

RESEARCH ARTICLE

FACHS: Adaptive Hybrid Storage Strategy Based on File Access Characteristics

YING SONG^{1,2,3}, QIANG ZHANG^{1,2}, AND BO WANG⁴¹Beijing Key Laboratory of Internet Culture and Digital Dissemination, Beijing Information Science and Technology University, Beijing 100101, China²Beijing Advanced Innovation Center for Materials Genome Engineering, Beijing Information Science and Technology University, Beijing 100101, China³State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100086, China⁴Software Engineering College, Zhengzhou University of Light Industry, Zhengzhou 450002, China

Corresponding author: Ying Song (songying@bistu.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61872043; and in part by the State Key Laboratory of Computer Architecture, ICT, CAS, under Grant CARCHA202103.

ABSTRACT With the widespread use of distributed systems and the development of big data technology, the storage redundancy, data availability and persistence of distributed systems have become a concern. Traditional multi-copy storage may cause large storage redundancy, thus wasting storage space. The erasure code is considered as the best alternative to the replica strategy. However, the existing methods do not fully consider the access characteristics of files, so that the data with high access popularity are stored using computationally complex erasure codes, resulting in poor parallel read and write performance. Moreover, due to insufficient consideration of file size, storage redundancy cannot be minimized, thus wasting storage space. Therefore, we propose an adaptive hybrid storage strategy based on file access characteristics called FACHS. For cold files, namely the files with low access frequency, we use RS Code (Reed Solomon Code) to store them. RS Code has low computing and storage costs. We use multi-copy and LRC code (local reconstruction code) to store small and large size hot files respectively. Multi-copy ensures the efficiency of file read/write, as well as the ability of parallel read/write. The recovery cost of LRC code in case of node failure is very low. The experimental results show that compared with the existing methods, FACHS can reduce the storage space occupation by 12% for cold files, and improve the read/write speed by 8% and the recovery efficiency by 29% for hot files.


INDEX TERMS Access characteristics, adaptive storage, erasure code, multi-copy.

I. INTRODUCTION

Now we live in a big data era [1]. With the development of information technology, big data technology is increasingly widely used in all walks of life [2]. How to reduce the storage redundancy of large-scale data, improve data availability, persistence and data recovery efficiency are major challenges [3] for distributed systems [4] in the big data scenario. Hadoop [5] is the most widely used distributed system at present. It has a distributed file system called HDFS [6] for storing large-scale data and a parallel computing model MapReduce [7] for large-scale data processing. Nowadays, many distributed systems [8] adopt multi-replica strategy to

effectively solve the problem of storage fault tolerance. Many versions of Hadoop use the multi-copy policy; however, the multi-copy policy may lead to excessive storage redundancy.

Therefore, nowadays, more and more distributed storage systems use erasure codes instead of multi-copy strategy [9]. Typical erasure codes are RS Code and LRC Code (LRC adds local check blocks on the basis of RS), which can provide the same level of fault tolerance with less storage redundancy. In a variety of erasure code types, the storage redundancy does not exceed 70%. Files stored with erasure codes will be cut into k blocks, and then m parity blocks are generated by encode. When a node fails, the erasure code is decoded to obtain the surviving data block information, and then the surviving block is transmitted to the reconstruction node for data recovery. Since it takes a long time for erasure

The associate editor coordinating the review of this manuscript and approving it for publication was Jjun Cheng .

code to recover data, all data blocks are placed in different data nodes to minimize data block loss in case of node failure.

However, the application effect and data recovery efficiency of erasure codes technology in Hadoop need to be improved. Many scholars are also studying how to improve the performance of data storage and recovery to better apply erasure codes. After all, as far as storage redundancy is concerned, erasure codes still have great potential in this field, and they have natural advantages over multi-copy strategy [10].

Distributed storage systems host files with varying heat and sizes. Files with different access characteristics have different performance requirements. The files with high popularity need high read/write efficiency and recovery efficiency, and the corresponding number of files is also small. The files with low heat need low storage redundancy to ensure the storage efficiency of the system, and the corresponding number of files is also large. The files of different sizes in the cluster also need appropriate storage methods, otherwise the storage redundancy cannot be minimized, resulting in a waste of storage space.

Nowadays, a single erasure code type or storage strategy has been difficult to meet the storage requirements of files with different access characteristics in distributed systems. Therefore, more and more scholars propose and study adaptive storage strategies. However, the existing methods do not fully consider the characteristics of file access. For example, DECPA [16] only considers the file size, which may lead to the use of erasure codes with low computational efficiency to store data with high access frequency, resulting in poor parallel read and write performance. In other methods [15], [17], [18], the access efficiency and recovery efficiency of hot files cannot be improved because the file size and heat are not considered. In the existing methods mentioned above, the file access characteristics have not been fully considered. This makes files with high access frequency adopt inappropriate encoding and storage methods, resulting in poor parallel read/write performance or high recovery costs [19], or files with low access frequency use erasure code with high storage redundancy [20], [21], resulting in a waste of storage space. How to combine different coding and storage methods, make full use of the advantages of different types of coding and storage methods, avoid their disadvantages, and form an efficient adaptive storage scheme is the main content of our research.

Therefore, we propose an adaptive hybrid storage strategy based on file access characteristics, FACHS. FACHS selects the appropriate storage mode according to the size and heat of the file to improve the cluster read/write efficiency and recovery efficiency as well as maintain low storage redundancy. The experimental results show that compared with the existing methods, FACHS can reduce the storage space occupation by up to 12% for cold files, and improve the read/write speed by up to 8% as well as the recovery efficiency by up to 29% for hot files.

The organization of this paper is as follows. In Section II, the related work and our motivation are introduced. In Section III, we describe the design of FACHS in detail. In Section IV, we evaluate FACHS through a set of analysis and experiments. Section V summarizes our work and proposes the future research directions.

II. RELATED WORK

This section first briefly introduces the basic erasure codes, and then focuses on the work related to adaptive hybrid storage.

A. BASIC ERASURE CODES

The files stored in the erasure codes will be cut into multiple stripes, with k blocks for each stripe, and then multiple parity blocks will be generated through coding. When a node fails, the erasure code is decoded to obtain the surviving data block information, and then transmits it to the reconstruction node for data recovery. Since it takes a long time for erasure code to recover data, all data blocks are placed in different data nodes to minimize data block loss in case of node failure [11]. Typical erasure codes include RS Code and LRC Code.

1) REED-SOLOMON CODE (RS CODE)

RS Code was proposed by Irving S. Reed and Gustave Solomon in 1960. It is a coding algorithm based on finite fields, also known as Galois Field. It is named after Galois, a famous French mathematician. It was first used in the field of communication. After decades of development, it has been widely used in storage systems. RS Code [12] is a maximum distance separable (MDS) code [13] with high storage efficiency. Its encode and decode operations are based on the Galois field. The specific layout of RS Code can be expressed as $RS(k, m)$, where k and m represent the number of data blocks and the number of parity blocks, respectively. RS Code encodes k data blocks into m parity blocks, so that when any one of the $k + m$ blocks is lost, k blocks can be arbitrarily selected from other surviving blocks for reconstruction (as long as the number of lost blocks does not exceed m blocks), and $k + m$ blocks together form a stripe, and each block is stored on different nodes.

2) LOCAL RECONSTRUCTION CODE (LRC CODE)

LRC Code is a non MDS code [13]. The specific layout of LRC Code can be expressed as $LRC(k, y, z)$, where k , y and z represent the number of data blocks, the number of local parity blocks and the number of global parity blocks, respectively. $LRC(k, y, z)$ encode k data blocks into a group, and further divide the k data blocks into y local groups on average. Each local group includes k/y data blocks, and calculates a local check block respectively. Then $LRC(k, y, z)$ calculates z global check blocks from all k data blocks. When a block is lost (not a parity block), it can be reconstructed by transmitting other data blocks in the data block group of the lost block and their corresponding local parity blocks. The

$k + y + z$ blocks together form a stripe, and each block is stored on a different node.

B. ADAPTIVE HYBRID STORAGE SCHEME

Now there are many researches on adaptive hybrid storage strategies, including adaptive hybrid strategies based on the same type of erasure codes and adaptive hybrid strategies based on multiple types of erasure codes. Next, we introduce the relevant work in this field.

Jinping et al. proposed a LRC-RS hybrid coding strategy [14] to solve the problem of high repair bandwidth and stable node resource waste caused by the application of low rate RS (Reed Solomon code) codes with the same parameters in decentralized storage. By classifying nodes according to their confidence level, high trusted nodes can use LRC coding, while low trusted nodes can use RS coding with low code rate to reduce the repair bandwidth and storage overhead of stable nodes.

The HACFS [15] proposed by Xia et al. is a classic hybrid storage scheme for erasure codes of the same type, using two erasure codes based on XOR series, LRC Code and Product Code. HACFS adjusts the code structure in the system according to the change of workload. It uses a fast code to optimize recovery performance, and a compact code to reduce storage redundancy. A novel conversion mechanism is adopted to efficiently encode data blocks up and down between fast code and compact code.

Chiniah et al. [16] designed and implemented the dynamic erasure code policy allocation - DECPA based on the minimum storage redundancy, which is an adaptive hybrid storage scheme of erasure code and multi-copy, realizing the hybrid storage of RS code and multi-copy with different parameter types. Three different algorithms are implemented to adapt to various application scenarios of single strip and multiple strips, and effectively improve the storage efficiency of the system. However, because the file heat is not considered, the files with different heat may not have an appropriate storage mode, resulting in a reduction in the cluster's read/write efficiency and data recovery efficiency.

Qiu et al. [17] proposed an efficient hybrid erasure code storage framework EC Fusion (Erasure Codes Fusion) for cloud storage systems to solve the problem that front and back office workloads can simultaneously and efficiently process in parallel. EC Fusion is a hybrid storage of RS Code and MSR Code. It dynamically selects the appropriate code according to the application workloads. For write intensive application workloads, RS Code is used to reduce both computing overhead and storage costs. For read intensive application workloads, using MSR Code can improve recovery efficiency. Therefore, better overall application performance and recovery performance can be achieved in a low-cost way.

On the basis of MSR Code and LRC Code, Xie et al. [18] design the coding and decoding algorithm of AZ Code, use the MSR Code with a specific parameter to generate

a local parity check block, and use the RS Code with a specific parameter to generate a global parity check block. This enables AZ Code to have high recovery efficiency and storage efficiency.

In the above existing approach, whether based on the internal hybrid coding of erasure codes, or the hybrid storage strategy of two erasure codes, or the hybrid storage strategy of erasure codes and multi-copy, failed to fully consider the file access characteristics. This makes the files with high access frequency adopt inappropriate encode and storage methods, resulting in poor parallel read/write performance or high recovery cost [19], or the files with low access frequency use correction codes with high storage redundancy [20], [21], resulting in a waste of storage space.

III. FACHS STRATEGY DESIGN

To improve the storage efficiency, parallel read/write ability and recovery efficiency of distributed systems, we analyze the size and heat distribution of files in the cluster, and design an adaptive hybrid storage strategy called FACHS based on file access characteristics. In this section, we first introduce the design concept of FACHS. In order to better analyze the access differences of files in the cluster, we define the expression of file heat. Next, we describe how to adopt different storage strategies according to the access characteristics of files. Finally, we introduce the implementation process of FACHS in detail.

A. DESIGN CONCEPT

In a distributed system, the size of files varies. Some files may be smaller than one stripe, while others may occupy multiple stripes. The file size also directly affects the storage redundancy caused by using erasure codes and multi-copy. The same is true for the popularity of files. The access frequency of files in the cluster is different. Only a small number of files may be frequently accessed in a short period of time, while other files may only have a small amount of access for a long time. If erasure code with low coding and decoding efficiency is used to store hot data, the read/write efficiency of the cluster will be affected. In addition, if the selected erasure code or storage mode is improper, when the data node where the hot data is located fails, it will not only cause low data recovery efficiency and high recovery cost, but also further worsen the high concurrency overload problem of the cluster, resulting in read delay.

Therefore, we believe that using efficient encoding and storage methods for hot files can improve the read/write performance and recovery performance of the cluster, while cold files only need to be encoded and stored at low storage costs to reduce the storage redundancy of the cluster. Large files require a single stripe to accommodate more encoding of data blocks to reduce the number of stripes. Small files need appropriate stripe size to reduce storage redundancy. We use different strategies to design FACHS according to different file size and heat.

B. HEAT CALCULATION AND CLASSIFICATION

Through research and analysis, we know that the popularity of files in distributed clusters depends on the frequency of access within a certain period of time. New files are often accessed frequently at the initial stage of creation, with a high frequency [25] of access, and most of them will be accessed less frequently over time. Therefore, this will affect the determination of file access frequency. Therefore, it is important to determine the access skew rate of the time period, so that new files can be distinguished and the access frequency of files can be calculated more accurately. Combining the real environment and the above analysis, we propose a more accurate heat calculation formula.

First, we calculate the average access frequency f_{avg} of each file. Set the heat statistics time cycle to T , and the total number of access to the file in the cycle is N . The formula for calculating the average access frequency f_{avg} in the cycle is as follows.

$$f_{avg} = \frac{N}{T} \quad (1)$$

After obtaining the average access frequency f_{avg} of the file, we will calculate the access slope I of the file during the period. Each statistical cycle T is divided into n periods $\{t_1, t_2, \dots, t_n\}$, where t_1 is the first period of the current statistical cycle, t_n is the last period of the current statistical cycle, and access weight $\lambda = 1/i$, the access frequency of t_i period is f_{ti} . That is, the access weight proportion in the period closer to t_n is higher, and the access weight proportion in the period closer to t_1 is lower. In this way, we can get the truest reflection of the access frequency in the cycle and avoid being affected by new files. The calculation formula of access slope I is as follows:

$$I = \left(\sum_{i=1}^n \frac{f_{ti}}{i} \right) / T \quad (2)$$

We set the heat of the file to H , calculate the average access frequency f_{avg} and access slope I of the file, and assign a balance factor β to the average access frequency of the file, Then the file access slope balance factor is $(1 - \beta)$, The heat H is calculated as follows:

$$H = (1 - \beta)I + \beta f_{avg} (0 < \beta < 1) \quad (3)$$

After the heat value of the document is calculated, we classify the heat of the document. In daily life, many phenomena conform to the law of Zipf distribution [22], including the frequency of words, the frequency of web page visits, etc. In a large-scale distributed cluster, it also conforms to this rule. 80% of users' access to the cluster is usually concentrated on 20% of the files. Therefore, in our design, we use the heat calculation formula mentioned above to calculate the heat value, and then count the top 20% of the files ranked by heat as hot files, and the remaining 80% of the files are defined as cold files.

TABLE 1. Units for magnetic properties.

	Storage Efficiency
Replica	0.33
RS(3,2)	0.60
RS(6,3)	0.67
LRC(6,2,2)	0.60

C. FILE SIZE CLASSIFICATION

We use the size of three data blocks (the default size of a data block in Hadoop 3.0 is 128MB) as the threshold of file size classification. Because when the file is smaller than three data blocks, RS (3,2) only needs to use one stripe storage. Although the multi-copy storage efficiency is only 0.33 shown in Table 1, when it is less than three data blocks, it will not cause excessive storage space occupation. Multi-copy can greatly improve the parallel read/write ability of files and reduce the read delay of files.

D. FILE SIZE CLASSIFICATION

The idea of designing the FACHS algorithm is as follows. Considering the file access characteristics (size and heat), we can select the best encode and storage mode for each file to improve the read/write performance and recovery performance while ensuring low storage redundancy.

The advantage of using multi-copy for the "hot" and "small" files has been mentioned in Section III-C. Although it may cause additional storage redundancy, it can improve the read/write efficiency and parallel read/write ability of files with high access frequency. Using LRC (6,2,2) coding for "hot" and "large" files can take advantage of its efficient reconstruction capability. Through our research, we know that more than 95% of data loss in distributed clusters occurs on a single node. When LRC Code handles data loss of less than two data blocks, it can directly call local check blocks for fast data recovery without calling global check blocks, reducing the time of data reconstruction and transmission, and effectively improving cluster recovery efficiency. For the "cold" files, we use RS Code to store them, and take advantage of the RS code's low storage cost to reduce the storage redundancy of the cluster. Among the "cold" files, the "small" and "cold" files use RS (3,2), the "large" and "cold" files use RS (6,3) to encode the maximum number of complete stripes respectively, and the last stripe uses RS (3,2) or RS (6,3) depending on the size of the remaining blocks to avoid too many empty blocks to save storage space.

We traverse all the files in the cluster, obtain the heat value of the files in the cycle based on the heat formula proposed in 3.2, count the heat value required for ranking as a hot file, and set the Heat flag ("cold" or "hot") for the files. Then we obtain the storage information of the file, that is, the specific storage space occupied by the file, and set the

Size flag (“large” or “small”) for the file. Then, we encode the file and select the storage method as follows. If the Heat flag of the file is “hot” and the Size flag is “small”, then we use multi-copy to store such file. If the Heat flag of the file is “hot” and the Size flag is “large”, then we use LRC (6,2,2) to code such file. If the Heat flag of the file is “cold” and the Size flag is “small”, RS (3,2) is used to encode such file. If the Heat flag of the file is “cold” and the Size flag is “large”, RS (6,3) is used to encode the maximum number of complete stripes, and RS (3,2) or RS (6,3) is used to encode the last stripe depending on the remaining block size. The specific algorithm is shown in Algorithm 1.

Algorithm 1 Adaptive Selection Algorithm of FACUS

Input: file_size, file_heat, hot_value;
Output: storage mode

```

1: if (file_size < block_size * 3) then
2:   set Heat = hot;
3:   if (file_size < block_size * 3) then
4:     set Size = small;
5:     Apply 3 Replica;
6:   else
7:     set Size = large;
8:     Apply LRC(6,2,2);
9:   end if
10:else
11:   set Heat = cold;
12:   if (file_size < block_size * 3) then
13:     set Size = small;
14:     Apply RS(3,2);
15:   else
16:     set Size = large;
17:     for all full strips do
18:       Apply RS(6,3);
19:     end for
20:     remaining strip Apply RS(3,2) or RS(6,3);
21:   end if
22:end if
  
```

We have clearly defined and classified the files in the cluster through Algorithm 1, and adopt appropriate storage methods for different types of files according to their characteristics to improve the storage efficiency, read and write efficiency and recovery efficiency of the cluster.

IV. EXPERIMENTAL EVALUATION

To verify the advantages and disadvantages of the FACHS strategy, we built an experimental platform on the basis of CloudSim [23] platform. We compare FACHS with the existing methods in the simulation scenario in terms of the storage efficiency, read/write performance, and recovery efficiency. To this end, we first introduce the experimental environment and the experimental design, then analyze the performance evaluation results.

A. EXPERIMENTAL ENVIRONMENT

CloudSim is a full system simulation platform, launched by the Grid Lab and Gridbus project of the University of Melbourne, Australia. It is universal enough to provide data centers, hosts, virtual machines and other cloud computing system simulation components for our experiments [24]. Moreover, CloudSim can simulate the hardware resources of a single physical node in a large-scale cloud environment, enabling us to simulate the situation of multiple nodes and multiple files during experimental design.

In our experiment, CloudSim will generate a cluster of multiple nodes every time according to the experimental needs. Each node has a single-core Intel(R)core(TM)-i7 CPU@2.00 GHz and runs 4GB RAM on Ubuntu 16.04.LTS. Our machines running CloudSim platform are configured as Intel (R) Core (TM) i5-1035G1 CPU @ 1.00GHz 1.19GHz and 16 GB RAM.

We extend the function on the basis of CloudSim to make the simulation environment close to Hadoop distributed cluster to meet the requirements of this experiment.

B. EXPERIMENTAL DESIGN

Our experiment simulates the distributed cluster application scenario of “write once and read many times”. In this scenario, Zipf distribution is applied to the read allocation of files, namely 80% of the access are concentrated on 20% of the files. To ensure the fairness of the experiment, we designed multiple groups of comparison experiments to store different numbers of files on different number of nodes. Each node has an average of 50 files, and the size of each file is randomly set within 3000 MB.

We select DECPA strategy [13], RS (3,2) and RS (6,3) for our comparison experiments. Among them, DECPA is a dynamic erasure coding strategy allocation. We implement DECPA-F and DECPA-MF strategies. The experiments are divided into three groups. The first group of experiments consist of 100 nodes, 5000 files, and 4000 reads. The second group of experiments consist of 500 nodes, 25000 files and 20000 reads. The third group of experiments consist of 1000 nodes, 50000 files and 40000 reads.

Our experimental performance evaluation includes the storage redundancy, write performance, read performance of the cluster, and the reconstruction time required to recover data blocks when a single node fails.

C. ANALYSIS OF EXPERIMENTAL RESULTS

We compare FACHS with DECPA-F, DECPA-MF, RS (3,2) and RS (6,3) respectively in terms of the storage redundancy, write performance, read performance, and the reconstruction time required to recover data blocks in the event of a single node failure.

1) STORAGE OVERHEAD ANALYSIS

We count the storage redundancy generated when the cluster stores files. As shown in Figure 1, the storage redundancy

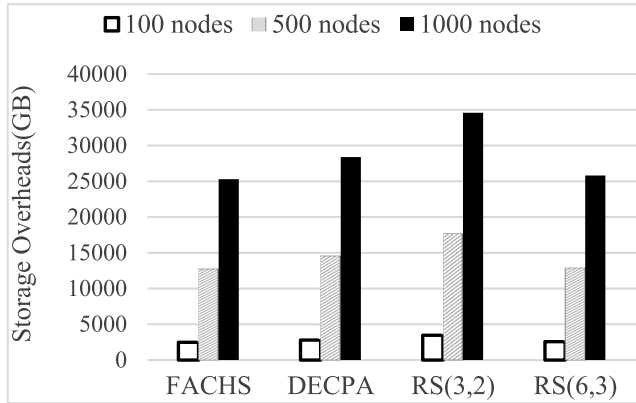


FIGURE 1. Comparison of storage overhead under different node sizes.

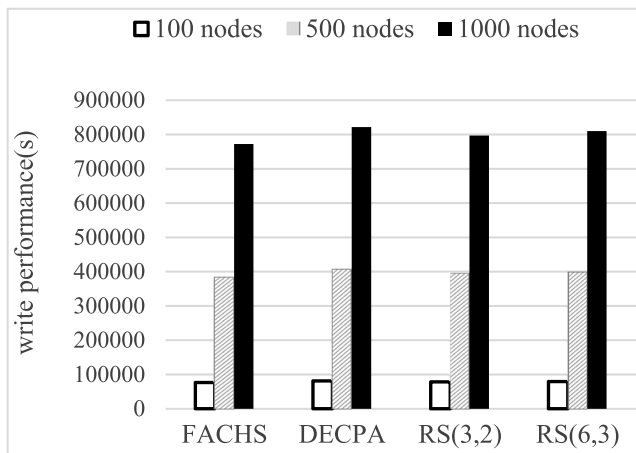


FIGURE 2. Write performance comparison under different node sizes.

generated by FACHS under different nodes and file numbers is better than DECPA and the basic RS code. In the 1000 node scale, when the data size is 50000 GB, FACHS can save up to 3000 GB of storage space compared with DECPA, and improve the storage efficiency by 12%. Compared with RS coding, FACHS can save up to 9000 GB of storage space and improve the storage efficiency by 37%. According to the file size, FACHS uses RS (3,2) and RS (6,3) to store cold files in the cluster. RS Code itself has a high storage efficiency. FACHS can avoid the generation of too many empty blocks in the stripe, so it can reduce the occupied storage space.

2) WRITE PERFORMANCE ANALYSIS

We verify the performance of writing data by writing multiple files to the cluster. The experimental results are shown in Figure 2. When there are 100 nodes, compared with RS (3,2) and RS (6,3) coding, FACHS can shorten the write time by 2.5% and 4% respectively. The reason is that some files in FACHS use multi-copy of storage, saving the encode transmission time of erasure codes. Compared with DECPA, FACHS shortens the write time by 6.3%. Because the coding

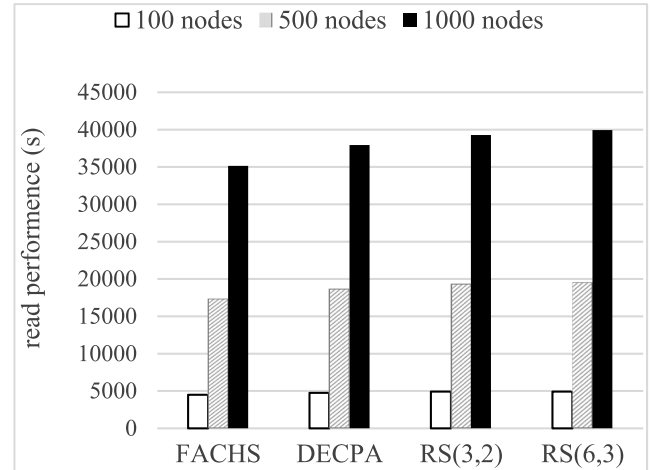


FIGURE 3. Comparison of read performance under different node sizes.

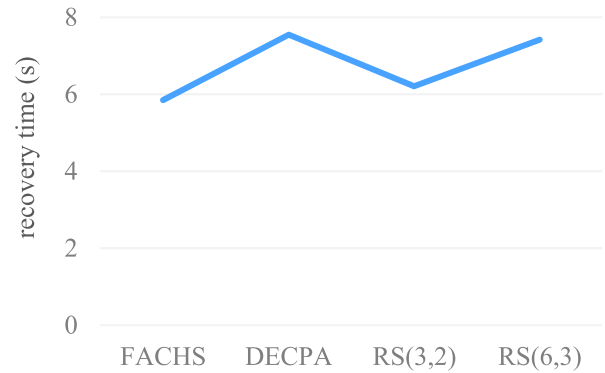


FIGURE 4. Single block loss recovery time comparison.

efficiency of LRC code is better than RS code, the write time can be shortened. With the increase of the number of files, the proportion of write performance improvement also increases slightly due to the increase of the time taken for erasure code encode. At the scale of 1000 nodes, compared with RS (3,2) and RS (6,3), FACHS can shorten the write time by 3.2% and 4.9%, and compared with DECPA, FACHS can shorten the write time by 6.4%.

3) READ PERFORMANCE ANALYSIS

We read data from files of multiple sizes, and the experimental results are shown in figure 3. The advantage of the multi-copy strategy is that its read and write performance is higher than that of the erasure code, and the advantage of RS code is that its storage space is saved when storing cold data. FACHS divides the heat of files, and uses the zipf distribution to focus most of the access on a small number of files, which also allows FACHS to take full advantage of the multi-copy, so fachs is better than decpa and traditional rs code in data read performance. at the scale of 100 nodes, compared with rs (3,2), RS (6,3) and decpa, fachs can shorten the data read time by up to 9.2%, 9.4% and 5.3%, respectively. with the increase of the number of files and the number of read operations,

the number of access to the hot data by fachs is more, and the efficiency of read operations is higher. in 1000 node scale, compared with rs (3,2), RS (6,3) and decpa, fachs can shorten the data read time by 11.8%, 13.7% and 8.0%, respectively.

4) RECOVERY TIME ANALYSIS

We evaluate the reconstruction time required for FACHS to recover lost data blocks when simulating single block loss. The experimental results are shown in Figure 4. Because FACHS uses LRC codes, compared with RS codes, it is more efficient in the number of transmission blocks and the encode and decode time, it can reduce the reconstruction time when blocks are lost. Compared with RS (3,2), RS (6,3) and DECPA, FACHS can shorten the recovery time by 6.2%, 26.8% and 29.1%, respectively.

From the experimental results of the above four indicators, we can see that in the three groups of experiments, the performance indicators of FACHS at each scale are better than those of other comparison objects, and the performance indicators at 1000 node scale have the highest proportion of improvement.

V. CONCLUSION

This paper designs and implements FACHS. This is a hybrid storage strategy, which integrates RS Code, LRC Code and multi-copy together, so that the cluster has lower storage redundancy as well as higher read/write efficiency and recovery efficiency. As to the infrequently accessed files, we use RS Code for storage; as to the frequently accessed files, we use LRC Code and multi-copy storage according to the file size. To verify the effectiveness of FACHS, we conducted several groups of experiments on the CloudSim simulation platform. Compared with the traditional RS Code, FACHS improves the write performance, the read performance, the recovery efficiency by up to 4.8%, 13.7% and 29.1%, respectively. Compared with DECPA, FACHS improves the write performance, the read performance, the recovery efficiency by up to 6.4%, 8.0% and 26.8%, respectively. Compared with the traditional RS Code and DECPA, FACHS saves up to 9000 GB and 3000 GB of storage space at 1000 nodes, respectively. Therefore, our experiments verify the performance improvement of FACHS compared with existing methods.

Our work has further optimized the use of erasure code hybrid storage strategy in distributed systems, improved the performance of clusters, and made a contribution to the worldwide distributed storage field. However, because FACHS uses different types of erasure codes, the performance requirements of nodes are also high, which may lead to the failure to give full play to the advantages of FACHS in heterogeneous environment. Therefore, in the future work, we will study how to optimize FACHS for heterogeneous environment.

REFERENCES

- [1] L. Qian, Z. Luo, Y. Du, and L. Guo, "Cloud computing: An overview," in *Proc. Cloud Comput., 1st Int. Conf., (Cloudcom)*, Beijing, China, Dec. 2009, pp. 626–631.
- [2] K. Michael, "Securing the cloud: Cloud computer security techniques and tactics," *Comput. Secur.*, vol. 31, no. 4, p. 633, Jun. 2012.
- [3] M. Hilbert, "Big data for development: A review of promises and challenges," *Develop. Policy Rev.*, vol. 34, pp. 135–174, Jan. 2016.
- [4] X. Li, R. Li, P. Lee, and Y. Hu, "OpenEC: Toward unified and configurable erasure coding management in distributed storage systems," in *Proc. USENIX FAST*, 2019, pp. 1–15.
- [5] R. G. Masur and S. K. McIntosh, "Preliminary performance analysis of Hadoop 3.0.0-alpha3," in *Proc. New York Sci. Data Summit (NYSDS)*, Aug. 2017, pp. 1–3.
- [6] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *Proc. IEEE Symp. Mass Storage Syst. Technol.*, May 2010, pp. 1–10.
- [7] J. Dean, "MapReduce: Simplified data processing on large clusters," in *Proc. Symp. Operating Syst. Design Implement.*, 2004, pp. 1–7.
- [8] D. Ford, "Availability in globally distributed storage systems," in *Proc. 9th USENIX Symp. Operating Syst. Design Implement. (OSDI)*, Vancouver, BC, Canada, Oct. 2010, pp. 1–14.
- [9] R. Li, Y. Hu, and P. P. C. Lee, "Enabling efficient and reliable transition from replication to erasure coding for clustered file systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 9, pp. 2500–2513, Sep. 2017.
- [10] K. Rashmi, N. B. Shah, D. Gu, H. Kuang, D. Borthakur, and K. Ramchandran, "A solution to the network challenges of data recovery in erasure-coded distributed storage systems: A study on the Facebook warehouse cluster," Presented as the 5th USENIX Workshop Hot Topics Storage File Syst., 2013.
- [11] S. Mitra, R. Panta, M.-R. Ra, and S. Bagchi, "Partial-parallel-repair (PPR): A distributed technique for repairing erasure coded storage," in *Proc. 11th Eur. Conf. Comput. Syst.*, Apr. 2016, p. 30.
- [12] J. Gu, C. Wu, X. Xie, H. Qiu, J. Li, M. Guo, X. He, Y. Dong, and Y. Zhao, "Optimizing the parity check matrix for efficient decoding of RS-based cloud storage systems," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2019, pp. 533–544.
- [13] C. Wu, S. Wan, X. He, Q. Cao, and C. Xie, "H-code: A hybrid MDS array code to optimize partial stripe writes in RAID-6," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, May 2011, pp. 782–793.
- [14] H. Jin-Ping, L. Gui-Yang, and L. Hui, "Hybrid coding LRC-RS in heterogeneous decentralized storage," *Comput. Eng. Des.*, vol. 42, no. 2, pp. 301–308, 2021.
- [15] M. Xia, "A tale of two erasure codes in HDFS," in *Proc. Usenix Conf. File Storage Technolog.*, 2015, pp. 213–226.
- [16] A. Chiniyah and A. Mungur, "Dynamic erasure coding policy allocation (DECPA) in Hadoop 3.0," in *Proc. 6th IEEE Int. Conf. Cyber Secur. Cloud Comput. (CSCloud)/ 5th IEEE Int. Conf. Edge Comput. Scalable Cloud (EdgeCom)*, Jun. 2019, pp. 29–33.
- [17] H. Qiu, C. Wu, J. Li, M. Guo, T. Liu, X. He, Y. Dong, and Y. Zhao, "EC-fusion: An efficient hybrid erasure coding framework to improve both application and recovery performance in cloud storage systems," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2020, pp. 191–201.
- [18] X. Xie, C. Wu, J. Gu, H. Qiu, J. Li, M. Guo, X. He, Y. Dong, and Y. Zhao, "AZ-code: An efficient availability zone level erasure code to provide high fault tolerance in cloud storage systems," in *Proc. 35th Symp. Mass Storage Syst. Technol. (MSST)*, May 2019, pp. 230–243.
- [19] Y. Chen, S. Alspaugh, and R. Katz, "Interactive analytical processing in big data systems: A cross-industry study of mapreduce workloads," *VLDB Endowment*, vol. 5, no. 12, pp. 1802–1813, 2012, doi: 10.14778/2367502.2367519.
- [20] Y. Hu, Y. Wang, B. Liu, D. Niu, and C. Huang, "Latency reduction and load balancing in coded storage systems," in *Proc. Symp. Cloud Comput.*, Sep. 2017, pp. 365–377.
- [21] S. Mitra, R. Panta, M.-R. Ra, and S. Bagchi, "Partial-parallel-repair (PPR): A distributed technique for repairing erasure coded storage," in *Proc. 11th Eur. Conf. Comput. Syst.*, Apr. 2016, p. 30.
- [22] Y. Chen, S. Alspaugh, and R. Katz, "Interactive analytical processing in big data systems: A cross-industry study of MapReduce workloads," *Proc. VLDB Endowment*, vol. 5, no. 12, pp. 1802–1813, Aug. 2012, doi: 10.14778/2367502.2367519.

- [23] R. N. Calheiros, R. Ranjan, C. A. F. D. Rose, and R. Buyya, "CloudSim: A novel framework for modeling and simulation of cloud computing infrastructures and services," in *Proc. CloudSim ICCP*, 2009, pp. 1–9.
- [24] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. D. Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw. Pract. Exper.*, vol. 41, no. 1, pp. 23–50, Jan. 2011, doi: [10.1002/spe.995](https://doi.org/10.1002/spe.995).
- [25] J. Liao, Z. Cai, F. Trahay, and X. Peng, "Block placement in distributed file systems based on block access frequency," *IEEE Access*, vol. 6, pp. 38411–38420, 2018, doi: [10.1109/ACCESS.2018.2851571](https://doi.org/10.1109/ACCESS.2018.2851571).



YING SONG received the Ph.D. degree in computer engineering from the Institute of Computing Technology (ICT), Chinese Academy of Sciences. She is currently an Associate Professor with the Computer School, Beijing Information Science and Technology University. Her work has covered topics, such as performance modeling, resource management, cloud computing, and big data computing platform. She has been authoring or coauthoring more than 30 publications in these areas, since 2007. Her research interests include computer architecture, parallel and distributed computing, and virtualization technology. She has served for various academic conferences.



QIANG ZHANG received the B.S. degree in computer science and technology from Beijing Information Science and Technology University, Beijing, China, in 2019, where he is currently pursuing the master's degree with the Computer School. His research interest includes distributed storage.



BO WANG received the B.S. degree in computer science from Northeast Forest University (NEFU), Harbin, China, in 2010, and the Ph.D. degree in computer science from Xi'an Jiaotong University (XJTU), Xi'an, China, in 2017. He was a Guest Student with the State Key Laboratory of Computer Architecture, Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), from 2012 to 2016. He is currently a Lecturer with the Software Engineering College, Zhengzhou University of Light Industry (ZZULI). He has published more than ten research articles in these areas. His research interests include distributed systems, cloud computing, edge computing, resource management, and task scheduling. He has served for various academic journals and conferences.

...