Frequent Block Access Pattern-Based Replication Algorithm for Improving the Performance of Cloud Storage Systems

Mohammed Sharfuddin
Computer Science and Engineering, Muffakham Jah College of Engineering and Technology, Hyderabad, India
Email: sharfuddin@mjcollege.ac.in

Thirumalaisamy Ragunathan
Computer Science and Engineering, SRM, University, Amaravati, A.P., India
Email: ragunathan.t@srmap.edu.in

***Abstract*—The data replication is a crucial process to increase the performance of distributed systems by replicating multiple replicas of same data at different sites. With the evolution of the cloud storage, many people are transferring their crucial data to the cloud storage. Replication is used in cloud storage systems to improve availability, replication factor, access time and storage efficiency. Replication factor of a file refers to the number of replicas or copies of that file in the system. Every file in HDFS is stored in the form of blocks. HDFS has a default replication factor of 3 for all file blocks. In our proposed strategy a novel replication algorithm is proposed it decides the replication factor based on the support values of the file block then we will find the Frequent Block Access Pattern of the popular files and the placement of the frequent access pattern of the file blocks is based on the local support values at each data nodes. According to the results of the performance analysis, the proposed strategy's algorithm outperforms the Hadoop distributed file system's replication process.**

***Index Terms*—cloud storage systems, distributed file systems, HDFS, replication, replication factor, access time, storage cost**

I. Introduction

A. The Cloud Storage System

Cloud Storage System (CSS) is a data storage model that consists of multiple geographically distributed servers or resources, providing a large storage capacity which the users can access at a low cost. These resources work together to provide single storage cloud. It allows users to store their files online so that they can access them from anywhere via the Internet [1]. Cloud storage provides benefits which include reduced costs, simplified IT management, improved user experience, disaster recovery.

To enhance the availability of data stored in cloud storage, a technique called data replication is widely used. When a user accesses a file or data stored on CSS, a replica of this file stored at a node closest to the user is returned thus ensuring less data access time.

B. Data Replication

Replication is a concept in cloud computing that allows many copies or replicas of a data to be stored at several sites. Since it allows creation of many copies of data that can be stored at several sites, in case of crash of one site where a potential replica is stored, there are backups available for use. This distinctive property thereby ensures security of data. Replication enables the users to access their data without any dependence on the network infrastructure and without having to worry about the network traffic. It also ensures that other users are given exclusive access to their desired data without any conflicts. Replications have many benefits which include disaster recovery, high availability and low access time among many others. Replication also has its fair share of limitations or drawbacks such as cost of storage space and the time that is required to update all the replicas during which these replicas become inaccessible. The possibility over striking a balance between the disadvantages and advantages for the correct number of replicas is still debatable. Many modern large-scale storage systems have the replication factor set to 3 by default. Hadoop for instance is one such storage system where 3 copies of a block of a file to be stored on the Hadoop File System are made and stored at different Data Nodes.

The cloud-based replication is critical to the seamless operation of business processes and systems. It also enables businesses to scale to meet their rising data needs as they expand across platforms, regions, and other areas without sacrificing performance.

Replication factor of a block refers to the number of copies of that file block in the system. The merits of replication include increased availability of data, improved reliability of CSS, improve access latency to cloud, balanced load of CSS nodes, and improved fault tolerance of CSS. Replication algorithms fall under two

Manuscript received May 6, 2022; revised August 2, 2022; accepted August 29, 2022.

types static replication algorithms and dynamic replication algorithms. Static algorithms are those in which the replication strategy is predefined for any application and the number of replicas and the location of these replicas are fixed The replication of the data is decided when the data is created. In dynamic replication algorithms, the replication strategy can adapt to changes i.e., depending on the changing access patterns of the users the number of replicas can vary and the location of replicas vary depending on the node capacity. The replication of the data changes during its life time in the system.

Data availability is one of the most important features of any cloud storage system since it ensures that data is accessible at all times. Data replication is a typical method used by service providers to increase data availability and improve cloud performance.

Research Motivation - One of the most convenient and efficient ways to save data online is through cloud storage. There are a plethora of storage service providers on the internet, and the market is so large that every major IT company now has its own storage facility, which helps to make a sizable income margin from consumers. Instead of putting data on a local storage device like a hard disc, the user saves it on a remote server that can be accessed through an internet connection. Various cloud storage service companies supply storage services in a variety of price ranges.

Data access can also be sped up with the use of a replica, especially in businesses with a large number of locations. When accessing data from North American data centers, users in Asia and EUROPE may experience latency. Bringing a copy of the data closer to the user can speed up access and balance network load.

Server speed can also be improved and optimized with replicated data. Users can get data more quickly when organizations run several replicates on multiple servers. Administrators can also save processing cycles on the primary server by diverting all read operations to a replica, allowing them to focus on more resource-intensive writing operations.

The drawback of replication is excess storage consumption and expense. These drawbacks can be overcome by solving three main challenges.

There are a few challenges when choosing a replication strategy are:

1) What to replicate: The first main issue is deciding what data to replicate. If a wrong file is replicated it will contribute to unnecessary storage space while on the other hand if a popular or important data is left out from being replicated, then it will lead to the unavailability of the data. Hence the first major decision to be taken is to decide which data should be replicated.
2) How many replicas: Once the data to replicate is decided, the next step is to decide how many replicas of the data has to be replicated. Again, making too many copies of unimportant data will lead to unnecessary storage cost while fewer copies will lead to unavailability issues. So, deciding the right number of replicas is crucial to any replication strategy. Hadoop by default makes 3 copies of any data.
3) Where to place them: Generally, when a user requests a file the Name node returns the blocks of the file from the Data nodes closest to the user. So, deciding on which Data nodes to place the block copies is essential to ensuring that the users get their data in less time and easily.

The rest of the paper is organized as follows:

In Section II we discuss related work, Section III we have discussed proposed replication algorithm, Section IV we have given performance evaluation and Section V is conclusion.

II. Related Work

The phrase “replication” refers to the technique of maintaining identical copies of data across various platforms. Replication must be one of the design requirements of any Distributed File System (DFS) [2]. Hadoop DFS is now utilized by the majority of academia and industry for data storage and processing. The Hadoop Distributed File System (HDFS) is a storage and processing system for massive volumes of data. Inefficient replication is the primary cause of a file system's performance decline in HDFS. The number of Hadoop-based applications is rapidly increasing due to the system's stability and dynamic features. To provide computing with dependability, scalability, and high availability, the HDFS, which is at the heart of Apache After carefully evaluating the disadvantages of HDFS architecture, this study presents a method for dynamically replicating information files based on predictive analysis. To overcome this problem, we offer a high-availability data replication mechanism in the Hadoop framework. When the file's access frequency decreases, the recommended technique does not operate well. The number of replicas is not taken into account by this procedure

Data replication is one of the most important methods in cloud storage systems, and numerous ways have been presented [3]. The fundamental goal of data replication is to improve performance for data-intensive applications by solving some key difficulties in this category, such as availability, reliability, security, bandwidth, and data access reaction time. Despite its importance and maturity, there is no systematic, thorough, and complete assessment of cloud data replication, to the author's knowledge. This study gives a detailed assessment and classification of state-of-the-art data replication methods among several existing cloud computing platforms, using a classical classification to characterize current schemes on the topic and identify unresolved difficulties. There are three basic categories in the classification: data deduplication schemes, data auditing schemes, and data classification schemes. The author just discussed the different classification of replication not the procedure to perform the replication

With the advent of the big data era, the research focuses on how to improve the cloud storage system's

reliability, availability, and high performance [4]. In Hadoop Distributed File System, a replication placement approach based on rack-awareness is used to deal with large amounts of data storage (HDFS). The creation of load imbalance in HDFS is unavoidable if the heterogeneity of the cloud storage cluster and the load disparities of each service node are not synthetically considered. This work proposes a multi-index evaluation replication placement strategy known as MERP to address the shortcomings of HDFS' default replication placement mechanism. MERP uses a combination-weighting approach to take a holistic perspective of each data node's load characteristics, hardware performance, and network topological distance. TOPSIS model to assess potential data nodes thoroughly and choose the best one for replication placement. In terms of load balancing for distributed cloud storage clusters, the simulation results definitely indicated that our MERP surpasses HDFS's default replication placement. When the file's access frequency decreases, the recommended technique does not operate well. The number of replicas is not taken into account by this procedure.

The number of cloud users and the amount of cloud resources are crucial parameters that influence how cloud-based applications are managed [5]. As the volume of traffic to cloud-based apps grows, resource provisioning is becoming one of the most difficult issues to address in order to service time-varying and diverse workloads in the resource management scope. In this study, we present a resource provisioning strategy based on workload clustering for cloud-based applications with heterogeneous workloads. To identify cloud workloads according to their Quality of Service (QoS) requirements, we used a Biogeography-Based Optimization (BBO) technique combined with K-means clustering. In addition, we employed Bayesian learning to specify appropriate resource provisioning operations in order to meet the QoS needs of cloud-based apps. The simulation results obtained through simulation show that the proposed method is viable. The author did not discuss the details related to, what to replicate and the number of replicas is not taken into account by this procedure.

The users of cloud computing can use a variety of services on demand, including storage. At the moment, a lot of data is produced and needs a lot of storage [6]. Users have the option of storing their data offsite and accessing it online. Of course, using the cloud provides the user with the storage they desire. Data storage and retrieval take a very long time and are challenging because data accumulates. Additionally, the storage technique now in use needs to be improved for performance. Response speed, data accessibility, and migration costs are the variables that impact how well cloud storage performs. Data can therefore be repeated to other sites in order to improve these factors. The main difficulties in dynamic replication include choosing which data should be replicated, choosing how many replicas to make, deciding where to put each replica, managing the replicated data, and giving users the best replica possible. In order to address the replication difficulties and for the management of cloud storage, we plan to provide a revolutionary dynamic data replication technique utilizing the Intelligent Water Drop (IWD) algorithm. Replication takes into account the data's popularity and size. The replication and administration of storage in the cloud are optimized using the IWD algorithm, which is based on swarm intelligence. The authors, on the other hand, have not explained how to choose an appropriate number of replicas.

The traditional cloud computing paradigm has failed to scale, in particular, latency and bandwidth use have significantly increased, resulting in a reduction in Quality of Service (QoS) [7]. The data management breadth in fog computing, on the other hand, necessitates far more speed and scalability considerations. This is due to the resource constraints and heterogeneity of fog nodes while installing IoT applications. However, to the best of our knowledge, there is no literature review that categorizes these difficulties in a systematic way. We classified data replica placement approaches into four groups in this paper: framework-based, graph-based, heuristic-based, and meta-heuristic-based algorithms. To summarise, the study's main contribution is as follows: I've been reading publications about data replica location in fog. The author just discussed fog computing not the procedure to perform the replication

Data replication is a typical approach for assuring data availability. In static replication policies, the number of replicas to be created and determined statically at the time of cloud system setup [8]. To accommodate for changing trends in user demands and storage capacity, data replication must be dynamic. This prevents storage space from being squandered by keeping duplicates of data that is infrequently used. This work proposes a method for dynamically adjusting the replica factor for each data item, taking into account the data's popularity, the current replication factor, and the number of active nodes in the cloud storage. HDFS (Hadoop Distributed File System) is used to accomplish the proposed approach. The proposed technique is implemented in (HDFS), and the experimental results show that the proposed strategy maintains an ideal number of clones for each data item based on its popularity while avoiding the cloud storage availability constraint. When a good site for replication is chosen, these methods will produce the best outcomes. The authors, on the other hand, have not explained how to choose an appropriate number of replicas.

Fault tolerance, data availability, data locality, and load balancing are all advantages of data replication in cloud storage systems, from both a reliability and performance standpoint [9]. To maintain replication level, data blocks held on the failed data node must be restored each time it fails. This might be a significant burden for the system, which is already overburdened by users' application workloads. Although various replication concepts have been made, the approach of re-replication has yet to be thoroughly addressed. We describe a postponed re-replication technique in this research that dynamically shifts the re-replication workload based on the system's current resource consumption condition.

Because the pattern of workload varies based on the time of day, simulation results from synthetic workload are used. The simple technique that adjusts re-replication based on current resource use demonstrates a great opportunity for minimizing impacts on users' application workloads. Our method can lessen performance implications on customers' application workloads while maintaining the same level of reliability as standard HDFS. The proposed strategy did not consider the details related to the number of replicas and storage cost.

The Hadoop Distributed File System (HDFS) is a distributed storage system that consistently stores massive volumes of data and provides high-bandwidth access to applications [10]. HDFS ensures excellent data availability and dependability by duplicating data three times and distributing it across numerous data nodes. One of the most important factors affecting HDFS performance is the placement of data replicas. Because replicas of data blocks cannot be uniformly distributed across cluster nodes under the existing HDFS replica placement strategy, the current HDFS must rely on the load balancing utility to balance replica distributions, which takes additional time and resources. These difficulties necessitate the development of intelligent approaches for resolving the data location problem. Without the use of a load balancing utility, great performance is achieved. In this research, we propose an intelligent data placement policy in cloud storage systems that address the aforementioned issues. The major drawback was that this strategy does not provide a mechanism for finding the appropriate number of replicas.

The authors proposed a system model which is a multi-tier hierarchical cloud system architecture consisting of multiple tiers each with data centers of different regions and sizes [11]. The participants are the users, super data Centre, main data Centre and ordinary data Centre. The cloud data server architecture consists of the replica broker who is the central managing broker, the scheduling broker who contains information regarding the locations of replicas and the data enters. The main contribution of this paper is the mathematical model that shows the relationship between system availability and the number of replicas, which during that time wasn't considered in many of the researches. The proposed Dynamic Data Replication strategy (D2RS) decides which files to be replicated, how many new replicas to create and were to place these replicas. These replication processes are carried out when popularity of the file exceeds a dynamic threshold. While the process of replication of the blocks is based on temporal locality the placement of these replicas is based on the access information of directly connected data centers. According to the experimental analysis conducted by the paper shows this strategy improves the efficiency as well as the effectiveness of the system. Previously the number of replicas were predefined i.e., they are static and moreover even the files that may not be accessed are also replicated. These issues are solved by specifying the only the most accessed files and number of replicas of the file. Popularity of data may change over a period of time and the strategy proposed did not consider the situation when there is a reduction in the popularity of data.

In Table I we have given the details related to literature survey conducted in extensive manner.

III. Proposed System Model

In this section we elaborate the proposed strategy and explain the algorithms. There are three main modules which are described in this section.

A. System Architecture

In Fig. 1 the proposed system architecture consists of a Hadoop cluster containing a single Name Node and a multiple Data Node. Hadoop Web browser is the web browser of the Name Node that shows the details of the Hadoop cluster. Such details include the health of the Data Nodes, the number of live and dead Data Nodes and we can also browse the files stores on HDFS [12].

When a user requests a file from the Hadoop cluster, this request is sent to the Name Node which decides from which Data Nodes the files blocks are to be fetched.

Once all the blocks of the requested file is fetched this is combined to form the original file which is then sent to the client that placed the request. The Data Nodes are not aware of the other nodes in the cluster and communicate only with the Name Node.

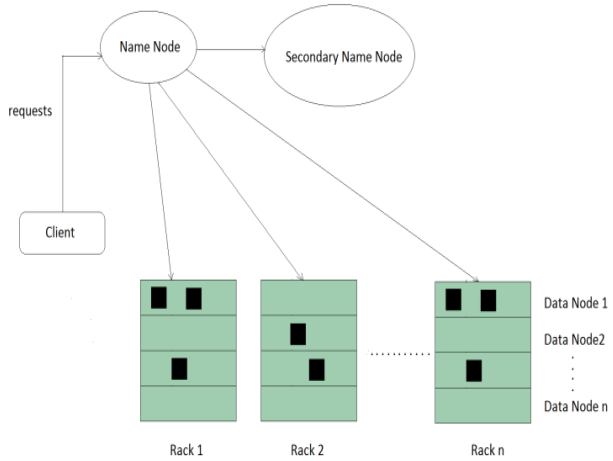


Figure 1. System architecture.

B. Selection of Popular Files

When selecting the number of replicas for the files only the popular file is chosen and considered for replication. The other files are regarded as unpopular files and this do not get considered for the replication process. Popular files are decided based on the concept of support values. The concept used in the proposed system is as follows:

- Analyzing the log to get the client file access behaviour that frequently accesses the files stored in DFS.
- Calculating the support values i.e. frequently accessed files or support value count.
- Determining how the support value for a file block is calculated.

Only the file blocks of popular files are used to establish the support values. For each file block, there are two forms of support: Local Support (LS) and Global Support (GS). The log entries of a single data node are used to determine local support, whereas the log entries of all data nodes are used to calculate global support.

TABLE I. DETAILS OF LITERATURE SURVEY PAPERS WITH RESEARCH GAPS

S-No.	Title	Strategy	Research Gaps	Parameters		
				Replication Factor	Storage space	Access Time
1	An efficient replication management system for HDFS management	The paper proposes a dynamically replicating information files depending on predictive analysis	In our research replicates the file block depending upon the support values and placement of file block based upon the local support value of the file block	✓	✓	×
2	Data replication schemes in cloud computing: a survey.	The classification of data reduplication, data auditing, and data handling	What to replicate, The number of replicas and placement of replicas	×	×	×
3	A Multi-index Evaluation Replication Placement Strategy for Cloud Storage Cluster	The MERP strategy takes the holistic view of the load characteristic, hardware performance	The proposed approach replicates the file block depending upon the support values	×	✓	×
4	A workload clustering based resource provisioning mechanism using Biogeography based optimization technique in the cloud based systems.	Our proposed mechanism utilized biogeography-based optimization (BBO) technique with K-means clustering to classify the cloud workloads	our proposed approach replicates the file block depending upon the support values and placement of file block based upon the local support	×	×	×
5	A novel dynamic data replication strategy to improve access efficiency of cloud storage	a novel dynamic data replication strategy with intelligent water drop (IWD) algorithm	our proposed approach replicates the file block depending upon the support values	×	✓	×
6	Data replica placement approaches in fog computing: a review	studying articles on data replica placement in fog computing	Complete replication process What to replicate, The number of replicas and placement of replicas	✓	✓	×
7	A Dynamic Replica Factor Calculator for Weighted Dynamic Replication Management in Cloud Storage Systems	This strategy classifies the data to hot, warm and cold by considering the access pattern of the data	In our research replicates the file block depending upon the support values	×	×	×
8	Avoiding Performance Impacts by Re-Replication Workload Shifting in HDFS Based Cloud Storage	we present a deferred re-replication algorithm to dynamically shift the re-replication workload based on current resource	The placement of file block based upon the local support value of the file block	×	✓	×
9	An intelligent data placement mechanism for replica distribution in cloud storage system	we propose an intelligent policy for data placement in cloud storage systems	Replicates the file block depending upon the support values and placement of file block based upon the local support value of the file block	×	×	×
10	Modeling a Dynamic Data Replication Strategy to Increase System Availability in Cloud Computing Environments	Evaluating and identifying the popular data and triggering a replication operation when the popularity data passes a dynamic threshold.	Replicates the file block depending upon the support values and placement of file block based upon the local support value of the file block	×	✓	×

In the Algorithm [1], Support is an indication of how frequently the files appear in the dataset [13]. For example that there is M numbers of entries collected in the log. A file F appears in X number of entries in the log then support will be calculated as:

Support (F) =Total number of entries for File F in log (X) / Total Number of entries

Using the above equation support is used as a metric that is used for finding the popular of files in the proposed system. When the popularity of the file (support) exceeds the dynamic threshold, replication process for that file is triggered. The above table shows how support values for file blocks are decided, as well as how table entries are calculated. F400 and F500 are the most popular files. Take (F400 B200) for example; if the total number of sessions in which F400 B200 appears is 8 and the total number of sessions in which F400 appears is 11, the support value for F400 B200 is 8/11= 0.72;

similarly, we can compute the support value for other file blocks from Table II. The Global Support value for each file block can then be calculated by taking into account all data node log entries [14].

Calculating the Frequent Block Access Patterns for the popular files

We will start by looking at how the block access pattern [F400 B200 B201] in Table III is extracted. F400 B200, F400 B201, or F400 B200 B201 appears in a total of 11 sessions, with the pattern F400 B200 B201 appearing in 8 of them. The block access pattern F400 B200 B201 support value is therefore calculated as 8/11. (72.7 percent). We will calculate the support value for each block access pattern in the same way, with a few examples in Table III. Then, based on the t value, a threshold (t) value is determined, and frequent block access patterns are obtained. The most common block access patterns are identified. If we set the threshold value to 60%, F400 B201 B202 and F400 B200 B201 will constitute the frequent block access group [14].

Calculating Support value for block access pattern (BAP).

TABLE II. CALCULATING SUPPORT VALUE FOR FILE BLOCKS IN A DATA NODE

Session Id	File / Blocks	Support for the Block
1.	F400 B200B201 B20	Support(F400,B200)= (8/11)= 0.727
2.	F500 B200B201 B202	
3.	F400 B201 B202 B203	Support(F400,B201)= (11/11) =1.0
4.	F400 B200 B201 B202	
5.	F400 B200 B201 B202	Support(F400,B202)= (9/11)= 0.818
6.	F500 B201 B202 B203	
7.	F400 B200 B201 B202	Support(F400,B203)= (5/11)=0.333
8.	F500 B201 B202B203	
9.	F400 B200 B201 B203	Support (F500,B200)= (2/4)=0.5
10.	F400 B200 B201 B203	
11.	F400 B200B201 B202	Support (F500,B201)= (4/4)=1.0
12.	F400 B201 B202B203	
13.	F500 B200 B201 B202	Support (F500,B202)= (4/4)=1.0
14.	F400 B200 B201 B202	
15.	F400 B201 B202 B203	Support (F500,B203)= (2/4)=0.5

TABLE III. CALCULATING THE SUPPORT VALUE FOR BLOCK ACCESS PATTERN

Calculating Support value for block access pattern (BAP)
Block Access Pattern (F400 B200 B201) = (8/11) = 72.7%
Block Access Pattern (F400 B201 B202) = (9/11) = 81.8%
Block Access Pattern (F400 B202 B203) = (3/11) = 27.2%
Block Access Pattern (F400 B201 B203) = (2/11) = 18.18%

In Fig. 2 the description of proposed architecture is discussed the client request the file blocks and the request is received by the name node. The name node find the rack and the data node where the file blocks are stored. The data node sends the file block pattern to the name node and in turn name nodes send the request to the client with best results of file block pattern.

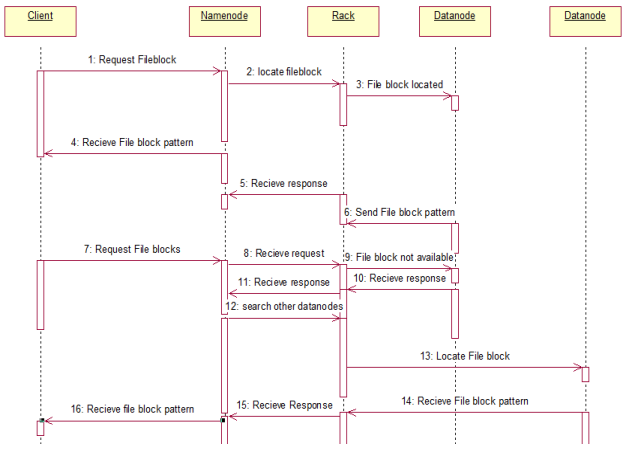


Figure 2. Sequence diagram of proposed system architecture.

C. Determination of Number of Replicas

In Algorithm [2], Once a popular file is selected for replication, the decision of how many additional replicas to be created for that file is taken. The decision involves dividing candidate files (popular files) into three categories depending upon the confidence values computed for them. Hadoop distributed file system has a default replication factor of three that is no matter how popular the data is it gets a fixed replication factor of three.

But in the proposed system we first generate the strong association rules of the popular files. We then calculate the support values of these rules. A dynamic threshold is also considered to measure against the Support values of the rules [14].

The replication factor of the popular file block is fixed in this section. We divided the replication factor into three categories in these. We regard MinSupp to be the threshold value for determining the blocks' Frequent Block Access Pattern. If (Frequent BAP(block) >=MinSupp), then files are frequent access patterns, and if a file block appears in any of the frequent access patterns generated from the log, it is deemed category 1. and if the file block is not available in any of the frequent access patterns but its global support value is greater than MinSupp1 (threshold), the file block is classified as category 2 and the file block is not available in any of the frequent access patterns but its global support value is greater than MinSupp1 (threshold). File blocks that are not available in any of the frequent access patterns but have a global support value >= MinSupp1 (threshold) are classified as category 2; those that are not available in any of the frequent access patterns but do not have a global support value >= MinSupp1 are classified as category 3. Blocks in category 1 have a replication factor of 4, whereas blocks in category 2 have a replication factor of 3. The replication factor for blocks in category 3 is set to 2 [14].

D. Placement of Replicas

In Algorithm [3] If (Ff Bi, Bj) belongs to category 3, the first replica is placed at a data node where the write client process is now executing. The second copy of category 3 will be placed in the data node of the rack, in

other than the write client process rack. Choose a data node that has higher local support than the other data nodes in the rack.

If (Ff Bi, Bj) falls under category 2, the above-mentioned placement method is used first. The third replica will be placed on the data node of a rack, which hosts the write client process. Select a data node with higher local support than the other data nodes.

If (Ff Bi, Bj) comes into category 1, the previous two phases are followed first. In addition, the fourth replica will be placed on the same switch's rack. Choose a data node in the rack that has higher local support compared to other data nodes.

E. Algorithms

The replication algorithm of the proposed strategy solves the three of the replication challenges by taking the following essential steps:

- Deciding what data to replicate
- How many replicas of the file to be created and
- Where to place these replicas

Algorithm 1: Calculating the support values for File Blocks in a data node

Input: Transaction log L and Minsupp, the minimum support count threshold
Output: List of popular files (LP).

1: for each File F **do**
2: for each File Block B **do**
/* the number of client requests for each block*/
3: Calculate number of request during a session for each block
/* The number of client requests for all blocks, **ST**= T(F1B1) + T(F1B2) +..... */
4: Calculate number of request during a session for all blocks
5: for each File Block B **do**
/* The number of client requests for each block, Support=T(FiBi)/ST*
6: if Support (Ff)>= MinSupp do
7: Store in popular file list (Popular File)
8: else do
9: Do not consider files
10: end for

Algorithm 2: Number of Replicas and Category Determination

Input: List of Popular Files (PF) and their support values MinSupp, the minimum support count threshold
Output: List of file name and its updated replicas

1: For each file Ff in Popular Files do

2: Generate Frequent Block Access Pattern for all files.
3: end for
4: for each single Combination of Block `Access Patterns (C-BAP) in each session do

5: Calculate Support for each Combination of Block Access Patterns (C-BAP) in each session (C-BAP)
C-BAP Support = The Total number of sessions in which C-BAP appears / Total number of sessions in which at least one of the C-BAP blocks appears
6: if (Support (C-BAP) > MinSupp)
7: Paste C-BAP into the Frequent BAP list. (The following are the entries in this list: C-BAP is the file name, and Frequent BAP list is the name of the list.)
8**: end if**
9: end for
10: for each file Ff present in the Log do
11: for each block Ff (Bi) of Ff do
12: if (Ff (Bi)) is present in any one of the C-BAPs present in Frequent BAP_list then
13: add Ff (Bi) in Category 1
else
14: if (GS (Ff (Bi))) >= MinSupp1
/* MinSupp1 - minimum support value */
15: add Ff (Bi) in Category 2
else
16: add Ff (Bi) in category 3
17: end for

Algorithm 3: Placement of replicas

Input: No. of replicas of each file block and List of Data Nodes and Racks with corresponding Local Support values
Output: Data Node that hosts the file block and File Blocks allocated according to Local Support value
/* Placement of first replica */

1**: for** each Rack RK do
2: for each Data Node DN do
3: for each File Block B do

4: If (Ff Bi, Bj) belongs to category 3, the first replica is placed at a data node where the write client process is now executing. The second copy of category 3 will be placed in the data node of the rack, in other than the write client process rack. Choose a data node that has higher local support than the other data nodes in the rack. **end for**
5: for each Rack RK do
6:for each Data Node DN not same as containing first replica do
7: for each File Block B do
8: If (Ff Bi, Bj) falls under category 2, the above-mentioned placement method is used first. The third replica will be placed on the data

node of a rack, which hosts the write client process. Select a data node with higher local support than the other data nodes. **end for**

10: for each Rack RK not containing a replica of selected file block do

11: for each Data Node DN do

12: for each File Block B do

13: If (Ff Bi, Bj) comes into category 1, the previous two phases are followed first. In addition, the fourth replica will be placed on the same switch's rack. Choose a data node in the rack that has higher local support compared to other data nodes. **end for**

IV. Performance Evaluation

In this section, first, the simulation parameters for running the simulation are defined. The details of the experimental setup are then discussed. Finally, the simulation results are presented.

A. Simulation Parameters

The simulation parameter values are taken from [15]-[19] and it is collected in Table IV. Requests generated in the ratio of 20 percent from log and 80 percent out of the log are considered.

TABLE IV. Access Time of File Blocks

SL.No	Accessing data block from	Communication time/ Transfer time (4kb data)	Average time (ms)
1	Local disk	-	0.0890
2	Main Memory	-	0.00009
3	-	From remote DN to Local DN	0.004
4	Remote memory	-	0.00409
5	-	From one DN to client DN (another rack)	0.00569

We built the simulation environment in Java and ran it on an i7 processor with 16 GB of RAM running Windows 10. In a DFS system, each rack consists of ten racks (Rk0–Rk9) and 100 data nodes (DaN0–DaN99). We created a synthetic log with 100000 read requests based on the assumption that 100 files are present in the DFS environment. Each file is made up of 100 blocks that are each 4KB in size. The length of each request varies between 5 and 15 blocks. We first verified that all of the files support values are calculated, and then we loaded the entire file block into RAM.

B. Results

1) Storage cost utilization

In this section we analyze the storage cost of HDFS and our proposed system. Considering that each file has an ideal size of 4 MB. We increase the read request from 1000 to 5000 and we have repeated each experiment five times then we have taken average at the end.

The storage cost usage of the proposed system and Hadoop is depicted in Fig. 3. The cost of storing is slightly higher than using the current algorithm. Hadoop employs the same replication factor for all file blocks, i.e. three replicas per file block, and each file block is 4 KB in size. The suggested algorithm, on the other hand, will give alternative replication factors based on the support values estimated for each file block, such as two replicas by default for each file block and three and four replicas depending on the support values of each file block. Currently, storage devices (hard drives) with the biggest storage capacity, such as Tera Bytes, are inexpensive; therefore a small increase in storage capacity has little impact.

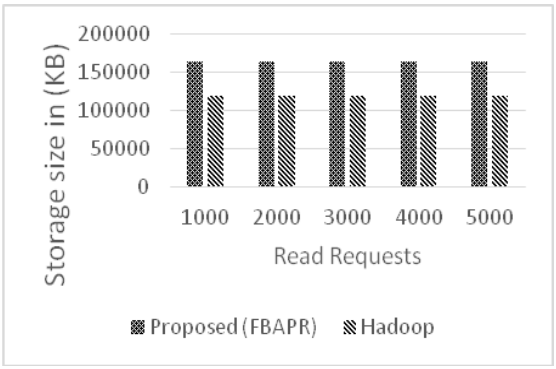


Figure 3. Disk storage utilization for HDFS and frequent pattern-based replication algorithm.

2) Replication factor analysis

A file block's replication factor is a property that specifies how many copies of the file block are replicated. HDFS has a default replication factor of three, which means three copies of the file are created and stored on three distinct Hadoop Data Nodes. This characteristic is important since it allows for more data availability. The average replication factor for both techniques and by varying read request rate is shown in Fig. 4. In this instance, the maximum average replication factor is calculated to be 4.02, which is slightly higher than the one used in HDFS. HDFS has a replica factor of three because each file is always replicated three times. The proposed system creates four replicas for category-1 file blocks with the greatest support values, three replicas for category-2 file blocks with the average of access, and two replicas for category-3 file blocks by default.

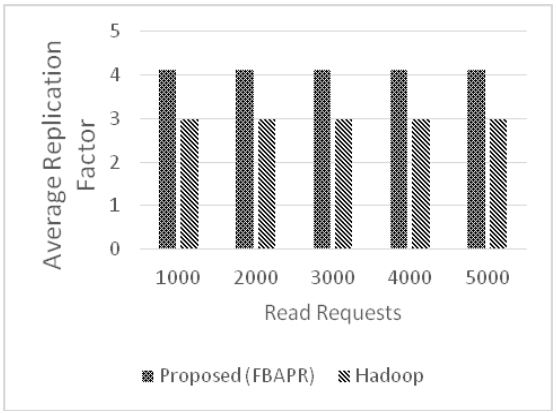
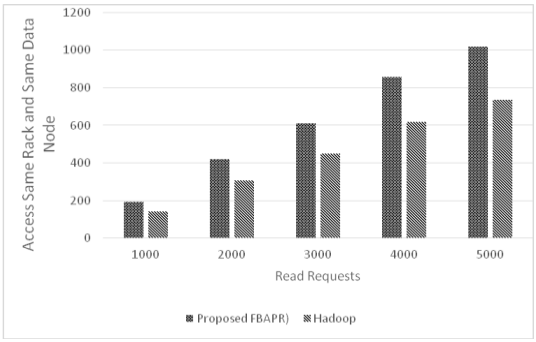


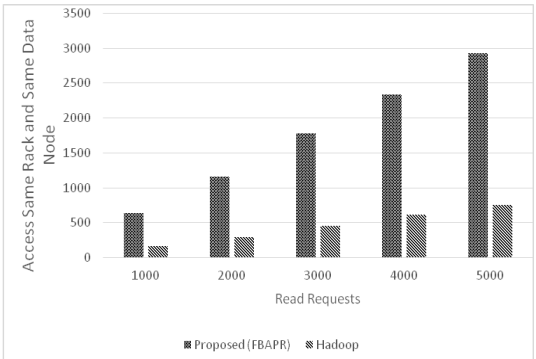
Figure 4. Average replication factor for HDFS (Vs) frequent pattern-based replication algorithm.

Fig. 5 for different percents of requests collected from log requests, Fig. 5(a) to Fig. 5(f) demonstrates the file request satisfied by the same rack and same data node. There are a total of 1000 file requests, each of which generates five random blocks to read, resulting in a total of 5000 sub requests. The 5000 sub requests from the Same Rack and Same Data node (SRSD) maximum request are accessed in the same way (SRSD). The proposed Frequent Pattern-Based Replication technique can be seen to work. By sending read requests to the same rack as the same data node, it outperforms the HDFS replication process (SRSD). Fig. 5(a) depicts a read request taken from a log with a percent of zero and a randomly generated read request with a percent of 100. As a result, our proposed Frequent Pattern-Based Replication algorithm outperforms the HDFS replication algorithm when compared to other data nodes. Fig. 5(b) shows that read requests derived from logs account for 20% of all read requests and randomly generated read requests account for 80% of all read requests.

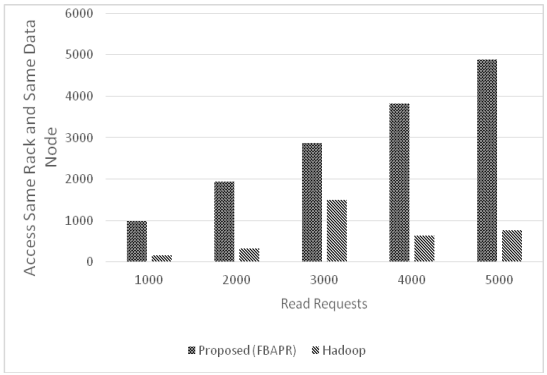
Fig. 5(c) considers the read request taken from log to be forty percent and randomly generated read request to be sixty percent, Fig. 5(d) considers the read request taken from log to be sixty percent and randomly generated read request to be forty percent; Fig. 5(e) considers the read request taken from log to be eighty percent and randomly generated read request to be twenty percent Fig. 5(f) shows that a read request taken from a log is worth a hundred percent of the time, but a randomly produced read request is worth zero percent of the time. The suggested Frequent Pattern-Based Replication algorithm outperforms the HDFS replication algorithm, as can be seen.



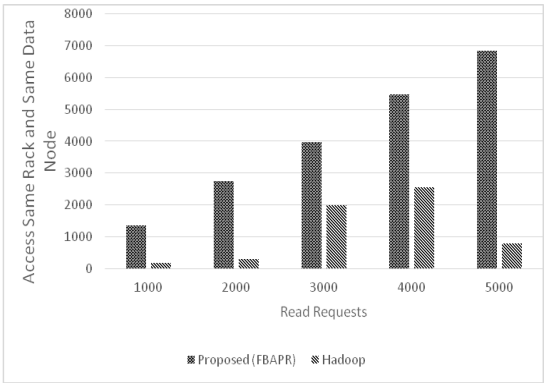
(a) 100% read requests generated randomly and zero percent from the system log



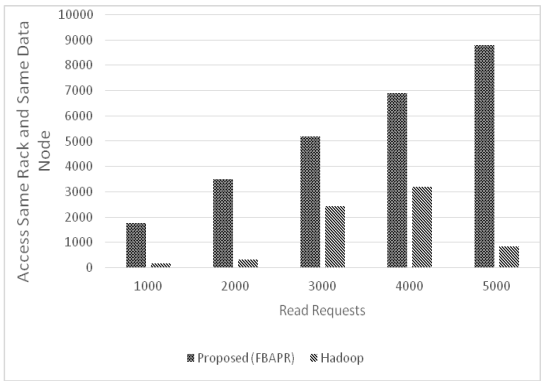
(b) 80% read requests generated randomly and 20% from system log



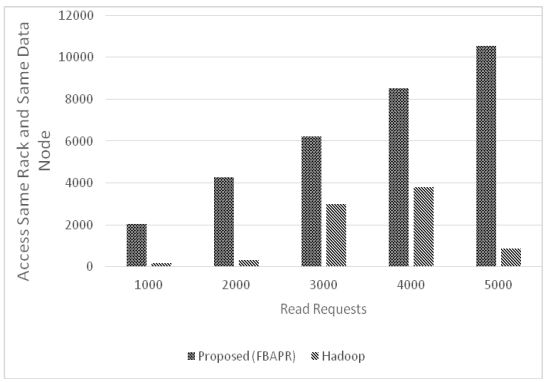
(c) 60% read requests generated randomly and 40% from system log



(d) 40% read requests generated randomly and 60% from system log



(e) 20% read requests generated randomly and 80% from system log



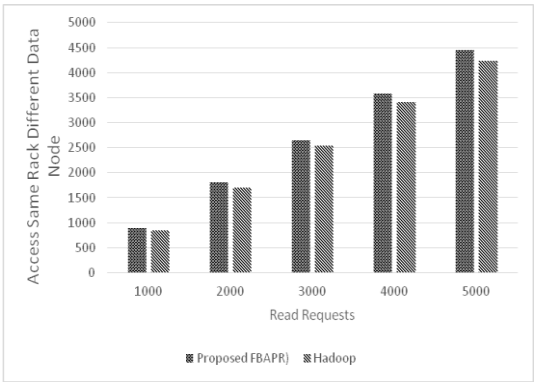
(f) 100% read requests generated from system log and zero percent randomly

Figure 5. File request satisfied by the same rack and same data node.

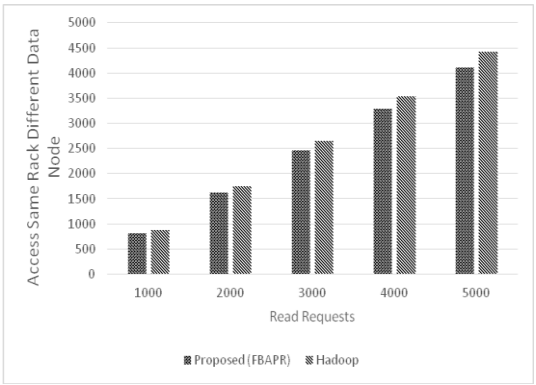
Fig. 6(a)-(f) depicts the file requests fulfilled by the Same Rack and Different Data nodes (SRDD) for various

percents of requests derived from log requests. There are a total of 1000 file requests, each of which generates five random blocks to read, resulting in a total of 5000 sub requests, with the 5000 sub requests from the same rack and different data node (SRDD) accessed in the maximum request (SRDD). By directing read requests to the same rack as the same data node, the proposed Frequent Pattern-Based Replication algorithm outperforms the HDFS replication algorithm (SRDD). Fig. 6(a) depicts a read request taken from a log with a percent of zero and a randomly generated read request with a percent of 100. Thus, when compared to other data nodes, our suggested Frequent Pattern-Based Replication technique outperforms the HDFS replication algorithm. Fig. 6(b) shows that read requests from the log account for 20% of the total, whereas read requests produced randomly account for 80%.

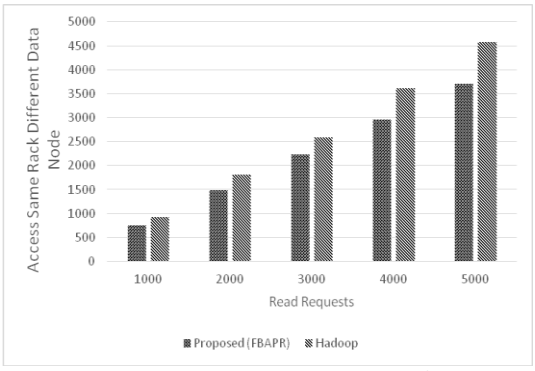
Fig. 6(c) considers the read request taken from log to be forty percent and randomly generated read request to be sixty percent; Fig. 6(d) considers the read request taken from log to be sixty percent and randomly generated read request to be forty percent; Fig. 6(e) considers the read request taken from log to be eighty percent and randomly generated read request to be twenty percent Fig. 6(f) shows that a read request taken from a log is worth a hundred percent of the time, but a randomly produced read request is worth zero percent of the time. As a result, our suggested Frequent Pattern-Based Replication technique outperforms the HDFS replication algorithm and outperforms all other data nodes.



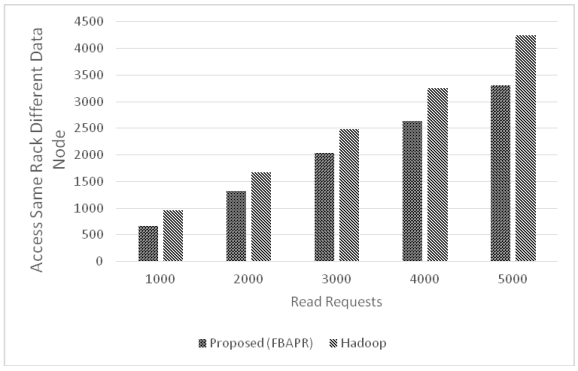
(a) 100% read requests generated randomly and zero percent from the system log



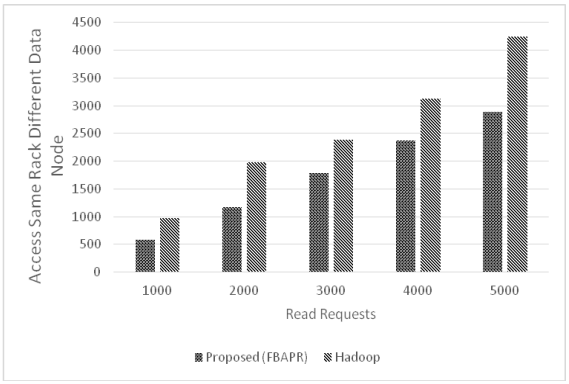
(b) 80% read requests generated randomly and 20% from system log



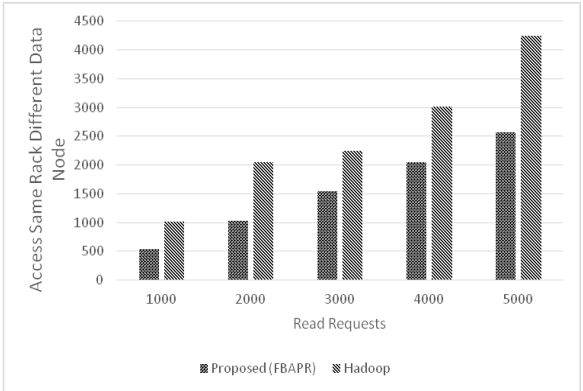
(c) 60% read requests generated randomly and 40% from system log



(d) 40% read requests generated randomly and 60% from system log



(e) 20% read requests generated randomly and 80% from system log



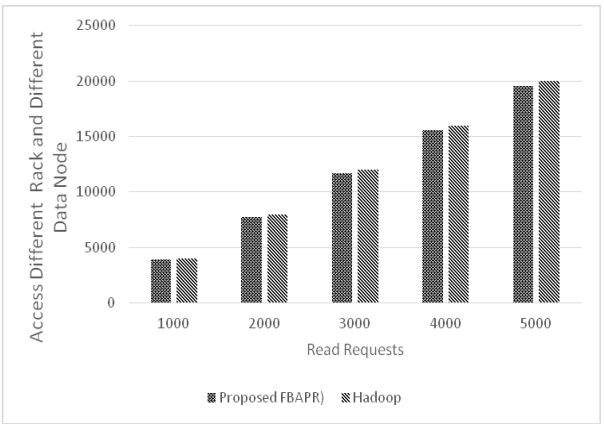
(f) 100% read requests generated from system log and zero percent randomly

Figure 6. File requests fulfilled by the Same Rack and Different Data nodes (SRDD).

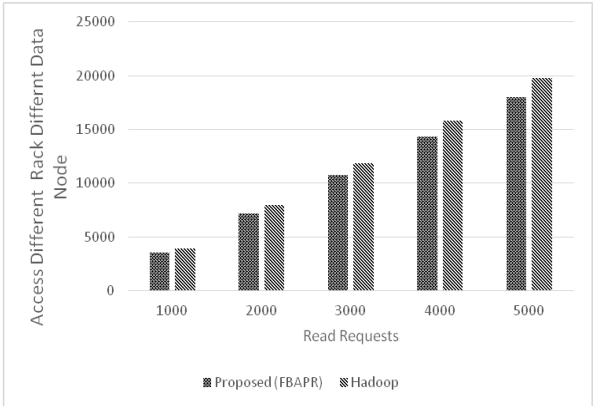
Fig. 7(a) through (f) depicts the file requests met by different racks and data nodes for various percents of requests derived from log requests. There are a total of

1000 file requests, with each file request creating five random blocks to read, resulting in a total of 5000 sub requests. The 5000 sub requests from various racks and data nodes (Diff R Diff D) are accessible in the maximum request (Diff R Diff D). By submitting the read request to Diff Rack as Diff Data node, we can see that the proposed Frequent Pattern-Based Replication algorithm outperforms the HDFS replication algorithm (Diff. R. Diff. D).

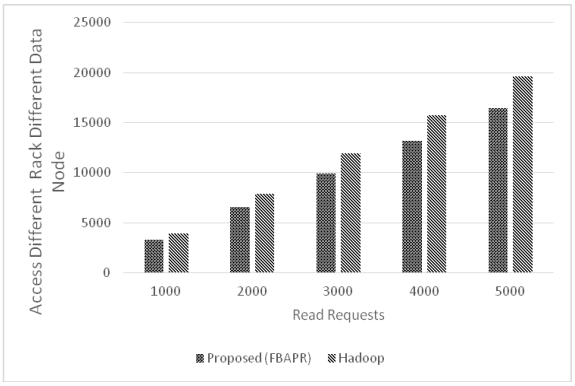
Fig. 7(a) depicts a read request taken from a log with a percent of zero and a randomly generated read request with a percent of 100. Thus, when compared to other data nodes, our suggested Frequent Pattern-Based Replication technique outperforms the HDFS replication algorithm. Fig. 7(b) shows that read requests from the log account for 20% of the total, whereas read requests produced randomly account for 80%. Fig. 7(c) considers the read request taken from log to be forty percent and randomly generated read request to be sixty percent; Fig. 7(d) considers the read request taken from log to be sixty percent and randomly generated read request to be forty percent; Fig. 7(e) considers the read request taken from log to be eighty percent and randomly generated read request to be twenty percent Fig. 7(f) shows that a read request taken from a log is worth a hundred percent of the time, but a randomly produced read request is worth zero percent of the time. As a result, our suggested Frequent Pattern-Based Replication technique outperforms the HDFS replication algorithm and outperforms all other data node replication algorithms.



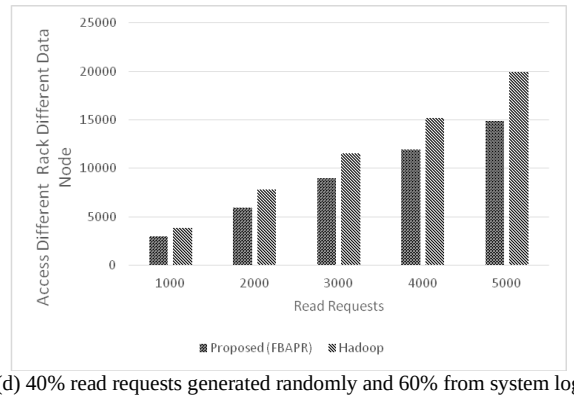
(a) 100% read requests generated randomly and zero percent from the system log



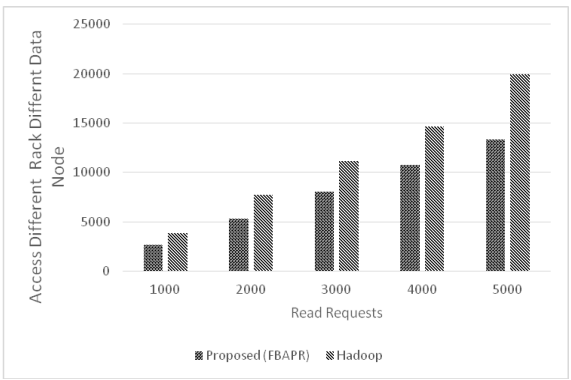
(b) 80% read requests generated randomly and 20% from system log



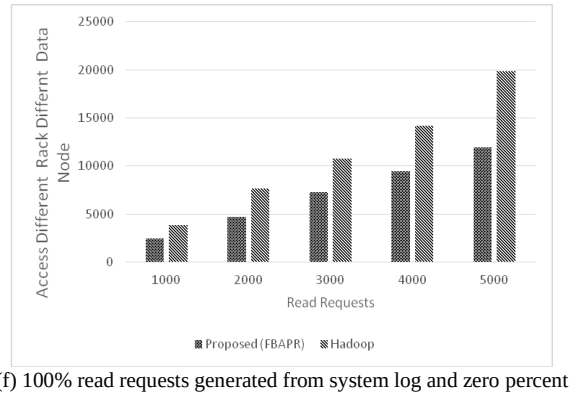
(c) 60% read requests generated randomly and 40% from system log



(d) 40% read requests generated randomly and 60% from system log



(e) 20% read requests generated randomly and 80% from system log



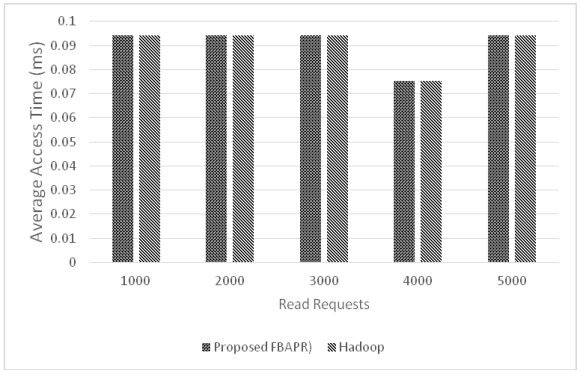
(f) 100% read requests generated from system log and zero percent randomly

Figure 7. File requests met by different racks and data nodes for various percents of requests derived from log requests.

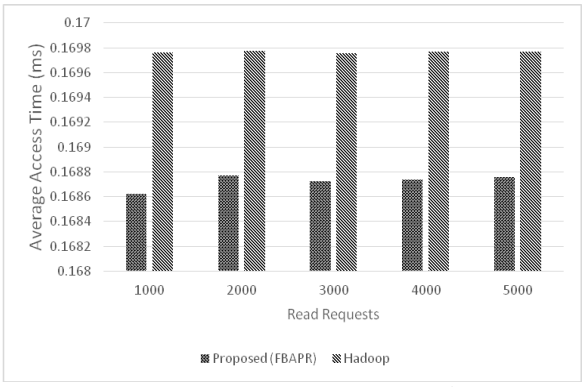
The Average Access Time of file blocks is depicted in Fig. 8(a)-(f). The read request taken from the log is given a percent of zero, whereas the read request produced randomly is given a percent of 100. Thus, when

compared to other data nodes, our suggested Frequent Pattern-Based Replication technique outperforms the HDFS replication algorithm. Fig. 8(b) shows that read requests from the log account for 20% of the total, whereas read requests produced randomly account for 80%.

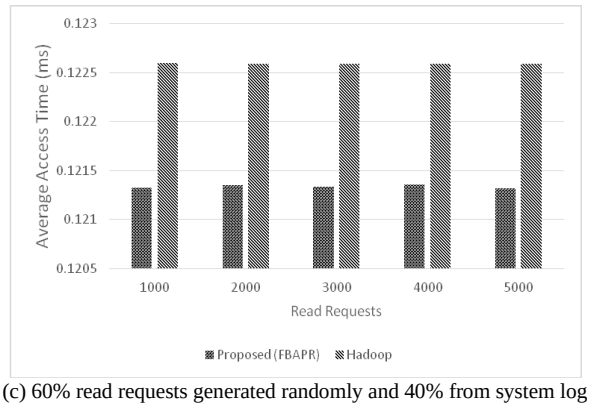
Fig. 8(c) considers the read request taken from log to be forty percent and randomly generated read request to be sixty percent; Fig. 8(d) considers the read request taken from log to be sixty percent and randomly generated read request to be forty percent; Fig. 8(e) considers the read request taken from log to be eighty percent and randomly generated read request to be twenty percent Fig. 8(f) shows that a read request taken from a log is worth a hundred percent of the time, but a randomly produced read request is worth zero percent of the time. As a result, our suggested Frequent Pattern-Based Replication technique outperforms the HDFS replication algorithm and outperforms all other data nodes.



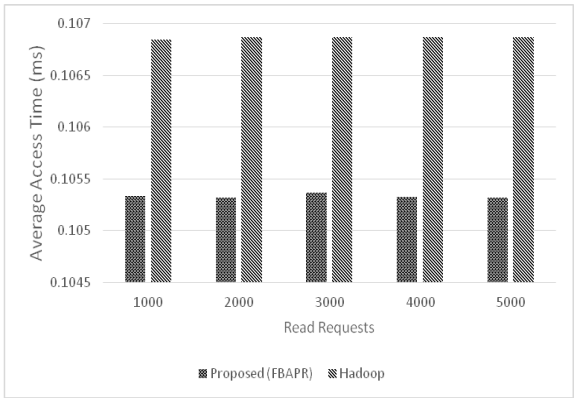
(a) 100% read requests generated randomly and zero percent from the system log



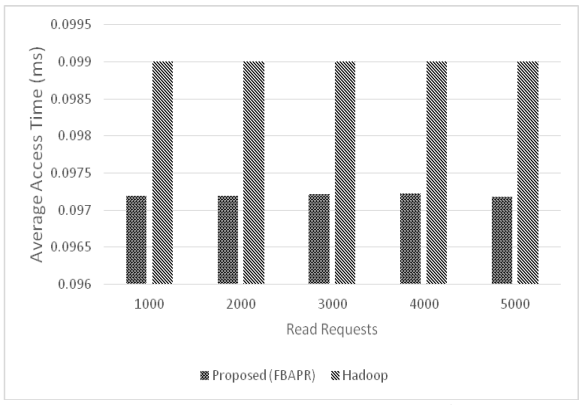
(b) 80% read requests generated randomly and 20% from system log



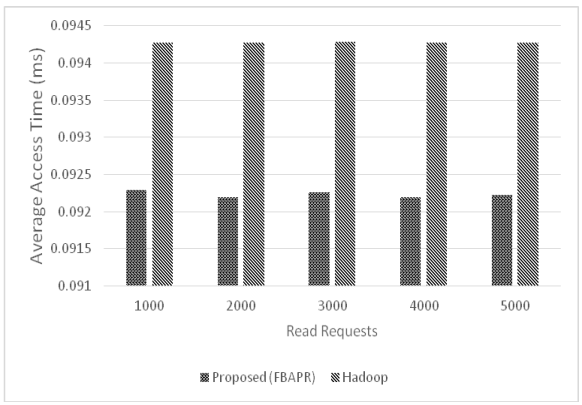
(c) 60% read requests generated randomly and 40% from system log



(d) 40% read requests generated randomly and 60% from system log



(e) 20% read requests generated randomly and 80% from system log



(f) 100% read requests generated from system log and zero percent randomly

Figure 8. Average access time of file blocks.

V. CONCLUSION

In this paper, we present a novel cloud storage system replication algorithm based on frequent block access patterns. In order to save storage space, the replication factor in the suggested technique was determined based on the support values of the file blocks and the access frequency of the blocks contained in the file. In order to boost performance, we have also proposed an effective method of storing replicas in cloud storage systems. Both the Hadoop replication strategy and the above-mentioned algorithm were developed in Java, for performance analysis. According to the findings, the suggested technique outperforms the Hadoop distributed file system's replication mechanism.

CONFLICT OF INTEREST

The author declares no conflict of interest.

AUTHOR CONTRIBUTIONS

Both the authors contributed in all the aspects of writing this paper like writing the Introduction, Performing extensive literature survey, Proposed Strategy, Performance Evaluation and conclusion.

REFERENCES

[1] B. Rajkumar, V. Christian, and S. S. Thamarai, *Mastering Cloud Computing*, Mc Graw Hill, 2013.

[2] K. Swaroopa, A. S. P. Kumari, N. Manne, *et al*., "An efficient replication management system for HDFS management," *Science Direct Material Proceedings*, July 2021.

[3] A. Shakarami, M. Ghobaei-Arani, A. Shahidinejad, *et al.*, "Data replication schemes in cloud computing: A survey," *Cluster Compute*, vol. 24, pp. 2545-2579, 2021.

[4] Y. Su and W. Zhang, "A multi-index evaluation replication placement strategy for cloud storage cluster," in *Proc. of the 4th International Conference on Cloud and Big Data Computing*, August 2020, pp. 20-26.

[5] M. Ghobaei-Arani, "A workload clustering based resource provisioning mechanism using biogeography based optimization technique in the cloud based systems," *Soft Compute*, vol. 25, pp. 3813-3830, 2021.

[6] S. N. John and T. T. Mirnalinee, "A novel dynamic data replication strategy to improve access efficiency of cloud storage," *Information System and e-Business Management*, vol. 18, pp. 405-426, 2020.

[7] E. Torabi, M. Ghobaei-Arani, and A. Shahidinejad, "Data replica placement approaches in fog computing: A review," *Cluster Compute*, pp. 1-29, 2022.

[8] S. Gopinath and E. Sherly, "A dynamic replica factor calculator for weighted dynamic replication management in cloud storage systems," *Procedia Computer Science*, vol. 132, pp. 1771-1780, 2018.

[9] T. Shwe and M. Aritsugi, "Avoiding performance impacts by re-replication workload shifting in HDFS based cloud storage," *IEICE Transaction on Information System*, pp. 2958-2967, 2018.

[10] I. A. Ibrahim, W. Dai, and M. Bassiouni, "An intelligent data placement mechanism for replica distribution in cloud storage system," in *Proc. IEEE International Conference on Smart Cloud*, December 2016.

[11] D. Sun, G. Chang, S. Gao, *et al.*, "Modelling a dynamic data replication strategy to increase system availability in cloud computing environments," *Journal of Computer Science and Technology*, vol. 27, pp. 256-272, 2012.

[12] ApacheHadoop. [Online]. Available: http://Hadoop.apache.org/

[13] H. Jiawei and K. Michelin, *Data Mining Concepts and Techniques*, 2nd ed., Morgan Kaufmann Publishing, 2007, pp. 23-29.

[14] T. Ragunathan and M. Sharfuddin, "Frequent block access pattern-based replication algorithm for cloud storage systems," in *Proc. Eighth* International C*onference on Contemporary Computing*, 2015, pp. 7-12.

[15] Resource & Design Center for Development with Intel. [Online]. Available: https://www.intel.com/content/www/us/en/design/resource-design-center.html

[16] Seagate enterprise performance 10K HDD review StorageReview.com-Storage reviews. (May 2015). [Online]. Available: https://www.storagereview.com/seagate_enterprise_performance_10k_hdd_review

[17] List of Intel SSDs. (2019). In Wikipedia. [Online]. Available: https://en.wikipedia.org/w/index.php?title=List_of_Intel_SSDs&oldid=898338259

[18] Corsair Vengeance LPX DDR4 3000 C15 2x16GB CMK32GX4M2B202000C15. [Online]. Available: fromhttps://ram.userbenchmark.com/Compare/Corsair-Vengeance-LPX-DDR4-3000-C15-2x16GB-vs-Group/m92054vs10F4000_wp_Ethernet.pdf

[19] Cisco Nexus 5020 Switch Performance in Market-Data and Back-Office Data Delivery Environments. [Online]. Available: https://www.cisco.com/c/en/us/products/collateral/switches/nexus-5000-series-switches/white_paper_c11-492751.html

Mohammed Sharfuddin received his M.Tech degree in Software Engineering from Jawaharlal Nehru Technological University (JNTU), Hyderabad, India in 2010. He is pursuing PhD in computer science and engineering from JNTU, Hyderabad. His research interests include distributed file system, and cloud computing. At present he is working as an Assistant professor at Department of Computer Science and Engineering at Muffakham Jah College of Engineering and Technology, Hyderabad, Telangana, India.

Dr. Thirumalaisamy Ragunathan received his PhD degree in computer science and engineering from IIIT, Hyderabad, India in 2010. Currently he is a professor & Associate Dean (Research & Development) at Computer Science and Engineering Department at SRM University, Amravati, and Andhra Pradesh India. His research interests include transaction processing, concurrency control, speculative processing, distributed file systems, cloud computing, and big data analysis.