iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 13 **Issue:** V **Month of publication:** May 2025

DOI: https://doi.org/10.22214/ijraset.2025.71584

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue V May 2025- Available at www.ijraset.com

Hadoop Distributed File System (HDFS) with Its Architecture

Vandana Malik
Baba Mastnath University, India

Abstract: ***The exponential growth of big data has catalyzed the development of robust, scalable, and fault-tolerant storage systems. The Hadoop Distributed File System (HDFS) stands as a key pillar in the Hadoop ecosystem, providing a distributed, resilient storage infrastructure for managing petabytes of data. This paper investigates the core architecture of HDFS, including its design principles, components, and operational workflow. It also analyzes practical implementations, advantages, limitations, and future trends. Through case studies and real-world applications, the paper illustrates how HDFS supports the ever-growing demands of modern data-driven enterprises.***

I. INTRODUCTION

With the rise of data-intensive applications in industries ranging from healthcare to finance and telecommunications, traditional centralized storage systems have proven inadequate in meeting the demands of scalability, performance, and reliability. Apache Hadoop addresses this issue through a distributed approach to both storage and processing. HDFS, a central component of Hadoop, is engineered to reliably store very large files across machines in a large cluster, offering high-throughput access to application data (White, 2015). Unlike conventional file systems, HDFS is designed with a focus on fault tolerance, minimal hardware dependency, and data locality. It forms the storage layer over which MapReduce and other distributed data processing frameworks such as Apache Hive, Apache Spark, and Apache HBase operate.

II. CORE PRINCIPLES AND DESIGN PHILOSOPHY

A. *Assumptions and Goals*

HDFS was developed based on several key assumptions:

- Hardware Failure is Common: HDFS is built to handle frequent hardware failures gracefully.
- Write-Once-Read-Many Access Pattern: Optimized for large-scale batch processing rather than random reads/writes.
- Large Data Sets: Ideal for gigabyte- to petabyte-scale files.
- Streaming Data Access: High throughput is prioritized over low latency.

B. *Design Goals*

- Simplicity and Scalability: Support for clusters with thousands of nodes.
- Portability: Platform-independent and easily deployable on heterogeneous systems.
- Fault Recovery: Ensures data integrity and availability even after node or disk failures.

III. DETAILED ARCHITECTURE OF HDFS

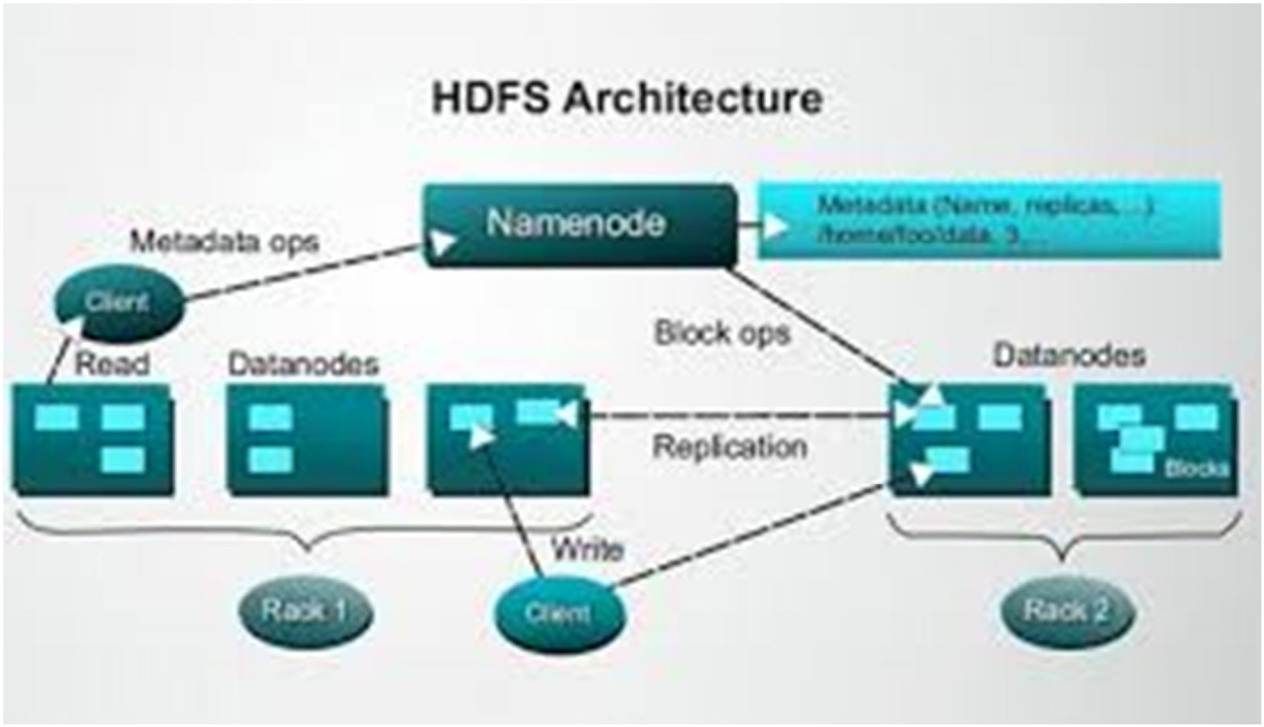
A. *System Components*

NameNode (Master)

- Maintains the file system namespace.
- Records metadata including file-to-block mappings and access permissions.
- Coordinates file operations like open, close, rename, and delete.

DataNodes (Slaves)

- Manage actual storage and serve read/write requests from clients.
- Perform block creation, deletion, and replication under NameNode instruction.
- Send periodic heartbeat and block reports to the NameNode.



Secondary NameNode

- Regularly checkpoints the file system state by merging the edit logs with the current file system image.
- Improves system stability and aids faster NameNode recovery.

HDFS Clients

- Interface with HDFS via a Java API or command-line utilities.
- Initiate requests to the NameNode and interact directly with DataNodes for data transfer.

B. Metadata and Data Separation

HDFS separates metadata (stored by the NameNode) from data (stored on DataNodes), allowing for parallel data access and better system performance.

C. File System Hierarchy

HDFS provides a POSIX-like hierarchical directory structure. Although not fully compliant with POSIX standards, it offers essential file operations like create, delete, append, and permission management.

IV. FILE MANAGEMENT AND DATA FLOW

A. File Write Operation

1) Client requests file creation via NameNode.
2) NameNode validates and returns block allocation details.
3) Client splits file into blocks and writes them sequentially to DataNodes.
4) Data is pipelined through a chain of DataNodes based on the replication factor.

B. File Read Operation

1) Client requests metadata from NameNode.
2) Based on proximity (data locality), the client retrieves data blocks directly from DataNodes.
3) Blocks are reassembled to reconstruct the complete file.

C. Block Replication and Recovery

Replication enhances fault tolerance. If a DataNode fails, the NameNode triggers replication of lost blocks to maintain the required replication factor.

V. ADVANCED FEATURES OF HDFS

A. High Availability (HA)

To mitigate the single-point-of-failure issue of the NameNode, HDFS supports HA configurations using:

- Active/Standby NameNodes
- Shared Edit Logs via Quorum Journal Manager or NFS
- Zookeeper-based Failover Controllers (ZKFC)

B. Federation

Introduced to scale the NameNode horizontally. Multiple independent NameNodes manage portions of the namespace, and DataNodes register with all NameNodes.

C. Erasure Coding (EC)

A more storage-efficient alternative to replication, EC reduces storage overhead while maintaining fault tolerance. It's particularly effective for cold or archival data.

VI. REAL-WORLD APPLICATIONS AND CASE STUDIES

A. Facebook

Facebook stores petabytes of log data and user content in HDFS clusters. It also utilizes Apache Hive and Presto for querying this data.

B. LinkedIn

LinkedIn uses HDFS as the backbone of its data infrastructure, supporting workflows including metrics processing, log ingestion, and machine learning pipelines.

C. Yahoo

Yahoo is one of the early adopters of Hadoop, running thousands of nodes with HDFS to store and process clickstream, email, and search data.

D. Healthcare and Genomics

Organizations such as the Broad Institute employ HDFS to store genome sequences and run distributed DNA analysis using MapReduce and Spark.

VII. ADVANTAGES AND LIMITATIONS

A. Advantages

1) Fault Tolerant: Self-healing through replication.
2) Cost Effective: Operates on commodity hardware.
3) Scalable: Can grow to thousands of nodes.
4) Flexible: Supports both batch and interactive workloads.

B. Limitations

1) Small File Inefficiency: Millions of small files overwhelm NameNode memory.
2) Latency: Not ideal for real-time applications.
3) POSIX Incompatibility: Lacks full support for Unix file system semantics.
4) Complex Maintenance: Requires skilled resources for administration.

VIII. FUTURE TRENDS AND ENHANCEMENTS

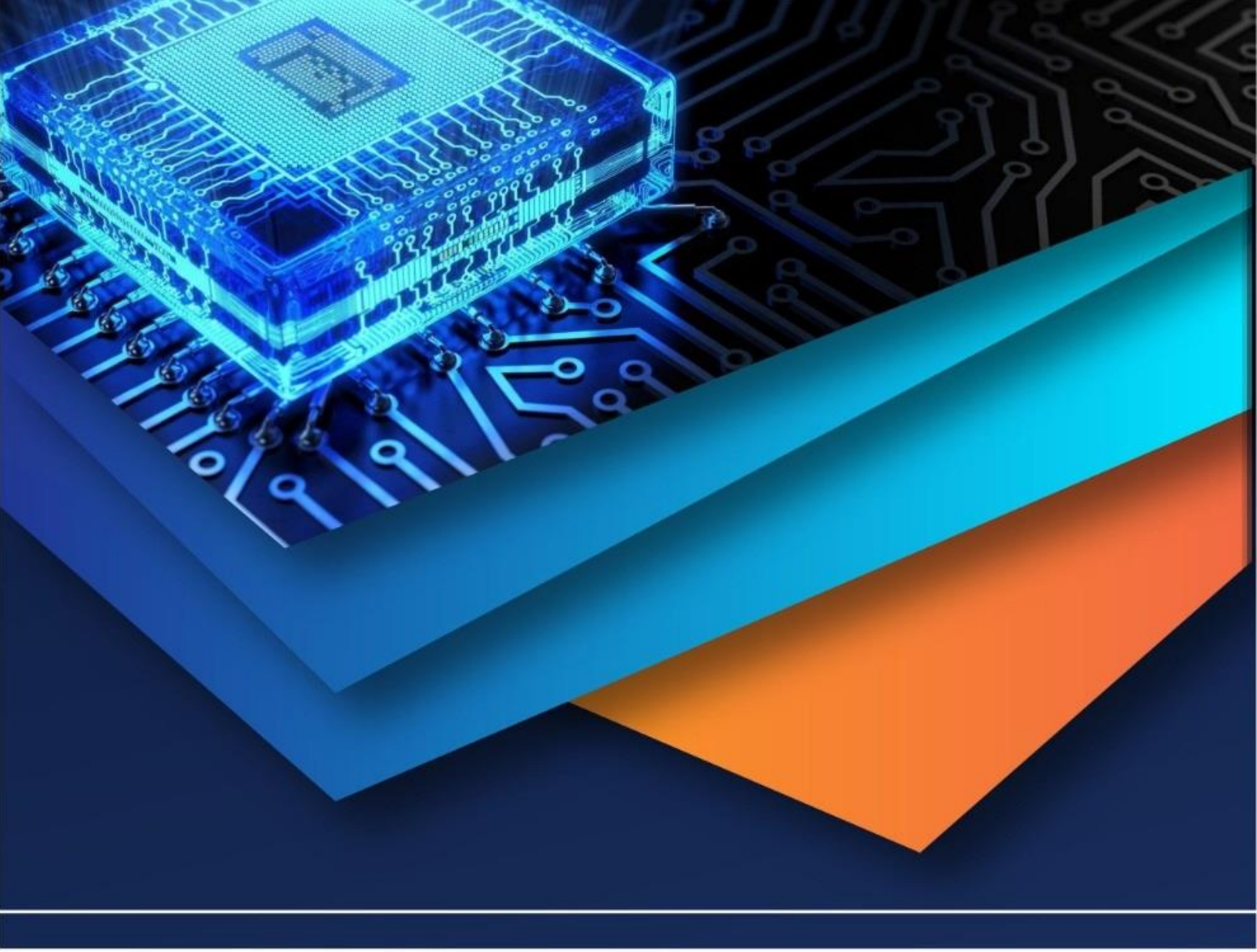
1) Cloud-Native HDFS: Integration with object stores like Amazon S3 and Azure Blob Storage.
2) Containerized Deployment: Using Kubernetes to orchestrate Hadoop services.
3) Enhanced Security: Kerberos integration, encryption-at-rest, and attribute-based access control (ABAC).
4) AI and ML Integration: Support for TensorFlow and PyTorch on Hadoop data lakes.

IX. CONCLUSION

HDFS remains a foundational technology in the big data ecosystem. Its architecture supports distributed, scalable, and reliable storage, empowering businesses and researchers to analyze massive datasets. While it faces challenges in managing small files and real-time processing, continuous innovation ensures that HDFS evolves alongside modern data needs. Understanding HDFS is essential for any professional dealing with large-scale data storage and processing.

REFERENCES

[1] Borthakur, D. (2007). *The Hadoop Distributed File System: Architecture and Design*. Apache Software Foundation. https://hadoop.apache.org/docs/r1.2.1/hdfs_design.pdf
[2] Shvachko, K., Kuang, H., Radia, S., & Chansler, R. (2010). The Hadoop Distributed File System. *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, 1–10. https://doi.org/10.1109/MSST.2010.5496972
[3] White, T. (2015). *Hadoop: The Definitive Guide* (4th ed.). O'Reilly Media.
[4] Vavilapalli, V. K., Murthy, A. C., Douglas, C., Agarwal, S., Konar, M., Evans, R., ... & Baldeschwieler, E. (2013). Apache Hadoop YARN: Yet Another Resource Negotiator. *Proceedings of the 4th Annual Symposium on Cloud Computing (SOCC '13)*. https://doi.org/10.1145/2523616.2523633
[5] Apache Software Foundation. (2023). *HDFS Erasure Coding*. https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HDFSErasureCoding.html



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY