ISSN: 0974-5823 Vol. 7 No. 2 February, 2022

International Journal of Mechanical Engineering

Efficient Recovery Method in Erasure Coding-based Distributed File System

[1]Dong-Jin Shin1 and [2]Jeong-Joon Kim

[1]Department of Computer Engineering, Anyang University, South Korea

[2]Department of ICT Convergence Engineering, Anyang University, South Korea

Abstract - The use of distributed file systems to efficiently store big data is increasing. Replication-based distributed file systems have the disadvantage of less storage efficiency than EC-based distributed file systems. However, in EC-based distributed file systems, the recovery throughput and time are slowed due to disk overhead that occurs when accessing the disk while using more nodes and disks. Therefore, in this paper, we propose a Store Random Block and Avoid Disk Concurrent Access method to address this. The proposed method was applied to EC-based distributed file systems to improve recovery throughput and recovery time compared to existing systems.

Index Terms - Replication, Erasure Coding, Distributed File System, Disk Overhead.

INTRODUCTION

Recently, the volume of data generated along with new technologies such as AI(Artificial intelligence), IoT(Internet of things), and blockchain is increasing and the number of fields that are utilized is increasing. Distributed file systems have emerged to store such big data safely and efficiently. Representatively, there are GFS(Google File System), HDFS(Hadoop Distributed File System), AFS(Andrew File system), GlusterFS(Gluster File System), etc. Among many distributed file systems, HDFS is easy to manage as it can be accessed in an open-source form developed through Apache Project. Since it has Map-Reduce technology that supports parallel processing internally, it is efficient for data processing as well as storing through multiple nodes [1].

When storing data, the existing distributed file system uses a replication technique that divides data into blocks of a certain size and then replicates them in K times and distributes them to multiple nodes. The current distributed file system has the advantage of storage efficiency because it uses the storage space more effectively than the replication technique through encoding and decoding by applying the Erasure Coding technique (hereinafter referred to as EC) and occupies less storage disk space [2- 3].

Distributed file systems with EC techniques have improved storage efficiency compared to replication techniques. However, in the event of a failure on the DataNodes storing the data, the disk is accessed more frequently to recover the original data, resulting in a lack of recovery throughput and recovery speed compared to traditional replication techniques [4-5]. Therefore, in this paper, we propose two ways to reduce the overhead of accessing disks when recovering data from a decoding request. Disk access overhead is a phenomenon in which multiple recovery-related daemons are operated to enhance the performance of parallel recovery, resulting in simultaneous access to limited disks.

Two ways to improve disk access overhead problems are to store random blocks and to allow only one disk access to a single disk. Therefore, when storing data blocks and parity blocks through encoding, they are stored randomly, not sequentially. In addition, if a data in the next recovery sequence accesses the disk that is currently recovering, re-insert the file you want to recover into the decode queue table that manages the recovery operation. The proposed disk overhead solution was experimented by applying it to HDFS, an EC-based distributed file system that is open source. Experimental results are shown by comparing recovery throughput with recovery speed during decoding.

Starting with the current description of Introduction, Section 2 Literature review introduces the principles of the distributed file system of replication and EC techniques. Section 3 Proposed Method introduces disk overhead problems arising from EC-based distributed file systems and describes ways to solve them. Section 4 Performance Evaluation compares the two methods proposed in this paper with EC-based HDFS and concludes with Section 5 Conclusion.

LITERATURE REVIEW

Section 2 introduces the replication techniques used in this paper and the principles of data storage and recovery of EC-based distributed file systems and compares the two techniques.

I. Replication-based Distributed File System

When a replication-based distributed file system is requested to store the original data, it is divided into blocks of data of a certain size through existing configured parameters, replicated among k, and distributed to each disk. By default, the value of the

parameter that divides the data block in an open source-based distributed file system is 128 megabytes, and the value of k used for replication is specified as 3 [6]. *Figure 1* shows the process of storing data in a replication-based distributed file system of distributed files.

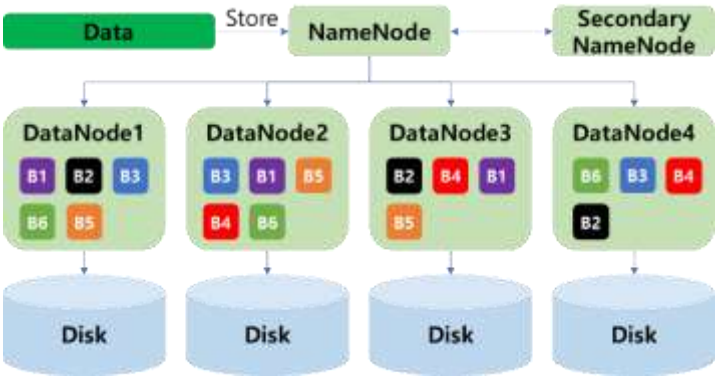

Figure 1 Process of storing data on replication-based distributed file system

In *Figure 1*, there are NameNode for the storage location of data blocks and other metadata management, and DataNodes where data blocks are to be distributed. When data storage is requested, the data is divided into six data blocks: B1, B2, B3, B4, B5, and B6. And because of the triple replication, each block is replicated three by three, and a total of 18 data blocks are distributed and stored in the DataNodes. The SecondaryNameNode acts as a checkpoint for metadata image backup because the location of the data block is unknown in the event of a NameNode failure. *Figure 2* shows the data recovery process through a data read operation.

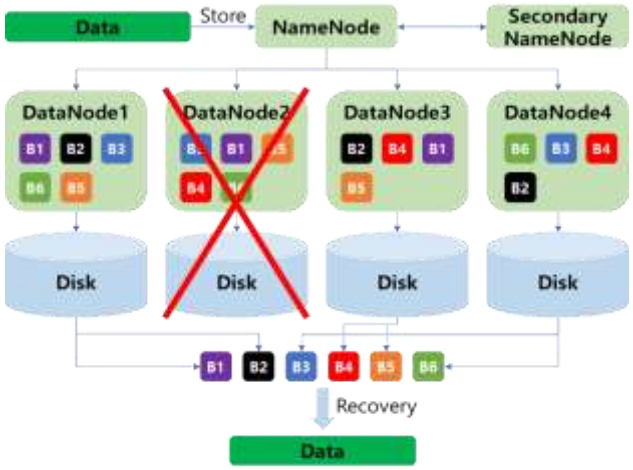

Figure 2 Process of recovering data from replication-based distributed file system

In *Figure 2*, DataNode2 represents the process of recovering the original data using blocks stored on other DataNodes, although the failure occurs and becomes unresponsive. That is, when a data read request starts, the original data can be restored by fetching B1 and B2 from DataNode1, fetching B4 and B5 from DataNode3, and fetching B3 and B6 from DataNode4

II. EC-based Distributed File System

In the EC-based distributed file system, data blocks are encoded and stored differently from the replication technique. Although encoding is the same as the replication technique up to the process of dividing the data block into data blocks, a parity block is generated through encoding without duplicating the divided data block among k. Encoding algorithms for generating parity blocks include a variety of algorithms such as RS(Reed-Solomon), Liberation, and Weaver Code [7-9]. EC is defined by parameters such as the number of data blocks to be stored separately from the algorithms used, and the number of parity blocks to be generated through encoding. For example, RS(6, 3) means that when the original data is stored using the Reed-Solomon algorithm, 6 data blocks and 3 parity blocks are generated and stored through encoding. *Figure 3* shows the data storage process of the EC-based distributed file system configured through RS(6, 3).

When saving the original data is requested, the NameNode is divided into DB1, DB2, DB3, DB4, DB5, and DB6 data blocks through the set number of split parameters. The unstacked data blocks are then stored from DataNode1 to DataNode6, the six DataNodes specified in RS (6, 3). Then, it generates parity blocks PB1, PB2, and PB3 through encoding and is stored on three DataNodes, DataNode7, DataNode8, and DataNode9.

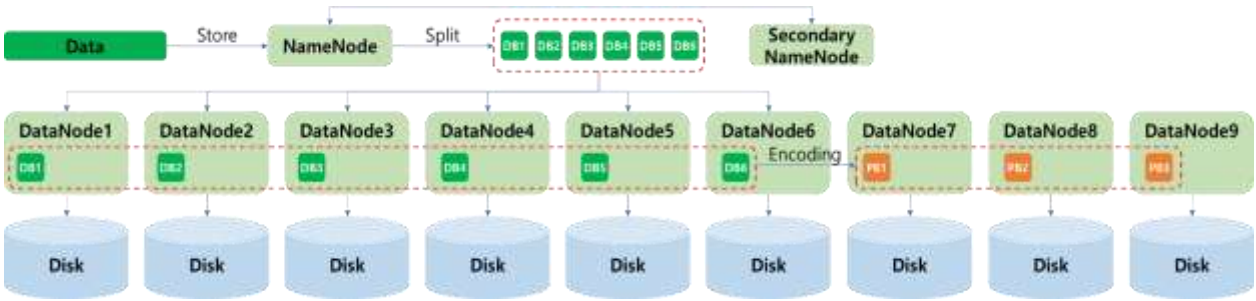

Figure 3 Process of encoding data on EC-based distributed file system

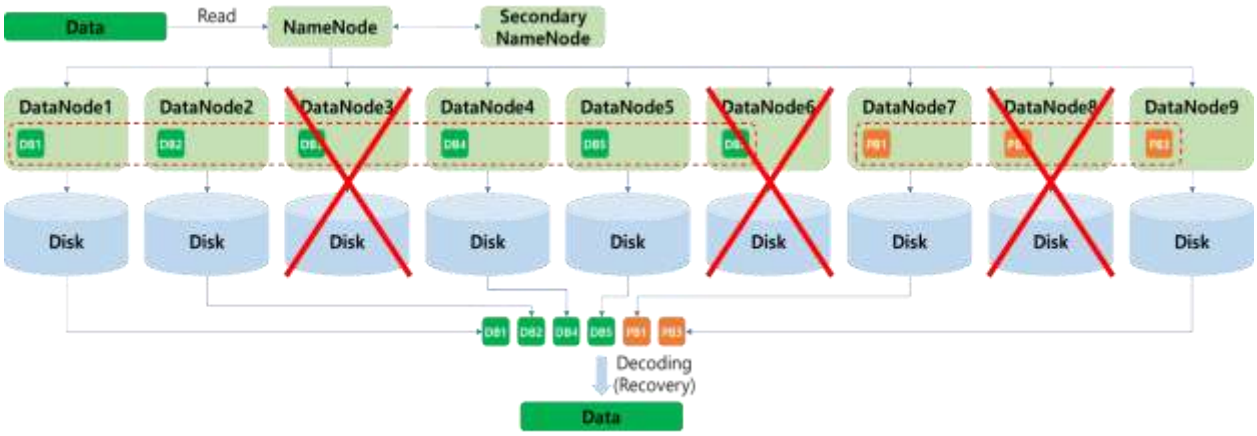

Figure 4 Process of decoding data on EC-based distributed file system

Figure 4 shows the process of recovering data by decoding through a complete data read request in a situation where DataNode3, DataNode6, and DataNode8 fail. When a data read request starts, the original data is restored by decoding the blocks from DataNodes that have data blocks and parity blocks except for the failed DataNodes.

III. Comparison of replication and EC

In the distributed file system using the replication and EC techniques described in *Figures 1 and 3*, the storage efficiency is calculated by Equation (1), and the comparison is shown in *Table 1* [10].

$$Storage\ \ Efficiency = \frac{K}{K+M} \qquad (1)$$

$$K = Data\ \ Blocks$$

$$M = Number\ of\ blocks\ required\ for\ recovery$$

Table 1 Comparison of replication and EC

Technique	Data Durability	Store Efficiency
Three-way Replication	2	33%
RS(6, 3)	3	67%

Table 1 means the storage efficiency calculated through Equation (1) in the replication techniques and the EC techniques. Data Durability means the maximum number of failed DataNodes. In case of triple replication, data can be recovered up to two DataNode failures, and RS(6, 3) can recover data up to three DataNodes failures. The storage efficiency of triple replication is

33% (3÷(3+6)=0.33) because when data is stored, 3 data blocks are created, and 6 data blocks required for recovery. However, in the case of RS(6, 3), when data is stored, 6 data blocks and 3 parity blocks necessary for recovery are generated, so it is calculated as 67% (6÷(6+3)=0.67). Comparing the two techniques shows that EC techniques are storage efficient by saving approximately twice the disk space.

METHOD

Section 3 analyses the problems that contribute to the two methods proposed in this paper and describes solutions.

I. Disk Access Overhead Issues

This section discusses the problems caused by disk overhead during decoding in EC-based distributed file systems.

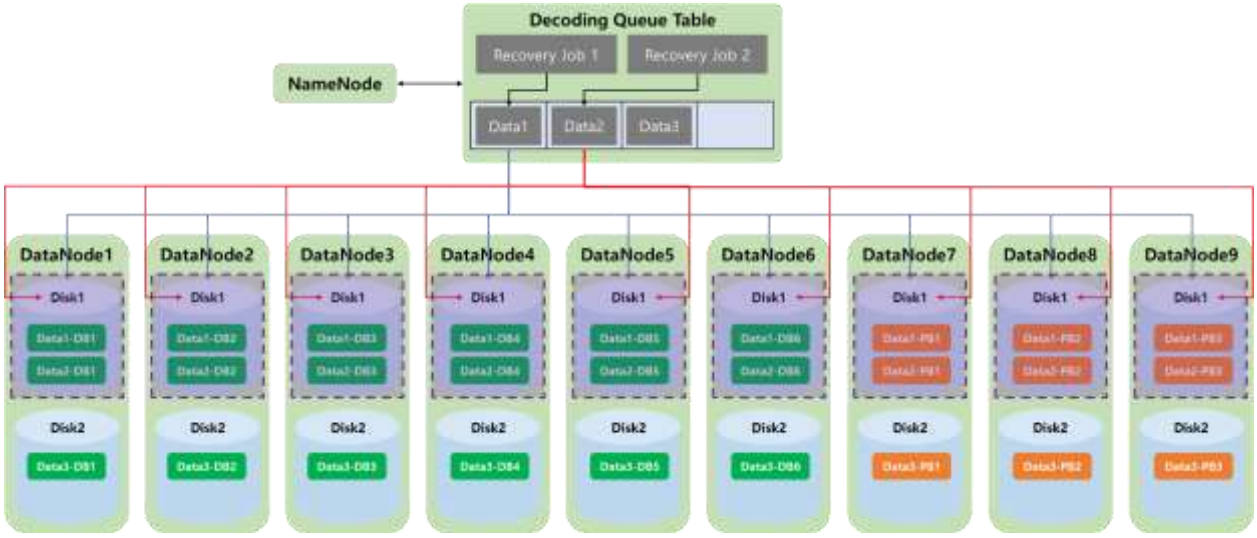

Figure 5 Disk access overhead issues with EC-based distributed file systems

Figure 5 shows disks of all DataNodes accessed during decoding operations according to data read requests in EC-based distributed file system structures using RS(6, 3). Recovery Jobs that perform parallel recovery select the data to be recovered from the Decoding Queue Table. In other words, parallel recovery causes Data1 and Data2 corresponding to Recovery Job1 and Recovery Job2 to perform simultaneous recovery operations, and Data3 is waiting for the next recovery destination. At this time, to recover Data1 and Data2, disk overhead (represented by a purple dotted rectangle) occurs while accessing Disk1 of all DataNodes at the same time. Therefore, in a typical EC distributed file system, these parallel recoveries present disk overhead problems because they store the data blocks and parity blocks generated by encoding sequentially. To address these challenges, this paper proposes two methods: Store Random Block and Avoid Concurrent Disk Access.

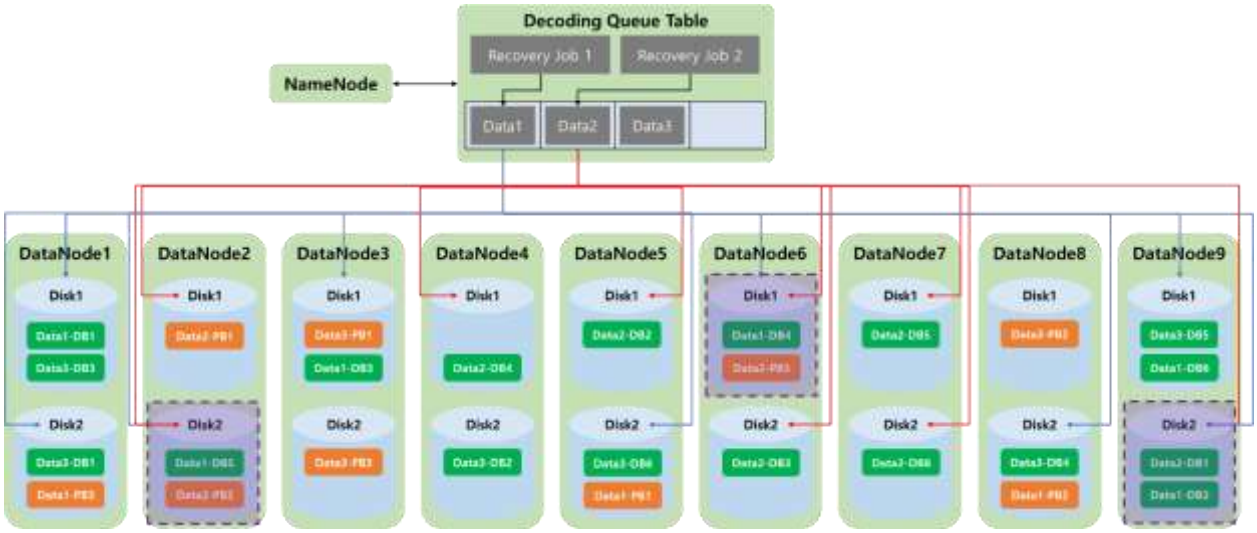

Figure 6 EC-based distributed file system with store random block applied

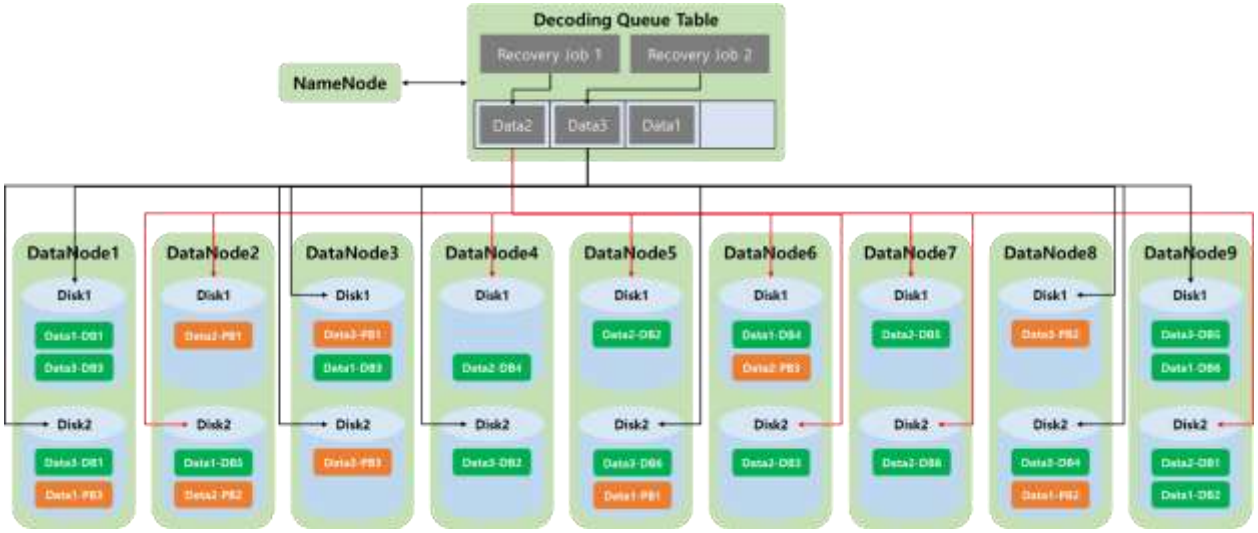

Figure 7 EC-based distributed file system with avoid concurrent disk access applied

II. Store Random Block

In EC-based distributed file systems, data blocks and parity blocks are basically stored through Store Sequential Block as shown in *Figure 5*. However, Store Random Block can somewhat solve the problem of multiple Recovery Jobs accessing disks at the same time due to the sequentially stored blocks described earlier. In other words, when data blocks and parity blocks are stored through encoding, they are randomly stored on disk within the DataNodes. When data is recovered using Store Random Block, the probability of simultaneous disk access shown in *Figure 5* can be reduced, reducing the frequency of disk overhead occurrence. The parallel recovery process of EC-based distributed file systems with Store Random Block is shown in *Figure 6*.

In *Figure 5*, there are nine disk overhead problems according to the Store Sequential Block method, occurring on Disk1 of all DataNodes. However, in the EC-based distributed file system to which the Store Random Block shown in *Figure 6* is applied, there are three disk overhead problems in Disk2 of DataNode2, Disk1 of DataNode6, and Disk2 of DataNode9, which is more efficient than the sequential block storage method. Therefore, the overhead frequency is reduced, and the recovery throughput and recovery speed can be improved because the disks of other DataNodes are utilized for parallel recovery operations.

III. Avoid Concurrent Disk Access

Avoid Concurrent Disk Access utilizes queues managed by Decoding Queue Table associated with NameNode. If Recovery Job1 occupies and uses the disks needed for recovery when recovering data in parallel due to a decoding request, Recovery Job2 checks whether the disks needed for recovery are used for recovery and proceeds with the recovery. In other words, if there is a case of accessing the disk already being used for recovery, it is a method of avoiding this and delaying the recovery request to the back of the queue managed in the Decoding Queue Table.

In *Figure 6*, Data1 and Data2 to be restored in parallel have less overhead through Store Random Block than the existing Store Sequential Block method, so more disks are participating. *Figure 7* shows the parallel recovery process of the EC-based distributed file system to which Avoid Concurrent Disk Access is applied. Data1, which is executed for recovery, is moved to the back of the decoding queue table, and the next sequence, Data2 and Data3, is first started in parallel recovery. In *Figure 6*, which uses Store Sequential Block method, three disk overhead problems occurred. However, the EC-based distributed file system with Avoid Concurrent Disk Access in *Figure 7* can completely reduce disk overhead because it checks the use of disks and changes the order of data to be recovered in parallel

In EC-based distributed file

The algorithm processing and implementation code for Avoid Concurrent Disk Access proposed in this paper is shown in *Figure 8*.

```
1  Start
2  Recovery_Job Data ← get Data from Decoding_Queue_Table
3  Data_Layout ← get Data_Layout from Metadata_Namenode by Recovery_Job
4
5  For Each block of Data_Layout
6  Int check_disk ← get disk_id from block
7  Int key ← get hash key by check_disk
8  If (used_disk_table[key] ← 1)
9  add Data to Decoding_Queue_Table
10 GO TO START
11 End If
12 End For
13
14 Recovery_Job Start and Data Recovery
15
16 For Each block of Data_Layout
17 Int check_disk ← get disk_id from block
18 Int key ← get hash key by check_disk
19 used_disk_table[key] ← 0
20 End_for
21
22 update Data_Layout to Metadata_Namenode
23 End
```

Figure 8 Pseudo code with avoid concurrent disk access applied

In the Decoding Queue Table of the EC-based distributed file system, assign the information of the data to the Recovery Job to the recovery job and obtain the layout of the data through Metadata_NameNode to determine which blocks to recover (Line 1-3). Verify that the disk where the blocks that need to be recovered in Data_Layout are in use, and if the disk is in use ("1"), put the data that you want to recover back into the Decoding Queue Table and restart it from scratch (Line 5-12). When a decoding request is entered, the recovery operation starts, and when the recovery is complete, the disks in use are marked as unused ("0") (Line 14-20). The last changed Data_Layout is re-saved to Metadata_NameNode and updated with new information (Line 22-23).

RESULTS

In this section, for performance evaluation, the EC-based distributed file system with Store Random Block and Avoid Concurrent Disk Access proposed in this paper is named EC-HDFS-SA, and the performance evaluation is compared with the basic EC-HDFS.

The environment configured for performance evaluation consisted of one NameNode and nine DataNodes through RS(6, 3). Therefore, six data blocks can be stored, three parity blocks can be stored, and the encoding algorithm uses Reed-Solomon. EC-HDFS-SA operates in virtual environment simulations, and as a detailed specification, CPU is XEON 4110 (8 core x 2), memory is 128GB DDR4 and hard disks are nine (1TB 7200 rpm) Ubuntu 20.04.4 operating systems. Each DataNode consists of two disks through partitioning. In EC-HDFS-SA, the block size to be stored on each disk was set to 128 MB, and 500 dummy data of 1152 MB in size were created using the dd command of Linux and stored in EC-HDFS-SA.

The main title is centred, and in times new roman 14-point, boldface type. Only the first letter of the first word in the title needs to be capitalized except for the letters and words that are originally capitalized. Leave one blank lines after the title.

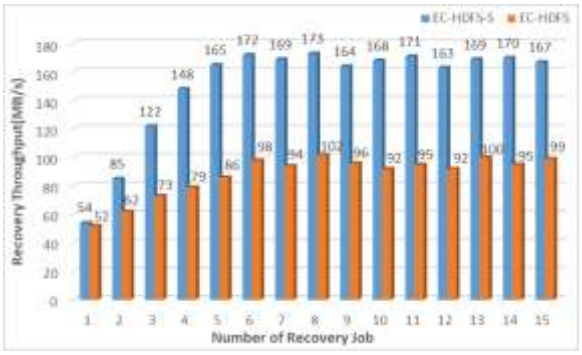

Figure 9 Recovery throughput with EC-HDFS-S and EC-HDFS on number of recovery jobs

Figure 9 shows the recovery throughput over the increasing number of parallel recoveries in EC-based distributed file systems with Storm Random Blocks. EC-HDFS does not increase recovery throughput because it utilizes Store Sequential Block method, but EC-HDFS-S uses Store Random Block, which increases recovery throughput because it accesses more disks and runs parallel recovery. A maximum of 173MB/s is measured, and the recovery throughput does not increase from the moment the Recovery Job is executed 6 times. This is because each DataNode has two disks designated, limiting the increase in recovery throughput. Performance improvements are expected to be higher if the number of disks used by Recovery Job for parallel recovery is increased by increasing the number of disks on each DataNode.

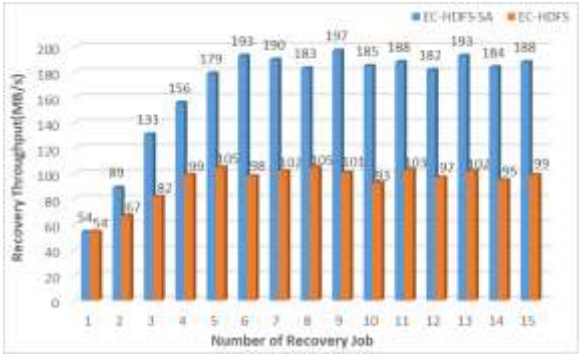

Figure 10 Recovery throughput with EC-HDFS-SA and EC-HDFS on number of recovery jobs

Figure 10 shows the recovery throughput over the increasing number of parallel recoveries in EC- based distributed file systems with Storm Random Block and Avoid Concurrent Disk Access. EC-HDFS is similar to *Figure 8,* so there is little change, and EC-HDFS-SA even applies Avoid Current Disk Access to avoid disk overhead as much as possible, skipping to the next parallel recovery data. Therefore, the recovery throughput is slightly higher than EC-HDFS-S because more disks are accessed. Capacities of up to 197 MB/s were measured, and the recovery throughput uniformity problem from the point of 6 recovery operations is the same as in *Figure 9*.

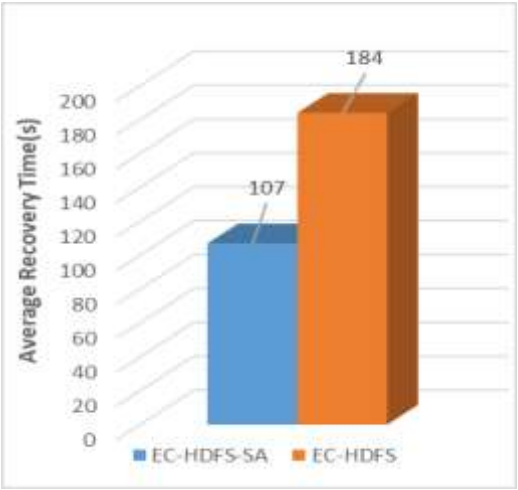

Figure 11 Average recovery time of 15 decoding requests for EC-HDFS-SA and EC-HDFS

Figure 11 shows a recovery time comparison between EC-HDFS-SA and EC-HDFS. Due to the decoding operation, the data used for parallel recovery were randomly requested for 15 pieces, and the average recovery time was graphedEC-HDFS-SA measured an average of 107 seconds for 15 decoding requests, and EC-HDFS measured 184 seconds for the same number of decoding requests. The average recovery time of EC-HDFS-SA was measured about 0.7 times faster than that of EC-HDFS.

CONCLUSION AND FUTURE WORK

The EC-based distributed file system increases storage efficiency compared to the replication technique, but disk overhead occurs as more disks are used. Therefore, in this paper, we propose ways to improve the disk overhead problem arising from EC-based distributed file systems that can efficiently store data. Store Random Block reduces disk overhead compared to the existing system because blocks are randomly placed and stored.

In addition, Disk Avoid Concurrent Access uses the Decoding Queue Table to check whether a disk is in use and to recover the data to be recovered next in parallel recovery, so disk overhead can be avoided. Comparing EC-HDFS-SA using Store Random Block and Avoid Concurrent Disk Access and conventional EC-HDFS, recovery throughput increases because more disks are utilized. In addition, the recovery time was also measured about 0.7 times faster due to the increase in throughput. In the future, research will be conducted that can be tested and improved in various environments by applying suggestions to various EC-based distributed file systems such as GlusterFS and GFS.

CONFLICTS OF INTEREST

- the authors have no conflicts of interest to declare.

REFERENCES

[1] Blomer J. A survey on distributed file system technology. In Journal of Physics: Conference Series, 2015; 608(1):012039.

[2] Weatherspoon, H. and Kubiatowicz, J. D. 'Erasure coding vs. replication: A quantitative comparison', In International Workshop on Peer-to-Peer Systems 2002 (pp. 328-337). IPTPS.

[3] Rodrigues R, Liskov B. High availability in DHTs: Erasure coding vs. replication. In International Workshop on Peer-to-Peer Systems 2005 (pp. 226-239). IPTPS.

[4] Shen J, Zhang K, Gu J, Zhou Y, Wang, X. Efficient scheduling for multi-block updates in erasure coding based storage systems. IEEE Transactions on Computers, 2017; 67(4):573-581.

[5] Fu Y, Shu J, Shen Z, Zhang G. Reconsidering single disk failure recovery for erasure coded storage systems: Optimizing load balancing in stack-level. IEEE Transactions on Parallel and Distributed Systems. 2015; 27(5):1457-1469.

[6] Shvachko K, Kuang H, Radia S, Chansler R. The hadoop distributed file system. 26th symposium on mass storage systems and technologies 2010 (pp. 1-10). MSST.

[7] Plank J S. A tutorial on Reed–Solomon coding for fault-tolerance in RAID-like systems. Software: Practice and Experience. 1997; 27(9):995-1012.

[8] Plank J S. The raid-6 liber8tion code. The International Journal of High Performance Computing Applications. 2015; 23(3):242-251.

[9] Hafner J L. WEAVER Codes: Highly Fault Tolerant Erasure Codes for Storage Systems. USENIX Conference on File and Storage Technologies 2005 (pp. 16-16). FAST.

[10] Introduction to HDFS Erasure Coding in Apache Hadoop. https://blog.cloudera.com/introduction-to-hdfs-erasure-coding-in-apache-hadoop/. Accessed 17 December 2021.