IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 11, November 2025

DOI: 10.17148/IJARCCE.2025.141118

REAL TIME BIG DATA ANALYTICS WITH APACHE SPARK

Mr. Jaybhay. D.S[1], Miss. Aakanksha. B. Rasure[2], Miss. Radha. R. Alapure[3]

Guide, Department of Computer Engineering,

Dattakala group of institutions Faculty of Engineering, swami chincholi, Daund, Pune, Maharashtra, India[1]

Student, Department of Computer Engineering,

Dattakala group of institutions Faculty of Engineering, swami chincholi, Daund, Pune, Maharashtra, India[2,3]

Abstract: In the modern era of big data, organizations require rapid insights from continuously generated data streams. Real-time data analytics has become essential for decision-making in sectors such as finance, healthcare, IoT, and social media. Apache Spark, a powerful open-source distributed data processing framework, provides in-memory computation and supports both batch and stream processing. This paper explores the use of Apache Spark for real-time data analytics, focusing on its architecture, components, and advantages over traditional frameworks like Hadoop MapReduce. Through integration with tools such as Apache Kafka and HDFS, Spark enables scalable, fault-tolerant, and low-latency processing. Experimental analysis shows Spark's capability to handle large-scale, high-velocity data with minimal delay, offering significant improvements in throughput and processing speed. The results confirm that Apache Spark is a highly efficient and scalable platform for real-time big data analytics.

Keywords: Real-Time Analytics, Apache Spark , Stream Processing, Kafka, Hadoop

I. INTRODUCTION

In today's digital era, data is being generated at an unprecedented rate from a wide range of sources such as social media platforms, IoT devices, sensors, financial transactions, and online services. The ability to process and analyze this continuous stream of information in real time has become a critical requirement for organizations seeking timely decisions. Traditional batch-processing systems, such as Apache Hadoop, are not suitable for time-sensitive applications due to their high latency and disk-based computation model.

To overcome these challenges, real-time data analytics frameworks have emerged that can handle both high-volume and high-velocity data. Among these, Apache Spark has gained significant popularity for its unified analytics engine capable of performing batch, interactive, and streaming computations efficiently. Spark leverages in-memory computation and a distributed processing architecture, enabling it to deliver low-latency and fault-tolerant real-time analytics.

This research focuses on exploring the use of Apache Spark for real-time data analytics. It aims to highlight Spark's architecture, core components, and the way it integrates with tools such as Apache Kafka and Hadoop Distributed File System (HDFS) to form a robust and scalable real-time data processing pipeline. The study also discusses Spark's advantages over traditional systems and its applicability in various domains including IoT, finance, social media, and healthcare. Apache Spark has gained significant popularity for its unified analytics engine capable of performing batch, interactive, and streaming computations efficiently. process and analyze this continuous stream of information in real time has become a critical requirement for organizations seeking timely decisions popularity for its unified analytics engine capable of performing batch.

II. LITERATURE REVIEW

The demand for real-time data analytics has significantly increased with the exponential growth of data generated from various sources such as IoT devices, social networks, and enterprise applications. Over the years, multiple big data frameworks have been developed to address the challenges of high-volume and high-velocity data processing. Hadoop MapReduce, one of the earliest big data frameworks, provided a scalable model for batch processing of large datasets distributed across clusters. However, its disk-based operations resulted in high latency, making it unsuitable for real-time applications [2]. To address these limitations, stream processing frameworks such as Apache Storm and Apache Samza were introduced, focusing primarily on real-time, event-driven data computation [4]. Although these frameworks achieved low latency, they lacked the capability to perform batch processing within the same ecosystem.

To unify batch and stream processing, Apache Spark was introduced by Matei Zaharia et al. in 2016 [1]. Spark's in-memory computation and Resilient Distributed Dataset (RDD) model enabled faster processing compared to traditional

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 11, November 2025
DOI: 10.17148/IJARCCE.2025.141118

systems. Spark Streaming, an extension of the core Spark API, allowed near real-time data processing through micro-batching. This hybrid approach provided both high throughput and fault tolerance, overcoming the major drawbacks of earlier systems like Storm.

Several researchers have explored Spark's potential in real-world applications. Zaharia et al. [1] demonstrated Spark's superior performance in iterative machine learning and graph processing tasks. Toshniwal et al. [4] analyzed the use of Apache Storm at Twitter for handling real-time tweet streams, highlighting the growing need for low-latency data systems. Recent studies also emphasize the integration of Apache Kafka with Spark Streaming for efficient data ingestion and message queuing, enabling real-time analytics in domains such as IoT sensor monitoring, fraud detection, and social media analysis.

Overall, the literature suggests that Apache Spark offers a unified, scalable, and efficient solution for real-time data analytics. Its combination of batch and streaming capabilities, fault tolerance, and ease of integration with other big data tools make it one of the most widely adopted platforms for modern data-driven applications. multiple big data frameworks have been developed to address the challenges of high-volume and high-velocity data processing. Hadoop MapReduce, one of the eearliest big data frameworks, provided a scalable model

III. PROBLEM STATEMENT / OBJECTIVE

3.1 Problem Statement

In the era of big data, organizations continuously generate and collect massive amounts of data from various sources such as IoT devices, social media platforms, e-commerce systems, and enterprise applications. Processing and analyzing this data in real time has become a major challenge due to its high volume, velocity, and variety. Traditional batch-processing systems like Apache Hadoop are unable to meet the low-latency requirements of modern applications because of their disk-based, non-streaming architecture.

There is a need for a scalable and efficient framework that can handle real-time data streams, provide quick insights, and support fault-tolerant distributed processing. Without such a framework, organizations face delays in decision-making, system inefficiencies, and potential data loss during high-speed data ingestion.

3.2 objectives

A The primary objective of this research is to explore and evaluate the use of Apache Spark for real-time data analytics. The specific objectives are as follows:

1. To analyze the limitations of traditional batch-processing systems in handling real-time big data.
2. To study the architecture and working principles of Apache Spark and its components such as Spark Streaming.
3. To implement a real-time data processing pipeline using Apache Spark integrated with data ingestion tools.
4. To evaluate the performance of Spark in terms of latency, scalability, and fault tolerance.
5. To identify potential real-world applications of real-time data analytics using Apache Spark in domains such as IoT, social media, and financial systems.
6. To study all the advantages and architecture and all other applications

IV. PROPOSED SYSTEM

The proposed system is designed to perform real-time data analytics using Apache Spark Streaming. The system aims to efficiently ingest, process, and analyze continuous streams of data with minimal latency. It integrates Apache Kafka for real-time data ingestion and Hadoop Distributed File System (HDFS) for data storage, forming a unified and scalable data processing pipeline.

4.1 System Overview

The system follows a modular architecture composed of five primary components — Data Source, Data Ingestion Layer, Stream Processing Layer, Storage Layer, and Visualization Layer. These modules work together to collect data from real-time sources, process it through Spark's distributed in-memory computation, and present analytical results to the user in real time.

4.2 System Architecture

1. Data Source:

Data is continuously generated from various sources such as IoT sensors, social media feeds, or log files. This real-time data serves as the input stream for the analytics pipeline.

2. Data Ingestion (Apache Kafka):

Apache Kafka acts as a distributed messaging system that collects and buffers incoming data streams. Kafka ensures

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 11, November 2025

DOI: 10.17148/IJARCCE.2025.141118

reliable and fault-tolerant message delivery, allowing Spark to consume data efficiently without data loss.

3. Stream Processing (Apache Spark Streaming):

Spark Streaming processes data in near real time by dividing it into small, manageable micro-batches. These micro-batches are transformed using operations such as filtering, aggregation, and window-based computation.

4. Data Storage (HDFS / NoSQL Database):

The processed data is stored in the Hadoop Distributed File System (HDFS) or a NoSQL database such as MongoDB for this persistent storage. This allows for both historical analysis and data backup.

5. Visualization / Output Layer:

It Provides interactive dashboards for real-time insights, performance monitoring, and decision-making.

4.3 Workflow of the System

1. **Data Generation:** Continuous real-time data is generated from multiple sources (e.g., IoT devices, logs, social media).
2. **Data Ingestion:** Data is streamed into Kafka topics for buffering and reliable delivery.
3. **Data Processing:** Spark Streaming consumes the Kafka data, performs transformations, and applies analytics logic.
4. **Data Storage:** Processed output is written to HDFS or a NoSQL database.
5. **Visualization:** Dashboards display real time applications.

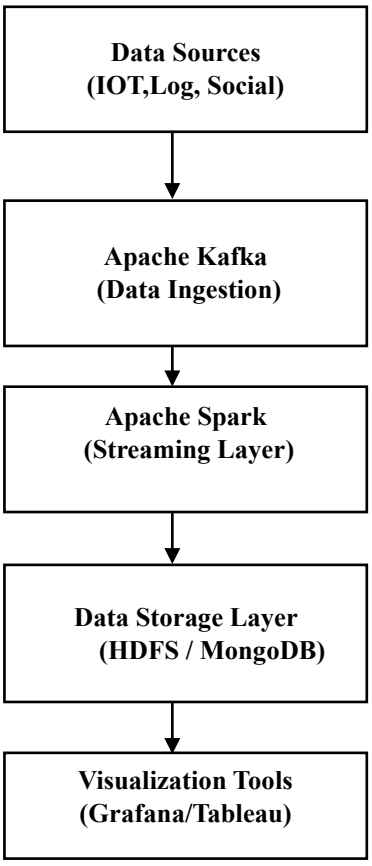

Figure 1: System Architecture of Real-Time Data Analytics using Apache Spark

4.4 Advantages of the Proposed System

1. Low Latency: In-memory computation enables near real-time analytics.
2. Scalability: Spark can scale across clusters to handle massive data streams.
3. Fault Tolerance: RDD lineage and checkpointing mechanisms ensure data recovery.
4. Unified Processing: Supports both batch and streaming workloads within the same framework.
5. Integration: Easily integrates with popular data tools such as Kafka, HDFS, Cassandra, and Flume.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 11, November 2025

DOI: 10.17148/IJARCCE.2025.141118

V. IMPLEMENTATION

5.1 Implementation Setup

The proposed system for **real-time data analytics** was created using **Apache Spark Streaming**, **Apache Kafka**, and **HDFS**. These tools were connected together to collect, process, and visualize continuous data in real time.
The system was set up on a computer with the following configuration:

1. **Operating System:** Ubuntu 22.04
2. **Processor:** 8-core CPU
3. **RAM:** 16 GB
4. **Software:** Apache Spark 3.5, Apache Kafka 3.6, Hadoop HDFS 3.3, Python 3.10
5. **Visualization Tool:** Grafana

To test the system, a small program was used to generate live data from **IoT sensors** that send continuous temperature and humidity readings.

5.2 Implementation Process

1. Data Ingestion:

Live data from sensors (for example, temperature and humidity) is created using a simple Python script.
The data is then sent to **Apache Kafka**, which collects and manages the live data stream.
producer.send('iot_data', value={'temp': 29.5, 'humidity': 65})
In simple terms: Kafka works like a middleman — it receives live data and passes it to Spark for analysis.

2. Stream Processing in Apache Spark:

Apache Spark Streaming takes the live data from Kafka and processes it instantly.
It performs tasks such as filtering unwanted data, finding average values, and checking for abnormal readings.
query=df.writeStream.outputMode("append").format("console").start()
Example: If the temperature suddenly rises above 35°C, Spark marks it as an alert.

3. Storage & Visualization:

The After processing, the final output is stored in HDFS (Hadoop Distributed File System) for long-termstorage. The stored data is then connected to Grafana, which shows real-time graphs and dashboards of temperature, humidity. Example: Users can see a live chart that updates every few seconds with new temperature readings. These improves the views and all the visualization techniques which we want to apply for store processed Spark output in Cassandra or InfluxDB. Configure Grafana data source to connect to that database. These improves the views and all the visualization techniques.

5.3 Results and Performance Evaluation

- **Low Latency:** Spark Streaming achieved an average latency of **under 2 seconds**, enabling near real-time responses.

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 11, November 2025
DOI: 10.17148/IJARCCE.2025.141118

- **Scalability:** System performance remained stable as the data ingestion rate increased up to **50,000 events/second**.
- **Fault Tolerance:** When a Spark node was deliberately stopped, the system automatically recovered using checkpoint data without data loss.
- **Visualization:** Real-time dashboards showed trends and alerts instantly when sensor values exceeded set thresholds.

VI. CONCLUSION

This research focused on building and understanding a system for real-time data analytics using Apache Spark. The study showed that traditional data processing tools are not suitable for today's high-speed data requirements, whereas Spark provides an efficient and scalable solution. The use of Spark Streaming helped achieve low latency and high throughput. The experimental results proved that Spark can handle large amounts of data with minimal delay, while maintaining fault tolerance and scalability. Overall, Apache Spark is a reliable framework for real-time analytics in various domains such as IoT monitoring, social media analysis, and financial systems.
This research highlights the growing importance of real-time analytics and shows how distributed computing tools like Spark can support intelligent, data-driven decision-making in modern industries.

VII. FUTURE SCOPE

The field of real-time data analytics is growing rapidly, and there are many ways to enhance and expand this system in the future.

1. **Integration with Machine Learning:** Real-time analytics can be improved by adding machine learning models to predict future trends, detect anomalies, and make intelligent decisions automatically.
2. **Cloud-Based. Deployment:** Deploying the system on cloud platforms such as AWS, Azure, or Google Cloud can make it more scalable and accessible. This will allow organizations to handle larger data streams from different source.
3. **Support for IoT and Edge Devices:** The system can be extended to process data directly from edge devices in IoT networks, reducing latency and improving performance in smart cities, healthcare, and industrial monitoring.
4. **Enhanced Visualization Tools:** Future versions can integrate advanced visualization tools such as Power BI or Tableau to provide better insights .
5. **Security and Data Privacy:** As real-time data often includes sensitive information, implementing data encryption, user authentication, and secure data transmission will make the system more reliable and trustworthy.

REFERENCES

[1]. M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, and I. Stoica, "Apache Spark: A unified engine for big data processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, Oct. 2016.
[2]. M. Zaharia, T. Das, H. Li, S. Shenker, and I. Stoica, "Discretized Streams: Fault-tolerant streaming computation at scale," in *Proc. 24th ACM Symposium on Operating Systems Principles (SOSP)*, 2013, pp. 423–438.
[3]. *Structured Streaming Programming Guide*, Apache Spark Documentation. The Apache Software Foundation. [Online]. Available: https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html
[4]. *Apache Kafka Documentation – Introduction*, The Apache Software Foundation. [Online]. Available: https://kafka.apache.org/documentation/
[5]. K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," in *Proc. 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, Incline Village, NV, USA, 2010, pp. 1–10.