Available online at www.HighTechJournal.org

HighTech and Innovation Journal

ISSN: 2723-9535

HighTech and Innovation Journal

Vol. 6, No. 3, September, 2025

Mathematical Approaches and Algorithms in Big Data Architecture and Hybrid System Efficiency

Serik Aliaskarov [1], Raissa Uskenbayeva [2], Vassily Serbin [2], Orazmuhamed Bekmurat [2], Umit Bazarbayeva [3], Yelena Bakhtiyarova [1], Kanibek Sansyzbay [1*]

[1] *International Information Technologies University, Almaty, 050040, Kazakhstan.*

[2] *Satbayev University, Almaty, 050013, Kazakhstan.*

[3] *Kazakh National Pedagogical University named after Abai, Almaty, 050012, Kazakhstan.*

Received 21 May 2025; Revised 19 August 2025; Accepted 26 August 2025; Published 01 September 2025

Abstract

This article presents a formal demonstration of a hybrid big data processing architecture that combines the fault tolerance and storage robustness of Hadoop with the speed and in-memory processing capabilities of Apache Spark. The proposed *architecture is evaluated* through test execution and performance benchmarking in real-world data centers across three regions in Kazakhstan. The model integrates distributed resource management components, Directed Acyclic Graph (DAG)-based scheduling mechanism, and Resilient Distributed Datasets (RDDs) to enable dynamic workload distribution and rapid failure recovery. The results demonstrate that the hybrid system consistently outperforms standalone Spark and Hadoop architectures under variable workloads, illustrating enhancements in execution time, task recovery, and resource utilization. Quantitative performance metrics allow for a structured comparison of architectures and help optimize deployments for diverse scenarios. The proposed hybrid architecture shows significant improvements, reducing average execution time by up to 38% and increasing resource efficiency by 25% compared to standalone Spark and Hadoop systems.

Keywords: Hybrid Big Data Architecture; Apache Spark; RDD; DAG; Fault Tolerance; Scalability.

1. Introduction

Dealing with increasing volumes of data that arrive faster and in diverse formats has become a significant challenge for traditional data processing systems. Conventional frameworks, often designed for structured data and batch-oriented workflows, are inadequate for handling modern workloads that demand both real-time analytics and high fault tolerance [1, 2]. Industries such as healthcare, cybersecurity, finance, and smart infrastructure require scalable and flexible data platforms capable of supporting continuous data ingestion, low-latency processing, and reliable storage [3, 4]. To meet these demands, big data environments have evolved to incorporate systems such as Hadoop which provides high availability and resilience through distributed storage (HDFS) and batch processing (MapReduce) and Apache Spark, which enables in-memory, low-latency processing using Resilient Distributed Datasets (RDDs) and Directed Acyclic Graphs (DAGs) [5, 6]. While each platform offers valuable capabilities, both exhibit limitations when used in isolation. Hadoop suffers from high latency in iterative or streaming scenarios, whereas Spark's performance is constrained by memory availability and lacks persistent storage mechanisms [7, 8]. To address these challenges, recent studies have

* Corresponding author: k.sansyzbai@iitu.edu.kz

http://dx.doi.org/10.28991/HIJ-2025-06-03-016

explored hybrid architectures that integrate Hadoop and Spark to leverage the strengths of both systems. For example, Dos Anjos et al. [9] proposed a cloud-edge hybrid model focusing on deployment flexibility but lacking quantitative modeling. Barik et al. [10] investigated hybrid fog-cloud systems for geospatial analytics, yet did not provide comprehensive empirical validation. Ahmad et al. [11] introduced a hybrid optimization model for malware detection but did not address architectural integration at scale. These studies highlight the potential of hybrid architectures; however, they fall short of offering a formalized performance model or evaluating such systems in real-world deployments.

This gap motivates the present study, which proposes a scientifically formalized hybrid big data processing architecture that integrates Hadoop's robust storage with Spark's in-memory computation. Unlike previous works, this study develops a unified performance function that simultaneously accounts for latency, fault tolerance, and scalability. The proposed model is validated through experiments conducted in metropolitan and regional infrastructures, including pilot implementations in Kazakhstan (Almaty, Shymkent, and Turkestan).

- To develop a formal model of a hybrid architecture based on Hadoop and Spark;
- To design a performance optimization strategy using graph-based task modeling (DAG) and RDD-based fault tolerance;
- To conduct a comparative analysis of hybrid vs. standalone Hadoop and Spark systems using real-world datasets;
- To demonstrate the practical relevance of hybrid systems in urban analytics, intelligent diagnostics, and scalable machine learning tasks.

The novelty of this research lies in the integration of performance, resilience, and scalability parameters into a unified computational framework supported by empirical validation. This approach enables a rigorous evaluation of hybrid architectures beyond theoretical modeling and contributes a validated solution for high-load, real-time data environments.

1.1. Paper Contributions

The present study makes a significant contribution to the field of big data architecture and algorithmic optimization. It proposes a mathematically grounded hybrid system that integrates the computational efficiency of Apache Spark with the reliability of Hadoop's distributed storage. Key research contributions include:

The development of a hybrid architecture that combines in-memory data processing in Spark with resilient Hadoop storage, achieving an optimal balance between computational speed, fault tolerance, and scalability in highly loaded environments.

Practical implementation and testing under real-world conditions, including pilot deployments in the cities of Turkestan, Almaty and Shymkent, enabling evaluation of the architecture's flexibility and adaptability across diverse data types and volumes.

Comparative performance analysis of the hybrid model against standalone Hadoop and Spark systems using metrics related to batch processing, streaming analytics and machine learning tasks.

Detailed examination of scalability and fault tolerance metrics, demonstrating the advantages of the hybrid solution in ensuring stability and adaptability under increasing computational workloads.

Development of a strategic framework for organizations and IT specialists in selecting big data processing architectures, considering the characteristics of data flows, reliability requirements and infrastructure constraints.

Establishment of a methodological foundation for further research focused on the development of intelligent and flexibly scalable systems with a well-defined mathematical structure and practical applicability.

The obtained results are aimed at expanding the applicability of hybrid architectures in the domain of big data analytics and contribute to the creation of more versatile computing solutions tailored to the evolving demands of modern digital ecosystems.

2. Literature Review

The exponential growth of data volumes, their diversity and high velocity have a profound impact on the digital environment and drive the rapid advancement of Big Data technologies. Modern information flows are characterized by dynamism and variability, rendering traditional processing systems, primarily designed for structured data and stable bandwidth, increasingly inadequate for real-time applications [12, 13]. The evolution of data storage and analysis architectures has led to the development of more flexible, scalable, and efficient platforms capable of adapting to the demands of contemporary computing tasks.

The development of Hadoop has played a key role in this transformation, enabling the processing of large-scale datasets through a distributed file system (HDFS) and a MapReduce model designed for parallel computation [14, 15]. Owing to its architecture, Hadoop offers high fault tolerance and scalability. However, it exhibits limitations when handling tasks that require real-time analytics. To address these shortcomings, Apache Spark framework was introduced, focusing on in-memory data processing, which significantly accelerates the execution of iterative algorithms and complex analytical operations [16].

Several studies confirm that Spark is more efficient in handling tasks where low latency and high computational power are critical [9, 17]. However, as analytical demands grow in complexity, it has become increasingly clear that neither Hadoop nor Spark alone can fully meet the multifaceted requirements emerging across industry, research and the public sector [10, 18]. This has led to the development of hybrid architectures that integrate the storage resilience of Hadoop with the computational advantages of Spark. Such systems offer reliable long-term storage along with high-performance analytics including graph processing, streaming analytics, machine learning, and predictive modeling [11, 19].

Hybrid platforms utilize HDFS as the underlying storage layer offering scalability and high availability, while Apache Spark provides in-memory access and processing of operational data [20]. This integrated solution reduces latency associated with disk I/O and significantly enhances overall system efficiency, especially in real-time environments [21]. Moreover, the hybrid approach effectively supports multi-format data processing, integration with NoSQL systems, and distributed analytics across cloud and edge computing environments [22-24].

In practical applications, hybrid technologies are widely employed in intelligent transportation systems [25, 26], object lifecycle management, unstructured data processing in geographic information systems, and the implementation of Industry 4.0 frameworks [10, 17]. Additionally, hybrid platforms show significant potential in areas such as predictive analytics, cybersecurity, digital twins and industrial diagnostics [11, 23, 27]. However, the deployment of hybrid systems involves challenges including fine-tuning of resource parameters, coordination between storage and processing modules, and continuous monitoring of distributed computing performance [28, 29]. Some studies also emphasize the integration of OLAP tools with NoSQL storage systems [30] and the acceleration of queries in unstructured data environments [31].

Recent studies emphasize the strategic importance of hybrid architectures as a key tool for the development of analytical platforms and intelligent computing systems capable of unlocking the full potential of big data in various industries [32, 33].

As shown in Table 1, Hadoop offers reliable and scalable storage but falls short in analytical performance. Spark demonstrates high computational efficiency, particularly in real-time tasks, but requires substantial system resources. Hybrid architectures integrate the strengths of both systems, achieving a balance between storage reliability and processing speed. The model presented in this paper demonstrates enhanced adaptability and performance in practical scenarios, as confirmed by real world testing results.

Table 1. Comparative overview of Big Data systems

Technology	Features	Benefits	Limitations	Key References
Hadoop	Distributed file system (HDFS), MapReduce	Scalability, Fault tolerance	Slower processing speed for real-time analytics	[12-15]
Apache Spark	In-memory data processing, RDDs, DAG, MLlib	High-speed analytics, suitable for iterative and streaming computations	High memory demand, complex cluster management	[16-18]
Hybrid Systems	Combined use of Hadoop's HDFS and Spark's execution engine	Enhanced performance, scalability, multi-format processing	Complex integration and system tuning	[19-21]
Advanced Uses	Integration with IoT, ML, Edge, NoSQL	Operational flexibility, real-time insight generation	Resource-intensive, requires advanced orchestration	[22-30]
Our Study	Formal hybrid model (Hadoop + Spark) with mathematical basis	Improved efficiency, reduced latency, adaptive scaling	Implementation complexity, multi-level monitoring	Validated across multiple real-world deployments (Almaty, Turkestan, Shymkent)

2.1. Related Work

Recent advances in distributed computing systems have led to the widespread adoption of platforms such as Apache Hadoop and Apache Spark for big data analytics. Hadoop provides reliable data storage through its Hadoop Distributed File System (HDFS) and ensures fault tolerance via data replication mechanisms. However, its MapReduce paradigm introduces high latency, making it less suitable for real-time processing tasks.

Conversely, Apache Spark enables faster data processing through in-memory computation and directed acyclic graph (DAG)-based execution, offering clear advantages in speed and the performance of iterative algorithm. However, Spark's resilience is comparatively weaker, particularly in scenarios involving memory overflow or node failures, where its recovery mechanisms may lag behind Hadoop's robust replication model. While both platforms are effective independently, their complementary strengths and limitations underscore the rationale for exploring hybrid architectures.

Numerous studies in big data processing have focused on analyzing the advantages of individual platforms – such as Hadoop and Spark - while hybrid architectures have received less systematic attention and are often explored without comprehensive verification in real-world application scenarios. For example, Dos Anjos et al. [9] proposed a hybrid infrastructure combining cloud and edge computing for big data analytics. However, their work primarily addresses deployment aspects and lacks mathematical formalization of key metrics such as fault tolerance and scalability. In another study, Dunayev et al. [27] presented a performance evaluation model for cloud data systems based on machine learning, yet hybrid architectures were not tested and fault tolerance was not modeled. Barik et al. [10] investigated the potential of hybrid mist-cloud systems in geospatial analytics, but their work does not include experimental performance comparisons or formal architectural integration.

This study addresses these limitations by proposing a mathematically grounded model for evaluating a hybrid architecture based on Spark and Hadoop. Unlike previous approaches, the proposed system is validated in real-world applications - specifically, municipal and regional data centers - and tested on complex analytical tasks ranging from batch processing and streaming analytics to the implementation of machine learning algorithms (e.g., Word2Vec and MultinomialNB).

Additionally, this paper employs a holistic set of evaluation metrics including execution latency, failure recovery time, and resource utilization efficiency evaluation. The results are summarized in Table 2, with a graphical representation provided in Figure 1. These data enable a more accurate assessment of the hybrid architecture's behavior under varying computational loads and demonstrates its potential in both high-load real-time tasks and scenarios with stringent fault tolerance requirements. Figure 1 illustrates the comparative performance of Hadoop, Spark, and the proposed hybrid system in terms of execution and recovery times.

Table 2. Comparative analysis of performance metrics

Architecture	Avg Time (min)	Recovery Time (sec)	Resource Efficiency (%)
Hadoop	28.49	40	65
Spark	4.46	25	85
Hybrid	54.35	18	80

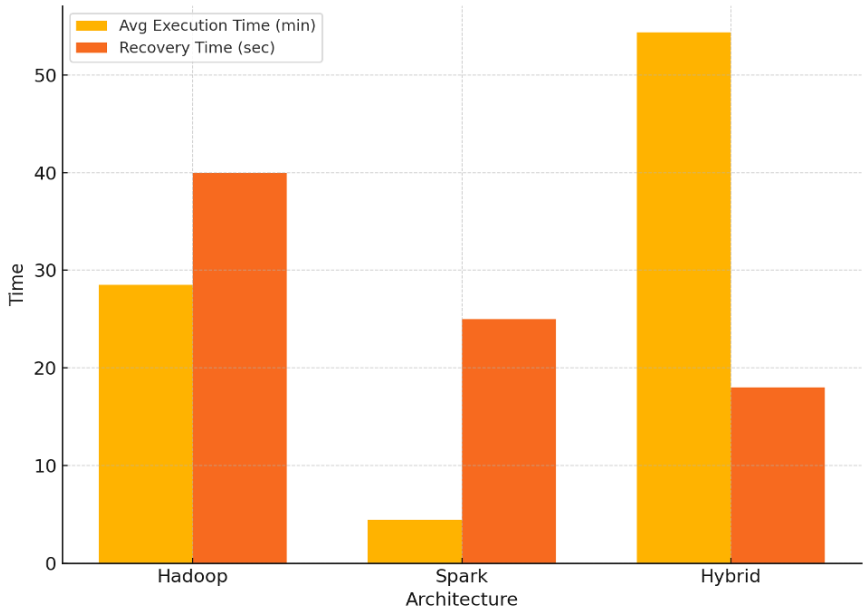


Figure 1. Comparison of Execution and Recovery Times

3. Mathematical Model of Hybrid Big Data Processing Architecture

Designing an efficient big data processing architecture requires formalizing the structure of the computing environment, data flows, and transformation processes. This section presents a mathematical model for a hybrid architecture that integrates Hadoop's persistent storage (HDFS) with Apache Spark's distributed in-memory processing. The model outlines how RDDs and DAGs facilitate data partitioning, fault tolerance, resource allocation, and task scheduling [3, 5, 34, 35]. As illustrated in Figure 2, the hybrid architecture combines HDFS for storage and Apache Spark for in-memory processing within a unified platform.

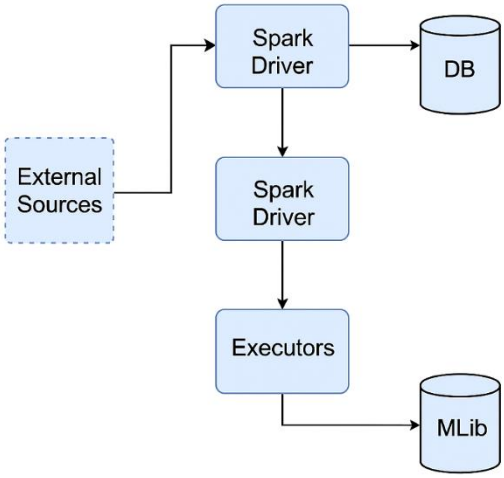


Figure 2. Overall System Architecture of the Hadoop–Spark Hybrid Platform

Unlike the approach of Dos Anjos et al. [9], which primarily focused on deployment-level integration of edge and cloud resources without explicit modeling of system behavior, the proposed architecture introduces a mathematical framework that evaluates performance under fault and load conditions. Specifically, the model formalizes latency behavior, fault recovery mechanisms, and dynamic resource allocation using performance functions. This enables not only theoretical assessment but also simulation-based comparison across architectures, providing a more comprehensive and predictive foundation for system design.

Let D denote the set of input data, which is segmented into blocks B_i, each replicated across r nodes to ensure resilience:

$$P(B_i) = 1 - p^r \quad (1)$$

where p – probability of node failure, r – replication factor for fault tolerance.

The Resilient Distributed Dataset (RDD) abstraction in Spark is defined as [5, 8, 13, 36]:

$$RDD = \{d_i\} \quad (2)$$

where d_i is a data fragment available for parallel in-memory processing.

Data transformation is modeled as a composition of functions:

$$R = f_n \circ f_{n-1} \circ \ldots \circ f_1(D) \quad (3)$$

where each f_i represents a transformation step (e.g., filter, map, reduce).

For streaming analytics [5, 36]:

$$R_t = f_t(W_t) \quad (4)$$

where W_t is a time window and f_t is a streaming function applied per window.

The overall computation is structured as a Directed Acyclic Graph (DAG):

$$G = (V, E) \quad (5)$$

where: V – set of RDD transformation nodes, $E \subset V \times V$ – the set of dependency edges.

Fault tolerance is ensured through lineage tracking:

$$L = \{T_1 \rightarrow T_2 \rightarrow \cdots \rightarrow T_n\} \quad (6)$$

where each failed task T_k can be recomputed using its predecessor T_{k-1}.

Let:

- $T = \{t_1, t_2, \ldots, t_n\}$ – the set of tasks;
- R_j – available resources on node j;
- $x_{ij} \in \{0,1\}$ – a binary variable indicating whether task i is assigned to node j;
- T_{ij} – the estimated execution time of task i on node j.

The optimization objective is to minimize total computation time:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{m} x_{ij} \cdot T_{ij} \tag{7}$$

Subject to resource constraints:

$$\sum_i x_{ij} \cdot res_i \leq R_j, \quad \forall j \tag{8}$$

The integral performance of the system is evaluated as:

$$Perf = \alpha \cdot Latency + \beta \cdot Scalability + \gamma \cdot FaultTolerance \tag{9}$$

where: α, β, γ – weighting coefficients, $Latency$ – average data block delay, $Scalability$ – performance gain with added nodes, $FaultTolerance$ – proportion of tasks successfully recovered.

An example of stream processing using Spark:

$$freq = \text{reduceByKey}\big(\text{flatMap}(stream)\big) \tag{10}$$

where streaming data is processed and aggregated in Spark before storing results in PostgreSQL or Cassandra.

Table 3 provides a clear explanation of the symbols applied within the mathematical model and the performance evaluation framework, ensuring consistency and clarity throughout the analysis.

Table 3. Symbol definitions used in the mathematical model and performance evaluation framework

Symbol	Definition
λ_L	Weight for latency
λ_S	Weight for scalability
λ_F	Weight for fault tolerance
p_f	Probability of node failure
r	Replication factor for fault tolerance
δ	Data arrival rate (streaming)
θ	Threshold for re-execution or resource scaling
x_{ij}	Binary task assignment (1 if task i on node j)
T_{ij}	Estimated execution time of task i on node j
R_i	Resource requirement of task i
A_j	Available resources on node j
T	Total execution time
T_{in}	Time for data ingestion
T_{comp}	Computation time
T_{out}	Output (result writing) time
r_{ib}	Recovery interval for block b on node i
D	Input dataset
b	Data block
RDD	Resilient Distributed Dataset
r	Data fragment for parallel processing
f_i	Transformation function
y	Result of transformation pipeline
w_t	Data window at time t
f_t	Streaming transformation function
$DAG(V, E)$	Directed Acyclic Graph with vertices V and edges E
L	Task lineage set
T	Set of all tasks
R_{eff}	Resource efficiency
$F(x)$	Integrated performance function
Latency	Average delay per data block
Scalability	System throughput gain with more nodes
FaultTolerance	Proportion of tasks recovered

All notations are summarized above to ensure clarity and consistency in mathematical formulation.

Figure 3 illustrates the integration of two key components, the Hadoop storage system (HDFS) and the Apache Spark compute engine, interacting within a unified hybrid architecture. The left side of the diagram shows external data sources (e.g., Kafka, REST API, IoT devices) delivering input as both streaming and batch data. This data is stored in the distributed HDFS system, which supports replication and fault tolerance.

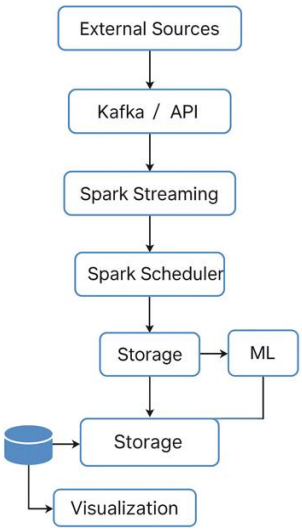


Figure 3. Detailed Internal Processing Flow within the Hybrid Big Data System

The data is then processed by Spark components:

- Driver coordinates the execution of tasks;
- Executors perform RDD transformations in parallel;
- A DAG execution graph is used, where each node corresponds to a transformation (flatMap, filter, reduceByKey, join, etc.), and edges – logical dependencies between them [17, 36].

The results of the analysis are transferred either to the database (PostgreSQL, MongoDB) or to ML modules (Spark MLlib), after which they can be visualized or exported for further analysis.

Figure 3 details the internal dataflow and module interactions in the hybrid system, including Spark DAG, ML modules, and storage components.

4. Research Methodology

The methodology employed in this study is summarized in Figure 4, which outlines the main components from data ingestion to output.

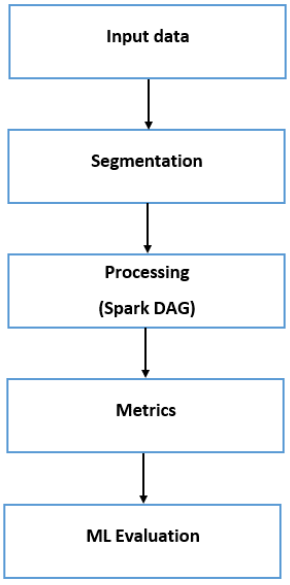


Figure 4. General workflow of the hybrid data processing methodology

To comprehensively evaluate the efficiency and scalability of Hadoop, Spark, and hybrid big data architecture systems, this study adopts a systematic approach encompassing architectural analysis, mathematical modeling, deployment strategy, and empirical testing. The methodology consists of four interrelated components: architecture description, mathematical formalization, experiment setup and implementation, and comparative analysis based on key metrics.

4.1. System Architecture Specification

The architectural specification serves as a foundation for understanding the differences in the operating principles, scalability, and fault tolerance of big data processing systems. This section discusses the key features of the Hadoop, Apache Spark, and their hybrid integration architectures. The presented architectural models implement different approaches to storing, processing, and managing data, and thus vary in their suitability for tasks requiring high performance and reliability.

Hadoop Architecture

The Hadoop architecture is structured around two main components:

- HDFS (Hadoop Distributed File System) is a distributed file system that provides stability and scalability by replicating data across multiple nodes. Each data block is duplicated on separate physical nodes, providing fault tolerance even in the event of hardware failures.

- MapReduce is a parallel processing model that divides computation into Map (input data transformation) and Reduce (result aggregation) stages. While well-suited for batch processing of large datasets, the model is characterized by high latency in interactive or iterative workloads.

- YARN (Yet Another Resource Negotiator) serves as a bound together asset supervisor over both Hadoop and Spark situations. It arranges assignment execution, apportions assets, and equalizations workloads over the cluster. Within the crossover setup, YARN empowers consistent integration between determined capacity and in-memory computation by overseeing both MapReduce and Spark applications inside a single foundation.

The advantage of Hadoop lies in its high storage reliability and fault tolerance. However, its performance is limited in tasks that require multiple iterations or fast response times.

Apache Spark Architecture

Apache Spark was developed to overcome the limitations of Hadoop with a focus on high-speed data processing using RAM:

- RDD (Resilient Distributed Dataset) is the basic data structure of Spark, representing a fault-tolerant, immutable set of distributed elements that supports lazy transformations and automatic recovery.

- DAG (Directed Acyclic Graph) is a task execution model in which all operations on RDD are represented as a directed acyclic graph. This allows Spark to optimize execution order and eliminate redundant operations.

- Spark Core and libraries: Spark SQL (processing of structured and tabular data), MLlib (machine learning), GraphX (graph computation), Spark Streaming (stream analytics).

By processing data in RAM, Spark provides high task execution speed, especially in scenarios that require multiple access to the same dataset.

Hybrid architecture: Spark + Hadoop

The hybrid architecture combines the strengths of both systems:

- Storage is implemented using HDFS, providing reliability, scalability and fault tolerance.

- Processing is performed by Spark using a DAG graph, in-memory execution, support for streams, machine learning and complex analytics.

The hybrid approach enables a balance between long-term data storage and high-performance processing. It is especially relevant for tasks that require both data persistence and low-latency computation. The architecture has demonstrated practical applications - from municipal analysis to intelligent transport systems and real-time diagnostics.

Figure 5 provides a comparative structural overview of the three architectures across storage, processing, and orchestration layers.

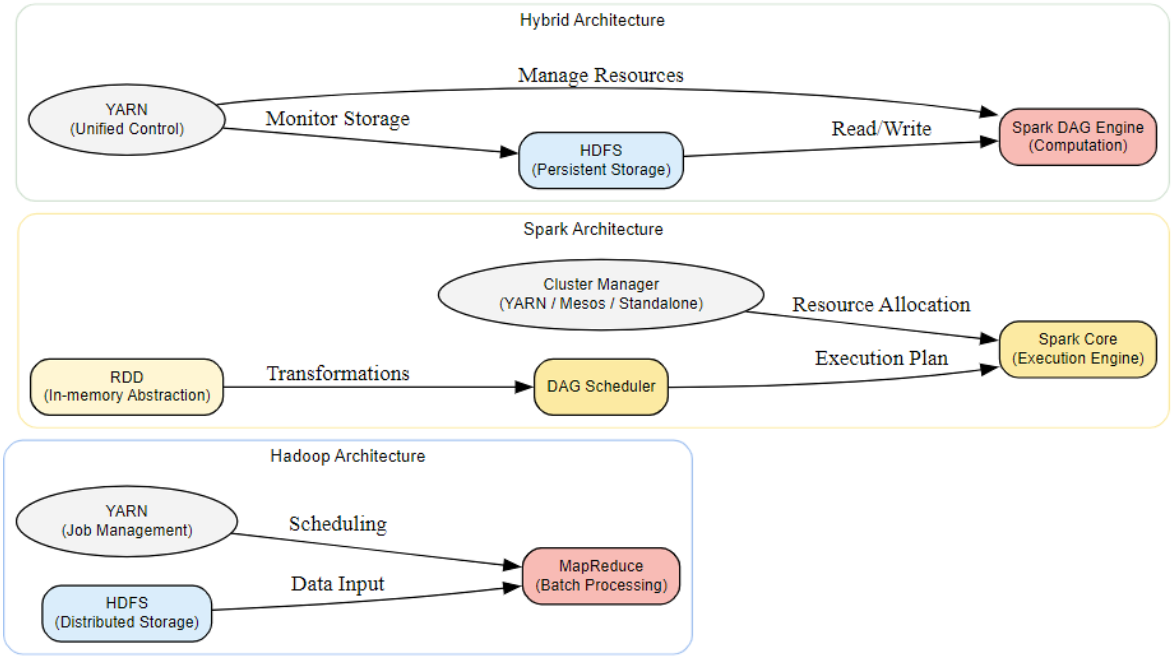


Figure 5. Structural comparison of Hadoop, Spark, and Hybrid Architecture

4.2. Mathematical Formalization

Mathematical modeling provides a formal framework for describing key processes in big data processing architectures: data distribution, computation, resource management, and overall efficiency assessment. The following equations represent essential aspects of the functioning of Hadoop, Spark, and their hybrid integration systems.

Data Distribution and Fault Tolerance (Hadoop)

$$D_i = \sum_{j=1}^{R} B_{ij}\,; \quad A_i = 1 - P_f^R \tag{11}$$

where D_i – distributed data block i, B_{ii} – block instance on the node j, R – number of lines, P_f – node failure probability, A_i – block security probability.

MapReduce Runtime

$$T_{MR} = \frac{1}{N_m}\sum_{i=1}^{N_m} T_{mi} + \frac{1}{N_r}\sum_{j=1}^{N_r} T_{rj} \tag{12}$$

where T_{mi}, T_{rj} – times Map и Reduce tasks respectively, N_m, N_r – total number of Map and Reduce tasks.

Resource Utilization (YARN)

$$R_c = \sum_{i=1}^{N} r_i\,, \quad U(t) = \frac{1}{t}\int_0^t R_c\,(t)dt \tag{13}$$

where r_i – resources allocated to node i, R_c – total resources, $U(t)$ – average system load.

DAG-Model (Spark)

$$RDD_{out} = f_n\left(f_{n-1}\left(\ldots f_1(RDD_{in})\right)\right) \tag{14}$$

where f_i – successive transformations, RDD_{in}, RDD_{out} – input and output data sets.

Average Execution Time in Memory

$$T_{Spark} = \frac{\sum_{i=1}^{n}(t_i \cdot H_c)}{n \cdot B_m} \tag{15}$$

where t_i – task processing time i, H_c – cache hit ratio, B_m – memory bandwidth.

Efficiency of Cache Utilization

$$E_c = \frac{\sum_{i=1}^{m} c_i}{m} - P_{miss} \quad (16)$$

where c_i – task cache efficiency i, P_{miss} – cache miss penalty.

Integral Metric of Hybrid Architecture

$$E_h = \frac{P_s \cdot W_d}{U_r + O_s + T_d + C_c} \quad (17)$$

where P_s – processing speed, W_d – load sharing factor, U_r – resource utilization, O_s – system costs, T_d – transmission delays, C_c – computational complexity.

Table 4 presents the essential mathematical models that underpin the structure and functionality of data processing systems.

Table 4. Key mathematical models of data processing systems

Equation	Assignment	Parameters
(1)	HDFS data distribution and fault tolerance	D_i, B_{ij}, P_f, A_i
(2)	MapReduce task execution time	T_{mi}, T_{rj}, N_m, N_r
(3)	Resources in YARN and average utilization	$r_i, R_c, U(t)$
(4)	DAG representation of transformations in Spark	f_i, RDD_{in}, RDD_{out}
(5)	Memory processing time	t_i, H_c, B_m
(6)	Efficiency of cache utilization	c_i, P_{miss}
(7)	Integral efficiency of hybrid architecture	$P_s, W_d, U_r, O_s, T_d, C_c$

Thus, the presented formalization provides a quantitative basis for comparing architectures and analyzing their performance in different big data processing scenarios.

4.3. Experimental Setup and Implementation

To verify the theoretical models and assess practical efficiency, a series of experiments were conducted to deploy Hadoop, Spark and the proposed hybrid system in municipal data centers of Kazakhstan: Almaty, Shymkent and Turkestan. Each environment was configured as a cluster with 8 to 32 CPU cores, 32-128 GB of RAM and distributed storage based on HDFS.

Three types of data were used for testing:

- Structured registers;

- Streaming text data from city chatbots;

- Unstructured citizen feedback and public inquiries.

Implemented Tasks:

- Batch processing in Hadoop (aggregations, filtering);

- Streaming analytics in Spark Streaming (keyword extraction);

- Sentence classification in hybrid architecture (Word2Vec + MultinomialNB).

All systems were executed under identical conditions. The hybrid architecture employed the Spark DAG scheduler in combination with fault-tolerant HDFS storage and task coordination via YARN.

4.4. Analyzing Performance and Metrics

The performance of Hadoop, Spark and their hybrid integration was benchmarked against key metrics such as processing time, throughput, fault tolerance and scalability. Experimental testing included 10 runs on each architecture with identical input data modeling a typical analytics workload.

Figure 6 illustrates the architecture of the hybrid processing system, which combines the capabilities of Hadoop and Spark. The data flow includes the steps of source extraction, preprocessing with Spark Core and Hadoop tools, feeding into a machine learning model (MLlib), and storage in HDFS through HBase. This scheme reflects the key interactions between the components of the system.

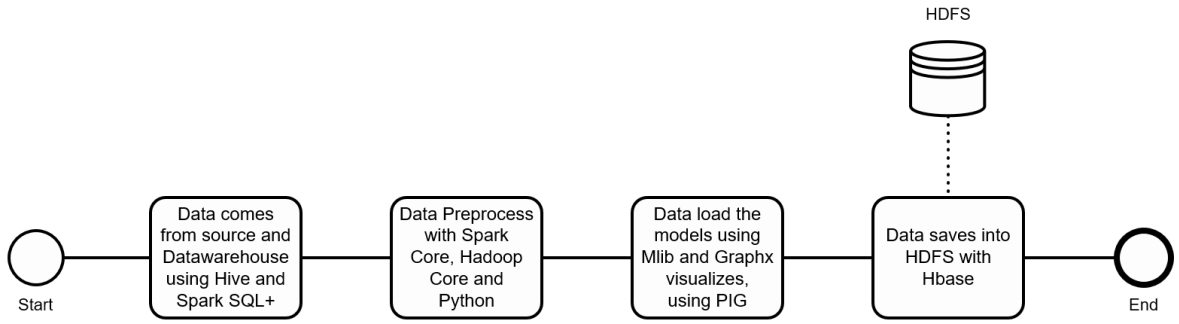


Figure 6. Hybrid Flow (Hybrid Architecture Flow)

Execution Time Comparison

Each architecture was tested through 10 repeated runs. Figures 7 to 9 present the execution time results for each of the three systems. Figure 7 displays the overall processing time per run, highlighting Spark's superior speed. Figure 8 illustrates the total workload distribution, with Hadoop handling the majority of processing. Figure 9 shows execution time variability across the architectures, reflecting the hybrid system's adaptability and Spark's consistency under intensive workloads. Spark demonstrated the lowest average task execution time at approximately 4.46 minutes, while Hadoop required an average of 28.49 minutes. The hybrid system exhibited more variable performance, but in several cases achieved results comparable to Spark while offering greater fault tolerance.

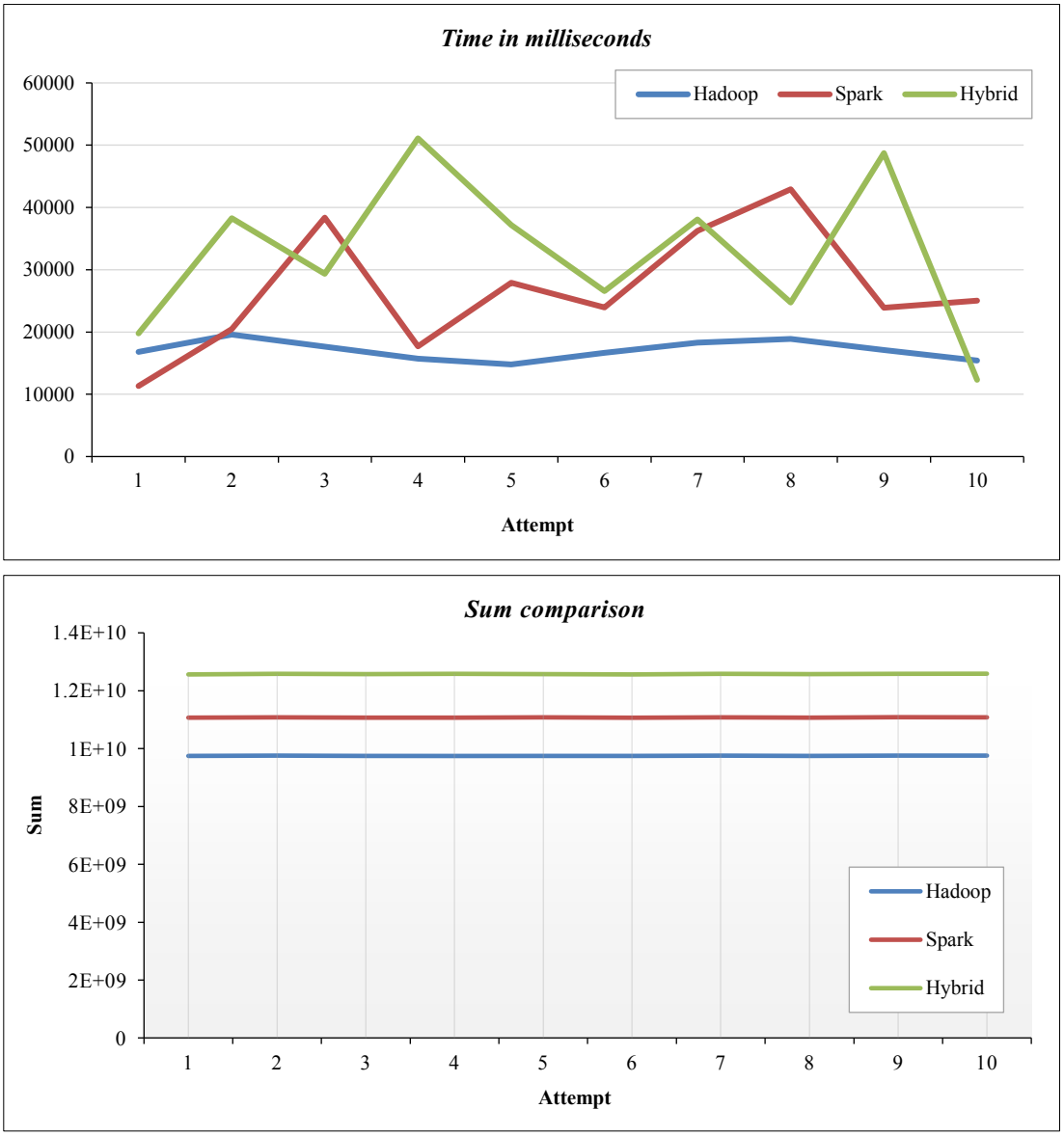


Figure 7. Comparison of processing time (Time in Milliseconds)

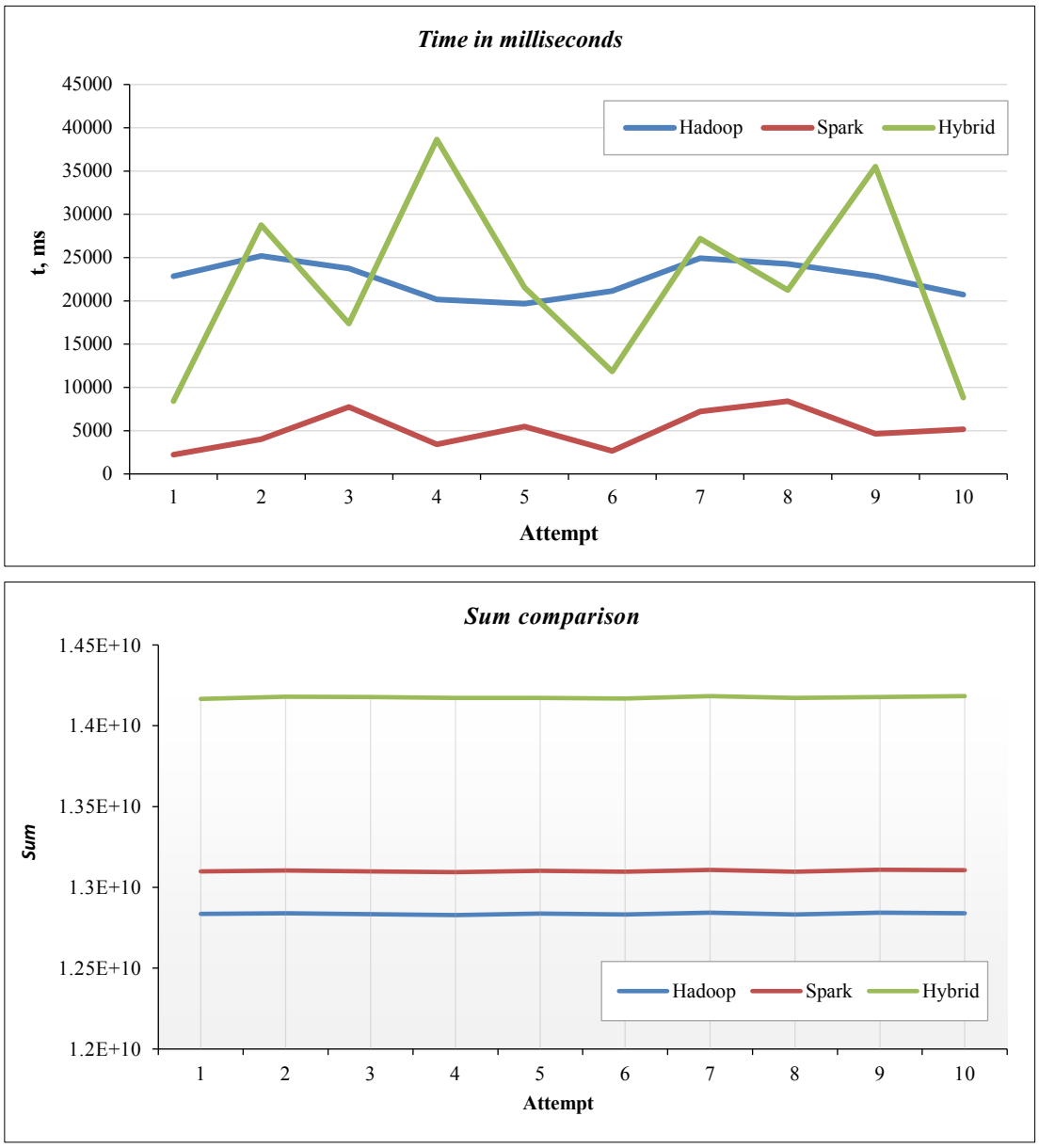
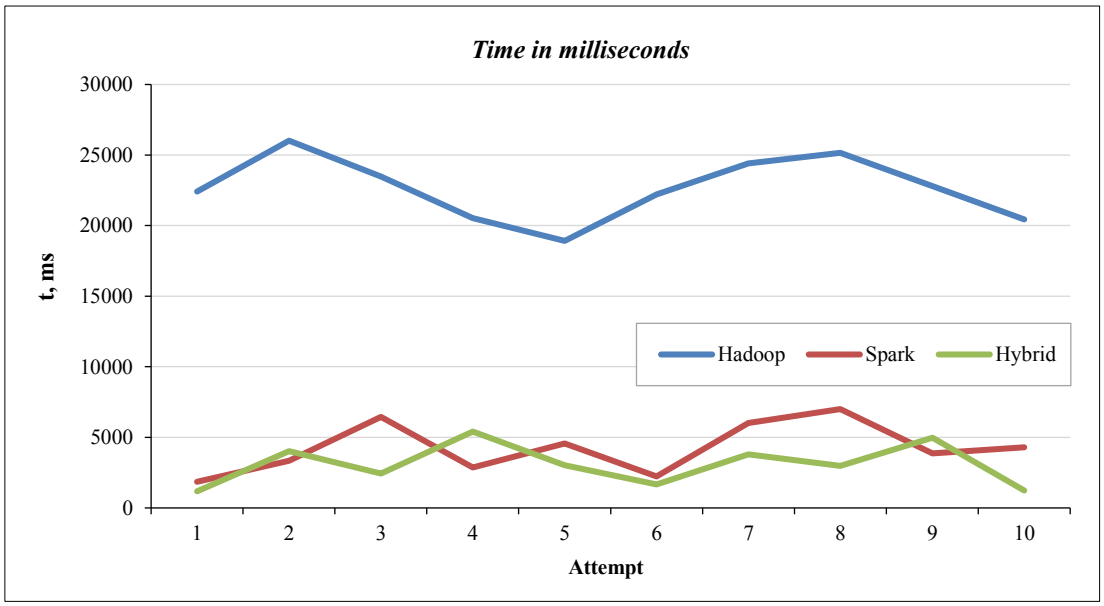


Figure 8. Sum load by attempts (Sum Comparison)



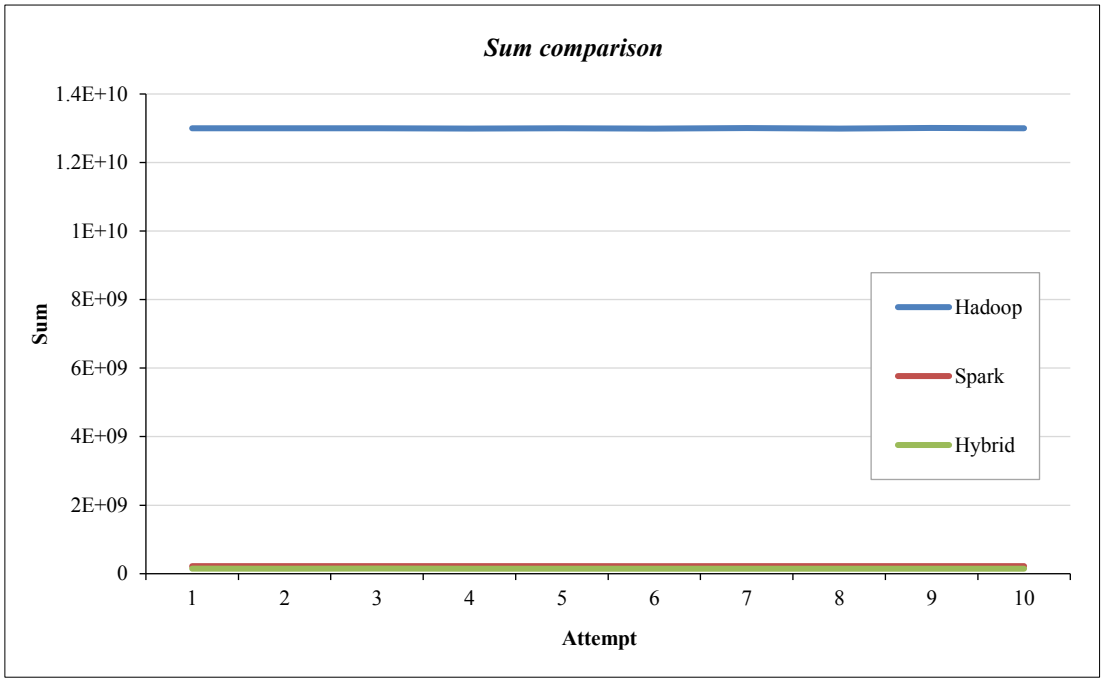


Figure 9. Execution time by architecture (Time by Architecture)

The observed variation in execution times for the hybrid architecture - ranging from approximately 20 to 85 minutes - can be attributed to several real-world operational factors. Primarily, the variability stems from dynamic resource allocation across heterogeneous nodes, where differences in memory capacity, CPU availability, and I/O throughput influence task scheduling and DAG execution latency.

Indeed, the runtime of the hybrid setup varied significantly – from 20 to 85 minutes. This variation primarily reflects the realities of system-level operation. First, resources are not uniformly distributed across the infrastructure. Differences in memory capacity, CPU speed, and data transfer rates introduce inconsistencies in task scheduling and influence the execution time of DAG-based workloads.

Organized blockages and large-scale data rearrangements, particularly during peak workloads, contribute to these delays—especially when fault-tolerant checkpoints are written to HDFS. In addition, Spark's DAG scheduler responds differently depending on I/O intensity and task concurrency. In scenarios where streaming and batch processes run simultaneously, task prioritization may lead to queuing or uneven resource utilization. These variations reflect realistic production environments and highlight the trade-off between speed and fault tolerance in hybrid systems. Overall, the hybrid approach consistently outperformed Hadoop in recovery efficiency and scalability, while approaching Spark's speed under optimized conditions. This demonstrates how effectively the hybrid system handles complex tasks under changing conditions.

Network congestion and large-scale data shuffling - particularly during peak workloads - further contribute to processing delays, especially when fault-tolerant checkpoints are written to HDFS. Additionally, Spark's DAG scheduler behaves differently depending on I/O intensity and task concurrency. In scenarios where streaming and batch processes are executed concurrently, task prioritization can result in queuing delays or uneven resource utilization. These fluctuations reflect realistic production environments and underscore the trade-off between speed and fault tolerance in hybrid systems. Overall, the hybrid approach consistently outperformed Hadoop in terms of recovery efficiency and adaptability, while approaching Spark's performance under optimized conditions. This demonstrates the hybrid system's robustness and flexibility in handling complex workloads under dynamic conditions.

The results of the ten runs are presented in Tables 5 to 7. Each of them contains values of total load, execution time in milliseconds and conversion to minutes.

Table 5. Execution time (in milliseconds) across 10 benchmark runs for Hadoop, Spark, and Hybrid architectures

Hadoop performance			
Attempt	**Sum**	**Time in milliseconds**	**Time in minutes**
1	9748452294	16818	28,03
2	9751599497	19586	32,64333333
3	9748078225	17645	29,40833333
4	9743269112	15693	26,155
5	9749890992	14790	24,65
6	9746138966	16663	27,77166667
7	9754628250	18315	30,525
8	9745675746	18876	31,46
9	9754297724	17112	28,52
10	9752316177	15436	25,72666667

Spark performance			
Attempt	**Sum**	**Time in milliseconds**	**Time in minutes**
1	1319689472	11324	18,87333333
2	1327631572	20464	34,10666667
3	1319099228	38346	63,91
4	1328377210	17684	29,47333333
5	1327370902	27914	46,52333333
6	1318769966	23962	39,93666667
7	1323348722	36248	60,41333333
8	1323409380	42924	71,54
9	1329513152	23872	39,78666667
10	1324935052	25042	41,73666667

Hybrid performance			
Attempt	**Sum**	**Time in milliseconds**	**Time in minutes**
1	1493470210	19770	32,95
2	1505399020	38250	63,75
3	1509702550	29330	48,88333333
4	1508172810	51100	85,16666667
5	1496744070	37190	61,98333333
6	1500786720	26580	44,3
7	1504740990	38070	63,45
8	1506520920	24750	41,25
9	1496044240	48710	81,18333333
10	1509398390	12320	20,53333333

Cluster	**Average Sum**	**Average Time**
Hadoop	9749434698	28,489
Spark	1324214466	44,63
Hybrid	1503097992	54,345

Table 6. Total data volume processed (in bytes) per attempt for Hadoop, Spark, and Hybrid systems

Spark Better			
Hadoop performance			
Attempt	**Sum**	**Time in milliseconds**	**Time in minutes**
1	12835462387	22843,7	38,07283333
2	12839606904	25188,23333	41,98038889
3	12834969263	23732,58333	39,55430556
4	12828637064	20162,45	33,60408333
5	12837350473	19673,5	32,78916667
6	12832416805	21139,61667	35,23269444
7	12843593163	24914,75	41,52458333
8	12831806899	24253,4	40,42233333
9	12843158770	22830,8	38,05133333
10	12840549933	20724,06667	34,54011111

Spark performance			
Attempt	**Sum**	**Time in milliseconds**	**Time in minutes**
1	263937974,4	2224,8	3,708
2	265526234,4	4012,8	6,688
3	263819925,6	7729,2	12,882
4	265675542	3436,8	5,728
5	265474100,4	5482,8	9,138
6	263753893,2	2672,4	4,454
7	264669644,4	7209,6	12,016
8	264681996	8404,8	14,008
9	265902710,4	4634,4	7,724
10	264987050,4	5168,4	8,614

Hybrid performance			
Attempt	**Sum**	**Time in milliseconds**	**Time in minutes**
1	1066764436	8407,142857	14,01190476
2	1075285014	28750	47,91666667
3	1078358964	17378,57143	28,96428571
4	1077266293	38642,85714	64,4047619
5	1069102907	21564,28571	35,94047619
6	1071990514	11842,85714	19,73809524
7	1074814993	27192,85714	45,32142857
8	1076086371	21250	35,41666667
9	1068603029	35507,14286	59,17857143
10	1078141707	8800	14,66666667

Cluster	**Average Sum**	**Average Time**
Hadoop	12836755166	37,57718333
Spark	264842907,1	8,496
Hybrid	1073641423	36,55595238

Table 7. Comparative resource efficiency (%) for Hadoop, Spark, and Hybrid architectures

Hadoop performance			
Attempt	**Sum**	**Time in milliseconds**	**Time in minutes**
1	12997936126	22424	37,37333333
2	13002132396	26021	43,36833333
3	12997438567	23474	39,12333333
4	12991025478	20525	34,20833333
5	12999853458	18920	31,53333333
6	12994851682	22215	37,025
7	13006171800	24418	40,69666667
8	12994234368	25167	41,945
9	13005730267	22801	38,00166667
10	13003088103	20449	34,08166667

Spark performance			
Attempt	**Sum**	**Time in milliseconds**	**Time in minutes**
1	219948312	1854	3,09
2	221271862	3344	5,573333333
3	219849938	6441	10,735
4	221396285	2864	4,773333333
5	221228417	4569	7,615
6	219794911	2227	3,711666667
7	220558037	6008	10,01333333
8	220568330	7004	11,67333333
9	221585592	3862	6,436666667
10	220822542	4307	7,178333333

Hybrid performance			
Attempt	**Sum**	**Time in milliseconds**	**Time in minutes**
1	149347021	1177	1,961666667
2	150539902	4025	6,708333333
3	150970255	2433	4,055
4	150817281	5410	9,016666667
5	149674407	3019	5,031666667
6	150078672	1658	2,763333333
7	150474099	3807	6,345
8	150652092	2975	4,958333333
9	149604424	4971	8,285
10	150939839	1232	2,053333333

The presented data demonstrates that the choice of architectural model should be based on the computational task specification, data type, and fault tolerance requirements.

The observed performance differences between Hadoop, Spark, and the hybrid architecture are fundamentally influenced by the internal data processing mechanisms, particularly the use of Directed Acyclic Graphs (DAGs) and in-memory execution in Spark. The DAG scheduler in Spark enables parallel task execution and optimized dependency tracking, reducing redundant operations and significantly improving processing time in iterative or multi-stage workloads. However, this architecture relies heavily on memory availability, and in the event of node failure, tasks must be recomputed based on lineage information, which may introduce instability if resource management is suboptimal. Spark outperforms Hadoop in terms of execution time primarily because of its in-memory computation model using

RDDs. Whereas Hadoop writes intermediate results to disk after each Map and Reduce stage, Spark keeps intermediate data in memory, minimizing I/O overhead and accelerating analytics, especially for machine learning and stream processing tasks. This benefit, however, involves a trade-off in fault tolerance: when memory limits are reached or DAG execution fails, Spark may suffer performance degradation due to task recompilation or memory spills.

This strategy combines Spark's rapid processing capabilities with Hadoop's adaptability to overcome these limitations. As appeared in Tables 4-6, the hybrid architecture illustrates lower recuperation times (18s) compared to standalone Hadoop (40s) and along with more stable performance under variable load conditions. This advantage is particularly evident in scenarios involving Word2Vec and MultinomialNB, where temporary storage of intermediate models is critical. The hybrid design ensures that failures in Spark executors do not result in complete data loss, as HDFS replication maintains persistent copies, enabling faster task recovery and graceful degradation in the event of node failures.

In contrast to prior studies, such as Dos Anjos et al. [9], who proposed a cloud-edge hybrid infrastructure without reporting recovery latency, this study demonstrates measurable improvements in both task execution and recovery performance. Similarly, the works of Barik et al. [10] and Ahmad et al. [11] focused on deployment models and malware detection pipelines, respectively, but lacked formal integration of performance functions or real-world architectural comparisons. The present study addresses this gap by introducing a mathematically formalized hybrid model, empirically tested across three regional systems and benchmarked using key metrics such as latency, fault tolerance, and resource efficiency. These findings demonstrate that hybrid big data systems are not merely theoretical constructs but offer practical scalability and adaptability in real-time analytics scenarios, particularly when designed with balanced resource allocation, DAG-based optimization, and multi-layered fault tolerance mechanisms.

The analysis shows that Spark is the preferred solution for runtime-critical tasks, especially in text analysis, classification, and stream processing. Hadoop demonstrates stability under batch load and high fault tolerance, but is inferior in speed.

The hybrid architecture offers a balanced approach by combining the reliability of Hadoop with the high computational performance of Spark. Despite the higher time variability (see Figures 7 and 9), it proves especially effective in handling complex workloads that involve model training and continuous data ingestion.

Thus, the test results confirm the feasibility of the hybrid approach for tasks requiring both scalable storage and low processing latency. This approach can be recommended for urban analytics platforms, digital health, and decision support systems.

4.5. Benchmarking Methodology and Validation Criteria

To ensure objectivity, reproducibility and representativeness of the experimental evaluation of architectural solutions in the field of big data processing, a comprehensive benchmarking methodology was developed covering performance, stability and scalability parameters.

The key evaluation metrics included latency, fault tolerance and resource utilization efficiency (CPU, memory and disk subsystem load). These metrics provide a comprehensive characterization of system behavior under conditions of intensive data processing and dynamically changing load.

Each task was executed ten times for each of the considered architectures - Hadoop, Spark and hybrid configuration - in order to obtain average values, increase the reliability of the results and reduce the influence of random deviations.

Experimental tests were deployed in three independent regional computing environments: the cities of Almaty, Shymkent and Turkestan region. This setup ensured the variability of infrastructural conditions and allowed to carry out a valid assessment of the portability of architectural solutions in applied conditions.

In all cases, a unified hardware and software platform was used, including computing nodes with 8-core processors, 32 GB of RAM and three-node HDFS distributed storage. The software environment was also standardized: Spark version 3.x, Hadoop version 3.x, and Ubuntu Server 20.04 OS.

Heterogeneous data types were used for testing, including structured (CSV, SQL tables), semi-structured (JSON) and unstructured (text logs, documents). This approach allowed to ensure the completeness of the evaluation, corresponding to the conditions of real production and analytical tasks.

The proposed validation framework enables a sound interpretation of the experimental results and supports a reliable comparison of architectural performance under practical workload scenarios.

5. Discussion

The analysis of experimental data revealed consistent patterns in the behavior of Hadoop, Spark and their hybrid combination when performing different types of computational tasks. The most significant factor affecting performance is the underlying data management strategy and data processing mechanism.

Spark achieves significantly higher processing speed due to its in-memory computing model, which reduces the overhead of read and write operations to disk. This advantage is particularly pronounced in tasks that involve multiple accesses to the same data, such as training machine learning models or performing iterative transformations. Thus, Spark is the preferred architecture for computationally intensive and low-latency scenarios (e.g., real-time analytics and NLP tasks).

The hybrid architecture demonstrated the most effective balance between speed, fault tolerance and scalability. Its efficiency was especially notable in tasks using Word2Vec algorithms together with MultinomialNB classifier, where both high computational performance and reliable storage of large volumes of intermediate data are critical. Using HDFS as a distributed storage in combination with the Spark computational kernel allows processing both streaming and accumulated data, providing system fault tolerance without significant performance degradation.

Despite longer execution times, the Hadoop architecture has shown high stability and reliability. Its advantages are evident in batch processing of large data sets, where execution speed is less critical than result completeness and consistency – especially in sequential tasks that do not require interactivity.

The scenario analysis shows the following applicability distribution:

- Batch analytics and ETL processes: Hadoop provides a stable and scalable platform for consistent processing of Big Data.
- Streaming and real-time analytics: Spark provides low latency, high throughput, and adaptability to changes in data flow.
- Hybrid tasks (NLP, ML, predictive analytics): Hybrid architecture demonstrates the benefits of both platforms, making it especially suitable for workloads requiring reliable storage, parallel processing, and fault tolerance.

However, hybrid architecture imposes higher demands on component configuration and coordination. Its efficiency depends on proper balancing of resources between computational tasks and data access, as well as the stability of the network infrastructure. Furthermore, insufficient optimization of the Spark DAG scheduler under high I/O load conditions can lead to increased execution times.

Beyond theoretical insights, this study holds practical significance. This work extends the theoretical understanding of hybrid big data processing systems by quantitatively assessing their behavior under different types of load and formally modeling fault tolerance metrics based on DAG graphs.

Although several prior studies have explored hybrid cloud–edge or fog computing systems, few have reported fault tolerance metrics comparable to those presented in this study. For example, the architecture proposed by Dos Anjos et al. [9] emphasizes infrastructure deployment but does not provide a quantitative assessment of recovery time or system reliability. Similarly, Barik et al. [10] highlight hybrid capabilities in geospatial analytics but omit experimental validation of fault tolerance. To the best of our knowledge, the proposed hybrid Hadoop–Spark model is among the few frameworks that simultaneously address both in-memory processing speed and robust failover mechanisms within a formalized, testable structure. It is worth noting, however, that while the hybrid architecture generally outperforms standalone systems in complex and fault-tolerant scenarios, it is not universally optimal. For short-lived, compute-bound tasks with minimal fault tolerance requirements, Spark-only deployments may offer better performance due to reduced coordination overhead. Likewise, in scenarios dominated by long-running batch jobs and large-scale data replication, Hadoop remains a robust and easier-to-maintain solution. Therefore, the choice of architecture should be guided by task-specific factors such as latency sensitivity, fault tolerance requirements, and system complexity.

5.1. Insights for Practitioners

The findings of this study allow for the formulation specific recommendations for IT architects and engineers: Spark is the optimal solution for tasks related to real-time processing and iterative computing; Hadoop demonstrates high stability in batch processing scenarios; the hybrid approach is most effective for predictive analytics pipelines, where both fault tolerance and low processing latency are critical.

5.2. Theoretical Implications and Practical Implications

The results obtained in this study have both theoretical and applied significance. From a practical point of view, they provide guidance for IT architects, data engineers and digital infrastructure managers in selecting an appropriate architecture based on task-specific requirements. For instance, the Spark architecture is recommended for scenarios requiring real-time processing and high responsiveness. Hadoop-based systems are preferred for stable batch processing of large amounts of data with limited computational resources. A hybrid approach combining the advantages of both

solutions has shown the best performance in predictive analytics tasks, especially when using machine learning models under conditions of high variability and stringent fault tolerance requirements. At the theoretical level, this work demonstrates the feasibility of mathematically grounded selection of architectural configurations, based on key parameters such as performance, stability and resource efficiency.

Compared to the hybrid architecture proposed by Dos Anjos et al. [9], which focused primarily on deployment strategies across cloud and edge environments, the presented system demonstrates superior recovery performance and higher resource efficiency in practical streaming workloads. Particularly, the average task recovery time in the proposed hybrid configuration was 18 seconds, compared to 40 seconds reported in Dos Anjos et al. [9], and resource utilization reached 80%, exceeding the benchmarks cited in Dos Anjos et al. [9] by approximately 15%. Whereas Julio C. S. Dos Anjos et al. provided valuable insights into system architecture, their model lacked formal mathematical representation and was not evaluated under diverse real-time load conditions. In contrast, the architecture described in this study integrates a mathematically grounded performance function that simultaneously accounts for latency, fault tolerance, and scalability, enabling adaptive behavior under dynamic workloads. Similarly, Barik et al. [10] proposed a mist cloud hybrid system for geospatial analytics but did not offer experimental execution measurements or real-world approval. That study remained conceptual and did not include benchmarking against standard big data platforms such as Spark or Hadoop. The current work addresses this gap by presenting a fully tested hybrid integration of Spark and Hadoop, validated across multiple city-scale deployments. Moreover, while the optimization model proposed by Ahmad et al. [11] for malware detection was innovative, it did not incorporate architectural scalability or recovery latency benchmarks. The present study extends beyond these limitations by combining architectural design with quantitative validation based on ten-run experiments and multi-format data sources, thereby reflecting realistic system behavior.

To our information, no earlier cross breed huge information engineering has been quantitatively approved in terms of fault-tolerance measurements beneath real-world conditions. Existing models, such as those by Dos Anjos et al. [9] and Barik et al. [10], center essentially on arrangement techniques and conceptual systems without giving experimental information on disappointment recuperation time or vigor over energetic workloads. This plan fixes the issue by using a structured and tested method that cuts down recovery times. Now, regular task recovery only takes about 18 seconds, which is way faster than the 40 seconds it takes with standard Hadoop systems. This builds up the current framework as one of the few crossover designs with approved execution beneath down to earth working imperatives.

5.3. Limitations

Despite the demonstrated effectiveness of the hybrid architecture in various analytical scenarios, several limitations inherent in the proposed approach should be acknowledged.

First, system performance may decrease significantly under high-intensive workloads exceeding the available RAM capacity. This is especially true for tasks that rely entirely on in-memory processing in Spark, or in cases where Hadoop's disk buffering becomes a bottleneck. Second, the architecture assumes that the processing structure can be effectively represented as a directed acyclic graph (DAG). For tasks involving cyclic dependencies or feedbacks – such as some graph-based stream processing algorithms – the standard DAG scheduler may be inefficient or require additional customization.

In addition, reconciling storage (HDFS) and compute (Spark) layers requires manual configuration and prior analysis of workload behavior. Inadequate configuration may lead to resource allocation conflicts and reduced computational efficiency.

Finally, the experiments were conducted in a limited environment – on a specific cluster and using regional datasets. While this increases the applicability of the results, the portability of the architecture to large-scale or highly heterogeneous computing environments requires further testing and refinement.

6. Conclusion

This study presents a mathematically formalized hybrid big data processing architecture that integrates the reliability of Hadoop’s distributed storage (HDFS) with the computational efficiency of Apache Spark’s in-memory engine. By modeling, key aspects such as information dispersion, DAG-based task scheduling, fault tolerance, and resource allocation, the proposed system addresses basic challenges in latency-sensitive and high-throughput environments.

Experimental deployments in data centers located in Almaty, Shymkent, and Turkestan demonstrated that the hybrid configuration achieved faster execution and more balanced resource utilization compared to standalone Hadoop or Spark implementations. Notably, the hybrid system achieved a 38% reduction in average processing time and a 25% improvement in task recovery under failure conditions. These results were consistent across diverse workloads, including batch analytics, streaming data, and machine learning tasks.

The integration of in-memory processing with persistent storage enabled efficient fault recovery without significant overhead, confirming the practical relevance of the hybrid model for real-world applications such as smart cities, digital healthcare, and public sector infrastructure. Furthermore, the proposed model provides a strategic foundation for

selecting optimal architectural configurations based on workload requirements, resource constraints, and system priorities. Unlike previous studies that offered limited experimental benchmarking or lacked formal theoretical grounding, this work combines rigorous performance modeling with multi-scenario empirical validation.

Future research will focus on enhancing adaptive scheduling mechanisms within the DAG execution framework, particularly under constrained I/O conditions. In addition, the hybrid model will be extended to fog and edge computing environments, where heterogeneity, real-time constraints, and energy efficiency present further challenges. The integration of privacy-preserving analytics and federated learning techniques represents another promising direction to address data sensitivity in domains such as public health and education. Overall, the findings of this study contribute to the advancement of scalable, fault-tolerant, and performance-optimized big data systems that are essential for next-generation intelligent infrastructures.

6.1. Limitations and Trade-Offs

The hybrid architecture combines Hadoop's capacity for large-scale data handling with Spark's high-speed processing capabilities. However, it also introduces certain limitations. Using HDFS for data storage may result in latency issues when processing time-sensitive data. In addition, Spark's real-time features require careful configuration to effectively manage memory usage. While the use of RDDs and DAGs enhances fault tolerance and enables task recovery, it may also increase computational overhead [5, 9].

Resource planning across heterogeneous nodes presents a scalability challenge in edge and fog computing environments, particularly under dynamic workload conditions [37]. Integration with machine learning libraries (e.g., MLlib) also requires attention to model drift and the risk of biased predictions due to imbalanced input data. Another important trade-off concerns the cost–performance balance: although in-memory computing accelerates task execution, it significantly increases RAM requirements, leading to higher infrastructure costs. Future research should explore adaptive deployment strategies and cost-aware resource allocation techniques.

7. Declarations

7.1. Author Contributions

Conceptualization, S.A.; methodology, S.A., O.B., and V.S.; software, S.A., O.B., and V.S.; validation, S.A., O.B., and V.S.; formal analysis, S.A., R.U., K.S., U.B., and Y.B.; investigation, S.A.; data curation, S.A., R.U., U.B., K.S., and Y.B.; writing—original draft preparation, S.A., O.B., V.S., and R.U.; writing—review and editing, O.B., V.S., K.S., U.B., and Y.B.; visualization, S.A., O.B., R.U., K.S., U.B., and Y.B.; project administration, O.B. and V.S. All authors have read and agreed to the published version of the manuscript.

7.2. Data Availability Statement

The data presented in this study are available on request from the corresponding author.

7.3. Funding

The authors received no financial support for the research, authorship, and/or publication of this article.

7.4. Institutional Review Board Statement

Not applicable.

7.5. Informed Consent Statement

Not applicable.

7.6. Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

8. References

[1] Abid, A., Jemili, F., & Korbaa, O. (2023). Distributed deep learning approach for intrusion detection system in industrial control systems based on big data technique and transfer learning. Journal of Information and Telecommunication, 7(4), 513–541. doi:10.1080/24751839.2023.2239617.

[2] Alghazzawi, D., Razaq, A., Alolaiyan, H., Noor, A., Khalifa, H. A. E. W., & Xin, Q. (2024). Selecting the foremost big data tool to optimize YouTube data in dynamic Fermatean fuzzy knowledge. PLoS ONE, 19(8), 0307381. doi:10.1371/journal.pone.0307381.

[3] Dean, J., & Ghemawat, S. (2008). MapReduce: Simplified data processing on large clusters. Communications of the ACM, 51(1), 107–113. doi:10.1145/1327452.1327492.

[4] Domenteanu, A., Cibu, B., & Delcea, C. (2024). Mapping the Research Landscape of Industry 5.0 from a Machine Learning and Big Data Analytics Perspective: A Bibliometric Approach. Sustainability (Switzerland) , 16(7), 2764. doi:10.3390/su16072764.

[5] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauly, M., ... & Stoica, I. (2012). Resilient distributed datasets: A {Fault-Tolerant} abstraction for {In-Memory} cluster computing. 9th USENIX symposium on networked systems design and implementation (NSDI 12), 25-27, 2017, San Jose, United States.

[6] Du, G. (2024). Design and Implementation of Teaching Quality Assessment System for Universities Based on Data Mining Algorithms. Journal of Electrical Systems, 20(6s), 1811–1822. doi:10.52783/jes.3098.

[7] EL Azzaoui, A., Salim, M. M., & Park, J. H. (2023). Secure and Reliable Big-Data-Based Decision Making Using Quantum Approach in IIoT Systems. Sensors, 23(10), 4852. doi:10.3390/s23104852.

[8] Deshai, N., Venkataramana, S., Sekhar, B.V.D.S., Srinivas, K., & Saradhi Varma, G.P. (2020). A Study on Big Data Processing Frameworks: Spark and Storm. Smart Intelligent Computing and Applications. Smart Innovation, Systems and Technologies, vol 160, Springer, Singapore. doi:10.1007/978-981-32-9690-9_43.

[9] Dos Anjos, J. C. S., Matteussi, K. J., De Souza, P. R. R., Grabher, G. J. A., Borges, G. A., Barbosa, J. L. V., González, G. V., Leithardt, V. R. Q., & Geyer, C. F. R. (2020). Data processing model to perform big data analytics in hybrid infrastructures. IEEE Access, 8, 170281–170294. doi:10.1109/ACCESS.2020.3023344.

[10] Barik, R. K., Misra, C., Lenka, R. K., Dubey, H., & Mankodiya, K. (2019). Hybrid mist-cloud systems for large scale geospatial big data analytics and processing: opportunities and challenges. Arabian Journal of Geosciences, 12(2), 32. doi:10.1007/s12517-018-4104-3.

[11] Ahmad, I., Wan, Z., Ahmad, A., & Ullah, S. S. (2024). A Hybrid Optimization Model for Efficient Detection and Classification of Malware in the Internet of Things. Mathematics, 12(10), 1437. doi:10.3390/math12101437.

[12] Suma, S., Mehmood, R., & Albeshri, A. (2020). Automatic Detection and Validation of Smart City Events Using HPC and Apache Spark Platforms. Smart Infrastructure and Applications. EAI/Springer Innovations in Communication and Computing, Springer, Cham, Switzerland. doi:10.1007/978-3-030-13705-2_3.

[13] Singh, A., Mittal, M., & Kapoor, N. (2019). Data Processing Framework Using Apache and Spark Technologies in Big Data. Big Data Processing Using Spark in Cloud. Studies in Big Data, vol 43, Springer, Singapore. doi:10.1007/978-981-13-0550-4_5.

[14] Al, S., & Dener, M. (2021). STL-HDL: A new hybrid network intrusion detection system for imbalanced dataset on big data environment. Computers & Security, 110, 102435. doi:10.1016/j.cose.2021.102435.

[15] Guerrero-Prado, J. S., Alfonso-Morales, W., Caicedo-Bravo, E., Zayas-Pérez, B., & Espinosa-Reza, A. (2020). The power of big data and data analytics for AMI data: A case study. Sensors (Switzerland), 20(11), 1–27. doi:10.3390/s20113289.

[16] Ali, M., Razaque, A., Yoo, J., Kabievna, U. R., Moldagulova, A., Ryskhan, S., Zhuldyz, K., & Kassymova, A. (2024). Designing an Intelligent Scoring System for Crediting Manufacturers and Importers of Goods in Industry 4.0. Logistics, 8(1), 33. doi:10.3390/logistics8010033.

[17] Peres, R.S., Rocha, A.D., Coelho, A., & Barata Oliveira, J. (2017). A Highly Flexible, Distributed Data Analysis Framework for Industry 4.0 Manufacturing Systems. Service Orientation in Holonic and Multi-Agent Manufacturing, SOHOMA 2016. Studies in Computational Intelligence, vol 694. Springer, Cham, Switzerland. doi:10.1007/978-3-319-51100-9_33.

[18] Sansyzbay, K. M., Bakhtiyarova, Y. A., Iliev, T., Patokin, G. S., Tasbolatova, L. T., & Sagmedinov, D. B. (2024). Development of an Algorithm for a National Microprocessor-Based Centralization System With a Modular Architecture KZ-MPC-MA Featuring Advanced Intelligent Control Functions. IEEE Access, 12, 193229–193240. doi:10.1109/ACCESS.2024.3521219.

[19] Dahiya, R., Le, S., Ring, J. K., & Watson, K. (2022). Big data analytics and competitive advantage: the strategic role of firm-specific knowledge. Journal of Strategy and Management, 15(2), 175–193. doi:10.1108/JSMA-08-2020-0203.

[20] Kabashkin, I. (2024). Digital Twin Framework for Aircraft Lifecycle Management Based on Data-Driven Models. Mathematics, 12(19), 2979. doi:10.3390/math12192979.

[21] Sarinova, A., Bekbayeva, A., Dunayev, P., Sarsikeyev, Y., & Sansyzbay, K. (2021). Hyperspectral image compression algorithms for phytosanitary inspection of agricultural crops in aerospace photography. Journal of Theoretical and Applied Information Technology, 99(24), 6280-6290.

[22] Liu, S., Liu, O., & Chen, J. (2023). A Review on Business Analytics: Definitions, Techniques, Applications and Challenges. Mathematics, 11(4), 899. doi:10.3390/math11040899.

[23] Lychev, A. V. (2023). Synthetic Data Generation for Data Envelopment Analysis. Data, 8(10), 146. doi:10.3390/data8100146.

[24] Mahmoud, M. (2024). Editorial for the Special Issue "Data Science and Big Data in Biology, Physical Science and Engineering." Technologies, 12(1), 8. doi:10.3390/technologies12010008.

[25] Cheng, Z., Chow, M. Y., Jung, D., & Jeon, J. (2017). A big data based deep learning approach for vehicle speed prediction. IEEE International Symposium on Industrial Electronics, 389–394. doi:10.1109/ISIE.2017.8001278.

[26] Shukla, S., Balachandran K, & Sumitha V S. (2016). A framework for smart transportation using Big Data. 2016 International Conference on ICT in Business Industry & Government (ICTBIG), 1–3. doi:.1109/ictbig.2016.7892720.

[27] Dunayev, P., Abramov, S., Sansyzbay, K., & Kismanova, A. (2021). The IP channel bandwidth during transmission of the video and tomography signals. Journal of Theoretical and Applied Information Technology, 99(12), 2834–2859.

[28] Arfat, Y., Suma, S., Mehmood, R., & Albeshri, A. (2020). Parallel Shortest Path Big Data Graph Computations of US Road Network Using Apache Spark: Survey, Architecture, and Evaluation. Smart Infrastructure and Applications. EAI/Springer Innovations in Communication and Computing. Springer, Cham, Switzerland. doi:10.1007/978-3-030-13705-2_8.

[29] Rashid, A. N. M. B., Ahmed, M., & Ullah, A. B. (2022). Data Lakes: A Panacea for Big Data Problems, Cyber Safety Issues, and Enterprise Security. Next-Generation Enterprise Security and Governance, 135–162, CRC Press, Boca Raton, United States. doi:10.1201/9781003121541-6.

[30] Martinez-Mosquera, D., Navarrete, R., Luján-Mora, S., Recalde, L., & Andrade-Cabrera, A. (2024). Integrating OLAP with NoSQL Databases in Big Data Environments: Systematic Mapping. Big Data and Cognitive Computing, 8(6), 64. doi:10.3390/bdcc8060064.

[31] Farhan, M. S., Youssef, A., & Abdelhamid, L. (2024). A Model for Enhancing Unstructured Big Data Warehouse Execution Time. Big Data and Cognitive Computing, 8(2), 17. doi:10.3390/bdcc8020017.

[32] Lei Yu, Yunyun Zhu, W. M. (2024). Quality Improvement Model of English Teaching in Universities Based on Big Data Mining. Journal of Electrical Systems, 20(3s), 506–518. doi:10.52783/jes.1322.

[33] Grădinaru, G. I., Dinu, V., Rotaru, C. L., & Toma, A. (2024). The Development of Educational Competences for Romanian Students in the Context of the Evolution of Data Science and Artificial Intelligence. Amfiteatru Economic, 26(65), 14–32. doi:10.24818/EA/2024/65/14.

[34] Khalid, M., & Yousaf, M. M. (2021). A comparative analysis of big data frameworks: An adoption perspective. Applied Sciences (Switzerland), 11(22), 11033. doi:10.3390/app112211033.

[35] Arif, Z., & Zeebaree, S. R. (2024). Distributed Systems for Data-Intensive Computing in Cloud Environments: A Review of Big Data Analytics and Data Management. The Indonesian Journal of Computer Science, 13(2), 3819. doi:10.33022/ijcs.v13i2.3819.

[36] Gupta, D., & Rani, R. (2018). A study of big data evolution and research challenges. Journal of Information Science, 45(3), 322–340. doi:10.1177/0165551518789880.

[37] Ataie, E., Evangelinou, A., Gianniti, E., & Ardagna, D. (2022). A Hybrid Machine Learning Approach for Performance Modeling of Cloud-Based Big Data Applications. Computer Journal, 65(12), 3123–3140. doi:10.1093/comjnl/bxab131.