumer lag
Backpressure Control	Consumer-driven flow regulation in Flink/Kafka	Prevents overload and improves stability
Columnar Storage	Parquet and ORC storage formats	10–50× faster analytical queries
Watermarking	Event-time watermark generation	Correct handling of late-arriving events
Zero-Copy I/O	sendfile() and memory-mapped file access	Near wire-speed throughput with lower CPU utilization
Distributed Caching	Redis, Memcached, ClickHouse Query Cache	Millisecond query response and reduced backend load

5. Scalability, Fault Tolerance, and Operational Excellence

Many streaming analytics platforms need to be resilient to fail worst case even in the face of failing hardware, software bugs, network congestion and a rapidly evolving workload. These days “scalability” and “fault tolerance” are essential components of streaming architectural design. Horizontal scalability is attained through scaling out topics across a multitude of partitions and brokers, which enables more computational power to be added in without reducing system availability. Or, a distributed messaging system are more scalable for throughput with adding Brokers Clusters and Processing Partitions horizontally, on the other hand, a vertically scaled system lapses in reliability due to it's hardware capacity. Similarly, the number of available TaskManagers for the processing of an event stream can be flexibly increased or decreased when the stream is growing, or when the number of required tasks for a specific stream is ready to be expanded or shrunk. This includes support for auto-resizes for resources and elastic scalability with container platforms such as Kubernetes for better operational efficiency. Some of the metrics that are observed by the autoscaling frameworks to dynamically scale up or down number of processing instances to support workloads include CPU usage, memory usage, processing latency, etc. Independent scales of messaging, processing and storage keep resources from ever getting "bottlenecked" and from having predictable response times during high traffic loads. In addition to scale, full pipeline visibility and monitoring for operational excellence requires streaming. Centralized monitoring platforms constantly observe broker health, the proportion of partitions, consumer lag, the duration of the checkpoint, processing throughput, storage consumption and bandwidth throughput on the network. These operational metrics are helpful in proactively planning capacity in advance, spotting anomalies in early, and reacting promptly to incidents so that high performance and uninterrupted real-time data processing would be achieved [8,9] shown in Figure 4.

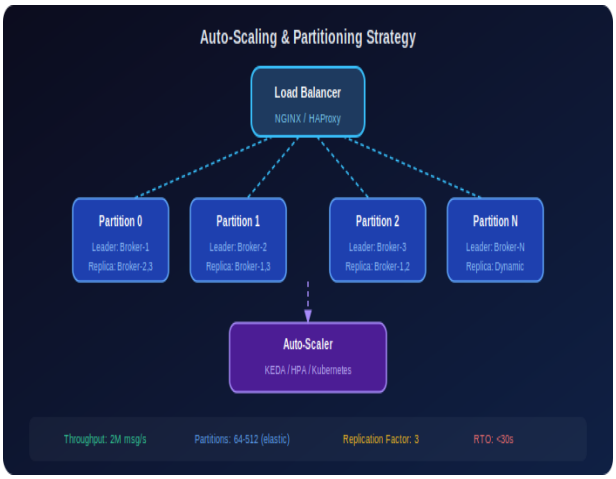

Figure 4 Partition topology and auto-scaling strategy: Kubernetes-based elastic scaling triggered by consumer lag and CPU metrics.

Modern streaming infrastructures must also have comprehensive recovery capabilities to avoid any failure points having an impact on the data pipeline. Common practice for error handling is to use Dead

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 07 July 2026
Page No: 5083-5090
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0668

Letter Queues (DLQs) or Dead Letter Topics (DLTs) that catch malformed, corrupted, duplicate, and unsupported events after some configurable retry iterations. Failed events are made it easier to flow through the system and are downlooped to dedicated recovery topics, thereby avoiding triggering failed events over and over and consuming resources within the system. Consumers in a Kafka application will include/relay topics that implement retries, consumers will use exponential backoff policies, producers will be idempotent, producers will have transactions and consumers will have consumer offsets. Because Kafka is able to store events persistently according to the user-configurable retention policies, historical events can be replayed by flushing consumer offsets, allowing for quick recovery from software errors, message loss at consumer end, or data corruption by the user without producers needing to re-send events. Apache Flink also improves resilience through distributed checkpointing and savepoints which will restore the state of an application without changing its exactly-once semantics. The ability to recover in similar fashion exists in Apache Pulsar, as native Dead Letter Topics, and in AWS Kinesis, with built-in support for the retry mechanism for downstream consumers. In addition to local fault tolerance, enterprise deployments more and more demand global replication for high availability and disaster recovery between distributed regions around the globe. For cross-region replication, the MirrorMaker 2 and Cluster Linking features of Apache Kafka and the built-in geo-replication feature of Apache Pulsar are available. Organizations can use an active/active architecture to process events from both regions at the same time or an active/passive architecture with redundant secondary regions that are kept in sync and take over when the primary region fails. These strategies have an impact on the reduction of the Recovery Point Objective (RPO) and the Recovery Time Objective (RTO), so that business continues to operate in the event of a regional failure. Together with TLS-based secure streaming, SASL or OAuth-based streaming authentication, topic-based authorization, encryption of the stored data, schema validation, and monitoring, these features create a scalable, resilient real-time streaming infrastructure for enterprise mission-critical applications that provides reliable streaming, high availability, and scalability [8,9].

Conclusion

Carefully designed systems must have a coordinated co-design of the messaging, compute, and storage layers for high-performance streaming and analytics systems. Another Lambda-to-Kappa transition is in the direction of stream-processing systems like Apache Flink, with their advantages of reduced complexity and guarantees of correctness. Previously, the time scale between dashboards and information was in minutes, but with the strength of columnar OLAP databases and Data Lakehouse formats it's no longer. The key capabilities of production-managed platforms are: exactly-once semantics out-of-the-box, Elastic autoscaling with consumer lag, fully comprehensive observability, and progressive alerting. Streaming platforms are in a prime position to provide a universal data fabric for connecting events, transaction systems, and analytics stores into a unified single data architecture with a consistent, predictable latency across all product sizes- real-time and near-real-time data and models serving into feature pipelines.

Reference List

[1]. 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).
[2]. Warren, J., & Marz, N. (2015). Big Data: Principles and best practices of scalable real-time data systems. Simon and Schuster.
[3]. Akidau, T., Bradshaw, R., Chambers, C., Chernyak, S., Fernández-Moctezuma, R. J., Lax, R., ... & Whittle, S. (2015). The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. Proceedings of the VLDB Endowment, 8(12), 1792-1803.
[4]. Carbone, P., Katsifodimos, A., Ewen, S., Markl, V., Haridi, S., & Tzoumas, K. (2015). Apache Flink: Stream and batch processing in a single engine. The Bulletin of the Technical Committee on Data Engineering, 38(4).
[5]. Kleppmann, M. (2017). Designing data-

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 07 July 2026
Page No: 5083-5090
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0668

intensive applications: The big ideas behind reliable, scalable, and maintainable systems. " O'Reilly Media, Inc.".

[6]. Noghabi, S. A., Paramasivam, K., Pan, Y., Ramesh, N., Bringhurst, J., Gupta, I., & Campbell, R. H. (2017). Samza: stateful scalable stream processing at LinkedIn. Proceedings of the VLDB Endowment, 10(12), 1634-1645.

[7]. Akidau, T., Chernyak, S., & Lax, R. (2018). Streaming systems: the what, where, when, and how of large-scale data processing. O'Reilly Media, Inc.

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

[9]. Flink, A. (2023). Apache Flink documentation.

[10]. Hive, A. (2017). Apache Hive Documentation-Apache Software Foundation.

[11]. Armbrust, M., Ghodsi, A., Xin, R., & Zaharia, M. (2021, January). Lakehouse: a new generation of open platforms that unify data warehousing and advanced analytics. In Proceedings of CIDR (Vol. 8, No. 1, p. 28). sn.

[12]. Shehbaz, S. (2026). Benchmarking and evaluating time-series databases for appliance-level energy data.

[13]. Schulze, R., Schreiber, T., Yatsishin, I., Dahimene, R., & Milovidov, A. (2024). ClickHouse- lightning-fast analytics for everyone. Proceedings of the VLDB Endowment, 17(12), 3731-3744.

[14]. Saha, D. (2024). Disruptor in Data Engineering: Comprehensive Review of Apache Iceberg. SSRN Electronic Journal.

[15]. Shafiq, M. H., Zhang, Z., & Stefanidis, K. (2025, September). Enhancing Data Interoperability in Multi-platform Lakehouses with Apache. In New Trends in Database and Information Systems: ADBIS 2025 Short Papers, Workshops, Doctoral Consortium and Tutorials, Tampere, Finland, September 23–26, 2025, Proceedings (p. 233). Springer Nature.