NJESR/October-2023/Volume-5/Issue-10 E-ISSN-2582-5836
DOI-10.53571/NJESR.2023.5.10.22-34

Electronic Path For Apache Hadoop In Cloud Computing: An Introduction

Patel Ram Suthar
Associate Professor (Physics)
Dr. Bhimrao Ambedkar Govt. College
Sriganganagar, Rajasthan, India
Email: prsuthara@gmail.com

(Received:25September2023/Revised:29September2023/Accepted:20October2023/Published:29October2023)

Abstract

Apache Hadoop is a free, open-source framework designed for the distributed storage and processing of large data sets across computer clusters using straightforward programming models. Doug Cutting and Mike Cafarella developed Hadoop in 2005, drawing inspiration from Google's MapReduce and Google File System (GFS) papers. Hadoop is architecturally designed to scale from a single server to thousands of machines, with each machine offering local computation and storage capabilities. This design allows for significant flexibility and efficiency in managing and processing vast amounts of data. The MapReduce function in Hadoop consists of two primary phases: the Map phase and the Reduce phase. In the Map phase, the task involves splitting the data and then mapping it, essentially organizing and preparing the data for processing. During the Reduce phase, the tasks involve shuffling the mapped data and then reducing it, which means aggregating or summarizing the data to derive meaningful insights or results. Hadoop's versatility allows it to run MapReduce programs in various programming languages, including Java, Ruby, Python, and C^{++}. This flexibility makes it accessible for a wide range of developers with different coding backgrounds. Hadoop's strength lies in its ability to store and process huge amounts of any type of data quickly, its scalability, cost-effectiveness, flexibility, and a robust ecosystem. These qualities make it a cornerstone for organizations dealing with big data analytics. While Hadoop is highly effective for certain types of big data applications, it's not a one-size-fits-all solution. Understanding its limitations is key to determining where it fits within an organization's data strategy.

Keywords: **HDFS, GFS, MapReduce, Map phase, YARN.**

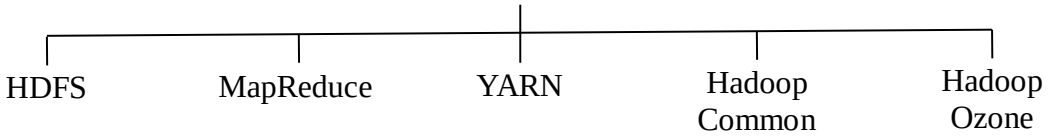
Introduction

Hadoop isn't an acronym and therefore doesn't have a full form. This name was chosen by one of its inventors, Doug Cutting, and has a rather straightforward, personal origin. It was actually the name of his young son's yellow toy elephant. Originally, Hadoop was developed as an open-source project to realize Google's MapReduce concept. It has since grown into a comprehensive framework for distributed storage and processing, capable of managing

enormous data sets across computer clusters. As a key player in the big data field, Hadoop is renowned for this capability. However, it's important to note that the name 'Hadoop' itself is not indicative of any specific technology or concept and doesn't stand for anything in technical terms.

Hadoop's genesis can be traced back to the nascent stages of the World Wide Web. During a period when the Web was expanding from millions to billions of pages, the task of conducting searches and delivering search results emerged as a significant challenge. This led startups such as Google, Yahoo, and AltaVista to develop frameworks aimed at automating the process of generating search results. Apache Hadoop represents a free, open-source framework engineered for the distributed storage and processing of substantial data sets across clustered computer systems, utilizing straightforward programming models. Conceived by Doug Cutting and Mike Cafarella in 2005, Hadoop's development was inspired by Google's MapReduce and Google File System (GFS) papers. Architecturally, Hadoop is designed to scale from single servers to thousands of machines, each providing local computation and storage. This scalable design is instrumental in affording remarkable flexibility and efficiency in the management and processing of extensive data volumes. Hadoop's versatility is evident in its compatibility with various programming languages, including Java, Ruby, Python, and C^{++}, for running MapReduce programs. This adaptability renders it accessible to a broad spectrum of developers with diverse coding proficiencies.

The strengths of Hadoop are manifold, encompassing its capacity for rapid storage and processing of massive volumes of diverse data types, scalability, cost-effectiveness, flexibility, and a robust ecosystem. These attributes establish it as a fundamental component for organizations engaged in big data analytics. Organizations frequently opt for deploying Hadoop clusters on public or hybrid cloud infrastructures as opposed to on-premises hardware. This preference is primarily driven by the desire for enhanced flexibility, improved availability, and better control over costs. Numerous cloud service providers now furnish fully managed Hadoop services. Such preconfigured, cloud-first Hadoop solutions significantly streamline operations. However, Hadoop's applicability is not universally optimal for all types of big data applications. Recognizing its limitations is crucial in ascertaining its role within an organization's broader data strategy. The core components of Hadoop are- Hadoop Distributed File System (HDFS), MapReduce, YARN, Hadoop Common, Hadoop Ozone, and some other Ecosystem Components.



1.1 Hadoop Distributed File System (HDFS)

The Hadoop Distributed File System (HDFS) stands as a pivotal element of Apache Hadoop, meticulously crafted to store vast volumes of data with high reliability and to facilitate the streaming of these data sets to user applications at substantial bandwidths. A comprehensive examination of HDFS reveals the following aspects:

1.1.1 Distributed Storage

HDFS is founded on the principle that the optimal approach to data processing involves a write-once, read-many-times pattern. This framework facilitates the storage of data across a dispersed network of nodes within a Hadoop cluster, ensuring both reliable and highly scalable storage capabilities.

1.1.2 Blocks Structured File System

In HDFS, files are segmented into blocks, usually 128 MB or 256 MB by default, though this size is adjustable. These blocks are then evenly distributed across the nodes of the cluster. This methodology of block-based distribution enhances efficient storage management and facilitates rapid data processing.

1.1.3 Architecture

HDFS operates on a master/slave architecture. The master node, termed the NameNode, oversees the file system namespace and controls client access to files. Meanwhile, the slave nodes, known as DataNodes, are tasked with the storage of the actual data.

1.1.4 Data Replication

HDFS employs a strategy of replicating data across various DataNodes, a measure designed to guarantee fault tolerance. Data blocks are duplicated and dispersed across multiple cluster nodes, with a standard replication factor of three, though this can be adjusted as needed. This approach significantly bolsters data reliability and availability, ensuring that even in the event of a DataNode failure, the data is still accessible from other nodes in the system.

1.1.5 Scalability

The architecture of the system is meticulously crafted to efficiently manage and scale up to thousands of nodes, with each node contributing to the storage of a segment of the file system's data. This scalability feature is central to HDFS's ability to process and store exceedingly large data sets with high efficiency. The scalability of HDFS does have some limitations, particularly in the management capabilities of the NameNode. As the size of the cluster grows, the NameNode's memory and processing requirements increase because it must manage the metadata for a larger number of files and blocks.

1.1.6 High Throughput

HDFS is engineered to deliver high data throughput, a feature that makes it particularly adept at handling applications involving large data sets where substantial read and write operations are frequent.

1.1.7 Data Locality Optimization

A distinctive characteristic of HDFS is its focus on data locality, prioritizing the movement of processing to where the data resides, rather than transferring data to the processing location. This strategy significantly reduces network congestion and enhances the total throughput of the system.

1.1.8 Fault Tolerance

HDFS, by default, generates multiple copies of data blocks, typically creating three replicas, and distributes them across different nodes. This redundancy is a key safeguard against data loss due to hardware failures.

1.1.9 Write-Once-Read-Many Model

HDFS is tailored for handling large files, adhering to a write-once-read-many access model. This design makes it exceptionally suitable for applications dealing with substantial data sets that require minimal updates.

1.1.10 Accessibility

Although primarily engineered for high-throughput, batch-processing tasks, HDFS also offers accessibility via more conventional interfaces such as HTTP. This feature extends its usability to a wider range of applications beyond its core batch-processing capabilities.

1.1.11 Integration with MapReduce

HDFS is intricately designed to integrate with MapReduce, a software framework that facilitates the parallel processing of substantial data volumes across a Hadoop cluster. This integration capitalizes on the data locality aspect of HDFS, enhancing the speed and efficiency of data processing. In summary, HDFS stands as a robust, scalable, and efficient file system, forming a crucial component of the Apache Hadoop ecosystem. Its design and functionality make it exceptionally well-suited for the storage and management of large-scale data sets within a distributed computing environment. Hadoop's functionality is fundamentally anchored by two core components: the Hadoop Distributed File System (HDFS) and MapReduce. Both play critical roles in its overall operation and efficiency.

1.2 MapReduce

MapReduce is indeed a pivotal component of the Apache Hadoop software framework, representing both a programming model and its corresponding implementation. It is designed for processing and generating large data sets using a parallel, distributed algorithm across a cluster. Key aspects of MapReduce include:

1.2.1 Programming Model

MapReduce operates on two fundamental user-defined functions: 'Map' and 'Reduce'. Hadoop users can develop applications by implementing these two essential functions, tailoring them to their specific data processing needs. The programming model in Hadoop Distributed File System (HDFS) is closely tied to the overall architecture of Hadoop, particularly its integration with MapReduce, which is the native processing framework of Hadoop.

1.2.2 Map Function

The Map function in MapReduce is designed to handle input data formatted as key-value pairs. It functions by reading data from a source, such as HDFS, and then processes this data to generate a collection of intermediate key-value pairs. To facilitate parallel processing, these Map tasks are allocated across various nodes within the cluster.

1.2.3 Reduce Function

The Reduce function in MapReduce acts on the intermediate key-value pairs outputted by the Map function. It aggregates these data tuples based on their keys, processing these grouped tuples to yield a more condensed set of tuples as its final output.

1.2.4 Fault Tolerance

MapReduce is adeptly engineered to manage failures at the application layer, ensuring reliable processing of extensive data sets across a sizable cluster of commodity servers. This resilience is maintained even in scenarios where individual nodes within the cluster experience failures.

1.2.5 Parallel Processing

MapReduce is inherently scalable due to its ability to process data in parallel across a cluster. The framework efficiently breaks down the job into smaller tasks, which are then executed concurrently on various nodes within the cluster. This parallel execution significantly enhances the processing capacity and speed of the system.

1.2.6 Scalability

The MapReduce framework is architecturally designed to be highly scalable, capable of expanding from a single server to a vast network of thousands of machines. Each of these machines contributes to the network by offering localized computation and storage capabilities, facilitating the framework's capacity to handle increasingly large data sets and complex computational tasks.

1.2.7 Data Locality Optimization

MapReduce emphasizes the principle of data locality, striving to position the processing as near to the data source as possible. This approach significantly reduces network congestion

and, consequently, enhances the overall throughput of the system, making the data processing more efficient.

1.2.8 Simplicity

MapReduce stands out for enabling the distributed processing of substantial data sets across compute clusters, with its most striking feature being its simplicity. This design allows developers to create applications based on the framework without requiring in-depth knowledge of the complexities involved in parallel processing or fault tolerance mechanisms. Essentially, MapReduce abstracts these intricate details, making it more accessible for developers to leverage its powerful capabilities.

1.2.9 Use Cases

MapReduce is employed in a diverse range of applications, extending from basic tasks like filtering and sorting to more intricate operations such as data analytics and pattern matching. This versatility allows it to be a crucial tool in various contexts where processing and analyzing large data sets are essential.

1.2.10 Integration with Hadoop Ecosystem

MapReduce, while a formidable tool for data processing, is frequently utilized in conjunction with other components of the Hadoop ecosystem to enhance its capabilities. HDFS is often paired with MapReduce for efficient data storage, YARN for effective resource management, and additional tools like Hive and Pig. Hive facilitates data warehousing and querying, while Pig offers high-level data processing functionalities. This integrated approach leverages the strengths of each component, creating a comprehensive and robust environment for handling big data challenges. MapReduce has established itself as a foundational pillar in the realms of big data and distributed computing. It provides a reliable and efficient framework specifically tailored for processing enormous datasets. This is achieved by distributing computational tasks across large clusters of computers, thereby harnessing the collective processing power to handle extensive data challenges effectively. This capability makes MapReduce a critical component in the toolkit for managing and analyzing big data in various industries and applications.

1.3 YARN (Yet another Resource Negotiator)

YARN, an acronym for Yet Another Resource Negotiator, represents a significant evolution in Apache Hadoop's capabilities, introduced in its 2.0 version. As the framework's resource management and job scheduling module, YARN fundamentally transforms Hadoop's efficiency, scalability, and flexibility. This innovation allows for more dynamic resource allocation, thereby optimizing the management of the Hadoop cluster's resources. Here are some essential components of YARN:

1.3.1 Resource Management

YARN plays a crucial role in the Hadoop ecosystem by managing and allocating system resources, such as CPU and memory, among the various applications executing within the Hadoop cluster. It effectively monitors and balances the resource consumption and requirements across all active applications, ensuring optimal utilization and efficiency. This resource management capability is key to maintaining the performance and stability of the Hadoop cluster, especially when handling multiple, resource-intensive tasks simultaneously.

1.3.2 Decoupling Job Scheduling and Resource Management

In Hadoop's earlier iterations, the MapReduce framework was tasked with both resource management and job scheduling. This conflation of roles often led to less efficient and flexible operations. However, with the introduction of YARN (Yet Another Resource Negotiator) in later versions, there was a significant architectural shift. YARN took over the responsibilities of resource management and job scheduling, effectively decoupling these functions from the MapReduce framework. This separation greatly enhanced the efficiency and flexibility of Hadoop, allowing for more effective utilization of resources and more adaptable job scheduling, independent of the MapReduce paradigm.

1.3.3 Architecture

The Architecture of YARN encompasses a range of critical elements, RM, NM, and AM, each contributing to its overall functionality.

1.3.3.1 ResourceManager (RM)

In the Apache Hadoop YARN architecture, the Resource Manager (RM) functions as the master daemon, overseeing resource allocation within the cluster. The RM is composed of two principal components: the Scheduler, responsible for allocating resources to various running applications, and the ApplicationsManager, which orchestrates the application lifecycle and manages the execution of application-specific tasks.

1.3.3.1.1 Scheduler

In the YARN framework of Apache Hadoop, the Scheduler's primary role is to allocate resources to the various applications running within the cluster. However, it is important to note that the Scheduler does not engage in monitoring or tracking the execution status of these applications. This task is managed separately, ensuring a more streamlined and efficient distribution of responsibilities within the YARN architecture.

1.3.3.1.2 ApplicationsManager

The ApplicationsManager component within YARN is tasked with overseeing the entire lifecycle of user applications, from initiation to completion, as well as managing their

scheduling. This integral part of YARN ensures that applications are efficiently queued, scheduled, and executed within the Hadoop ecosystem.

1.3.3.2 Node Manager (NM)

Functioning as a dedicated sentinel on each node in the Hadoop cluster, the NodeManager rigorously monitors and documents resource utilization parameters, including CPU, memory, disk space, and network bandwidth. This essential data is systematically communicated to the ResourceManager, playing a pivotal role in the efficient distribution and oversight of resources throughout the Hadoop ecosystem.

1.3.3.3 ApplicationMaster (AM)

For every application within the Hadoop environment, a unique ApplicationMaster instance is deployed. This entity engages in resource negotiations with the ResourceManager, collaborating closely with NodeManagers to facilitate the execution and vigilant supervision of tasks across the cluster.

1.3.4 Scalability

YARN significantly amplifies the scalability aspect of the Hadoop ecosystem. This enhancement empowers Hadoop to efficiently manage and process data across even larger clusters and accommodate a broader array of workloads, thereby extending its applicability and performance in big data environments.

1.3.5 Improved Cluster Utilization

YARN's sophisticated resource allocation mechanism markedly enhances the utilization efficiency of cluster resources. This optimization leads to a notable improvement in the overall operational efficacy of the Hadoop cluster, ensuring resources are used more effectively and judiciously.

1.3.6 Support for Multiple Workloads

YARN significantly broadens the scope of Hadoop's functionality by facilitating the execution of diverse data-processing frameworks beyond the traditional MapReduce. This capability enables Hadoop to support a wide array of processing models and applications, encompassing interactive processing, real-time streaming, and graph processing, thereby catering to a more varied set of computational requirements and use cases.

1.3.7 Flexibility

The integration of YARN into Hadoop has transformed it into a remarkably adaptable platform. Developers can now simultaneously deploy a multitude of data processing engines within the same Hadoop cluster. This convergence encompasses a wide spectrum of computational strategies, including batch processing, interactive SQL, real-time streaming,

and machine learning. Such versatility significantly enhances Hadoop's utility, making it an ideal solution for a broad range of data processing scenarios.

1.3.8 Fault Tolerance

In line with Hadoop's overarching design principles, YARN exemplifies robust fault tolerance. This is evident in its ability to recover from failures seamlessly: the ResourceManager possesses the capability to reboot a malfunctioning ApplicationMaster. Concurrently, NodeManagers hold the proficiency to reassign tasks that encounter failures, thereby maintaining continuous operation and ensuring the resilience of the system against disruptions.

Hence, YARN has been pivotal in evolving Hadoop from a platform solely dedicated to MapReduce operations into a multifaceted ecosystem. This transformation enables Hadoop to adeptly manage an expansive array of data processing and analytical techniques, significantly broadening its application spectrum and enhancing operational efficiency.

1.4 Hadoop Common

Hadoop Common serves as the cornerstone of the Apache Hadoop ecosystem, comprising a suite of shared utilities and libraries. This foundational layer underpins and streamlines the functionality of key Hadoop modules, including HDFS, YARN, and MapReduce, thereby constituting an integral component that enhances the cohesion and efficiency of the overall Hadoop framework. Here are some essential components of Hadoop Common:

1.4.1 Foundation of Hadoop

Embedded within the Hadoop ecosystem, Hadoop Common functions as the essential infrastructure layer, delivering key foundational services and capabilities vital for the operation of other Hadoop modules. It encompasses a comprehensive set of Java libraries and scripts pivotal for initiating Hadoop, as well as the source code forming the bedrock of Hadoop's core architecture.

1.4.2 Utilities and Libraries

Hadoop Common is composed of an array of utilities and libraries that underpin crucial functions such as filesystem and I/O abstractions, security features, networking capabilities, and administrative tools. These components are integral to the seamless operation and interconnectivity of the diverse elements within the Hadoop ecosystem.

1.4.3 Abstraction and Interoperability

Hadoop Common serves as the cornerstone for simplifying complex underlying systems, encompassing filesystems, operating systems, and network architectures. This level of abstraction guarantees that Hadoop seamlessly interfaces with a wide array of data sources

and systems, fostering robust integration and interoperability within diverse computing environments.

1.4.4 Configuration Management

Hadoop Common is pivotal in orchestrating the configuration parameters and operational settings for Hadoop. This involves managing configurations for essential Hadoop daemons including the NameNode, DataNode, ResourceManager, NodeManager, among others, ensuring seamless and efficient functionality across the Hadoop ecosystem.

1.4.5 Serialization And Deserialization

Hadoop Common encompasses APIs dedicated to the serialization and deserialization processes. These APIs are crucial for encoding data into a format suitable for storage or transmission and subsequently decoding it back to its original form. This functionality is essential for the effective and efficient movement of data across the various nodes within a Hadoop cluster.

1.4.6 Remote Procedure Call (RPC) and HTTP/FTP Support

Hadoop Common includes support for Remote Procedure Call (RPC) and HTTP/FTP protocols, which are fundamental to enabling communication between different nodes within the Hadoop cluster. This capability is a cornerstone for the operation of a distributed computing environment, as it ensures seamless and efficient data exchange and task coordination across the cluster's nodes.

1.4.7 Platform Independence

Hadoop Common is equipped with the essential tools and mechanisms that grant Hadoop the versatility to operate across diverse hardware configurations and operating systems. This feature endows Hadoop with platform independence, ensuring its adaptability and functionality in a wide array of computing environments, thus broadening its applicability and ease of integration into different technological ecosystems. Hence, Hadoop Common equips the Hadoop ecosystem with essential tools and abstractions, ensuring its compatibility across diverse hardware platforms and operating systems, thus enhancing its platform independence.

1.4.8 File System and I/O Operations

The module encompasses a range of interfaces and classes designed for interaction with and management of files, not only within local file systems but also across various supported distributed file systems, including HDFS.

1.4.9 User Interface Tools

Hadoop Common includes a suite of command-line tools and interfaces designed for direct interaction with Hadoop, encompassing a range of functionalities from file system operations

to job submission and cluster management tasks. Delivers insights into the overall condition of HDFS, encompassing details about the NameNode's status, block pool metrics, and a comprehensive list of DataNodes within the cluster.

1.4.10 JAR Files and Dependencies

Hadoop Common simplifies the development process for Hadoop applications by providing essential JAR files and dependencies, streamlining the creation of applications designed to integrate seamlessly with Hadoop's core components.

In essence, Hadoop Common serves as the fundamental backbone of the Hadoop ecosystem, offering a suite of essential utilities and services. This foundational component is vital for the seamless integration and efficient operation of various Hadoop modules, solidifying its role as an integral aspect of the Hadoop platform.

1.5 Hadoop Ozone

Hadoop Ozone, a component of the Apache Hadoop ecosystem, stands as a scalable and distributed object store. In an era where data volumes are skyrocketing, the demand for storage solutions that can adeptly scale and manage large datasets is paramount. Ozone rises to this challenge, presenting itself as a formidable and scalable storage option, poised to complement or, in specific use cases, supplant the Hadoop Distributed File System (HDFS). The following points offer a comprehensive overview of Hadoop Ozone's capabilities and features:

1.5.1 Object Store

Ozone distinguishes itself from HDFS by functioning as an object store rather than a traditional file system. It handles data as discrete objects, akin to the approach adopted by AWS S3. This method offers enhanced flexibility and scalability, particularly adept at managing vast volumes of data, a significant advantage over the conventional file-based system of HDFS.

1.5.2 Scalability

Ozone's architecture is engineered to efficiently manage immense namespaces, encompassing tens of billions of files and objects. This capacity makes it particularly adept at handling scenarios with an extensive number of small files, a task where it surpasses HDFS in efficiency, thereby catering effectively to applications with such demanding storage requirements.

1.5.3 Block Storage Layer

Ozone innovatively separates its block storage layer from the namespace layer, a design choice that enhances its scalability and streamlines the management of data blocks. This

architectural distinction allows for more efficient handling of the data, facilitating Ozone's ability to scale effectively as data demands grow.

1.5.4 Compatibility with HDFS

Ozone maintains compatibility with HDFS APIs, ensuring seamless integration with existing Hadoop applications. This feature facilitates an easy transition for systems currently reliant on HDFS, as they can interact with Ozone without requiring significant alterations.

1.5.5 Ozone Components

Ozone Components includes, Ozone Manager (OM), Storage Container Manager (SCM), and Data Nodes.

1.5.5.1 Ozone Manager (OM)

Functions akin to the NameNode in HDFS, managing the namespace and orchestrating client operations within the Ozone environment.

1.5.5.2 Storage Container Manager (SCM)

Oversees the distribution of data blocks and maintains the health and functionality of the data nodes.

1.5.5.3 DataNodes

Safeguards the actual data, with Ozone repurposing HDFS DataNodes for the storage of Ozone-specific data blocks. DataNodes in Hadoop Ozone play a crucial role in data storage and management.

1.5.6 Space Utilization

Ozone is engineered for enhanced space efficiency relative to HDFS, particularly adept in scenarios characterized by an abundance of small files.

1.5.7 Robustness and Fault Tolerance

Ozone incorporates mechanisms for replication and fault tolerance, capable of duplicating data across various DataNodes to guarantee data resilience and maintain high availability.

1.5.8 Support for S3 API

Ozone facilitates integration with applications and tools tailored for Amazon S3 by providing compatibility with the S3 API.

1.5.9 Use Cases

Ozone is optimally designed for applications that demand large-scale, distributed object storage, such as big data analytics, machine learning, and environments handling extensive volumes of small files.

1.5.10 Security and Governance

Ozone incorporates robust security and governance features, including Kerberos integration for authentication and mechanisms for enforcing access control policies.

Hadoop Ozone marks a pivotal advancement within the Hadoop ecosystem, effectively overcoming HDFS's constraints in handling vast quantities of files. It offers a versatile and

scalable storage solution adeptly tailored to the demands of contemporary big data applications.

1.6 Acknowledgement

I am thankful to the Commissionerate of College Education, Rajasthan, Jaipur, and Dr. Bhimrao Ambedkar Government College, Sriganganagar, for giving me opportunities to work and providing the facility of an e-library.

References

1. White, T. (2015). Hadoop: The Definitive Guide (4th ed.). O'Reilly Media.
2. Hadoop. Web Page. hadoop.apache.org/core/
3. Almansouri, H. T., & Masmoudi, Y. (2019). Hadoop Distributed File System for Big data analysis. In 2019 4th World Conference on Complex Systems (WCCS) (pp. 1-5). Ouarzazate, Morocco. https://doi.org/10.1109/ICoCS.2019.8930804
4. Bante, P. M., & Rajeswari, K. (2017). Big Data Analytics Using Hadoop Map Reduce Framework and Data Migration Process. In 2017 International Conference on Computing, Communication, Control and Automation (ICCUBEA) (pp. 1-5). Pune, India. https://doi.org/10.1109/ICCUBEA.2017.8463824
5. Abouzeid, A., Bajda-Pawlikowski, K., Abadi, D., Silberschatz, A., & Rasin, A. (2009). HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads. In Proceedings of the VLDB Endowment International Conference on Very Large Data Bases (pp. 1-4). Lyon, France: VLDB 09.
6. Saraswat, P., & Raj, S. (2021). A Review Paper on Hadoop Architecture. International Journal of Innovative Research in Computer Science & Technology (IJIRCST), 9(6), 96-99. https://doi.org/10.55524/ijircst.2021.9.6.22
7. OpenAI. (2023). ChatGPT. https://www.openai.com/chatgpt.