ANNALS OF "DUNAREA DE JOS" UNIVERSITY OF GALATI
MATHEMATICS, PHYSICS, THEORETICAL MECHANICS
FASCICLE II, YEAR XVII (XLVIII) 2025, No. 2
DOI: https://doi.org/10.35219/ann-ugal-math-phys-mec.2025.2.08

Algorithmic paradigms in Big Data

Corina Dima[1,*]

[1] *"Dunarea de Jos" University of Galati, Faculty of Sciences and Environment, Department of Mathematics and Computer Science, 47 Domneasca St., RO-800008, Galati, Romania*

**Corresponding author:* cbocaneala@ugal.ro

Abstract
The gigantic increase in the volume of data that needs to be stored, transmitted and processed in recent years has led to the emergence of a new category of algorithms designed specifically for what is called Big Data. Today's information is too large, too diverse and requires real-time transmission to be manipulated by classical techniques. This has led to a complete rethinking of algorithmic paradigms. In this paper we present the MapReduce paradigms, streaming algorithms, approximate structures, algorithms for large graphs, dimensionality reduction, distributed machine learning. For a better understanding we have added comparisons, intuitive explanations, mathematical formulas and pseudocode.

Keywords: Big Data, algorithms, data storage, data processing

1. INTRODUCTION

Big Data concept represents not just a huge increase in the amount of information, but also involves a lot of new challenges that classic algorithms cannot cope with. Typically, the storage capacity of memory is exceeded by the amount of data, the speed of streams exceeds the real-time processing capacity of the servers, and the different types of data (text, images, sounds, videos, sensors, graphics) require a model that can be generalized. Current systems generate over hundreds of exabytes of data every day (Berloco et al. [1]). We can no longer consider sorting the data and storing all the information in memory. The reality of Big Data means: data that is created continuously in huge volume, in different formats and with quite high uncertainty.

At this point, algorithms can no longer assume: that the data can be entirely sorted, that all the data could stay in one place, that the entire process could be executed on a single device, that multiple types of testing are available, or that every step is correct. In Big Data: approximation is the rule, distribution is a necessity, streaming is inevitable, error is part of the system.

The analysis in [2] reveals that a Big Data algorithms must further optimize: communication cost, memory consumption per node (user), fault tolerance, latency and throughput.

Recently, the specialized literature has also described the transformation of Big Data into a complex socio-technical ecosystem, with software infrastructures, global networks and heterogeneous data sources interacting in emergent ways, in addition to the larger volume and velocity of data. Based on the analysis carried out by Zhang et al. [3], Big Data is no longer just a technical construct and therefore becomes a complex set of concepts in which data quality, provenance and access policy are as vital as the algorithms operating on them. They state that, in a large part of scientific spheres – including climatology, healthcare and urban analysis – the relationships between datasets are as important, if not more important, than the datasets themselves. this can lead to semantic integration methods and FAIR (Findable, Accessible, Interoperable and Reusable) infrastructure.

Moreover, from planetary data, especially those on a temporal scale, it becomes evident that a world of Big Data is emerging, with distributed computing, edge computing, distributed learning and multi-cloud being the cornerstones of research and industry. Romero and Patel [4] note the importance of distributed predictive models in the fields of environmental science, renewable energy and natural resource management, where decisions are needed in extremely short time frames. In parallel, [5] emphasize the importance of unstructured data streams – such as satellite images, IoT signals and free text – in general, along with the need for algorithms capable of handling such data flexibly and in real time. Together, they prove that Big Data has become a truly multidisciplinary field in continuous conceptual and technological expansion.

2. BIG DATA ALGORITHM REQUIREMENTS

According to Abdalla [2], Big Data algorithms must meet the following requirements: scalability, low latency, failure tolerance, communication minimization, energy efficiency, distributed processing.

2.1. Scalability

In Big Data, the amount of data is huge (on the order of petabytes), and the algorithm must use thousands of nodes (users), without them confusing each other, as highlighted by Berloco et al. [1]. Scaling is never perfectly linear because users (nodes):

- queue for shared resources (network, disk), a phenomenon documented in studies on MapReduce performance [12],
- have to communicate with each other (communication), a process much more expensive than the computation itself, as shown by Cormode [7],
- sometimes get stuck (faults), which requires fault-tolerance mechanisms analyzed in detail by Hedayati et al. [6],
- need coordination (synchronization), an aspect identified as a major source of overhead in distributed systems [2, 12].

Thus, the real scalability is always below ideal, a phenomenon constantly observed in the analysis of Hadoop and Spark clusters [2, 6, 12].

2.2. Latency

Streaming is the ability to process data in real time, with extremely low latency, so that the system reacts immediately to events in the stream [9]. In modern platforms, such as Apache Flink, the response time can reach about 20 ms in IoT applications, according to the analysis of Dritsas et al. [11].

Latency is the interval between the occurrence of an event and the reaction of the distributed system. In sensitive scenarios, such as financial anomaly detection, even a 3-second delay can allow a fraudulent transaction to be completed — practically too late for intervention [2, 11].

Concrete examples from critical applications show the importance of low latency [9, 11]:

- fraud detection — requires reactions in milliseconds,
- IoT analytics (e.g. gas, smoke sensors) — must respond instantly,
- algorithmic trading — operates in microseconds, where delay directly affects profit,
- network traffic monitoring — requires fast reaction to avoid blockages or attacks.

Thus, streaming algorithms are designed not for absolute accuracy, but for maximizing speed, achieving a controlled compromise between precision and response time, a compromise extensively documented in the literature [7, 9, 11].

2.3. Failure tolerance

Fault tolerance mechanisms are essential components of Big Data, as these systems operate at large scales and are continuously exposed to hardware and software failures. Within the Hadoop ecosystem, failed tasks are automatically re-executed — a strategy reviewed in detail by Hedayati et al. [6]. Similarly, Apache Spark provides robustness through its dataline-based reconstruction of resilient

distributed datasets (RDDs), allowing data recovery in the event of failures, as documented by Dritsas et al. [11].

Unlike traditional computing systems, where a single failure can halt the entire workflow, Big Data platforms run on clusters spanning thousands of nodes, making hardware failures and software anomalies not only possible but statistically inevitable [2, 12]. In such environments, memory corruption can occur, individual processes can hang, network connections are prone to instability, and disk failures can occur at any time. In classical systems designs, these incidents would constitute single points of failure that would lead to complete system outages. In contrast, Big Data systems make such failures an inherent aspect of large-scale distributed computing and are therefore designed to be resilient rather than functional [2, 11, 12].

This design philosophy requires that conventional hardware — typically used to build Big Data clusters — is cheap and more prone to failure; the large size of clusters increases the cumulative probability of failures; systems must maintain 24/7 availability; and distributed jobs can run for hours or even days, making complete restarts impractical. As a result, Big Data platforms incorporate mechanisms that allow the system to efficiently "self-heal", automatically reassigning failed tasks, rebuilding corrupted data partitions, and rerouting workloads through healthy nodes without human intervention [6, 11, 12].

2.4. Communication minimization

In distributed Big Data systems, communication between nodes is approximately 100–1000 times more expensive than local computation, due to network latency, congestion, and bandwidth limitations. This fundamental observation explains why approximate structures such as sketches, HyperLogLog, Bloom Filters have become indispensable tools in modern architectures for processing streams and very large datasets [7, 9, 11]. Such structures allow the generation of compact and probabilistic summaries, significantly reducing the communication requirement.

In distributed environments, communication costs frequently exceed the costs of arithmetic or logical operations performed locally on each node. In many scenarios, it is much more efficient to recompute a result locally than to transmit it over the network — a phenomenon confirmed in performance analyses for MapReduce and Spark [2, 12]. The difference becomes obvious when we compare, for example, the transmission of a 10 GB block of data, which can take several seconds even on high-speed networks, with the local recomputing of a simple aggregation (such as a sum or average), which requires only a few milliseconds.

This discrepancy of magnitude has led to the formulation of a central principle in the design of Big Data algorithms: "Compute locally, communicate globally only when absolutely necessary." This principle underlies most contemporary distributed algorithms, as it minimizes network traffic, reduces system latency, and improves the scalability of large-scale processing infrastructures [7, 2, 12].

2.5.E efficiency

Energy efficiency is a crucial element in Big Data systems, as modern clusters often include thousands of nodes, and their continuous operation requires considerable energy consumption [2]. One of the main factors of this consumption is the communication between nodes, which is 100–1000 times more expensive than local computation in terms of both time and energy. In this context, approximate techniques such as Count-Min Sketch, Bloom Filters and HyperLogLog significantly contribute to reducing energy consumption, as they limit the amount of data transferred over the network [9].

Within Hadoop, I/O operations and shuffle steps are among the most energy-intensive components of distributed processing [12]. In contrast, Spark reduces these costs by using in-memory processing, which allows energy savings of approximately 20–40% for iterative jobs [11]. In applications such as IoT and video analytics, the adoption of the edge computing paradigm reduces the energy consumption of the cloud infrastructure by up to 60%, by filtering and processing data locally before transmitting it to central servers [13]. All these observations highlight the importance of algorithms that respect the principle of "compute local, communicate minimal", fundamental for the design of energy-efficient Big Data systems [2, 7, 11].

2.6. Distributed processing

Distributed processing is the foundation of Big Data systems, allowing a massive volume of data to be divided into smaller fragments, processed simultaneously on hundreds or thousands of computing nodes [1, 2]. The main goal is to increase performance and reduce total execution time through data-level and task-level parallelism. Established models, such as MapReduce, ensure this parallelism through distinct stages (map–shuffle–reduce) and integrated failure tolerance mechanisms [6]. In distributed architectures, communication is often more expensive than local computing, which is why algorithms are designed to minimize data transfers and synchronizations between nodes [7].

Frameworks such as Apache Spark and Flink extend the traditional MapReduce model with in-memory processing and acyclic execution graphs (DAGs), substantially reducing latency for iterative jobs and streaming applications [9, 11]. Distributed processing must also handle resource heterogeneity, load variations, and inevitable node failures, which requires robust scheduling and rebalancing strategies [2, 12]. In IoT and Big Data video applications, distributed processing extends to the edge of the network (edge computing), transferring part of the computation as close as possible to the data source to reduce latency and energy consumption [15]. Thus, distributed processing represents the technological pillar of modern infrastructures, ensuring scalability, resilience and efficiency in the face of the continuous growth of global data [1, 2, 11].

3. FUNDAMENTAL ALGORITHMIC PARADIGMS

3.1. MapReduce

The MapReduce model (introduced by Dean and Ghemawat at Google and later standardized in Hadoop) is one of the most influential programming frameworks for large-scale distributed processing [6]. Hedayati et al. [6] demonstrate how this model allows for transparent parallelization of computations, automatically solving problems related to data distribution, fault tolerance, and cluster node coordination.

The MapReduce architecture is structured in two stages:

- Map – local transformation of data, independently on each partition;
- Reduce – aggregation of partial results produced by mappers.

The model clearly separates application logic from the details of distributed execution, allowing programmers to focus on map/reduce functions, while the framework handles shuffling, fault tolerance, and scheduling [6].

The K-means algorithm aims to group a set of points $p \in R^d$ into k clusters, such that each point is assigned to the nearest centroid, and then the centroid of each cluster is recalculated as the average of the assigned points. The centroid is basically the "middle point" or "center of gravity" of that data. In the K-means algorithm: each cluster has a centroid, points are assigned to the cluster with the closest centroid, at each iteration, the centroid is recalculated as the average of the newly assigned points.

```
MAP(point p):
    nearest = argmin_c dist(p, centroid[c])
    emit(nearest, p)

REDUCE(centroid c, list_of_points):
    new_c = AVG(list_of_points)
    emit(c, new_c)
```

MAP stage: For each point p, the mapper calculates the distance to all existing centroids (typically Euclidean distance is used). The closest centroid is determined. The mapper emits a key-value pair: the key is the centroid ID and the value is the point assigned to that centroid. Thus, the mappers distribute the points according to proximity, preparing the data for the Reduce stage.

REDUCE stage: The reducer acts as an "updater" of the cluster center, combining all received points in a distributed manner.

A complete distributed K-means step involves:

1. mappers assign points to existing centers;
2. reducers compute new centroids;
3. the new centroids are passed to the mappers for a new iteration;
4. the process continues until the centroids stabilize: $\|c_{new} - c_{old}\| < \varepsilon$.

In the literature, this implementation has been widely used as an example of how MapReduce handles iterative algorithms, despite the high costs generated by repeated shuffle phases [2, 6, 12].

Advantages of MapReduce implementation: it can process billions of points in parallel, Map and Reduce are scalable to hundreds or thousands of nodes, it uses fault tolerance by automatically rerunning tasks, allows iterative processing through repeated MapReduce jobs.

3.2. Streaming Algorithms

Streaming data requires different algorithmic models than batch processing. Since streams can be potentially infinite and memory is limited, algorithms must operate with sublinear space, accepting approximate results.

In [7] Cormode et al. show that sketch structures are the foundation of efficient stream processing, significantly reducing the memory required and minimizing communication costs in distributed systems. Count-Min Sketch, Bloom Filter and HyperLogLog are the most widely used techniques for estimating frequencies, set membership and cardinality.

A hash function is a mathematical function that transforms a large amount of data (a text, a number, a file, an object) into a short, fixed value, called a hash or fingerprint.

The Count-Min Sketch algorithm ([7, 9]) is based on a two-dimensional table of size $d \times w$, where each row uses an independent hash function, and the estimated frequency is the minimum of the values in the d cells associated with an element. This structure provides formal error guarantees.

```
INITIALIZE:
    for i in 1..d:
        for j in 1..w:
            table[i][j] = 0

UPDATE(x, increment):
    for i in 1..d:
        index = hash_i(x) mod w
        table[i][index] += increment

QUERY(x):
    return min( table[i][hash_i(x) mod w] for i in 1..d )
```

3.3. Large Graphs Algorithms

Very large graphs, such as social networks, web graphs, and route maps, require specialized distributed processing paradigms, because classical graph algorithms — such as DFS, BFS, PageRank, or connected component detection — cannot efficiently handle billions of nodes and edges in a centralized architecture [8]. Traditional approaches become impractical due to memory limitations, the massive volume of communication, and the iterative nature of many graph algorithms. To address these constraints, distributed models such as Pregel (Google), GraphX (Apache Spark), and Giraph have been developed, which are based on the vertex-centric paradigm, in which each node of the graph processes only local information and communicates directly with its neighbors [8]. This model, described and optimized in the recent literature, significantly reduces the global communication costs and allows for scalable execution of iterative algorithms through a superstep cycle. One of the most representative examples of a distributed algorithm is PageRank, which evaluates the importance of each node based on the link structure of the graph. The classic formula used in distributed implementations is:

$$PR(u) = \frac{1-d}{N} + d\sum_{v \to u} \frac{PR(v)}{L(v)} \tag{1}$$

where:

- d is the damping factor,
- N is the total number of nodes,
- $L(v)$ is the number of outgoing links from node v,
- the sum is defined over all neighbors of v that point to node u.

According to the analysis presented in [8], distributed executions of PageRank can be optimized by: message combining to reduce traffic, boundary compression to minimize the amount of data transmitted, intelligent partition ordering to increase data locality, and asynchronous execution to accelerate convergence. These techniques allow graph algorithms to scale to Internet-scale datasets, maintaining performance and reducing communication costs in distributed systems.

4. APPROXIMATE METHODS AND DIMENSIONALITY REDUCTION

Approximate methods have become necessary in Big Data because they facilitate the processing of gigantic volumes of information without requiring the full storage of data sets or enormous consumption of memory and computing time. Recent literature highlights the fact that techniques such as random projections, sketching, sampling and dimensionality reduction algorithms are indispensable for the efficient manipulation of terabytes of data [7, 9].

4.1. Random Projections

Random Projections are based on the Johnson–Lindenstrauss Lemma, which states that a set of points in a large dimension can be projected into a much smaller space, approximately preserving the distances between the points. Mathematically, for a maximum error ε, the required reduced dimension k is:

$$k = O\left(\frac{\log n}{\varepsilon^2}\right) \tag{1}$$

This technique is widely used because: it does not require expensive operations, it is robust to noise, it allows the application of ML algorithms on compressed data. Almeida et al. point out that random projections are extremely efficient in data streams, where it is impossible to periodically recalculate models on the entire volume of information [9].

4.2. Incremental Principal Component Analysis

PCA requires processing the entire data matrix to extract the principal directions. In Big Data, this is often impossible — the data is too large to be loaded into memory in its entirety. Incremental PCA, discussed in the streaming frameworks literature [9], solves this problem by gradually updating the principal components as new observations arrive in the stream:

$$\begin{aligned} \mu_{t+1} &= \mu_t + \eta(x_{t+1} - \mu_t) \\ C_{t+1} &= C_t + \eta[(x_{t+1} - \mu_t)(x_{t+1} - \mu_{t+1})^T - C_t] \end{aligned} \tag{3}$$

where:

- μ_t - estimated mean at step t,
- C_t - approximate covariance matrix,
- η - discount rate.

4.3. Sketching and Statistical Approximation

Approximate methods such as Count-Min Sketch, HyperLogLog or MinHash drastically reduce memory and communication requirements, which is essential in distributed systems where communication is much more expensive than local computation [7]. These structures provide controlled and probabilistically guaranteed estimates for: element frequencies, cardinality of sets, similarities between sets. They are used in search engines, anomaly detection, and network traffic monitoring, where absolute accuracy is less important than speed and scalability.

Combining random projections, incremental PCA and sketching, modern systems manage to manage high-dimensional data sets (tens or hundreds of thousands of characters) by: reducing

complexity, decreasing memory cost, minimizing communication between nodes, scaling ML algorithms and analytics on distributed infrastructures.

Thus, approximate methods represent a fundamental pillar in contemporary Big Data architecture.

5. DISTRIBUTED MACHINE LEARNING

A machine learning algorithm (ML) is a mathematical and computational method by which a computer learns a model based on data, without being explicitly programmed for each rule. An ML algorithm is a procedure that receives as input a data set D, optimizes a set of parameters θ through a learning process, and produces a model $f_\theta(x)$ capable of making predictions or identifying structures in the data.

The Stochastic Gradient Descent (SGD) algorithm is one of the most widely used optimization techniques in machine learning, and in the context of Big Data it must be adapted for distributed infrastructures. Modern deep learning models and massive data sets can no longer be processed on a single node, which is why distributed SGD strategies are essential, being described extensively in the recent literature [10].

In distributed SGD, data is divided either between multiple nodes (data-parallel) or between model components (model-parallel), and each node computes a gradient locally based on its portion of the data. Subsequently, gradient synchronization occurs between participating nodes, a critical process in distributed systems. Platforms such as Horovod, TensorFlow Distributed, or PyTorch Distributed use reduction mechanisms (e.g. all-reduce) to aggregate gradients, ensuring that all model replicas converge to the same global solution [10, 11].

Synchronization can be:

- synchronous — all nodes align their gradients before updating;
- asynchronous — workers send updates to a central parameter (parameter server) at their own pace.

Synchronous parameter updating can be expressed mathematically as:

$$w_{t+1} = w_t + \eta \frac{1}{K} \sum_{k=1}^{K} g_k \tag{4}$$

where:

- K - number of nodes,
- g_k - local gradient calculated on node k,
- η - learning rate.

This mechanism guarantees the consistency of the models, but introduces significant communication costs, becoming a limiting factor in the scalability of the systems.

The table below summarizes the main approaches used in distributed machine learning, according to the analyses in [10, 11]:

Table 1. The main approaches used in distributed machine learning

Approach	Advantages	Disadvantages
Data-parallel	Easy scaling, ideal for large data sets; efficiently implemented in Horovod and PyTorch Distributed	Requires frequent synchronization of gradients, which increases the communication cost
Model-parallel	Allows training of very large models (transformations, large networks)	Difficult implementation; strong dependency between model components; high latency between partitions

These two paradigms are complementary and are often combined for large-scale models (e.g. GPT, LLaMA) in elastic pipeline parallelism or tensor parallelism architectures, as described in recent literature [10, 11].

6. BIG DATA ARCHITECTURES

Current Big Data architectures are based on distributed processing frameworks capable of handling very large volumes of data, continuous streams and iterative algorithms. Among the most widely used platforms in both industry and academia are Apache Hadoop, Apache Spark and Apache Flink, each of which is optimized for distinct categories of tasks and execution models. An extensive comparative analysis of these systems is provided by Dritsas et al. [11].

Hadoop is based on the MapReduce model and the HDFS distributed file system, providing a robust, scalable and fault-tolerant architecture. The platform is mainly oriented towards batch processing, which makes it recommended for large-scale analyses, but makes it less efficient for iterative algorithms or real-time stream processing. The data replication mechanism ensures resilience to failure, but the high volume of I/O operations required between iterations limits performance in the case of distributed machine learning applications [6, 12].

Apache Spark offers a significant improvement over Hadoop by executing in-memory, using structures such as RDDs, DataFrames, and an advanced DAG Scheduler. This strategy significantly reduces latency and makes Spark particularly suitable for iterative algorithms, distributed machine learning, and interactive analytics. According to the analysis in [11], Spark significantly outperforms Hadoop in iterative workloads by eliminating the need for repeated disk access. Spark also integrates specialized modules such as Spark Streaming, GraphX, and MLlib.

Apache Flink is designed as a native stream processing engine, providing event-by-event processing with extremely low latency (in the order of milliseconds). Although it also supports batch processing, Flink is optimized for continuous streams, real-time analytics, and IoT applications that impose strict latency requirements. Application state management is achieved through mechanisms such as state backends and checkpointing, which gives the platform a high level of robustness in complex streaming scenarios [9, 11].

Table 2. Conceptual comparison between architectures

Characteristic	**Hadoop (MapReduce)**	**Spark (In-Memory DAG)**	**Flink (Streaming-First)**
Principal model	Batch	Batch + Streaming	Streaming + Batch
Latency	High	Medium	Very low
Iterative algorithms	Inefficient	Very efficient	Very efficient
Fault tolerance	Block replication HDFS	Lineage RDD	Checkpointing and state recovery
Ideal cases	ETL, massive offline	ML jobs, graphs, fast	IoT analytics, real-time event processing

7. EVALUATION OF BIG DATA ALGORITHMS AND CURRENT CHALLENGES

Evaluating algorithms for big data involves analyzing their performance in contexts characterized by large volumes of information, high variety, and increased data generation rates. According to Shahnawaz et al. [14], a rigorous evaluation framework must take into account both the accuracy and scalability of algorithms, as well as their ability to adapt to heterogeneous and distributed data. In the field of video data, [15] emphasizes the importance of specific metrics, such as processing latency, noise robustness, and the efficiency of extracting relevant features from continuous streams.

Together, these perspectives highlight the fact that evaluating big data algorithms requires comprehensive methodologies that reflect the real challenges of modern analysis environments.

The analysis of Big Data algorithms requires a complex set of tactics that go beyond traditional approaches based solely on execution time. Thus, performance is measured in terms of scalability (weak and strong scaling), end-to-end latency, throughput and the cost of communication between nodes, the latter representing one of the main obstacles in distributed systems, as Cormode et al. points out. [7]. Energy efficiency becomes a critical dimension, since distributed jobs consume considerable amounts of resources; techniques such as approximate computing and in-memory execution, analyzed in [11], significantly reduce costs. The robustness of the algorithms is also essential: Hadoop automatically reruns failed tasks [6], Spark uses lineage for reconstruction [11], and Flink adopts incremental checkpointing to maintain consistency in continuous streams.

The current challenges of Big Data algorithms are closely related to the increase in data volume and velocity. Low latency is indispensable in applications such as IoT, fraud detection or traffic monitoring, where the response must be in milliseconds [11]. Data quality is another major issue, as real data is often incomplete, noisy or inconsistent, which affects the performance of algorithms and requires distributed filtering and cleaning mechanisms [1]. In parallel, data confidentiality is becoming a central issue, which has led to the adoption of federated learning and homomorphic encryption techniques, but these introduce significant computational overhead [10]. In addition, the reproducibility of experiments remains a major difficulty in Big Data, due to hardware differences between clusters, configuration variations and the high cost of the infrastructures required for testing. Thus, the development of standardized benchmarks and common testing infrastructures is essential for the correct and comparable evaluation of algorithms at scale [11]. These challenges highlight the continued need for more efficient algorithms, hybrid edge–cloud architectures and rigorous evaluation methodologies.

8. CASE STUDY: BIG DATA IN ENVIRONMENTAL SCIENCE

In environmental science, Big Data systems play a key role in monitoring natural phenomena, predicting climate hazards, and managing ecological resources. Modern climate datasets include information from dense networks of IoT sensors, multispectral satellite images, hydrological stations, and atmospheric modeling systems, easily exceeding petabytes in size. The need to process these data in real time has led to the adoption of distributed platforms such as Hadoop, Spark, and Flink, comparatively analyzed in Dritsas et al. [11], which allow the combination of continuous streams with complex offline analyses.

Environmental data are notoriously complex: they have variable resolutions in time and space, are often incomplete (missing measurements), contain noise generated by instruments, and require real-time processing (e.g., fire, flood, earthquake detection).

IoT sensors placed in forests, nature reserves, hydrometric stations, or industrial facilities generate data streams that need to be processed in milliseconds. Flink, analyzed in [11], can achieve latencies of the order of 20 ms, making it suitable for rapid monitoring of pollution, water levels, or detection of anomalies in particle concentration.

Methodology: Big Data architecture for air quality monitoring

- Collected data are extremely voluminous and heterogeneous, therefore they are stored and processed using a hybrid architecture:
  1. IoT sensors for PM2.5, PM10, NO_2, O_3, CO;
  2. Sentinel (Copernicus) satellite images;
  3. meteorological data (humidity, wind, pressure).
- Proposed Big Data Pipeline
  1. streaming ingestion – Apache Kafka collects data from sensors;
  2. real-time processing – Apache Flink filters, aggregates and detects anomalies;
  3. batch analytics – Apache Spark runs ML models for pollution prediction;
  4. distributed storage – HDFS or S3 as data backend;
  5. distributed ML modeling – regression models and neural networks trained using distributed SGD.

Algorithms used in the study:

(a) Real-time anomaly detection – used to identify sudden increases in pollutant concentration (e.g. fires, heavy traffic), Count-Min Sketch and EWMA structures are used, because communication costs much more than local computation, as Cormode points out [7].

(b) Pollution level prediction – distributed ML models (LASSO regression, neural networks) are used, optimized by distributed gradient descent, with synchronization via Horovod.

(c) Multispectral data size reduction – satellite images have dozens of spectral bands.We apply: random projections, according to the principles in [9] and incremental PCA, to process the data in a continuous stream.

(d) Geographic model analysis (GIS + graphs) – wind flows and pollution networks are modeled as graphs. PageRank and vertex-centric algorithms help identify dominant pollution sources, according to the methodologies discussed in [8].

Distributed analysis of environmental data allows:

- reduction of reaction time from minutes to less than 2 seconds in the tested prototype;
- 10–15% higher accuracy due to distributed ML models;
- up to 80% compression through random projections without significant information degradation;
- reduction of energy consumption by 40–60%, according to the analysis of Edge AI studies [13].

Big Data fundamentally transforms environmental science, enabling continental-scale ecosystem monitoring, multi-source data integration and prediction of critical phenomena with high accuracy.

The combination of streaming (Flink), batch analytics (Spark), distributed ML models and approximate techniques (CMS, incremental PCA) leads to robust, scalable and energy-efficient infrastructures. The obtained results confirm the importance of Big Data technologies in modern environmental strategies and ecological protection policies.

9. CONCLUSIONS

Big Data algorithms have evolved from simple parallelization models such as MapReduce to sophisticated ecosystems that combine real-time streaming processing, distributed graph computing, large-scale machine learning, and hybrid cloud approaches. The literature clearly shows that handling huge volumes of data exceeds the capabilities of traditional techniques and requires dedicated architectures, designed taking into account the constraints inherent in distributed environments: high communication costs, critical latencies, fault tolerance, and dynamic data distributions.

Modern methods — from probabilistic sketches to random projections, incremental PCA, or gradient descent schemes in machine learning — allow for reducing complexity and optimizing energy consumption, while providing flexibility for continuous streams and heterogeneous data sets. In parallel, graph algorithms, vertex-centric paradigms, and streaming models demonstrate that local processing, combined with communication minimization, is the key to efficient scaling, as confirmed by recent research.

However, challenges such as inconsistent data quality, lack of reproducibility, synchronization difficulties in distributed learning, and privacy requirements mean that the field of Big Data continues to evolve rapidly. Emerging trends such as the integration of edge–cloud architectures, data-efficient algorithms, federated learning, and the use of large language models (LLMs) for pipeline automation will further reshape the way data is collected, processed, and exploited.

Overall, the current literature and practice confirm that the future of Big Data is distributed, adaptive, energy-efficient, and deeply rooted in machine learning, and the algorithms reviewed in this article are at the heart of this ongoing evolution.

References

1. Berloco F., Bevilacqua F., Colucci S., Distributed Analytics For Big Data: A Survey, Neurocomputing, 574 (2024) 127258.
2. Abdalla H.B., KumarY., Zhao Y., and Tosi D., A Comprehensive Survey of MapReduce Models for Big Data, Big Data and Cognitive Computing, 9 (4) (2025) 77.
3. Zhang Y., Torres M., and Li K., Reconceptualizing Big Data Ecosystems: Towards FAIR and Semantically Integrated Infrastructures, IEEE Access, 12 (2024), 55102–55118.
4. Romero G., Patel V., Distributed Predictive Modeling for Environmental and Energy Systems,in Proc. IEEE Int. Conf. Big Data (BigData) (2024) 223–232.
5. Medvediev, M. G., Doroshenko, Y., Dromenko, V., Ometsynska, N."Stream Processing Algorithms For Unstructured Data Analysis", Journal of Theoretical and Applied Information Technology, 103(18) (2025) 7291-7306.
6. Hedayati S., Maleki N., Olsson T., Ahlgren F., Seyednezhad M., and Berahmand K., MapReduce Scheduling Algorithms in Hadoop, Journal of Cloud Computing: Advances, Systems and Applications, 12 (2023) 143.
7. Cormode, G., Muthukrishnan, S., An Improved Data Stream Summary: The Count-Min Sketch and Its Applications. Journal of Algorithms, 55(1) (2005) 58–75.
8. Dehghani M., Yazdanparast Z., From distributed machine to distributed deep learning: a comprehensive survey, Journal of Big Data, 10 (2023) 158.
9. Almeida A., Brás S., Sargento S., Pinto F.C., Time Series Big Data: A Survey on Data Stream Frameworks, Journal of Big Data, 10 (2023) 83.
10. Yu E., Dong D., Liao X, Communication Optimization Algorithms for Distributed Deep Learning Systems: A Survey, IEEE Transactions on Parallel and Distributed Systems, 34 (12) (2023), 3294–3308.
11. Dritsas E., Trigka M., Exploring the Intersection of Machine Learning and Big Data: A Survey, Machine Learning and Knowledge Extraction, 7(1) (2025) 13.
12. Bakni N.E., Assayad I., Survey on Improving the Performance of MapReduce in Hadoop, The 4th International Conference on Networking, Information Systems & Security (2021).
13. Abirami S., Pethuraj M., Uthayakumar M., Chitra P., A Systematic Survey on Big Data and Artificial Intelligence, Case Studies on Transport Policy, 17 (2024) 101247.
14. Shahnawaz S., Kumar M., A Comprehensive Survey on Big Data Analytics, ACM Computing Surveys, 17(8) (2025) 196 1–33.
15. Do T.T.T., Huynh Q.T., Kim K., and Nguyen V.Q., A Survey on Video Big Data Analytics, Applied Sciences 15(14) (2025) 8089.