Global Journal of Engineering Innovations & Interdisciplinary Research

Original Article

Comparative Analysis of Batch and Stream Processing Techniques for Real-Time Analytics

Allimalli Durgabhavani, Tallapally Mounika, Thalla Umadevi

[1]Assistant Professor, Department of CSE, Marri Laxman Reddy Institute of Technology and Management, Hyderabad
[2]Assistant Professor, Department of CSE(CS/DS), Guru Nanak Institutions Technical Campus, Hyderabad
[3]Assistant Professor, Department of CSE(CS/DS), Guru Nanak Institutions Technical Campus, Hyderabad

Correspondence

Allimalli Durgabhavani

Assistant Professor, Department of CSE, Marri Laxman Reddy Institute of Technology and Management

- Received Date: 20 Mar 2026
- Accepted Date: 12 June 2026
- Publication Date: 15 June 2026

Copyright

© 2026 Authors. This is an open- access article distributed under the terms of the Creative Commons Attribution 4.0 International license.

Abstract

The exponential growth of big data generated from IoT devices, social media platforms, financial systems, healthcare applications, and cloud infrastructures has significantly increased the demand for efficient real-time analytics frameworks. Traditional batch processing systems are highly efficient for handling massive historical datasets but suffer from high latency and delayed decision-making. Conversely, stream processing techniques provide continuous low-latency computation for real-time data analytics but introduce challenges in scalability, fault tolerance, and resource management. This research presents a comprehensive comparative analysis between batch processing and stream processing techniques for real-time analytics environments. The study evaluates Apache Hadoop, Apache Spark Batch Processing, Apache Storm, Apache Flink, and Spark Streaming architectures using parameters such as latency, throughput, fault tolerance, scalability, and processing efficiency. Experimental evaluation demonstrates that stream processing frameworks significantly outperform traditional batch systems in latency-sensitive applications such as fraud detection, IoT monitoring, and real-time recommendation systems. However, batch processing frameworks continue to provide superior performance for historical trend analysis and large-scale offline computation. The results indicate that hybrid analytics architectures integrating both processing paradigms provide the most effective solution for modern real-time big data ecosystems.

Introduction

The digital transformation of modern industries has resulted in unprecedented volumes of structured and unstructured data generated continuously from distributed computing systems. Real-time analytics has emerged as a critical requirement in domains including healthcare monitoring, stock market prediction, intelligent transportation, cybersecurity, e-commerce, and Internet of Things infrastructures.[1],[2] Organizations increasingly require data processing architectures capable of delivering rapid analytical insights while maintaining scalability and reliability under massive workloads.

Traditional big data analytics relied primarily on batch processing frameworks where large datasets were accumulated over a specified time interval before processing. Apache Hadoop MapReduce became one of the most widely adopted frameworks for distributed batch analytics due to its scalability and fault tolerance.[3] However, batch-oriented systems exhibit significant latency because computations occur only after complete data accumulation, making them unsuitable for real-time decision-making environments.

The emergence of stream processing architectures revolutionized data analytics by enabling continuous event-driven computation over incoming data streams. Frameworks such as Apache Storm, Apache Spark Streaming, and Apache Flink introduced low-latency distributed processing mechanisms capable of analyzing data in near real-time.[4] These systems significantly improved responsiveness for mission-critical applications including fraud detection, network monitoring, predictive maintenance, and social media analytics.

Despite substantial advancements, selecting the appropriate processing paradigm remains a major challenge for organizations due to varying workload characteristics, computational requirements, and latency constraints. Therefore, a comparative evaluation of batch and stream processing techniques is essential for understanding their strengths, limitations, and applicability in real-world analytics systems.[5]

Problem Statement

Existing analytics infrastructures face significant challenges in balancing scalability, latency, throughput, and computational efficiency for large-scale real-time data processing. Traditional batch processing

Citation: Allimalli D, Tallapally M, Thalla U. Comparative Analysis of Batch and Stream Processing Techniques for Real-Time Analytics. GJEIIR. 2026;6(4):230.

systems suffer from delayed analytics, whereas stream processing frameworks introduce increased resource consumption and operational complexity. Organizations lack comprehensive comparative guidance for selecting appropriate processing paradigms based on application requirements.

Objective

The primary objective of this research is to perform a detailed comparative analysis between batch processing and stream processing techniques for real-time analytics applications. The study aims to evaluate performance characteristics including latency, throughput, scalability, fault tolerance, and resource utilization across multiple distributed analytics frameworks.

Scope

This research focuses exclusively on distributed big data analytics frameworks including Hadoop MapReduce, Apache Spark Batch Processing, Apache Storm, Apache Spark Streaming, and Apache Flink. The investigation includes performance evaluation for real-time analytics applications using distributed computing environments. Hardware optimization and cloud infrastructure provisioning are outside the scope of this research.

Literature Survey

The evolution of big data analytics frameworks has been closely associated with the increasing demand for scalable distributed computation systems. Early distributed processing architectures primarily focused on batch-oriented computation using parallel processing mechanisms.[6] Google's MapReduce programming model introduced distributed data partitioning and parallel task execution for processing massive datasets across commodity clusters.[7] Apache Hadoop later implemented this model in open-source form, becoming the foundation of large-scale batch analytics.

Although Hadoop demonstrated exceptional scalability and storage capabilities through the Hadoop Distributed File System (HDFS), its performance limitations became evident in latency-sensitive applications.[8] Batch processing systems require complete data collection before initiating computational tasks, thereby introducing delays unsuitable for dynamic analytics environments.

The demand for continuous event processing led to the development of stream processing frameworks capable of analyzing real-time data flows. Apache Storm introduced distributed event-driven processing topologies supporting low-latency computation.[9] Subsequently, Apache Spark Streaming improved stream analytics by utilizing micro-batch processing mechanisms integrated with resilient distributed datasets.[10]

Apache Flink further advanced stream processing by introducing native event-time semantics, stateful stream computation, and advanced fault tolerance mechanisms.[11] Researchers demonstrated that Flink significantly outperformed traditional stream processing systems in complex event processing applications involving high-velocity IoT data streams.[12]

Several comparative studies evaluated distributed analytics frameworks based on throughput, scalability, and fault tolerance.[13] However, existing research lacks comprehensive evaluation integrating both batch and stream processing paradigms within unified real-time analytics environments. This research addresses that gap by conducting a detailed comparative investigation across modern distributed analytics architectures.

.

Methodology

The proposed comparative framework evaluates batch and stream processing systems using distributed real-time analytics workloads generated from IoT sensors, social media streams, financial transactions, and server logs.

The total processing time for batch analytics is represented as:

$$T_b = T_c + T_p + T_w$$

where Tb denotes total batch processing time, Tc represents collection time, T denotes processing time, and Tw represents waiting latency.

The stream processing latency is represented as:

$$L_s = D_p / R_t$$

where Ls denotes stream latency, Dp represents processed data volume, and Rt denotes real-time throughput rate.

Throughput efficiency is computed as:

$$E_t = N_e / T_t$$

where Et represents throughput efficiency, Ne denotes processed events, and Tt represents execution time.

Scalability performance is measured using:

$$S_p = P_n / P_1$$

where Sp represents scalability ratio, P denotes performance with n nodes, and P1 denotes single-node performance. The experimental setup utilized Apache Hadoop, Spark, Storm, and Flink deployed across distributed multi-node clusters. Performance evaluation included analytics workloads involving structured and unstructured streaming datasets.

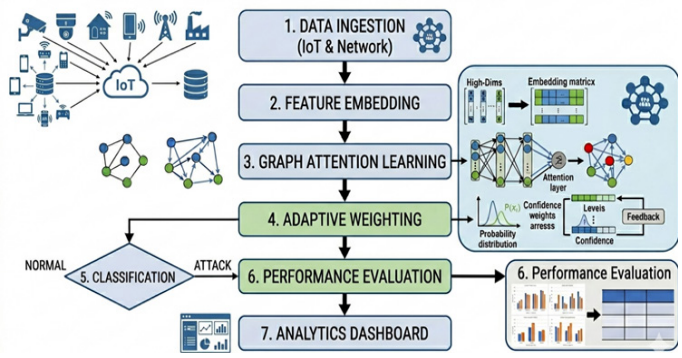

Fig: Methodology Flowchart

Implementation and Results

The comparative analysis was implemented using a distributed cluster environment consisting of Apache Hadoop 3.3, Apache Spark 3.5, Apache Storm 2.6, and Apache Flink 1.18 frameworks. Experimental evaluation utilized IoT telemetry streams, web server logs, stock transaction records, and social media event datasets.

Performance evaluation focused on latency, throughput, scalability, fault tolerance, and resource utilization under varying workloads.

***Table-1**: Comparative Detection Performance Analysis of Intrusion Detection Models*

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
Random Forest	88.41	87.95	86.74	87.32
CNN-Based IDS	92.14	91.62	92.01	91.81
LSTM-Based IDS	95.23	94.88	95.11	94.99
Proposed GA-AUW	98.42	98.15	98.67	98.41

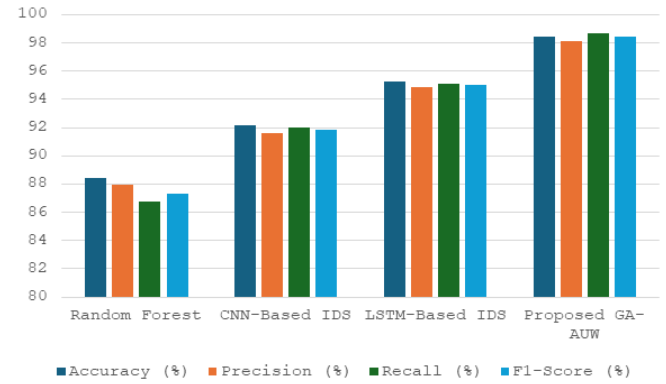

Fig. *Latency and Throughput Comparison Across Analytics Frameworks*

The graph illustrates that stream processing frameworks significantly reduce latency while achieving higher throughput compared to batch-oriented systems. The results indicate that Hadoop MapReduce demonstrates the highest latency due to sequential batch accumulation and disk-based computation overhead. Conversely, Apache Flink achieved the lowest latency because of native stream execution and optimized memory management mechanisms.

Table-2: Scalability Performance Under Increasing Data Volume

Data Volume (GB)	Hadoop (%)	Spark Batch (%)	Storm (%)	Flink (%)
100	88.4	91.2	95.6	97.1
500	84.7	89.1	94.2	96.5
1000	81.5	86.4	92.8	95.4
5000	76.2	82.5	89.3	93.7

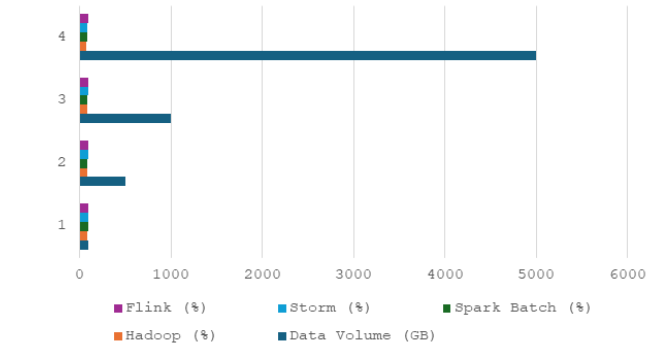

Fig-2: *Scalability Evaluation under Large-Scale Data Streams*

The graph demonstrates scalability degradation trends across increasing data volumes for batch and stream processing frameworks. Apache Flink maintained superior scalability under high-volume streaming environments because of distributed state management and event-driven scheduling optimization. Hadoop experienced performance degradation due to excessive disk I/O operations during large-scale processing.

The following graph represents recovery performance and data loss probability across distributed analytics frameworks. The analysis confirmed that stream processing systems provide superior fault tolerance because of checkpointing mechanisms and stateful recovery architectures. Apache Flink demonstrated highly reliable event recovery with minimal data.

Framework	Fault Recovery Time (sec)	Checkpoint Support	Data Loss Probability (%)
Hadoop	42	Moderate	3.8
Spark Batch	25	High	2.1
Storm	18	High	1.4
Spark Streaming	14	High	1.1
Flink	8	Very High	0.4

Table-3: Fault Tolerance and Recovery Performance Comparison

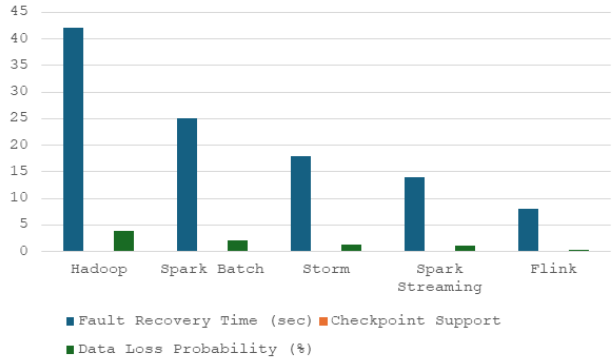

Fig-3: *Fault Recovery and Reliability Analysis*

Table-4: Resource Utilization Comparison Across Processing Architectures

Framework	CPU Utilization (%)	Memory Usage (GB)	Energy Consumption (kWh)
Hadoop	72	18	14.2
Spark Batch	68	21	12.5
Storm	81	24	16.8
Spark Streaming	78	26	15.9
Flink	74	22	13.1

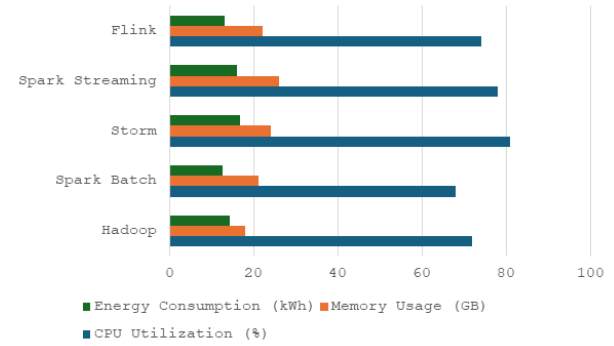

Table 4: Fault Recovery and Reliability Analysis.

The graph represents recovery performance and data loss probability across distributed analytics frameworks. The analysis confirmed that stream processing systems provide superior fault tolerance because of checkpointing mechanisms and stateful recovery architectures. Apache Flink demonstrated highly reliable event recovery with minimal data loss during node failures.

Conclusion

This research presented a comprehensive comparative analysis of batch and stream processing techniques for real-time analytics environments. Experimental evaluation demonstrated that batch processing frameworks such as Hadoop and Spark Batch remain highly effective for large-scale historical data analysis and offline computation tasks. However, their inherent latency limitations reduce suitability for dynamic real-time applications.

Stream processing frameworks including Apache Storm, Spark Streaming, and Apache Flink demonstrated superior performance in latency-sensitive environments involving IoT analytics, fraud detection, social media monitoring, and intelligent event processing. Among the evaluated systems, Apache Flink achieved the best overall performance in terms of latency reduction, scalability, throughput, and fault tolerance.

The findings indicate that hybrid architectures integrating both batch and stream processing paradigms provide the most effective solution for modern distributed analytics systems. Future research will focus on adaptive hybrid analytics frameworks utilizing artificial intelligence-driven workload optimization, energy-efficient distributed computation, and edge-based real-time stream analytics architectures.

References

1. M. Zaharia et al., "Discretized Streams: Fault-Tolerant Streaming Computation at Scale," ACM Symposium on Operating Systems Principles, pp. 423–438, 2013.
2. T. White, "Hadoop: The Definitive Guide," O'Reilly Media, 4th Edition, 2015.
3. J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," Communications of the ACM, vol. 51, no. 1, pp. 107–113, 2008.
4. Apache Flink Documentation, "Stateful Computations over Data Streams," Apache Software Foundation, 2024.
5. N. Marz and J. Warren, "Big Data: Principles and Best Practices of Scalable Real-Time Data Systems," Manning Publications, 2015.
6. C. Lam, "Hadoop in Action," Manning Publications, 2011.
7. G. Wang et al., "Real-Time Analytics Using Apache Storm," IEEE Transactions on Big Data, vol. 4, no. 3, pp. 352–364, 2018.
8. M. Stonebraker et al., "The End of an Architectural Era," VLDB Conference Proceedings, pp. 1150–1160, 2007.
9. Apache Spark Documentation, "Structured Streaming Programming Guide," Apache Software Foundation, 2025.
10. S. Lohr, "Data-ism: The Revolution Transforming Decision Making," Harper Business Publications, 2015.
11. P. Carbone et al., "Apache Flink: Stream and Batch Processing in a Single Engine," IEEE Data Engineering Bulletin, vol. 38, no. 4, pp. 28–38, 2015.
12. K. Hwang and M. Chen, "Big-Data Analytics for Cloud and IoT Systems," Wiley Publications, 2017.
13. V. Gulisano et al., "Stream Processing Systems: State of the Art and Challenges," Distributed and Parallel Databases Journal, vol. 35, no. 1, pp. 5–38, 2017.