ECTI Transactions on Computer and Information Technology
Journal homepage: https://ph01.tci-thaijo.org/index.php/ecticit/
Published by the ECTI Association, Thailand, ISSN: 2286-9131

Improving Performance of Cloud Storage Systems Using a Support-Based Replication Algorithm

Mohammed Sharfuddin[1] and Thirumalaisamy Ragunathan [2]

ABSTRACT

Data replication is a mechanism for creating a copy of the same file block on many sites. It is used in cloud storage systems to improve the performance of data availability. The current replication technique helps users fix the number of replicas required. The problem with the existing approaches is that the user cannot determine which file should be duplicated and how many copies are necessary, causing the cloud storage system's performance to suffer. The problem mentioned above has an impact on the performance of cloud storage systems. Thus, our proposed replication method determines the replication factor based on the support values of the file blocks to determine the precise number of duplicates to be replicated. We have also proposed an efficient technique to place the replicas based on the local support values to increase the performance of the cloud storage system. Our results indicate that our proposed replication algorithm performs better than the algorithm used in the Hadoop distributed file system.

Article information:

Keywords: Cloud Storage Systems, Distributed File Systems, HDFS, Replication, Replication Factor, Access Time, Storage Cost

Article history:
Received: December 30, 2021
Revised: September 12, 2022
Accepted: October 28, 2022
Published: November 25, 2022 (Online)

DOI: 10.37936/ecti-cit.2023171.247333

1. INTRODUCTION

Cloud Storage System (CSS) is a data storage model that consists of multiple geographically distributed servers or resources, providing a large storage capacity that the users can access at a low cost. CSS allows users to store their files online to access them anywhere via the Internet [1]. Cloud storage provides benefits that including reduced costs, simplified IT management and improved user experience and disaster recovery. There are different types of Cloud Storage Systems includes Relational Database Storage Systems (RDS) and Distributed File Storage Systems. Distributed File systems (DFS) form an essential part of the CSS.

They enable users to store and access the data as if it were stored on the user's own system Unlike the traditional client-server model, data in the distributed file systems are stored on many different nodes that is present in the system. A technique called data replication is widely used to increase the availability of data stored in cloud storage systems. For example, 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.

Replication is a concept in cloud computing that allows several replicas of data to be stored at different locations. Since it allows creation of several copies of data that can be stored at many sites, there are backups available for use in case of a crash of one site where a potential replica is stored. Replication enables the users to access their data without any dependence on the network infrastructure and without worrying about the network traffic. It also ensures that other users are having exclusive access to their desired data without any conflicts. Replication has many benefits including disaster recovery, high availability, and low access time. Replication also has its fair share of limitations or drawbacks, such as the cost of The correct number of replicas, storage space and the time required to update all the replicas. The best 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 as Hadoop,

[1] The author is with Computer Science and Engineering, Muffakham Jah College of Engineering and Technology, Hyderabad, India, E-mail: sharfuddin@mjcollege.ac.in
[2] The author is with Computer Science and Engineering, SRM University, Amravati, Andhra Pradesh, India, E-mail: ragunathan.t@srmap.edu.in

for instance, is one such storage system where three replicas of file block to be stored on the HDFS are made and stored at different data nodes.

The replication factor of a file block refers to the number replicas or copies of that file block in the system. Improve access latency to cloud. The merits of replication include increased availability of data, improved reliability of CSS.To improve access latency to cloud Replication algorithms fall under two types: Static replication and dynamic replication. 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 and do not change. Static algorithms are those in which the replication strategy is predefined for any application and the number of

1.1 Mainissues when choosing replication strategy are as follows:

What to replicate: The first main issue is deciding what data to replicate. If a wrong file is replicated, itwill waste storage space. On the other hand, if famous or essential data is left out from beingreplicated, itwill lead to theunavailability of the data. Hence thefirst significant decision is to decide which data should be replicated.

How many replicas: Once the data to replicate is decided the next step is to decide how many replicas to have. Again, making toomany copies of unimportant datawill lead to unnecessary storagecosts. In contrast, not enough copies will lead to unavailability issues. So, deciding the correct number of replicas iscrucialtoanyreplicationstrategy.Hadoopbydefaultmakes 3copiesofanydata.

Where to place them generally, when a user requests a file, the name node returns the file blocks fromthe data nodes closest to the user. So, deciding which data nodes to place the block replicas is essential toensuringthat theusersgettheir data lesstimeandefficiently. Our proposed algorithm addresses the above questions to improve the performance of Hadoop by (a)Selecting popular file blocks for replication, (b) Adjusting replication factor based on file block support value sand (c)Selecting better at a nodes for placement to replicas.

The paper is presented as follows, In section II,we discuss related work, section III architecture ofthe HDFS, calculation of support values and proposed replication algorithm. Section IV, Performance evaluation andinsectionV. We present our conclusion.

2. RELATED WORK

In [2] 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 (DRS). 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 Hadoop, employs a static replication technique. The percent of people who are accepted into the program. Due to the uniqueness of similar functions on several layers, HDFS, on the other hand, is fully unique. As a result, utilizing the same duplicating method for each file could result in poor performance. 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.

In [3] 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 (DRS). 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 Hadoop, employs a static replication technique. The percent of people who are accepted into the program. Due to the uniqueness of similar functions on several layers, HDFS, on the other hand, is fully unique. As a result, utilizing the same duplicating method for each file could result in poor performance. 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.

In [4] data replication is a typical approach for assuring data availability. In static replication policies, the number of replicas to be created is determined

statically at the time of cloud system setup. To accommodate for changing trends in user demands and storage capacity, data replication must be dynamic. This prevents storagespace 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 Hadoop Distributed File System (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 replication.

In [5] data replication is a typical approach for assuring data availability. In static replication policies, the number of replicas to be created is determined statically at the time of cloud system setup. 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 Hadoop Distributed File System (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 replication site.

In [6] Fault tolerance, data availability, data locality, and load balancing are all advantages of data replication in cloud storage systems. 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.

In [7] 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. 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, good 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.

In [8] proposed a system model which is a multitier hierarchical cloud system architecture consisting of multiple tier each with data centers of different regions and sizes. 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 centers. The main contribution of this paper is the mathematical model that shows the relationship between system availability and the number of replicas, The proposed Dynamic Data Replication strategy (D2RS) decides which files to be replicated, how many new replicas to create and where 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 [9] proposed a Cost-effective Dynamic Replication management which addresses two major issues namely how many replicas to create to satisfy availability requirement and where to place replicas to maximize performance and load balancing. This strategy was implemented in HDFS and it tries to maintain the minimum number of replicas while meeting the availability criteria. The probability of availability of a file F and the expected availability of a file are used to calculate the minimum number of replicas. Once the number of replicas has been decided blocking probability is used to determine the location for these replicas. Parameters such as CPU power, memory capacity and network bandwidth are taken into account for deciding the location. The experimental results show that this strategy is cost effective and improves the availability, performance and load balance of the system. CDRM ignores availability of data having less number of replicas.

3. PROPOSED REPLICATION ALGORITHM

In this section, we first discuss the architecture of the HDFS, which we use to propose our method. Following that, the method used to calculate support values for file blocks will be described.

The following is a summary of HDFS's components.

Name Node: One of the main Hadoop distributed file system component. It stores the Hadoop file system's metadata.Data Nodes: It saves application data. To do this, two files are used to represent each block in the Data Node. The first file includes the contents, while the second provides the block information. Secondary Name Node (SNN): It is a backup for the Name Node. SNN connects to NN at regular intervals to copy the updated.

HDFS File(s): A new file is created when a user stores data on HDFS. File Block(s): Every file in HDFS is stored in the form of blocks. A block in HDFS stored on a Data Node, is a chunk of data having a default size of 64 MB [10].

Procedure for storing HDFS files: The Name Node determines which Data Node should host a file block from a list of data nodes. It assigns a unique block ID to every block before storing it in the data nodes Name Node act s as a Replication Manager. One of the main functions of the name node is replication management, ensuring that every block has the intended number of replicas. The name node removes a replica of the over-re replicated block and assigns a block to the replication priority queue if the block is under-replicated. The name node also ensures that all the replicas of one block are never placed on a single rack. The name node handles the functions of opening-closing and renaming files. On receiving a file request from the user; the name node also decides which data nodes will be selected to return the files to the user.The Data Nodes function as File block Hosts: On the other hand, the data nodes are tasked with storing HDFS data in its local file system. It stores these files in the form of blocks in the HDFS.

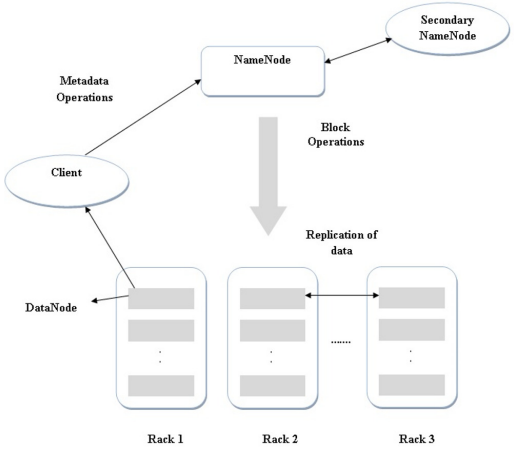

Fig. 1: *Architecture of the HDFS.*

A cloud storage system consists of a cluster of data nodes (DNs) grouped in several racks (RKs). Data is stored in DNs, while metadata is kept in the name node system (NN). Client applications are also run in DNs. The cloud storage system includes a secondary name node to help with fault tolerance (SNN). In the case, NN fails, data can be recovered via SNN. We have assumed that the cloud storage system's NN keeps a system log file to keep track of the client applications that have accessed the file system. Each entry in the system log file comprises the client's IP address, the blocks or files the client has visited, the time (t1) at which the client contacted the and the time (t2) the NN has generated the response. Note that the log file only maintains the client read access requests only.

3.1 Calculating support for the files

Support (SP) is exciting criteria that can be used for finding the popularity of a given file block in a data node. If more client requests are received for a particular block, then we can say the support of the block is high, and hence the popularity of this block is high compared to other blocks. We calculate the total number of block access requests generated by various clients for a particular block, denoted by Req(FrBr). We also calculate the total number of block access requests generated by various clients for all blocks by summing up each block's Req(FrBr). Block support is computed as Support (FrBr) = Req(FrBr)/ ΣReq(FrBr) [11].

Table 1: *Session information and File block support calculation.*

Session Id	FileBlocksinformation	FileBlockSupport
1	F200B99 B100B101	Support(F200,B101)=(4/4)=1.0
2	F200B99B100B101	
3	F100B99 B100B102	Support(F200,B99)=(2/4) =0.5
4	F100B99 B100B101	
5	F100 B100B101 B102	Support(F100,B100) =(11/11=1.0
6	F200 B100B101 B102	
7	F100B99 B100B101	Support(F100,B102)=(5/11)=0.333
8	F200 B100B101B102	
9	F100B99 B100B101	Support(F100,B101)=(9/11)=0.818
10	F100B99 B100B102	
11	F100B99B100B101	Support(F200,B100)=(4/4)=1.0
12	F100 B100B101B102	
13	F100B99B100B101	Support(F100,B99)=(8/11)=0.727
14	F100B99 B100B101	
15	F100 B100B101 B102	Support(F200,B102)=(2/4)=0.5

Table 1 explains how to calculate a block's support value. For instance, consider [F100 B99] file block. The support value for F100 B99 is 8/11 because the total number of sessions in which F100 B99 happens is 8, and the total number of sessions in which the file F100 appears is 11. Table 1 further calculates and displays support values for extra file blocks.

3.2 Determining number of replicas

This section discusses the procedure we are using to decide number of replicas for a block. We divide the candidate blocks for replication into three categories depending on the support value computed. We use predefined minimum support values minsupp1 and minsupp2 (where minsupp2 > minsupp1) to divide the candidate blocks into three categories. The procedure for this categorization of blocks is given below. In the procedure, Supp (block) indicates Supp value of the block. If (Supp (block) > = minsupp2), block belongs to category. 1. If (Supp (block) < minsupp2 and Supp (block) > = minsupp1), block belongs to category 2; else the block belongs to category 3.

(i) For popular blocks, we make more replicas to achieve better response time at the cost of increased storage.

(ii) We make fewer replicas for unpopular blocks because they will be accessed infrequently and clients can wait for some more time in the system to get the response. A block placed in one category may have to be transferred to another category as the Support value for that block may increase or decrease during its lifetime in the system. Our algorithm dynamically detects the popularity value of the block and based on that new replicas are created or existing replicas are deleted.

Blocks in category 1 have a replication factor of 4, whereas blocks in category 2 have 3 replication factor. The replication factor for blocks in category 3 is set to 2.

3.3 The Placement of Replicas

The first replica is placed in the rack and data node where the write client process is executing. This is same for all the categories. The second replica is placed in a different rack and data node that has the highest local support value. For placement of third replica of the file block, select the rack of write client process executing and select a different data node where the local support is highest and place the third replica. Placement of fourth replica. The systems select the leftover racks and find the data node where the local support of the file block is highest then place the fourth replicas.

Proposed Replication Algorithm Pseudo code

The proposed algorithm contains two modules based on the functionalities:

(i) File Block Support Value Calculation and Determine the Categorization.

(ii) Placement of replicas.

3.3.1 FileBlockSupportValueCalculationand Determine the Categorization

Algorithm 1: Category determination

```
01:  for each file Fr do
02:  for each block FrBr of Ff do
03:  Calculate T(FrBr)
/* The number of client requests for each block*/
04:  end for
05:  for each block FrBr of Fr do
06:  Calculate T
/* Total number of client requests for all the blocks belonging
to all the files i.e. T= T(F1B99) + T(F1B100) +. */
07:  Calculate Support (Ff Bb)
/* Support for forFrBr = T(FrBr) / T */
08:  end for
09:  for each block FrBr of Fr do
10:  if (Supp (FrBr) > = minsupp2)
11:  Copy block name (FrBr ) in Cat1
12:  end if
13:  if ((Supp (FrBr )< minsupp2) and (Supp (FrBr)
     > = minsupp1))
14:  Copy block name (FrBr) in Cat2
15:  else
16:  Copy block name (FrBr) in Cat3 17: end if
18:  end if
19:  end for
20:  end for
```

3.3.2 **Algorithm 2 :** Placementof replicas

Input: List of Data Nodes and Racks with corresponding Local Support values.
Output: File Blocks allocated according to support value.

```
/* Placement of first replica */
1:   for each Rack RK do
2:   for each DataNode DN do
3:   for each File Block B do
4:   The first replica is placed in the rack and data
     node where write client process is executing
/* Placement of second replica*/
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: Place the second replica in the different rack and different data node that is having highest local support value.
/* Placement of third replica*/
9: for each Rack RK not containing a replica of selected file block do
10: for each Data Node DN not same as containing first replica do
11: for each File Block B do
12: The file block with the sorted local support value high and select the rack of write client process is executing and select the different data node where the local support is highest and place the third replica
/* Placement of fourth replica*/
13: for each Rack RK do
14: for each Data Node DN do
15: for each File Block B do
16: Place the fourth replica in the leftover racks and select a data node that is having highest local support value.

4. PERFORMANCE EVALUATION

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

4.1 Simulation Parameters

The simulation parameter values are taken from [12][13][14]15][16] [17]. Requests generated in the ratio of 20 percent from log and 80 percent out of the log are considered.

Table 2: *Simulation Parameter values.*

S.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 conducted the simulation using java, we have implemented in i7 processor, 16 GB RAM and windows 10 operating system. We assumed that the DFS environment has ten racks (Rk0–Rk9) and 100 data nodes (DaN0–DaN99) for each rack. We created a synthetic log containing 100000 entries of reading requests based on the assumption that 100 files are present in the DFS environment. Each file is made up of 100 blocks, with each block being 4KB in size. The length of each request ranges from 5 to 15 blocks. We first determined support for all of the files blocks, and then we loaded the file block into RAM.

4.2 Storage Cost Utilization

In this section we analyze 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.

Figure.2 describes the storage cost utilization of the proposed system and Hadoop. The storage cost is slightly higher than the existing algorithm. This is because Hadoop uses same replication factor for all file blocks i.e. three replicas of each file block and size of the each file block is 4 KB. Whereas the proposed algorithm will provide different replication factors based on the support values calculated for the each file block i.e. default two replicas for every file block and three and four replicas based on the support values of each file block. Currently in the market the storage devices (hard disks) with the largest storage device such as Tera Bytes are available in less cost, thus slight increase of storage space does not impact much.Fig.2.Disk storage cost utilization for HDFS and Support Based Replication (SBR) algorithms.

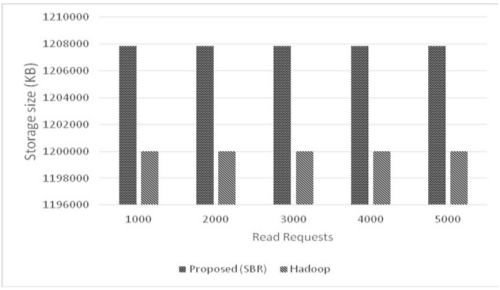

Fig.2: *Disk storage cost utilization for HDFS and Support Based Replication (SBR) algorithms.*

4.3 Replication factor Analysis

The replication factor of a file block is a property that defines how many copies of that file block are maintained. HDFS allows for a default replication factor of three meaning three replicas of the file are made and stored at three different Data Nodes in the Hadoop cluster. The significance of this property is that it enables increased data availability.

The average replication factor for both techniques and by altering read request rate is shown in Figure 3. In this instance, the highest average replication factor determined for the (Support Based Replication) SBR method is 3.02, which is marginally higher than the one used in HDFS. Since HDFS always creates

three replicas of each file therefore its replica factor as three. The proposed system creates four replicas for the highest support values of file blocks which belongs to category-1, three replicas for average of access of files blocks belonging to category-2 and default two replicas for every file block belonging to category- 3.

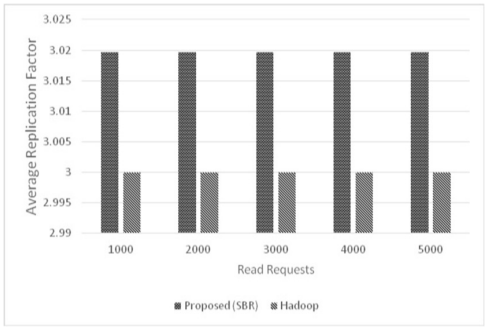

Fig.3: *Average Replication factor for HDFS (Vs) Support Based Replication (SBR)algorithms.*

4.4 Comparison of File requests satisfied by Same Rack as Same Data node (SRSD) for proposed method and Hadoop Replication Algorithm

Figures 4. (a) through (f) illustrate the file request satisfied by same rack and same data node for different percents of request taken from log requests. There are total of 1000 file request and each file request is generating five random blocks to read for a total of 5000 sub request is generated and The 5000 sub request from Same Rack and Same Data node (SRSD).We can observe that the proposed Support Based Replication (SBR) algorithm outperform the HDFS replication algorithm by sending the read request to Same Rack and Same Data node (SRSD).

The figure 4(a) considers the read request taken from system generated log is of zero percent and hundred percent randomly generated log read request, Thus our proposed Support Based Replication (SBR) algorithm outperform the HDFS replication algorithm.The figure 4(b) considers the read request taken from log is of twenty percent and randomly generated log log read request is of eighty percent.

The figure 4(c) considers the read request taken from log is of forty percent and randomly generated log read request is of sixty percent, The Figure 4(d) considers the read request taken from log is of sixty percent and randomly generated log read request is of forty percent. The figure 4(e) considers the read request taken from log is of eighty percent and randomly generated log read request is of twenty percent and The figure 4(f) considers the read request taken from log is of hundred percent and randomly generated log read request is of zero percent .Hence our proposed algorithm outperforming the existing replication algorithm.

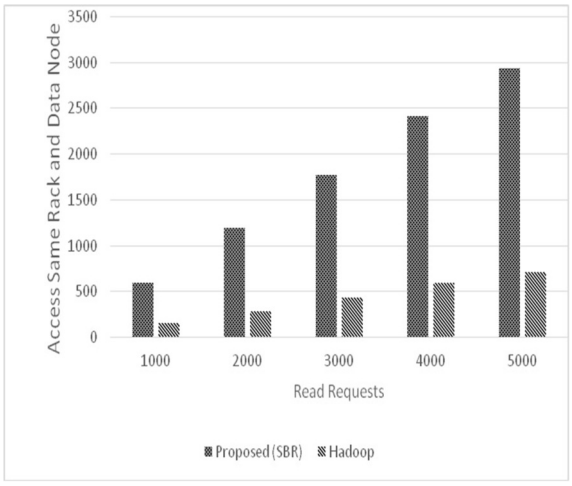

Fig.4: *a)Read request from log is 40 percent and randomly generated log 60 percent.*

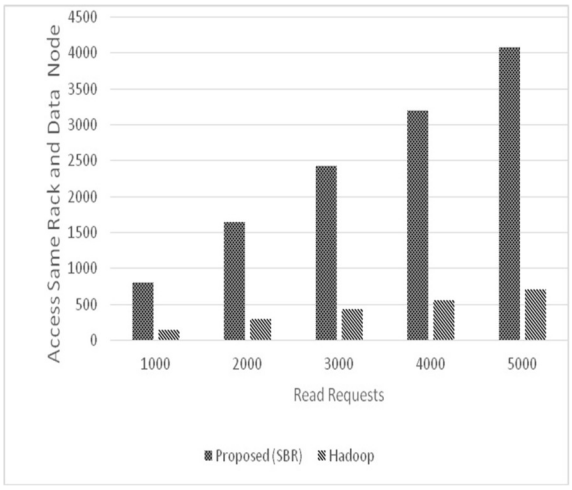

Fig.4: *b) Read requests from log is 60 percent and randomly generated log 40 percent.*

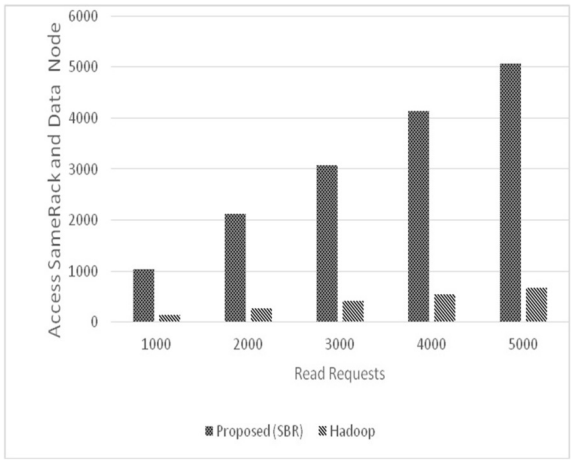

Fig.4: *c) Read requests from log is 80 percent and randomly generated log 20 percent.*

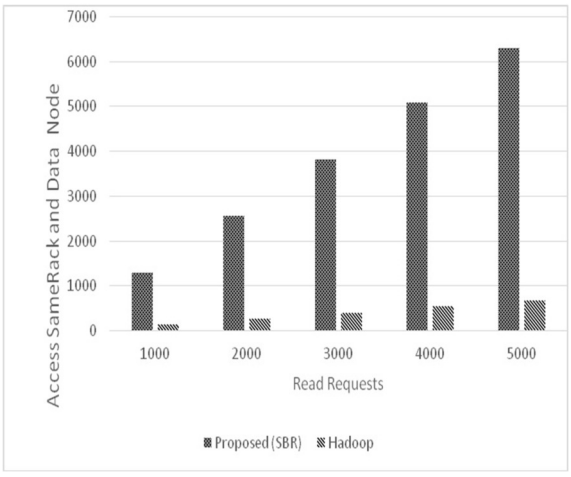

Fig.4: *d)Read request from log is 100 percent and randomly generated log zero percent.*

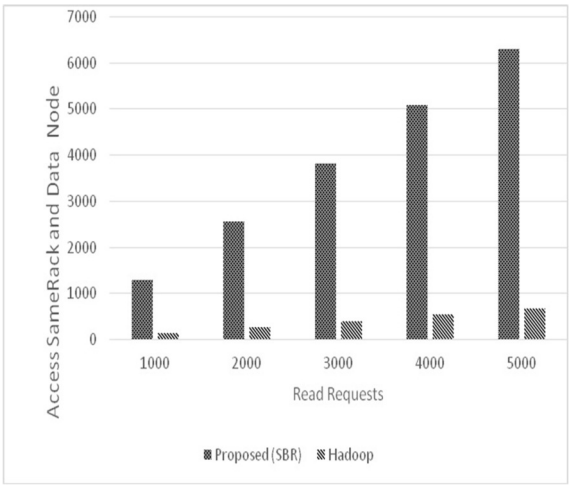

Fig.4: *e) Read request from log is 80 percent and randomly generated 20 percent.*

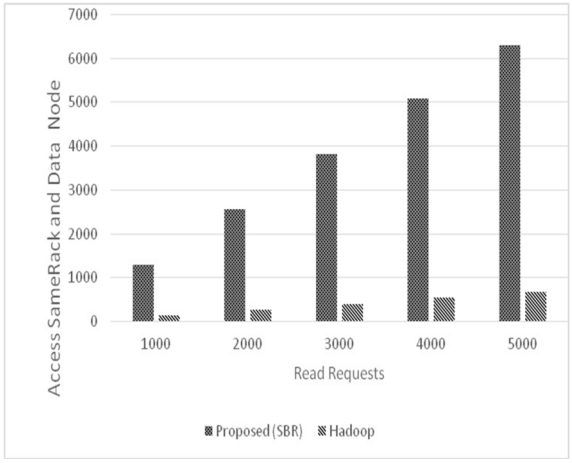

Fig.4: *f) Read Request from log is 100 percent and Randomly generated log zero percent.*

4.5 Comparison of File requests satisfied by Same Rack as different Data node (S.R.Diff.D) for proposed method and Hadoop

Figures 5. (a) through (f) illustrates the file request satisfied by Same Rack and Different Data node for different percents of request taken from log request. There are total 1000 file request and in each file request is generating five random blocks to read then a total of 5000 sub request is generated and the 5000 sub request from Same Rack and Different Data node (SR Diff D).We can observe that the proposed Support Based Replication (SBR) algorithm outperform the HDFS replication algorithm by sending the read request to Same Rack and Different Data node (SRDiffD).The figure 5(a) considers the read request taken from log is of zero percent and randomly generated log read request is of 100 percents. Thus our proposed Support Based Replication (SBR) algorithm outperform the HDFS replication algorithm is highest compared to other data nodes. The figure 5(b) considers the read request taken from log is of twenty percents and randomly generated log read request is of eighty percent.

The Figure 5(c) considers the read request taken from log is of forty percents and randomly generated log log read request is of sixty percent. The figure 5(d) considers the read request taken from log is of sixty percent and randomly generated log read request is of Forty percent. The figure 5(e) considers the read request taken from log is of eighty percent and randomly generated log read request is of twenty percent and the figure 5(f) considers the read request taken from log is of hundred percent and randomly generated log read request is of zero percent. Hence our proposed Support Based Replication (SBR) algorithm outperform the HDFS replication algorithm is highest compared to other data node.

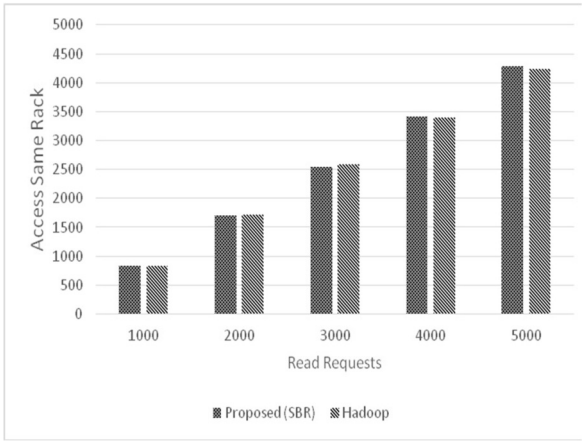

Fig.5: *a) Read request from log is zero percent and randomly generated log 100 percent accessed in Same Rack and Different Data Node.*

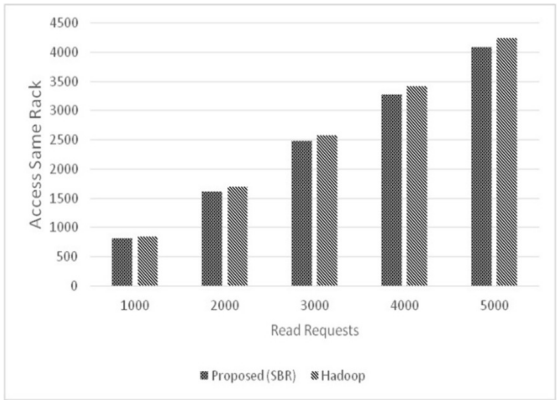

Fig.5: *b) Read request from log is 20 percent and randomly generated log 80 percent accessed in Same Rack and Different Data Node.*

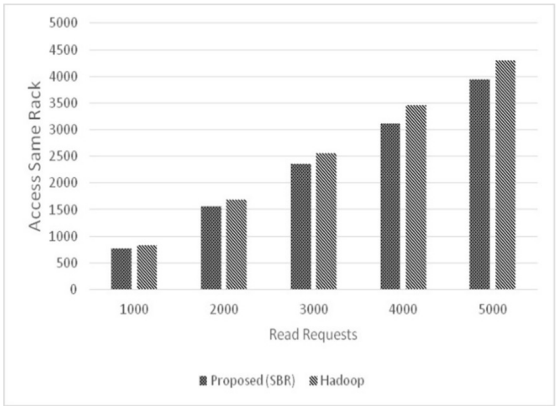

Fig.5: *c) Read requests from log is 40 percent and randomly generated log 60 percent accessed in Same Rack and Different Data Node.*

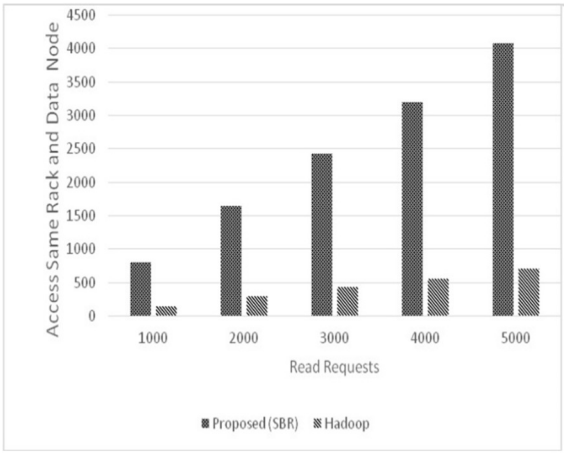

Fig.5: *d) Read request from log is 60 percent and randomly generated log 40 percent accessed in Same Rack and Different Data Node.*

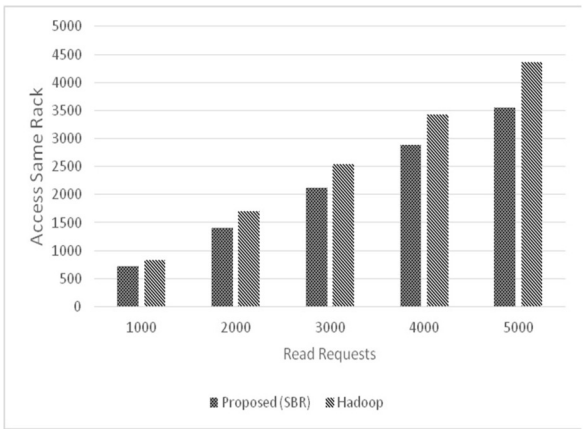

Fig.5: *e) Read request from log is 80 percent and randomly generated log 20 percent accessed in Same Rack and Different Data Node.*

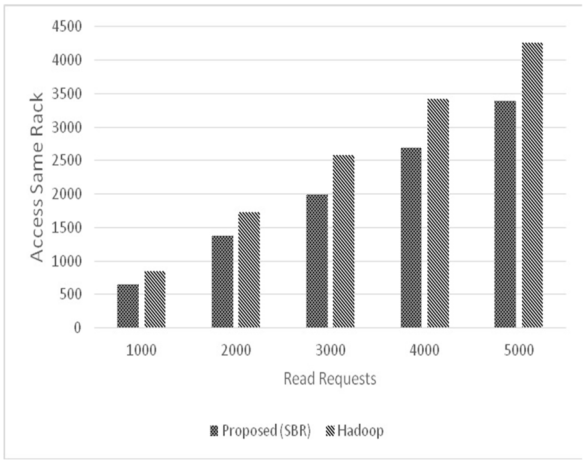

Fig.5: *f) Read request from log is 100 percent and randomly generated log zero percent accessed in Same Rack and Different Data Node.*

Figure.5 (a) through (f) Comparison of File requests satisfied by Same Rack as different Data node(S.R.Diff.D) for proposed method and Hadoop.

4.6 Comparison of File requests satisfied by Different Rack as Different Data node (Diff R.Diff.D) for proposed method and Hadoop

Figures 6. (a) through (f) illustrates the file request satisfied by Different Rack and Different Data node (Diff R Diff D) for different percents of request taken from log request. There are total 1000 file request and in each file request is generating five random blocks to read then a total of 5000 sub requests is generated and The 5000 sub requests from Different Rack and Different Data node (Diff R Diff D) maximum request are accessed in the (Diff R Diff D).We can observe that the proposed Support Based Replication (SBR) algorithm outperform the HDFS replication algorithm by sending the read request to Diff Rack and Same Data node (Diff. R .Diff. D).

The figure 6(a) considers the read request taken

from log is of zero percent and randomly generated log log read request is of 100 percent, Thus our proposed Support Based Replication (SBR) algorithm outperform the HDFS replication algorithm is highest compared to other data nodes. The figure 6(b) considers the read request taken from log is of twenty percent and randomly generated log log read request is of eighty percent The figure 6(c) considers the read request taken from log is of forty percents and randomly generated log log read request is of Sixty percent. The Figure 6(d) considers the read request taken from log is of sixty percent and randomly generated log read request is of Forty percent, The Figure 6(e) considers the read request taken from log is of eighty percent and randomly generated log log read request is of twenty percent and The figure 6(f) considers the read request taken from log is of Hundred percent and randomly generated log log read request is of zero percent .Hence our proposed Support Based Replication (SBR) algorithm outperform the HDFS replication algorithm is highest compared to other data node.

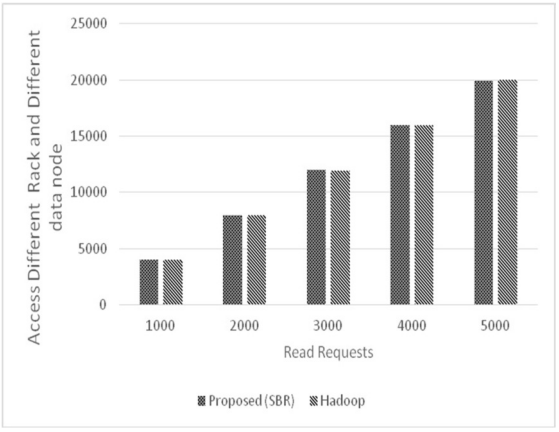

Fig.6: *a) Read request from log is zero percent and randomly generated log 100 percent.*

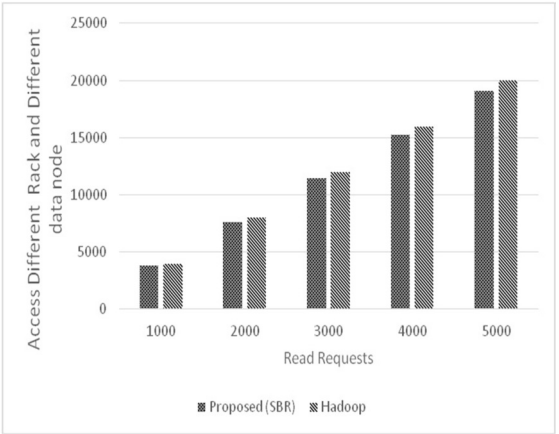

Fig.6: *b) Read requests from log is 20 percent and randomly generated log 80 percent.*

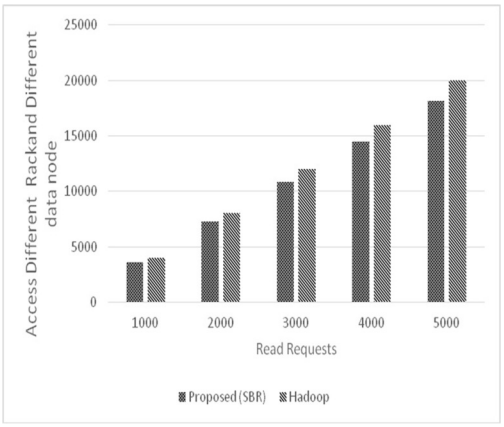

Fig.6: *c) Read requests from log is 40 percent and randomly generated log 60 percent.*

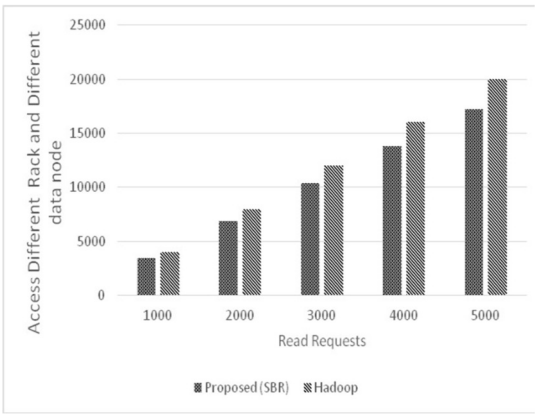

Fig.6: *d) Read requests from log is 60 percent and randomly generated log 40 percent.*

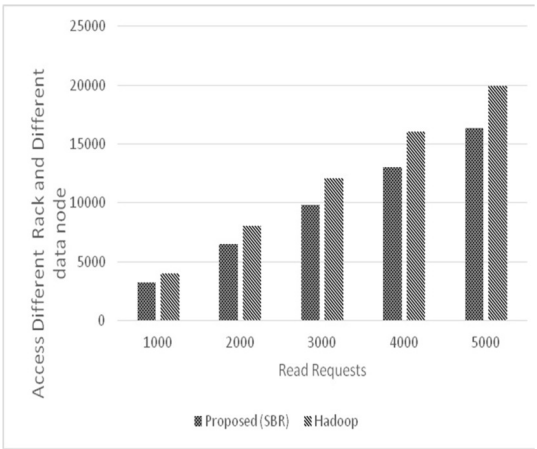

Fig.6: *e) Read requests from log is 80 percent and randomly generated log 20 percent.*

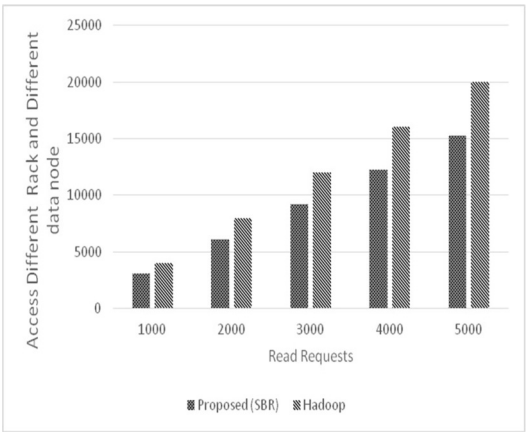

Fig.6: *f) Read request from log is 100 percent and randomly generated log zero percent.*

Fig.6.(a) through (f) Comparison of File requests satisfied by Different Rack as Different Data node (Diff .R.Diff.D) for proposed method and Hadoop.

4.7 Comparison of Average Access time for HDFS and Support Based Replication (SBR) algorithm

Figures 7. (a) through (f) illustrates the average access time of file blocks .It considers the read request taken from log is of zero percent and randomly generated log read request is of 100 percent, Thus our proposed Support Based Replication (SBR) algorithm outperform the HDFS replication algorithm is highest compared to other data nodes, The Figure 7(b) considers the read request taken from log is of twenty percent and randomly generated log read request is of eighty percent.

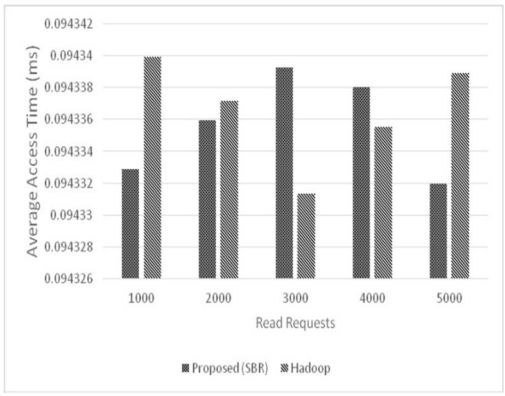

Fig.7: *a) Average Access time calculated from log is zero percent and randomly generated log 100 percent.*

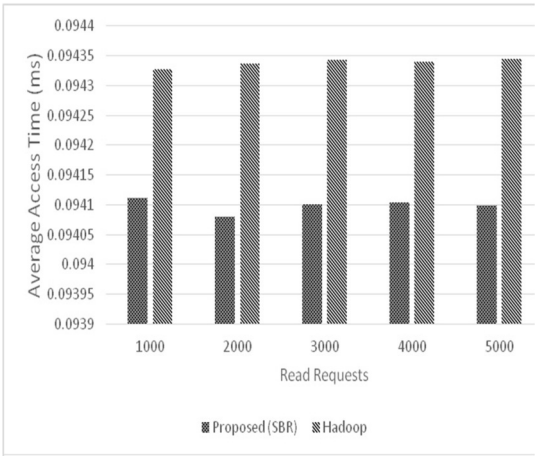

Fig.7: *b) Average Access time calculated from log is 20 percent and randomly generated log 80 Percent.*

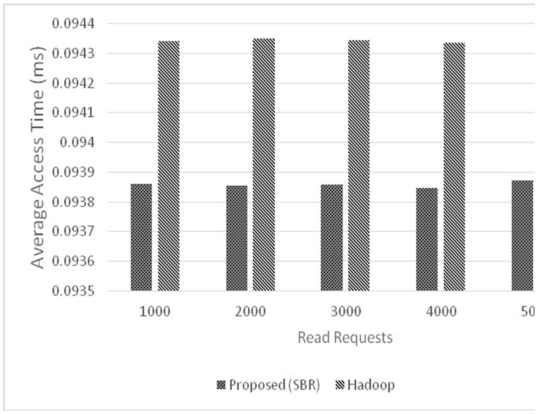

Fig.7: *c) Average Access time calculated from log is 40 percent and randomly generated log 60 percent.*

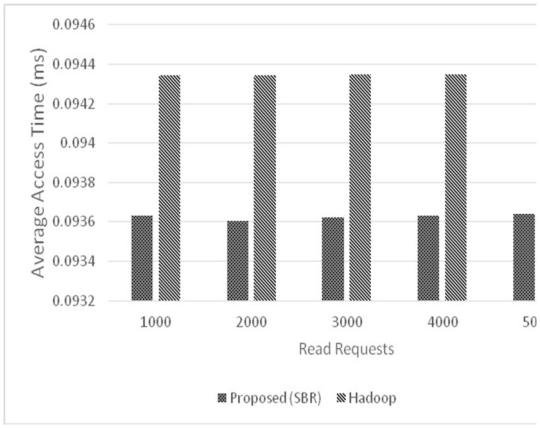

Fig.7: *d) Average Access time calculated from log is 60 percent and randomly generated log 40 Percent.*

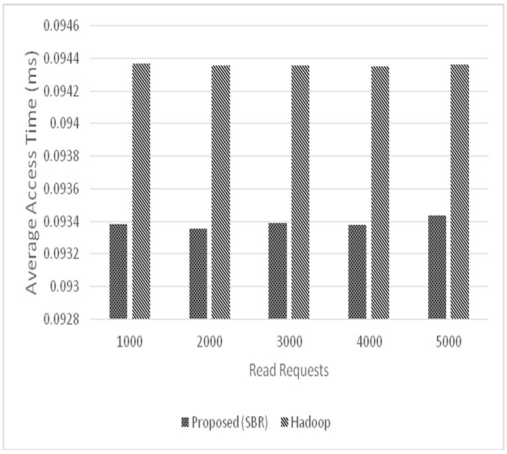

Fig.7: *e) Average Access time calculated from log is 80 percent and randomly generated log 20 Percent.*

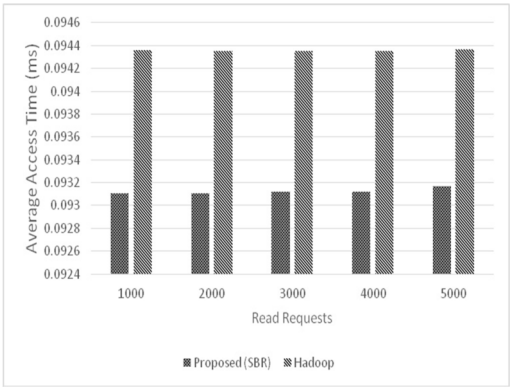

Fig.7: *f) Average Access time calculated from log is 100 percent and randomly generated log zero Percent.*

Figures.7 (a) through (f) Average Access time for HDFS and Support Based Replication (SBR) algorithm The figure 7(c) considers the read request taken from log is of forty percents and randomly generated log log read request is of Sixty cent, The figure 7(d) considers the read request taken from log is of sixty percent and randomly generated log read request is of Forty percent. The figure 7(e) considers the read request taken from log is of Eighty percent and randomly generated log log read request is of twenty percent and The Figure 7(f) considers theread request taken from log is of Hundred percent and randomly generated log read request is of Zero percent. Hence our proposed Support Based Replication (SBR) algorithm outperform the HDFS replication algorithm is highest compared to other data node.

5. CONCLUSIONS

The proposed replication strategy finds an appropriate data node to replicate file blocks to reduce the file read access time and increases the cloud storage system performance. The proposed work validates a replication method for cloud storage systems, the replication factor for a file block is determined based on the file block's support. We implemented both the Hadoop replication strategy and the above said algorithm in java. The results of the experiments show that the proposed method outperforms the Hadoop distributed file system's replication technique.

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, R. Satpathy and T.P. Kumare, "An efficient replication management system for HDFS management," *science direct material proceedings*, july 2021.

[3] A. Shakarami, M. Ghobaei-Arani, A. Shahidinejad, M. Masdari and H. Shakarami , "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," *ICCBDC '20: Proceedings of the 2020 4th International Conference on Cloud and Big Data Computing*, pp. 20-26, 2020.

[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 Systems 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*, vol. 25, pp. 3561-3589, 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 2018*, vol. E101-D, pp. 2958-2967, 2018.

[10] I. A. Ibrahim, W. Dai and M. Bassiouni, "Intelligent Data Placement Mechanism for Replicas Distribution in Cloud Storage Systems," *2016 IEEE International Conference on Smart Cloud (SmartCloud)*, pp. 134-139, 2016.

[11] D. Sun, G. Chang, S. Gao, L. Jin and X. Wang, "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*, `http://Hadoop.apache.org/`

[13] H. Jiawei and K. Michelin, *Data Mining Con-*

cepts and Techniques, 2nd Edition, pp. 23-29, 2007.

[14] T.Ragunathan and M. Sharfuddin, "Frequent block access pattern-based replication algorithm for cloud storage systems," *2015 Eighth International Conference on Contemporary Computing (IC3)*, pp. 7-12, 2015.

[15] Resource & Design Center for Development with Intel. (n.d.). Retrieved June 18, 2019, from Intelwebsite: https://www.intel.com/content/www/us/en/design/resource-design-center.html

[16] Seagate Enterprise Performance 10K HDD Review StorageReview.com-Storage Reviews. (2015,May18).RetrievedJune18,2019,from https://www.storagereview.com/seagate_enterprise_performance_10k_hdd_review

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

[18] Corsair Vengeance LPX DDR4 3000 C15 2x16GB

[19] CMK32GX4M2B202000C15.RetrievedJune18, 2019, from https://ram.userbenchmark.com/Compare/Corsair-Vengeance-LPX-DDR4-3000-C15-2x16GB-vsGroup/m92054vs10F4000_wp_Ethernet.pdf.(n.d.). Retrieved from http://www.force10networks.com/whitepapers/pdf/F4000_wp_Ethernet.pdf Cisco Nexus 5020 Switch Performance in Market-Data and Back-Office Data Delivery Environments. (n.d.). Retrieved June 18, 2019,from Ciscowebsite: 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.

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