Available online at www.HighTechJournal.org

HighTech and Innovation Journal

ISSN: 2723-9535

Vol. 6, No. 1, March, 2025

Smart Data Placement Strategy in Heterogeneous Hadoop

Nour-Eddine Bakni [1*], Ismail Assayad [2]

[1] *LIS Lab, Faculty of Sciences, University Hassan II of Casablanca, Casablanca, Morocco.*

[2] *LIS Lab, Faculty of Sciences, ENSEM, University Hassan II of Casablanca, Casablanca, Morocco.*

Received 29 September 2024; Revised 30 January 2025; Accepted 08 February 2025; Published 01 March 2025

Abstract

Big Data platforms are becoming increasingly essential these days, given the volume of data generated every moment by millions of people around the world. The Hadoop framework is a solution that allows storing and processing these large amounts of data in parallel on a cluster of machines. The default data placement strategy adopted by the Hadoop Distributed File System (HDFS), initially designed for a homogeneous cluster where all machines are considered identical, relies on distributing data to nodes based only on their disk space availability. Implementing this strategy in a heterogeneous environment, where nodes have varying computing or disk storage capacities, may result in performance degradation. In this paper, we propose a smart data placement strategy (SDPS) in heterogeneous Hadoop clusters that aims to place high-access data on high-performance nodes. It takes cluster heterogeneity into account when distributing data by first dividing nodes into groups based on their performance levels using a clustering algorithm and then allocating data blocks to appropriate nodes based on their hotness. SDPS also allows dynamically specifying the replication factor of data blocks to reduce storage space waste while maintaining data availability. Experimental results show that SDPS is more efficient in a heterogeneous environment compared with the default data placement policy of HDFS, and it improves MapReduce data processing, data locality, and storage efficiency.

Keywords: Big Data; Data Placement; Hadoop; HDFS; Heterogeneous Cluster.

1. Introduction

The accelerated growth in data volumes has prompted researchers to think differently about how to manipulate or analyze this data, which involves imposing new orders of magnitude in terms of capturing, searching, sharing, storing, and data analysis. This is how the "Big Data" [1] emerged. Big Data is the collection of massive amounts of diverse data, stored digitally and then processed using advanced technologies to establish diagnoses, make decisions accordingly, and establish action plans. Big data is experiencing constant growth and has become widely used in many fields due to its advantages. Among the reasons that have contributed to the expansion of big data are, on the one hand, the arrival and development of storage media, particularly through the spread of cloud computing [2], and on the other hand, the transformation of adaptive processing mechanisms, including the implementation of revolutionary databases supporting unstructured data (Hadoop) [3] and the design of new high-performance processing models (MapReduce) [4].

MapReduce is a programming model that promotes the parallel processing of large data sets across multiple computers in a cluster. It is one of the components that constitute the core of the Hadoop framework, which handles the processing of large data sets in a flexible and distributed manner across a set of machines or nodes, where each node

* Corresponding author: nour-eddine.bakni@taalim.ma

http://dx.doi.org/10.28991/HIJ-2025-06-01-03

forms both a processing and storage unit. The purpose of MapReduce is to provide developers with an abstraction that hides the complexity of operations related to parallelism, distribution of data processing, management of their execution in the cluster, and management of failures that may occur in the cluster during processing.

Data locality remains one of the factors that determine MapReduce performance. In general, data locality is related to the data placement strategy at the cluster level. The default data placement policy implemented by HDFS [5] disperses data across nodes based only on disk storage availability. This policy can be useful in homogeneous clusters where all nodes have the same performance characteristics. But in reality, it is very difficult to achieve a completely homogeneous environment, especially when dealing with clusters of thousands of nodes, which need to be renewed or updated from time to time, so we are talking about several generations of machines running in the same cluster. Implementing such a policy in this type of environment where nodes have varying performance, such as computing power or disk storage, may result in overall performance degradation due to over- or under-utilization of resources, slower data processing, unbalanced data distribution, etc. Several works [6-10] have addressed the problem of unbalanced workload in a heterogeneous environment and proposed dynamic placement policies aiming to balance data among nodes based on their performance. However, these research studies have not considered the heat or data access rate when placing blocks, which may have an effect on the overall system performance.

Another drawback of HDFS's default data placement policy in a heterogeneous environment is excessive data traffic, as high-performance nodes can finish processing their local data faster than low-performance nodes, requiring transportation of unprocessed data from slower nodes to faster nodes. This data movement between nodes will use the cluster's network bandwidth, which may impact Hadoop MapReduce performance. The default data placement policy also uses a constant replication factor for all data blocks. Since most of this data is actually cold data and has a low access throughput, adopting a fixed replication factor for all data without considering the access rate or hotness of this data will result in significant waste of disk storage resources. However, regardless of the frequency of data access, availability remains critical for any data center. In this context, several research studies [11-15] have proposed placement strategies that dynamically distribute data blocks based on their hotness and have also proposed dynamic replication mechanisms to reduce storage space consumption. However, it should be noted that these works have not set limits on data replication. Thus, a very large factor may have an effect on storage efficiency, and a too small factor < 2 may have an impact on data availability on the cluster.

To address the above shortcomings, we propose a smart data placement strategy in heterogeneous Hadoop clusters (SDPS) that allows data distribution across nodes based on their hotness, such that high-access data is placed on the highest-performing nodes. According to the experiment results, SDPS improved MapReduce execution time by more than 23% on average compared to the default HDFS policy and data locality by more than 9%. We also propose a dynamic replication strategy that aims to increase storage efficiency by reducing storage space consumption. Not to mention that the proposed model does not require any prior setting. Our work can be summarized as follows:

- DataNodes Clustering Algorithm (DCA) that groups nodes according to their performance into Virtual Racks (VR) using K-means clustering.
- Hotness-Aware Block Replication algorithm (HABR), which allows you to specify the hotness of data blocks and then assign a replication number to each of those blocks based on their hotness.
- Smart Data Placement Strategy (SDPS) that uses DCA to divide nodes into VRs and HABR to determine the hotness and the replication factors of data blocks and then distribute them across DataNodes in the cluster, with the condition of placing hot data blocks on high-performance nodes.

The rest of the document is organized as follows. Section 2 reveals the background and some details regarding Hadoop and HDFS. Section 3 deals with related work. Section 4 details the proposed SDDP and its relevance algorithms. Section 5 evaluates the performance of SDPS in a heterogeneous cluster and discusses the results. Section 6 concludes the paper and highlights some future work.

2. Background

In this section, we give an overview of the Hadoop framework and some details about the HDFS system and its default block placement policy.

2.1. Hadoop

Apache Hadoop is primarily an open-source software framework for distributed data processing and management based on Java, which can run on different environments such as clouds or data centers. Hadoop uses a set of machines configured in computing clusters to process and store huge amounts of datasets. It allows distributed processing of Hadoop analytics and Big Data tasks at the cluster level by breaking them down into smaller tasks that can be processed in parallel mode. Hadoop comprises a set of components with various functionalities. HDFS, as the Hadoop file system,

supports the management of datasets and metadata stored in the cluster. Then, Hadoop MapReduce, which is the distributed data processing module. Finally, YARN [16] distributes the workloads across the cluster. The Hadoop architecture is described in Figure 1.

A Hadoop cluster is a special type of computing cluster (server cluster) designed specifically to store and process large volumes of data, which promotes distributed computing. The work of analyzing the data is distributed across the cluster nodes (servers). These clusters are used to run the open-source distributed computing software Hadoop on low-cost computers. Typically, one machine in the cluster is designated as the NameNode, and another machine is designated as the JobTracker. These two machines are the masters. The other machines in the cluster are worker nodes and act as both DataNodes and TaskTrackers.

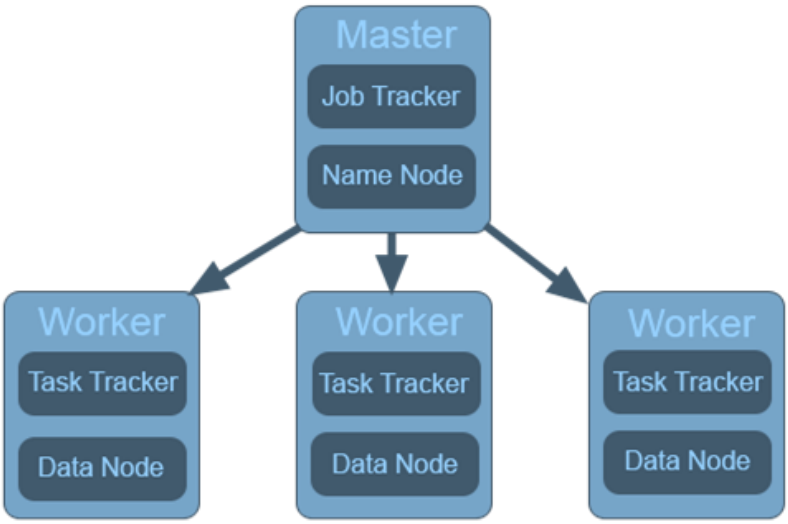


Figure 1. Hadoop architecture

The introduction of Hadoop enabled businesses to quickly benefit from massive data storage and processing capacity, higher computing power, modules with many options, better fault tolerance, more flexible data management, reduced costs compared to traditional warehouses, high scalability, and extensibility. Finally, Hadoop has played a role in the emergence of other big data analytics tools, such as the arrival of Apache Spark.

2.2. HDFS

Hadoop Distributed File System (HDFS) is one of the core components of the Hadoop framework, responsible for storing and managing large volumes of data across all nodes that form a common cluster database. The data in this system is stored and replicated across multiple nodes, making it fault-tolerant. This redundancy can also ensure high data availability at the server level. HDFS is also characterized by its flexibility, high scalability, ease of use, and integration into the entire Hadoop ecosystem software suite. Figure 2 describes the architecture of HDFS.

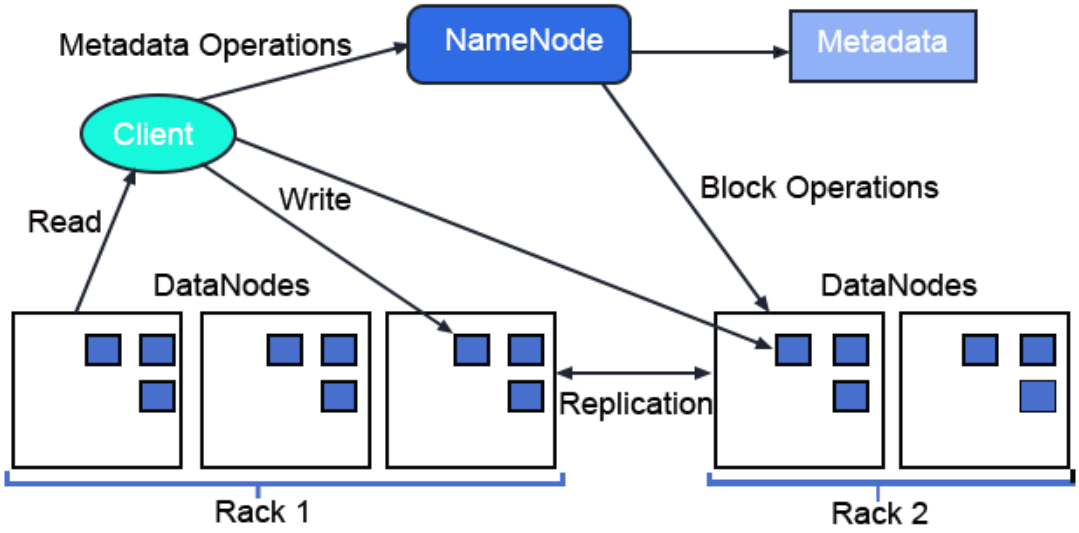


Figure 2. Architecture of Hadoop Distributed File System (HDFS)

As shown in Figure 2, HDFS consists of two components:

- **NameNode:** This node is the master demon (main program); it is the conductor of the HDFS cluster, managing the file system metadata, including information about files, directories, access permissions, data blocks, and block locations. The NameNode also maintains information about the DataNodes in the cluster and monitors their availability. Clients access the HDFS file system through the NameNode, which coordinates file read and write operations.
- **DataNode:** DataNode is the slave daemon (secondary program). These nodes are responsible for storing the actual file data in the HDFS file system. DataNodes periodically communicate with the NameNode to send information about their status and to receive instructions on how to store, read, or delete data. Depending on storage needs, DataNodes can be dynamically added or removed from the cluster.

Clients can perform data read/write operations on HDFS using its file system interface. To access the desired data, clients can request its location by connecting to NameNode, which will respond by proposing the addresses of the nodes containing the data blocks, and hence clients can read or write the data directly by connecting to these nodes. Random file access is also available on HDFS, and clients can read or write data at any offset in the file.

2.3. The Default HDFS Block Placement

When uploading a file to HDFS, it is divided into data blocks of a predefined size in HDFS, usually 128 or 256 MB blocks, and then distributed across the cluster nodes. To ensure system reliability and data availability, each block is replicated using a replication factor; the default factor of three adopted by HDFS means that each block is available on three different nodes.

The default HDFS block placement strategy randomly assigns the first replica of a data block to any node with enough space on the cluster. It then places the second copy on any node in a different rack than the first replica, but if there are no available racks, it can be hosted on the same rack as the first replica. And finally, it allocates the third replica on any node in the same rack as the second copy.

Whenever a data node goes down or fails, the node name instructs the data nodes hosting replicas of lost data blocks to redo the replication and placement of blocks on other nodes to ensure availability of data files by reaching their assigned replication factor again.

3. Related Works

Data placement policies occupy a fairly important place in scientific research, considering their importance and impact on MapReduce performance and Hadoop system efficiency. Xie et al. [17] proposed to allocate data on heterogeneous nodes according to their computing capacity. Shah et al. [6] presented an algorithm for balancing data across nodes by dividing them into two categories according to a specific criterion. Vengadeswaran et al. [18] addressed the data locality issue through a data placement mechanism based on data interdependence. A workload-driven approach [19] aimed at reducing timespan by co-locating frequently accessed data elements between queries. SLDP proposed by Xiong et al. [11] divides nodes into several virtual tiers based on their performance and then distributes data blocks across these tiers based on their hotness. Lui et al. [12] used a gray forecast model to predict the heat and replica number of data blocks, then placed them on the appropriate nodes. Wu et al. [20] proposed DGAD data placement according to task execution frequency to improve data locality. Xiong et al. [13] used a heat-aware data clustering to specify the heat of the data, Double Sort Exchange to allocate cold data, and a dynamic replication placement mechanism to place hot data. CoHadoop [14] aimed to co-locate data processed by a job in the same node. CoHadoop added a file-level property (locator) to extend HDFS, and files placed on the same set of DataNodes have the same locator. Qureshi et al. [21] proposed a data placement called RDP to address unbalanced workloads and network traffic to improve Hadoop performance. Lee et al. [7] introduced a data placement algorithm to balance workloads across nodes according to their computing capacity to reduce data transfer time to improve Hadoop system efficiency. Bae et al. [10] proposed a new data placement policy that aims to improve data locality with a minimal amount of replicated data by replicating only the data blocks that have the highest access rate. Hussain et al. [15] addressed performance degradation by placing block data on the highest-performing nodes. Liu et al. [22] presented a replica placement policy based on the TOPSIS entropy weighting method to calculate the efficiency scores of each node, rack, and cluster. Then, block placement is performed based on the obtained scores. Vengadeswaran et al. [23] proposed a data placement (CLUST) based on grouping semantics in data to distribute blocks optimally across nodes.

Managing data replication can help improve storage efficiency while maintaining enough availability. Liu et al. [24] presented BRPS, a Big Data replica placement strategy reducing data movement across the cluster to improve task processing performance. Ciritoglu et al. [25] proposed a workload-aware balanced replica deletion algorithm to handle imbalanced data, hot spots, and performance degradation. Ciritoglu et al. [26] presented HARD, a heterogeneity-aware replica deletion, which is an extension of the algorithm in [25] to deal with the replica deletion issue in a heterogeneous

environment. Dai et al. [27] suggested a new replica placement policy to distribute replicas of data across nodes without resorting to a load balancer. Bui et al. [28] proposed an approach to dynamically replicate data based on popularity, with an erasure code to ensure reliability for low-popularity data. Ahmed et al. [29] presented a replication policy to group data files based on their importance in various clusters and apply an appropriate replication policy to each cluster to reduce storage waste while maintaining data availability and reliability. Fazul et al. [30] proposed a metric observation model that automatically determines when to start corrective action and triggers the reactive balancing process in the file system, based on standardized trigger events. A. Zayed et al. [31] presented an optimization of the HDFS replication policy using predictive categorization to minimize storage consumption while ensuring data availability and system reliability. He et al. [32] addressed the shortcoming of static replication by adopting a dynamic decision strategy for the number of replicas based on data popularity. He et al. [33] proposed a copy placement strategy based on evaluation value and load balancing to improve the performance of cloud storage systems.

Most previous research that addressed the issue of cluster heterogeneity relied on one or at most two performance criteria for node clustering to perform data distribution. However, in our case, the proposed model considers multiple performance criteria when clustering nodes (CPU speed, memory size, disk capacity, disk IOPS, etc.) to further refine the classification. Moreover, and unlike previous works that proposed dynamic data block replication strategies, our model imposes limits on the replication operation in order to reduce storage space consumption while maintaining data availability.

4. Proposed Model Design

This section presents in detail the design of the proposed SDPS model and its auxiliary algorithms, which aim at the efficient utilization of the cluster resources and better performance of Hadoop MapReduce. Figure 3 illustrates the implementation of SDPS in a heterogeneous Hadoop cluster. The model architecture includes 3 components: the DataNodes Clustering Algorithm (DCA), which is responsible for dividing DataNodes into groups (VRs); the Heat-Aware Block Replication (HABR) algorithm, which determines the heat and number of replications of data blocks; and SDPS, which relies on the information provided by DCA and HABR to place data blocks on appropriate DataNodes.

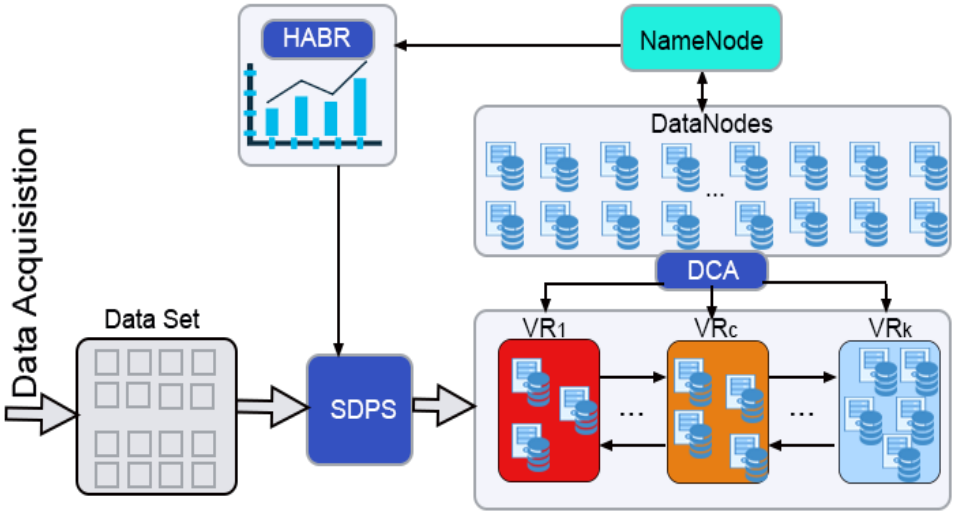


Figure 3. Implementation of SDPS in a heterogeneous Hadoop cluster

4.1. DataNodes Clustering Algorithm (DCA)

Cluster heterogeneity presents an additional constraint on the overall performance of Hadoop MapReduce, especially if there were sincere differences in the performance of the DataNodes in the cluster. and this performance lag between nodes can result in significant data transfer between fast nodes and slower nodes, which can impact data processing time and limit overall system throughput. This requires solutions that can overcome these limitations, increase the system's performance and efficiency, and make the most of its capabilities.

In this context, we present our solution for this problem, which is called DataNodes Clustering Algorithm (DCA), which aims to classify DataNodes according to several performance criteria (processor speed, memory capacity, storage, IOPS, ...) using the K-Means method, the idea is to partition the machines that have close capacities into groups or what we will call Virtual Racks (VR). In another sense, to achieve node clustering in VRs (Virtual Racks), we perform comparisons between nodes based on the performance parameters mentioned above, so that nodes grouped in the same VR (Virtual Rack) will have almost the same performance. The number of VRs depends on the degree of variance between nodes in the cluster.

The K-means algorithm belongs to the family of unsupervised machine learning algorithms, used to analyze and classify a data set in order to group "similar" data into groups (or clusters). Given a number K in advance, the algorithm will split the dataset into K clusters, and the goal is to find the best cluster where each cluster has the data points closest to each other. Here, the DataNodes will be the datasets of this algorithm which are denoted $DN = \{dn_1, \ldots, dn_n\}$, and the set of performance criteria are denoted $CP = \{cp_1, \ldots, cp_m\}$. The relation between a DataNode dn_i and a performance criterion cp_j is denoted $x_{i,j} = \mathrm{d}n_i \times cp_j$, and thus the observation matrix X is as follows:

$$X = \begin{pmatrix} x_{1,1} & x_{1,2} & \ldots & x_{1,m} \\ x_{2,1} & x_{2,2} & \ldots & x_{2,m} \\ \ldots & \ldots & \ldots & \ldots \\ x_{n,1} & x_{n,2} & \ldots & x_{n,m} \end{pmatrix} \quad (1)$$

To classify the datasets into K clusters, the algorithm will seek to compare the degree of proximity or similarity between the data points, this process is repeated several times until having a good classification. The K-means algorithm typically involves the Euclidean distance in the proximity comparison. In our case to test the degree of similarity between two DataNodes dn_i and dn_j, we calculate the distance $d_{i,j}$ between the two elements with this formula:

$$d_{i,j} = \sqrt{\sum_{u=1}^{m} \left(x_{i,u} - x_{j,u}\right)^2} \quad (2)$$

where,

- $x_{i,u}$: relationship between node dn_i and performance parameter u.
- $x_{j,u}$: relationship between node dn_j and performance parameter u.
- m: the number of performance parameters.

We will start by manually specifying the value of K the number of groups (VRs) that will be generated by the algorithm, and follow these steps:

Step 1: Select K Data Nodes or centroids randomly.

Step 2: Assign each DataNode to its nearest centroid to build the K predefined groups. (The nearest in terms of similarity which can be calculated using Equation 2).

Step 3: Calculate a new centroid of each group, for example for a group of DataNodes VR_a, the new centroid can be calculated as follows:

$$\forall_{j \in 1,\ldots,m} \; x_{c,j} = \frac{1}{|VR_q|} \sum_{dn_i \in VR_q} x_{i,j} \quad (3)$$

where,

- $x_{c,j}$: the relationship between the new centroid c and the performance parameter j.
- $x_{i,j}$: the relationship between node dn_i and performance parameter j.
- $|VR_q|$: the number of DataNodes in VR_q.
- m: the number performance criteria.

Step 4: We repeat the steps 2 and 4 until the new centroids are stable or no reassignment occurs.

Step 5: The model is ready and the classification of the DataNodes is done.

Note: It is not always obvious to choose the number of clusters K. Especially for a large dataset where we do not have any assumptions or priors about the data. The most common way to choose the number of clusters is to run the K-Means algorithm each time with a new value of K in an attempt to find the optimal value of K.

4.2. Hotness Aware Blocks Replication (HABR)

Data stored in a cluster typically does not have the same access rate, it does not maintain a constant access rate and can vary from time to time. For most data, we see a significant decrease in its access rate over time, and sometimes we find data that has not been accessed for a long time (Cold Data) or is rarely accessed in a certain period of time (warm data), which makes us question the usefulness of adopting a constant replication factor for all data in the HDFS system, especially with this variation between data in terms of access frequency and with the existence of a significant amount of cold data and sometimes warm data, while the hot data represents a lower percentage. Which can lead to wasted storage space at the cluster level.

Analyzing the access logs provided by NameNode that record information about all the requests performed, such as the date and time of the request, allows to obtain the frequency of access to data blocks and thus to determine the heat level of these blocks. In this regard, we propose Hotness Aware Block Replication (HABR) algorithm, which aims to determine the number of replications of data blocks according to their hotness using a dynamic data replication strategy [34], which is a method to dynamically determine the replication factor based on data popularity to optimize storage on HDFS and reduce space waste. For frequently accessed data (hot data), a large replication factor will reduce network utilization and improve application execution, while for infrequently accessed data (cold data), a small replication factor can reduce storage space consumption while avoiding performance degradation.

This method determines the replication factor by using the historical access frequency. to express the effect of historical information on the popularity of the object in the recent period, in many fields, the half-life is taken into account. The half-life is the period of time during which a substance undergoes decay to half of its concentration [34]. In our algorithm, we also take the half-life as the weight of the historical records, which means that the weight will change over time and will be halved after a defined period of time, so the more recent the records are, the higher their weight will be.

Let AR_i^n be the sum of all access rates of data block i over the entire cluster during the time interval T_n, n the number of time periods that have elapsed, and Pl_i^n be the popularity of data block i for the whole cluster during the time interval T_n, calculated by

$$\boldsymbol{Pl_i^n = \sum_{j=1}^{n} AR_i^j \times 2^{j-n}} \tag{4}$$

we will consider that the hotness of a data block i to be its average popularity per time interval T_n, denoted by $h(b_i)$, given by

$$\boldsymbol{h(b_i) = \frac{Pl_i^n}{n}} \tag{5}$$

where n denotes the periods of time that have elapsed, we can then determine the replication factor of data block i, denoted by rf(b_i), based on its hotness value $h(b_i)$, by

$$\mathbf{rf}(\boldsymbol{b_i}) = \begin{cases} \mathbf{4}, & \boldsymbol{h(b_i)/\bar{H} \geq 2} \\ \mathbf{3}, & \boldsymbol{1 \leq h(b_i)/\bar{H} < 2} \\ \mathbf{2}, & \boldsymbol{h(b_i)/\bar{H} < 1} \end{cases} \tag{6}$$

where $\bar{H}$ is the average of hotness values of all data blocks.

To conclude, HABR using the above equations will measure the hotness of data blocks and determine their replication factors, which can provide useful information for the next operation of placing data blocks on the appropriate nodes.

4.3. Smart Data Placement Strategy (SDPS)

Based on the information obtained from the two subsections, such as the set of VRs where each VR collects a number of DataNodes with almost similar performance characteristics, the hotness and replication factor of each block data, we design a smart data placement strategy (SDPS) for heterogeneous Hadoop environments, which enables placing hot data on high-performance nodes, its pseudocode is illustrated in Algorithm 1, and its main steps are as follows:

Step 1: Using DCA algorithm, SDPS groups DataNodes into K VRs, each VR is a set of DataNodes with almost the same performance. DCA takes as parameters the set of DataNodes and the initial number K of VRs.

Step 2: SDPS sorts the data blocks according to their hotness obtained by the HABR algorithm, and then it specifies the replication factor of each data block based on its own hotness values using Equation 6.

Step 3: SDPS sorts the DataNodes contained in each VR_C according to several performance parameters and calculates the overall storage space S_C of each VR_C, by summing the storage space of the DataNodes in that VR.

Step 4: SDPS redistributes the data blocks across the DataNodes, so that the data block with the highest hotness value will be assigned to the first highest performing DataNode, and the second hottest block is assigned to the second-best performing DataNode, and so on, after checking whether it still has enough storage space and whether it does not contain a copy of that data block to ensure data reliability. Each time a block is assigned, the overall remaining storage space S_C of the VR_C is recalculated.

The SDPS algorithm, by placing high-access data on the most performing DataNodes, ensures efficient utilization of cluster capacity and improves the execution time of workloads on Hadoop MapReduce by reducing data traffic between cluster nodes and network bandwidth usage. SDPS can also improve data locality by applying dynamic data replication and placement while ensuring data availability.

Algorithm 1. Smart Data Placement Strategy

```
Input: DN set of n DataNodes, B set of m data blocks
Output: PM[n][m] data blocks placement matrix
VR = {VR_c | 1 ≤ c ≤ K} ← DCA(DN, K);
H = {h(b_i) | 1 ≤ i ≤ m} ← get the hotness of data blocks;
B* = {b_i | 1 ≤ i ≤ m} ← apply descending sorting on data blocks based on their hotness;
RF = {rf(b_i) | 1 ≤ i ≤ m} ← specify the replication factor of each data block according to its hotness.
for c = 1 → K do
    VR*_c = {dn_j | j ≥ 1} ← sort all DataNodes in VR_c by multiple performance parameters;
    S_c ← calculate the total storage space in VR_c;
end for
□ ← 128; // default block size
for i = 1 → m do
    for l = 1 → rf(b_i) do
        for c = 1 → K do
            if S_c ≥ □ then
                for each dn_j from VR_c do
                    if dn_j still has enough space then
                        if b_i is not in dn_l then
                            PM [j][ i] ← 1; // assign the replica of the data block i of node j
                            S_c ← S_c − □; // recalculate remaining space in VR_c
                        end if
                    end if
                end for
            end if
        end for
    end for
end for
return PM[n][m];
```

5. Performance Evaluation

In this section, we performed a set of experiments to evaluate the effectiveness of SDPS and the other sub-algorithms on a heterogeneous Hadoop cluster. We begin by clarifying the experimental environment and then discuss the results obtained.

5.1. Experimental Setup

The experimental setup is a heterogeneous Hadoop cluster consisting of 1 Master Node and 29 Data Nodes contained in the same physical rack. Data nodes come in 4 different configuration types. The cluster configuration is detailed in Table 1. The work was realized on Hadoop 3.3.1; the data block size in the HDFS system was set to 128 MB, and each worker node in the cluster has 2 map slots and 2 reduce slots.

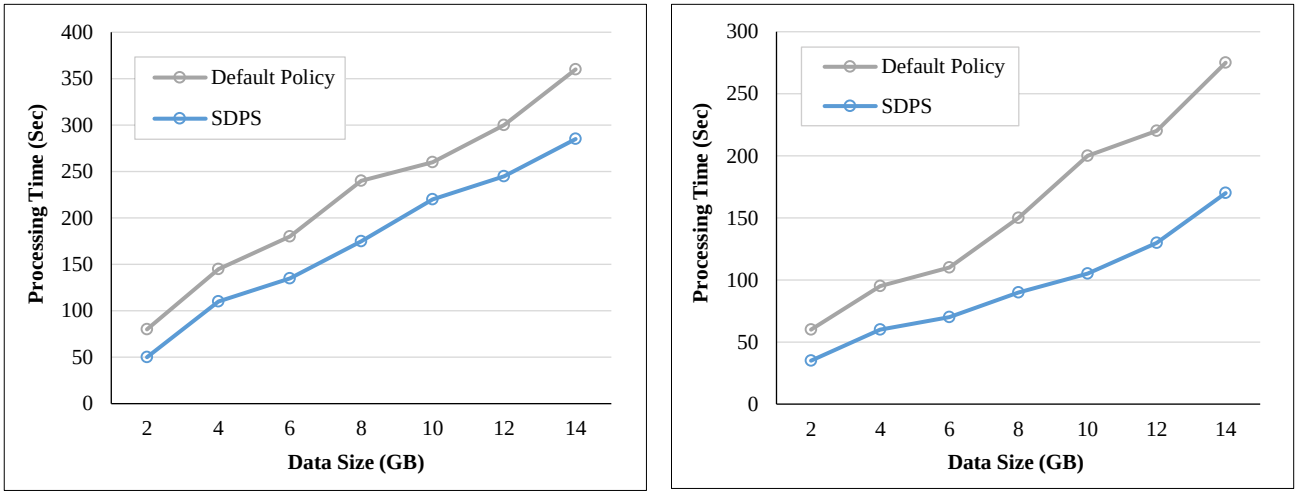
For the experimental workloads, we used usual MapReduce applications (WordCount and Grep) with different dataset sizes ranging from 2 GB to 14 GB. The experiments consisted of a series of tests comparing our placement model to the default HDFS policy. The results of each experiment are averages of 10 to 15 runs for each application with each dataset size in order to obtain more accurate values. The total number of runs is about 600 for all experiments.

Table 1. Cluster Configuration

	CPU			RAM	Disk		
	Model	Speed	Cores		Type	IOPS	Size
1 Master Node / 6 Data Nodes	Xeon E5-2670	2.6 GHz	8	32 GB	5.4k SATA	80	1 TB
7 Data Nodes	Xeon L5640	2.26 GHz	6	16 GB	7.2k SAS	95	1 TB
9 Data Nodes	Xeon E-2314	2.8 GHz	4	16 GB	7.2k SAS	90	500 GB
7 Data Nodes	Xeon E5-2603	1.8 GHz	4	8 GB	10K SAS	110	300 GB

5.2. Evaluation Results

The default data placement policy of HDFS has shown its limitations, especially in heterogeneous clusters, and this has a relationship with data locality, which remains one of the factors that determine MapReduce performance. We attempted to compare the performance of MapReduce under the proposed SDPS and the default placement policy of HDFS in order to evaluate the performance of SDPS and its auxiliary algorithms. In this comparison, we ran two usual applications (Grep and WordCount) on datasets of different sizes. For the default data placement strategy of HDFS, we set the default data replication factor to 3. The comparison results are shown in Figure 4.



(a) Processing time of WordCount **(b) Processing time of GREP**

Figure 4. Comparison of processing time between SDPS and default policy

From Figure 4, we can clearly notice that MapReduce performed better with SDPS compared to the default data placement policy, such that the processing time improved by more than 23% for the WordCount application and more than 40% for the Grep application. This can be explained by the good distribution of data across the cluster, by placing hot data on the best performing nodes, which reduces network usage to transmit data between nodes and therefore improves data processing on Hadoop MapReduce. In other words, placing high-frequency data blocks on high-performance nodes, which have the capacity to process this data in much less time than the lowest-performing nodes, will allow better utilization of the cluster capacity and reduce the data transfer between nodes that requires network bandwidth usage at the cluster level, which implies avoiding an additional time due to network usage. These facts can have an impact on the processing time of applications by MapReduce. Thus, the runtime improvement achieved by SDPS, as shown in Figure 4, is due to its data placement strategy across nodes.

Given the importance of data locality, especially in a heterogeneous system, and to evaluate the impact of our SDPS model on the data locality rate compared to the default policy across the cluster, we performed a cluster-wide comparison of data locality when running the WordCount and Grep applications with both data placement strategies. The results are shown in Figure 5.

We can observe from Figure 5 that there is a variation in the data locality rate between the two strategies; we see that the data locality is around 46% in the default policy, while it is more than 75% in SDPS. This means that by placing high-frequency access data on high-performance nodes, the proposed SDPS has managed to increase the data locality on the cluster by more than 29%. The availability of hot data on the best-performing nodes gives them a high probability of processing local tasks, which explains the increase in the data locality rate for the SDPS compared to the default

policy. Furthermore, we noticed that the hot data rate does not exceed 25% (of which the warm data is 19% and 6% is the intense hot data), while the remaining 75% is cold data. Thus, for our SDPS model, which adopts a dynamic replication strategy, the replication factor for the 6% that are intense hot data is 4, for the 19% that are warm data, a number of replicas of 3, and a factor of 2 for the 75% that are cold data. Thus, the overall data replication ratio will be 4×6% + 3×19% + 2×75% = 231% applied on a given dataset of 400GB gives 924GB of data. While in the default policy, which adopts a fixed replication factor that is 3 by default, we get 3×100% = 300%×400GB = 1200GB of data. which means we could save more than 29% of storage space by using this dynamic replication model, which will surely contribute to efficient storage space management across the entire cluster.

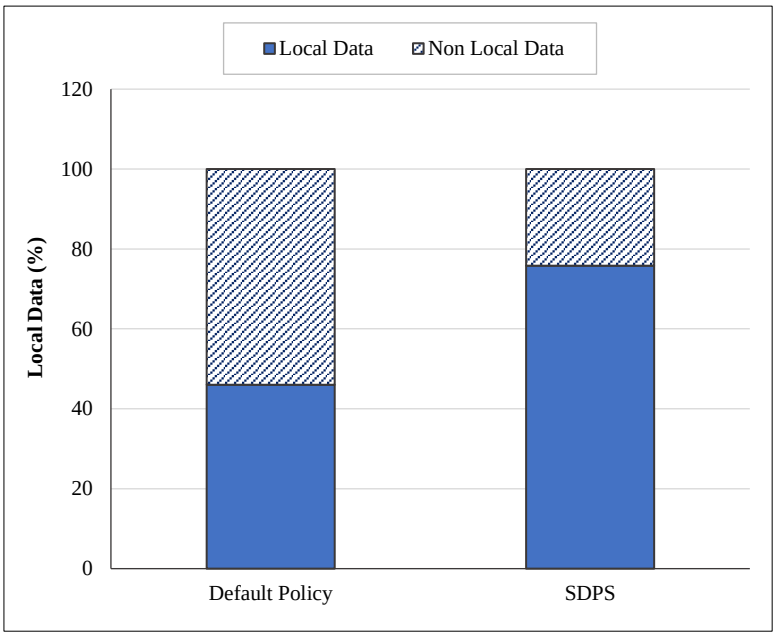


Figure 5. Comparison of Data Locality between SDPS and default policy

To summarize this section, the SDPS model has performed better than the default HDFS policy according to the experiments, whether it is its impact on application execution time or on data locality across the cluster. This means that SDPS was able to enhance data balance across nodes while taking into account node performance and data hotness when placing data. Which can improve the overall performance of Hadoop MapReduce. SDPS flexibly adapts to cluster variance, so the higher the cluster variance, the more virtual racks (VRs) are built, making it suitable for even the most complex heterogeneous environments. It is also scalable with the extension of the system, as new nodes are added to the cluster; the set of VRs will be rebuilt to accommodate the new nodes and refine the classification. Moreover, adopting a dynamic data replication strategy, which determines the number of data replicas based on the hotness factor or data access frequency, has shown its storage efficiency compared with the HDFS static replication strategy that adopts a fixed replication factor. Thus, SDPS can help reduce storage space consumption and make the system storage more efficient.

6. Conclusion

The default data placement of HDFS has some limitations, especially when applied in a heterogeneous environment, and may lead to MapReduce performance degradation. HDFS also adopts a constant replication factor for all data blocks, which may increase storage space consumption, especially in the case of huge amounts of cold data. In this paper, we proposed a smart data placement strategy (SDPS) to address these limitations. SDPS divides nodes based on their performance into homogeneous groups (VRs) using the DataNodes Clustering Algorithm (DCA) and determines the hotness and replication factors of data blocks using the hotness-aware block replication (HABR) algorithm, and then SDPS places each block into the appropriate DataNode. As proven by the experiments, SDPS performed better than the default strategy in data distribution, which helped improve MapReduce performance and cluster-level data locality. Additionally, adopting a flexible replication factor based on data hotness can increase storage efficiency.

As part of our future work, we aspire to extend our approach by adding an energy-saving component. Since the SDPS data placement policy takes into account the temperature criterion when distributing data to heterogeneous nodes, it leaves the possibility of having idle nodes, especially those that contain cold data. Therefore, designing a mechanism to put these inactive nodes into hibernation mode, or to put others into standby mode depending on the load, will save power in the cluster and improve energy efficiency without affecting system performance. We also plan to test our model in a broader environment.

7. Declarations

7.1. Author Contributions

Conceptualization, N.E.B. and I.A.; methodology, N.E.B. and I.A.; software, N.E.B.; validation, N.E.B. and I.A.; formal analysis, N.E.B.; investigation, N.E.B.; resources, N.E.B.; data curation, N.E.B.; writing—original draft preparation N.E.B.; writing—review and editing, N.E.B. and I.A.; visualization, N.E.B.; supervision, I.A.; project administration, I.A.; funding acquisition, N.E.B. 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] Sagiroglu, S., & Sinanc, D. (2013). Big data: A review. Proceedings of the 2013 International Conference on Collaboration Technologies and Systems, CTS 2013, 42–47. doi:10.1109/CTS.2013.6567202.

[2] Gong, C., Liu, J., Zhang, Q., Chen, H., & Gong, Z. (2010). The characteristics of cloud computing. Proceedings of the International Conference on Parallel Processing Workshops, 275–279. doi:10.1109/ICPPW.2010.45.

[3] White, T. (2012). Hadoop: The Definitive Guide. O'Reilly Media, California, United States.

[4] Khezr, S. N., & Navimipour, N. J. (2017). MapReduce and Its Applications, Challenges, and Architecture: a Comprehensive Review and Directions for Future Research. Journal of Grid Computing, 15(3), 295–321. doi:10.1007/s10723-017-9408-0.

[5] Dev, D., & Patgiri, R. (2015). Performance evaluation of HDFS in big data management. 2014 International Conference on High Performance Computing and Applications, ICHPCA 2014, 9, 1–7. doi:10.1109/ICHPCA.2014.7045330.

[6] Shah, A., & Padole, M. (2018). Load Balancing through Block Rearrangement Policy for Hadoop Heterogeneous Cluster. 2018 International Conference on Advances in Computing, Communications and Informatics, ICACCI 2018, 230–236. doi:10.1109/ICACCI.2018.8554404.

[7] Lee, C. W., Hsieh, K. Y., Hsieh, S. Y., & Hsiao, H. C. (2014). A Dynamic Data Placement Strategy for Hadoop in Heterogeneous Environments. Big Data Research, 1, 14–22. doi:10.1016/j.bdr.2014.07.002.

[8] Reddy, K. H. K., Pandey, V., & Roy, D. S. (2019). A novel entropy-based dynamic data placement strategy for data intensive applications in Hadoop clusters. International Journal of Big Data Intelligence, 6(1), 20. doi:10.1504/ijbdi.2019.097395.

[9] Shithil, S. M., Saha, T. K., & Sharma, T. (2017). A dynamic data placement policy for heterogeneous Hadoop cluster. 4th International Conference on Advances in Electrical Engineering, ICAEE 2017, 302–307. doi:10.1109/ICAEE.2017.8255371.

[10] Bae, M., Yeo, S., Park, G., & Oh, S. (2021). Novel data-placement scheme for improving the data locality of Hadoop in heterogeneous environments. Concurrency and Computation: Practice and Experience, 33(18), 5752. doi:10.1002/cpe.5752.

[11] Xiong, R., Luo, J., & Dong, F. (2015). SLDP: A Novel Data Placement Strategy for Large-Scale Heterogeneous Hadoop Cluster. Proceedings - 2014 2nd International Conference on Advanced Cloud and Big Data, CBD 2014, 158, 9–17. doi:10.1109/CBD.2014.57.

[12] Liu, Y., Wu, C. Q., Wang, M., Hou, A., & Wang, Y. (2018). On a Dynamic Data Placement Strategy for Heterogeneous Hadoop Clusters. 2018 International Symposium on Networks, Computers and Communications, ISNCC 2018, 5, 1–7. doi:10.1109/ISNCC.2018.8530970.

[13] Xiong, R., Du, Y., Jin, J., & Luo, J. (2018). HaDaap: a hotness- aware data placement strategy for improving storage efficiency in heterogeneous Hadoop clusters. Concurrency and Computation: Practice and Experience, 30(20), e4830. doi:10.1002/cpe.4830.

[14] Eltabakh, M. Y., Tian, Y., Özcan, F., Gemulla, R., Krettek, A., & McPherson, J. (2011). CoHadoop: Flexible data placement and its exploitation in Hadoop. Proceedings of the VLDB Endowment, 4(9), 575–585. doi:10.14778/2002938.2002943.

[15] Hussain, M. W., & Roy, D. S. (2022). A Counter-Based Profiling Scheme for Improving Locality through Data and Reducer Placement. Intelligent Systems Reference Library, 218, 101–118. doi:10.1007/978-981-16-8930-7_4.

[16] Vavilapalli, V. K., Murthy, A. C., Douglas, C., Agarwal, S., Konar, M., Evans, R., Graves, T., Lowe, J., Shah, H., Seth, S., Saha, B., Curino, C., O'Malley, O., Radia, S., Reed, B., & Baldeschwieler, E. (2013). Apache hadoop YARN: Yet another resource negotiator. Proceedings of the 4th Annual Symposium on Cloud Computing, SoCC 2013, 1–16. doi:10.1145/2523616.2523633.

[17] Xie, J., Yin, S., Ruan, X., Ding, Z., Tian, Y., Majors, J., Manzanares, A., & Qin, X. (2010). Improving MapReduce performance through data placement in heterogeneous Hadoop clusters. Proceedings of the 2010 IEEE International Symposium on Parallel and Distributed Processing, Workshops and Ph.D. Forum, IPDPSW 2010, 1–9. doi:10.1109/IPDPSW.2010.5470880.

[18] Vengadeswaran, S., Balasundaram, S. R., & Dhavakumar, P. (2024). IDaPS — Improved data-locality aware data placement strategy based on Markov clustering to enhance MapReduce performance on Hadoop. Journal of King Saud University - Computer and Information Sciences, 36(3), 101973. doi:10.1016/j.jksuci.2024.101973.

[19] Kumar, K. A., Deshpande, A., & Khuller, S. (2013). Data placement and replica selection for improving co-location in distributed environments. arXiv preprint, arXiv:1302.4168. doi:10.48550/arXiv.1302.4168.

[20] Wu, J. xuan, Zhang, C. sheng, Zhang, B., & Wang, P. (2016). A new data-grouping-aware dynamic data placement method that take into account jobs execute frequency for Hadoop. Microprocessors and Microsystems, 47, 161–169. doi:10.1016/j.micpro.2016.07.011.

[21] Qureshi, N. M. F., & Shin, D. R. (2016). RDP: A storage-tier-aware robust data placement strategy for hadoop in a cloud-based heterogeneous environment. KSII Transactions on Internet and Information Systems, 10(9), 4063–4086. doi:10.3837/tiis.2016.09.003.

[22] Liu, J., Xie, M., Chen, S., Xu, G., Wu, T., & Li, W. (2023). TS-REPLICA: A novel replica placement algorithm based on the entropy weight TOPSIS method in spark for multimedia data analysis. Information Sciences, 626, 133–148. doi:10.1016/j.ins.2023.01.049.

[23] Vengadeswaran, S., & Balasundaram, S. R. (2020). CLUST - Grouping aware data placement for improving the performance of large-scale data management system. ACM International Conference Proceeding Series, 1–9. doi:10.1145/3371158.3371159.

[24] Liu, L., Song, J., Wang, H., & Lv, P. (2016). BRPS: A Big Data Placement Strategy for Data Intensive Applications. IEEE International Conference on Data Mining Workshops, ICDMW, 813–820. doi:10.1109/ICDMW.2016.0120.

[25] Ciritoglu, H. E., Saber, T., Buda, T. S., Murphy, J., & Thorpe, C. (2018). Towards a Better Replica Management for Hadoop Distributed File System. Proceedings - 2018 IEEE International Congress on Big Data, BigData Congress 2018 - Part of the 2018 IEEE World Congress on Services, 104–111. doi:10.1109/BigDataCongress.2018.00021.

[26] Ciritoglu, H. E., Murphy, J., & Thorpe, C. (2019). HaRD: a heterogeneity-aware replica deletion for HDFS. Journal of Big Data, 6(1), 1-21. doi:10.1186/s40537-019-0256-6.

[27] Dai, W., Ibrahim, I., & Bassiouni, M. (2016). A New Replica Placement Policy for Hadoop Distributed File System. Proceedings - 2nd IEEE International Conference on Big Data Security on Cloud, IEEE BigDataSecurity 2016, 2nd IEEE International Conference on High Performance and Smart Computing, IEEE HPSC 2016 and IEEE International Conference on Intelligent Data and Security, IEEE IDS 2016, 262–267. doi:10.1109/BigDataSecurity-HPSC-IDS.2016.30.

[28] Bui, D. M., Hussain, S., Huh, E. N., & Lee, S. (2016). Adaptive Replication Management in HDFS Based on Supervised Learning. IEEE Transactions on Knowledge and Data Engineering, 28(6), 1369–1382. doi:10.1109/TKDE.2016.2523510.

[29] Ahmed, M. A., Khafagy, M. H., Shaheen, M. E., & Kaseb, M. R. (2023). Dynamic Replication Policy on HDFS Based on Machine Learning Clustering. IEEE Access, 11, 18551–18559. doi:10.1109/ACCESS.2023.3247190.

[30] Fazul, R. W. A., & Barcelos, P. P. (2022). An event-driven strategy for reactive replica balancing on apache hadoop distributed file system. Proceedings of the ACM Symposium on Applied Computing, 255–263. doi:10.1145/3477314.3507311.

[31] Zayed, N. A., Saleh, Y. N. M., Aboelfarag, A. A., & Shaheen, M. A. (2024). Optimizing Hadoop Distributed File System Replication Policies with Predictive Categorization. ACM International Conference Proceeding Series, 26–32. doi:10.1145/3694860.3694864.

[32] He, Q., Zhang, F., Bian, G., Zhang, W., Li, Z., & Chen, C. (2023). Dynamic decision-making strategy of replica number based on data hot. Journal of Supercomputing, 79(9), 9584–9603. doi:10.1007/s11227-022-05029-7.

[33] He, Q., Zhang, F., Bian, G., Zhang, W., Li, Z., Yu, Z., & Feng, H. (2024). File block multi-replica management technology in cloud storage. Cluster Computing, 27(1), 457–476. doi:10.1007/s10586-022-03952-1.

[34] Wang, Z., Li, T., Xiong, N., & Pan, Y. (2012). A novel dynamic network data replication scheme based on historical access record and proactive deletion. Journal of Supercomputing, 62(1), 227–250. doi:10.1007/s11227-011-0708-z.