ODCP: Optimizing Data Caching and Placement in Distributed File System Using Erasure Coding

Shuhan Wu, Yunchun Li, Hailong Yang, Zerong Luan, Wei Li

ODCP: Optimizing Data Caching and Placement in Distributed File System using Erasure Coding

Shuhan Wu[†], Yunchun Li[†], Hailong Yang[†¶⋆], Zerong Luan[§], and Wei Li[†]

School of Computer Science and Engineering[†],
Beihang University, Beijing, China, 100191[†]
College of Life Sciences and Bioengineering[§],
Beijing University of Technology, Beijing, China, 100083[§]
State Key Laboratory of Mathematical Engineering and Advanced Computing, Wuxi, China, 214125[¶]

Abstract. Many current distributed file systems use erasure-coding based data redundancy techniques to improve the reliability of data storage. Such techniques can significantly improve the effective storage utilization. However, there are several drawbacks to the above techniques. Firstly, they introduce non-negligible computation overhead for decoding. Secondly, traditional data caching and placement strategies become less effective in such cases. To solve the above drawbacks, this paper proposes a new data cache allocation mechanism based on simulated annealing and a new data placement strategy based on convex optimization, which effectively reduces data block transmission delay and decoding delay. We have implemented the proposed data placement strategy in the real-world distributed file system *Alluxio*, and evaluated the performance of our strategy. Experiment results show that our strategy can significantly reduce the file read delay compared to traditional data placement strategies.

Keywords: Distributed File System · Erasure Coding · Decoding Latency · Data Placement Strategy · Cache Allocation Strategy

1 Introduction

The distributed file system provides an excellent solution for storing and processing large scale data. However, the unbalanced accesses for hot data in distributed file system often cause severe performance degradation. For instance, studies [11] have shown that in a Facebook cluster, for more than 50% of the time, the frequency of visits to the most popular links exceeds $4.5\times$ than the average visits of ordinary links. Such a phenomenon of extremely unbalanced data accesses often leads to overwhelmed load on storage nodes and causes the slowdown of the entire distributed file system [8, 15]. The widely adopted method to address the load unbalance problem is to cache data and optimize data placement. Particularly, erasure-coding based techniques have been proposed to reduce the storage cost

[⋆] Corresponding author, hailong.yang@buaa.edu.cn of keeping multiple data copies in the distributed file system for reliability [11, 6], such as Ceph [2] and HDFS [12]. However, erasure-coding based techniques have their own drawbacks [6, 9]. Firstly, the decoding process introduces extra computation overhead and thus leads to longer data access delay. Secondly, the decoding overhead cannot be addressed by traditional data caching and placement strategies, because the overhead highly depends on the data block to be decoded. Therefore, a new approach needs to be designed to optimize the data caching and data placement targeting the distributed file system with erasure-coding.

To solve the above challenge, we propose a data cache allocation mechanism based on simulated annealing and a data placement strategy based on convex optimization. Evaluate results on real-world distributed file system *Alluxio* with erasure coding shows that our approach can significantly reduce file access delay.

Specifically, the main contributions of this paper are as follows:

– We propose a new data placement strategy that decomposes data placement problem into two stages of sub-optimization problems, and solves them using simulated annealing algorithm and convex optimization method, which dramatically reduces the complexity for computing the optimization results.
– We propose a new data cache allocation mechanism based on simulated annealing algorithm, which identifies the most profitable file blocks and allocates them in the data cache to achieve low read latency.
– We build a file read delay model that incorporates the decoding delay model and data transmission delay model. This model can effectively guide the design of data placement strategy for optimizing the file read latency.
– We implement the proposed approaches in real-world distributed file system *Alluxio*, and demonstrate the effectiveness of our approaches for reducing file read latency by comparing with traditional data placement approaches.

The rest of this paper is organized as follows. Section 2 introduces the background of data caching and data placement as well as the motivation of this paper. We present the design and implementation of our *ODCP* in Section 3. We evaluate the effectiveness of *ODCP* in Section 4. Section 5 presents the related work on the data caching and placement, and we conclude this paper in Section 6.

2 Background and Motivation

2.1 Erasure coding

The most commonly used Erasure Code is RS code, which divides the original file into k file blocks, and then generate r redundant blocks through the encoding matrix. k plus r equals n, which is denoted as (n, k) coding. It is an MDS code [5] (maximum distance separable code), which means that for (n, k) encoding, any subset with k blocks is obtained from the set of n blocks, the original file can be recovered.

2.2 Data cache allocation

In the erasure-coded scenario, due to the MDS code's characteristics, the read delay of the entire file is determined by the k-th arriving file block. Therefore, when the current k-1 block contains a cached block, the traditional backup cache mechanism can not accelerate file access. To solve this problem, Aggarwal V et al. proposed a functional caching mechanism in [1]. In this caching mechanism, the cache area stores the re-encoded blocks that still satisfy the MDS code's characteristics. With this caching mechanism, the cache block can always speed up file access. Therefore, based on functional caching, this paper proposes a new caching strategy for decoding delay optimization.

2.3 File read probability

Unlike traditional file access, for erasure-encoded files, the read request is uncertain, because of the characteristics of its MDS code, not every block must respond to the read request [10]. Therefore, we set a probability value for each block to represent its response probability when the read-write agent sends requests for a file. If the sum of all block probabilities is k, then there are k file block responses per access on average. The file access frequency and file block read probability are directly related to the storage node load. Therefore, this paper takes the read probability distribution as the optimization variables and converts it into a file placement strategy.

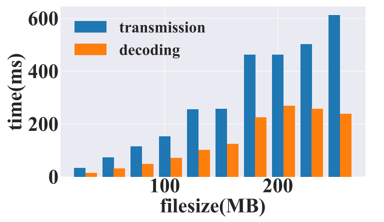
2.4 Motivation

In the distributed file system, due to the uneven frequency of data access, there are often severe data hot spots. For erasure-encoded files, because of its MDS code character and decoding overhead, traditional optimization techniques are not sufficient [3]. Secondly, many optimizations for erasure codes tend to pay more attention to the transmission delay, while the decoding delay is less studied or ignored. However, Figure 1(a) shows that the decoding delay is significant compared to the transmission delay. And the results in Figure 1(b) show a strong linear relationship between the decoding delay and the number of redundant blocks, which indicates there is room for optimization of the decoding delay. In order to maintain data consistency and high performance, many current distributed file systems such as HDFS use write-once design, in which the read operations are much more popular than write. Therefore, our work mainly focuses on the read operations.

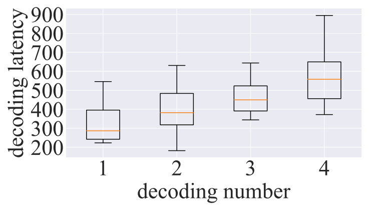
3 Design and Implementation

3.1 Data cache allocation using simulated annealing

There are two problems with the functional cache [1]: first, the re-encode process increases the encoding workload and write delay. Second, it does not have any



(a) transmission and decoding delay



(b) decoding delay and number

Fig. 1. Early experiment results

optimization in terms of decoding delay. Placing the re-encoded block in the cache area increases the probability that it participates in decoding greatly. Therefore, we limit the file blocks stored in the cache area to the original blocks. For the allocation of the cache area, we define the selection matrix mtx, which indicates which file blocks are selected to enter the cache area, where the elements can only take 0 or 1. The elements with value of 1 represent the file block where file i can be placed on node j. To search for the optimal selection matrix mtx, we continuously reducing the cost C as shown in Equation 1, where α is scale factor, P is the read probability distribution, indicating the probability of sending a file block read request for file i on node j, λ is the read frequency of each file.

$$C = \alpha \cdot mean(\sum_{i=1}^{m} p_{ij} \cdot mtx_{ij} \cdot \lambda_i) + (1-\alpha) \cdot std(\sum_{i=1}^{m} p_{ij} \cdot mtx_{ij} \cdot \lambda_i) \tag{1}$$

Then we use the search algorithm based on simulated annealing to find a near-optimal cache allocation scheme. The algorithm has a chance to accept a worse solution than the current optimal solution, which can prevent it from falling into the local optimal. Simultaneously, the algorithm's random factors will gradually decrease with the increase of the number of iterations to ensure the final convergence of the algorithm.

3.2 Building file read delay model

When accessing erasure-coded files, the read-write agent reads a file in two steps: transmission and decoding. The decoding process starts only after the transmission of the k-th block is completed. Therefore, this paper will model the two processes and combine them as the read delay model.

The upper limit O of the weighted average read delay of all files is as shown in Equation 2, where $\bar{U}_i$ and $\bar{T}_i$ represent transmission delay and decoding delay.

$$O = \sum_{i=1}^{m} \frac{\lambda_i}{\sum_{i=1}^{m} \lambda_i} (\bar{U}_i + \bar{T}_i) \tag{2}$$

Transmission delay model This paper cites a read latency estimation formula based on queuing theory proposed in [13], which has been mentioned and verified in many works [1, 13, 14]. But in order to combine it with the decoding delay, we added a subscript k to represent the difference between the original block and the redundant block.

First the statistics required in the model need to be calculated as follows. the average value of each node's transmission time $E[X_j]$ in Equation 3, variance σ_i^2 in Equation 4, the total request amount of node j Λ_j in Equation 5, the second-order origin moment Γ_j^2 in Equation 7, the third-order origin moment Γ_j^3 in Equation 8. And μ_j represents the service rate of node j. ρ_j in Equation 6 representing the request strength.

$$E[X_j] = \frac{1}{\mu_j} \tag{3}$$

$$\sigma_i^2 = E[X_j^2] - E[X_j]^2 \tag{4}$$

$$\Lambda_j = \sum_{i=1}^{m} (P_{ij1} + P_{ij2})\lambda_i \tag{5}$$

$$\rho_j = \Lambda_j \mu_j \tag{6}$$

$$\Gamma_j^2 = E[X_j^2] \tag{7}$$

$$\Gamma_j^3 = E[X_j^3] \tag{8}$$

Where X_j is the transmission delay of a single file block, λ_i is the file access rate. The optimization variable P_{ijk} represents the probability of sending a read request to file i on node j, where k represents the block type ($k = 1$ represents the original block, and $k = 2$ represents redundant block). m represents the total number of files, and n represents the total number of nodes.The transmission delay $\bar{U}_i$ is as shown in Equation 9:

$$\bar{U}_i = z_i + \sum_{j=1}^{n} \frac{P_{ij1} + P_{ij2}}{2} (E[Q_j] - z_i) + \sum_{j=1}^{n} \frac{P_{ij1} + P_{ij2}}{2} \sqrt{(E[Q_j] - z_i)^2 + Var[Q_j]} \tag{9}$$

Where z_i is an auxiliary variable. Besides, Q_j represents the total transmission delay of reading file blocks on node j. The form of $E[Q_j]$ and $Var[Q_j]$ is as defined in Equation 10 and 11:

$$E[Q_j] = \frac{1}{\mu_j} + \frac{\Lambda_j \Gamma_j^2}{2(1-\rho_j)} \tag{10}$$

$$Var[Q_j] = \sigma_j^2 + \frac{\Lambda_j \Gamma_j^3}{3(1-\rho_j)} + \frac{\Lambda_j^2 (\Gamma_j^2)^2}{4(1-\rho_j)^2} \tag{11}$$

Decoding delay model The encoding and decoding process of the erasure code is essentially matrix multiplication operation. The *coding matrix* is composed of an *identity matrix* and a *redundant matrix*. After multiplying with the *data matrix*, the *original blocks* and the *redundant blocks* are obtained. Pick any k blocks from them, and then take the corresponding rows from the *coding matrix* to form a *residual matrix*, and inverse it to get the *recovery matrix*. Use the *recovery matrix* to recover the original *data matrix*. However, the *original blocks* do not need to be calculated during the recovery process, and other lost blocks need to be calculated. So there is a linear relationship between the number of *redundant blocks* and the decoding delay. A mathematical model is established to estimate the decoding delay $\bar{T_i}$ and its form is as defined in Equation 12:

$$\bar{T_i} = \eta \frac{S_i}{k_i + r_i} \sum_{i=1}^{m} \lambda_i (1 \cdot r_{i,Cached} + \sum_{j=1}^{n} P_{ij2}) \tag{12}$$

The decoding time is positively correlated with the load of the agent, the file block size, the number of redundant blocks participating in decoding. For file i, the redundant blocks in cache will certainly participate in decoding, other redundant blocks of this file will participate in decoding according to probability. Where η is the scale factor, S is the file size, k_i and r_i are the original and redundant blocks of file i.To minimize decoding delay, let $r_{i,Cached} = 0$. Then the form of $\bar{T_i}$ turns into Equation 13.

$$\bar{T_i} = \eta \frac{S_i}{k_i + r_i} \sum_{i=1}^{m} \lambda_i (\sum_{j=1}^{n} P_{ij2}) \tag{13}$$

3.3 Formulating the optimization problem

In order to find the probability distribution under which the lowest average read delay can be achieved, a convex optimization problem needs to be formulated. We use Equation 2 as the objective function. Read probability distribution P and the auxiliary variable z are used as optimization variables. And constraints are added as follows.

$$P_{ij1} \cdot P_{ij2} = 0 \tag{14}$$

$$\sum_{i=1}^{m} d_i \leq C \tag{15}$$

$$\sum_{j=1}^{n} P_{ij1} \cdot P_{ij2} = k_i - d_i + 1 \tag{16}$$

$$ceil(P_{..1} + P_{..2}) = \neg mtx \tag{17}$$

To minimize the read delay and reduce the complexity of the transmission delay model, we use Equation 14 to ensure that a node can only store one block of a file at most. Inequality 15 ensures the sum of the number of blocks of all files in the cache cannot be greater than the capacity of the cache area, where d_i is the number of blocks of file i in the cache, and C is the capacity of the cache. Equation 16 ensures the sum of the read probability of each file on all nodes plus the number of blocks already in the cache minus redundant request equals to k. Add an additional redundant read request is to reduce the influence of straggler. Equation 17 ensures that the read probability distribution obtained must meet the premise of the previous cache allocation. The convex optimization problem is defined as 18.

$$\begin{aligned} & min \sum_{i=1}^{m} \frac{\lambda_i}{\sum_{i=1}^{m} \lambda_i} (\bar{U_i} + \bar{T_i}) \\ & s.t. \ 14, 15, 16, 17 \\ & var. \ P_{ijk}, z_i \end{aligned} \tag{18}$$

3.4 Solving the convex optimization problem

This subsection introduces the challenges of this optimization problem and the solutions we proposed. First of all, the constraints of the real number range and the integer constraint d_i coexist in the constraint conditions, which belong to the mixed-integer constraint problem. Usually, it is difficult to find the optimal solution for this type of problem directly. Secondly, the optimization problem has two sets of variables, reading probability distribution P and auxiliary variable z. It is difficult to optimize both at the same time.

Our solution to the integer mixed constraint problem is to decompose the originally unified problem into two stages: cache allocation and storage node data placement. When the cache allocation strategy is determined, the variable d_i becomes a constant, Equation 15 will be satisfied, and Equation 16 becomes a real range constraint. Therefore, there is no integer constraint.

And inspired by other work [1, 13, 14], this paper uses a method called *alternate optimization* to solve the two sets of variable problem. It alternately optimizes two sets of variables until the objective function converges.

Then the mapping relationship between the optimal solution and the data placement strategy must be established. First, according to the variable d_i, the corresponding number of blocks of each file are put into the cache area. The original blocks are placed preferentially. Secondly, if the block probability of requesting a file to a node is not 0, the corresponding block must be prepared. In sum, our approach optimizes reading delay by placing data according to the data placement strategy and initiating file requests based on the solution of Equation 18.

4 Evaluation

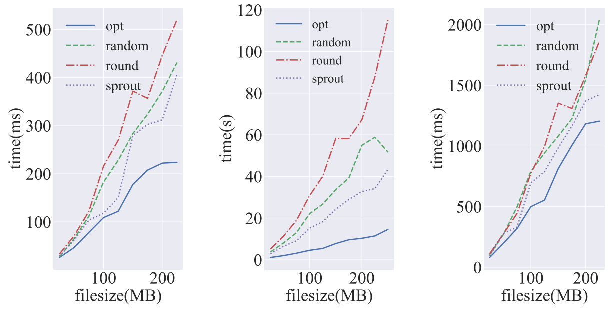
4.1 Experimental Setup

We evaluate the distributed file system *Alluxio* with our proposed approaches on a cluster of 10 nodes. The cluster contains one *Master node*, one *Client node* (as a read-write agent), and eight *Slave nodes*. There are 3 types of configuration of the *slave nodes*, including one model1 (Intel Xeon Phi 7210, 200GB Memory, 12TB HDD), four model2 (Intel Xeon E5-2620 V4, 12GB Memory, 100GB HDD), and three model3 (Intel Xeon E5-2620 V2, 8GB Memory, 200GB HDD). The network bandwidth is $1gbps$ between all nodes.

4.2 File read latency optimization

We test and analyze the overall file access latency and the latency of the two stages it contains. We test and compare common random strategies, round-robin strategies, the optimization strategy *sprout* proposed in [1], and our optimization strategy. They are written down as *random, round, sprout, opt*, respectively. According to our cluster size, we set the standard parameters of erasure coding in the experiment to (k, n)=(4, 8). The reading probability λ of each file are taken as 0.009, 0.011, 0.01, 0.012, 0.014, 0.013. The capacity of the cache area is 8 file blocks. The file sizes are randomly selected between 25MB and 250 MB. For smaller files, they are usually merged into bigger files in distributed file systems to achieve better performance.

Average block transmission delay Accessing a file requires obtaining a sufficient number of file blocks before entering the next stage of the decoding process. Therefore, the read delay of a file block dramatically affects the entire file's overall access delay. As shown in Figure 2(a), we evaluate the file block access delay under different data access patterns.



(a) blockfetching delay (b) decode delay (c) file read delay

Fig. 2. Different types of delays under four different data access patterns.

The results show that the round-robin strategy has the longest block read delay among all strategies. This is because under the round-robin strategy, the density of access requests received by a single slave node fluctuates periodically. Therefore, when the slave node is busy, it will slow down the transmission process of a series of file blocks. The random strategy alleviates this periodicity, so the block read delay decreases.

The opt strategy reaches the best of all strategies. The reasons are as follows. In the sprout strategy, cache allocation and transmission delay optimization problems are simultaneously modeled in an optimization problem. But in this article, these two problems are divided into two steps. This method significantly reduces the complex constraints in the optimization problem and the difficulty of solving. And a more reasonable cache allocation scheme makes the file block read delay have a better optimization effect. Compared with the other three strategies, the opt strategy reduces the average block read latency by 36.9%, 45.6%, and 27.9%, respectively.

We also evaluate and analyze the change in the average read latency of file blocks for each strategy with different cache sizes, as is shown in Figure 3.

We use *ms/MB* to express the average block read delay to eliminate the impact of the delay change caused by the file block size. It can be seen that, overall,

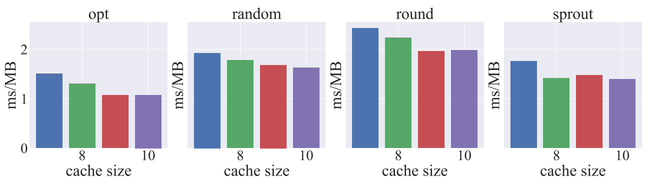


Fig. 3. Impact of data cache capacity on block read delay.

the increased capacity of the cache area under each strategy will reduce the average read latency of the file block. The average decline rate of each strategy are 10.3%, 5.5%, 6.5%, 6.9%, indicating that opt strategy reaches the highest cache utilization efficiency.

Decoding delay We count the decoding delays of the four strategies under various file sizes and express them through the cumulative decoding delay of the entire test process.

As shown in Figure 2(b), decoding delay of random and round-robin strategies increase rapidly with the size of the files. The decoding delay of the round-robin strategy is the longest, and the frequency of occurrence of redundant blocks during reading is periodic. When the read-write agent is busy at decoding, the decoding process of each file is slowed down; when it is idle, the decoding capability is wasted. Therefore, the total decoding delay is greatly increased. The random strategy eases this periodicity. The optimization of the request queue by the sprout strategy further eases the periodicity mentioned above. At the same time, because we place redundant blocks on nodes with lower IO bandwidth, the optimization process reduces the probability of requests on low IO bandwidth nodes, which indirectly reduces the number of redundant blocks, which also reduces decoding delay. But because there is no specialized optimization for decoding delay, there is still optimization space. Opt strategy has been specifically optimized for decoding delay. While ensuring the request queue's optimization effect, it also greatly reduces the number of redundant blocks. So opt strategy achieves the lowest decoding delay among the four strategies. Compared with the other three strategies, opt strategy reduces the decoding delay by 36.1%, 46.3%, and 30.9%, respectively.

Average file read delay The average file reading delay is a comprehensive reflection of the strategy optimization effect. It is the composite result of the previous average block read delay and decoding delay.

As shown in Figure 2(c), due to the disadvantages of the previous average block read delay and decoding delay, the round-robin strategy has the longest file access delay among all four strategies. The random strategy is slightly lower than the round-robin strategy. The sprout strategy optimizes the request queue to reduce the average file read delay significantly. Finally, due to the more detailed optimization of opt strategy, including cache allocation, file block transmission delay optimization, and decoding delay optimization, we obtain the lowest file read delay among all strategies. After standardizing the file read latency according to file size, opt strategy reduces the file read latency on average of 29.8%, 32.6%, and 19.9% compared to the other three strategies.

4.3 Parameter sensitivity analysis

Redundancy Higher redundancy has better reliability, but it is usually inferior in decoding delay and file size. During the experiment, we found that redundancy is an important factor that affects the decoding delay optimization effect. We

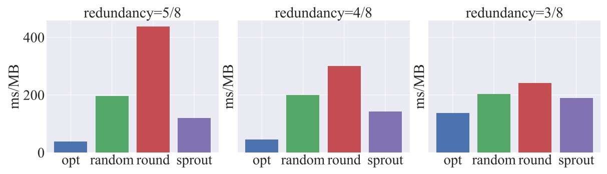


Fig. 4. Decoding delay under different settings of parameter k.

use the form of decoding delay at different k values in Figure 4 to show the effect of varying redundancy on the optimization effect of decoding delay. The redundancy is 5/8, 4/8, 3/8. As the redundancy reducing, the gap between various strategies is decreasing. That is because as the proportion of redundant blocks decreases, the optimization for redundant blocks in the optimization strategy gradually disappears. Finally, the decoding delay approaches the unoptimized strategy.

Straggler The straggler is a widespread problem in distributed file systems and distributed computing frameworks. Once a straggler appears in the system, the entire task is often slowed down due to the *bucket effect*. Opt optimization strategy uses redundant read requests to reduce the impact of a straggler.We artificially restrict *slave2*. Its network bandwidth was limited to *100mbps* to simulate the situation of a straggler.

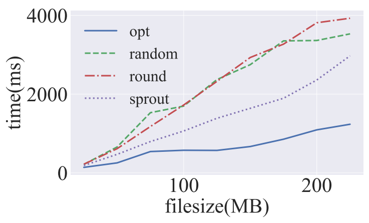


Fig. 5. File read delay when encountering straggler.

As shown in Figure 5, in the case of a straggler, the file read delay of each strategy increases. But opt strategy initializes a redundant read request before the end of the transmission process. Before the last file block arrives, it can enter the decoding stage, greatly reducing the straggler impact. Therefore, in this particular case, compared to the other three strategies, opt strategy decreases the file read delay by 67.5%, 66.9%, 48.0%.

5 Related Work

Liao *et al.* proposed a data layout strategy for distributed file systems based on data access frequency in [7]. This method first analyzes the history information of block access sequence of a specific application, and then uses the *k partition algorithm* to divide the files into multiple groups according to the frequency of data access. Afterward, the data is distributed in groups. In short, this newly proposed data placement strategy makes the data evenly distributed, and the data block access rate tends to be balanced.

Vaneet Aggarwal *et al.* proposed a functional caching method to minimize service delay in erasure code storage clusters in [1]. This paper optimizes the caching mechanism in the erasure code storage system. Making the cache blocks in the cache and the data blocks in the cluster constitute a decoding combination that conforms to the MDS code. And optimizing the data placement according to the file access delay.

HE and others investigated the redundancy setting of the Hadoop cluster in [4]. The default number of backups used in HDFS is 3. That is, each file block must be stored three times. A higher number of backups means higher storage resource consumption, but it also brings higher data availability and data locality. Therefore, a backup method is proposed in the article to backup more frequently accessed files to improve the performance of data access.

6 Conclusion

To address the long file read delay in distributed file system using erasure-coding based redundancy policy, we propose a new data cache allocation mechanism and data placement strategy using simulated annealing and convex optimization, respectively. In addition, we implement our approach in real-world distributed file system *Alluxio*. The experiment results show that our approach can effectively reduce the file read delay by an average of 29.8%, 32.6%, and 19.9%, compared to traditional *random*, *round* and *sprout* data placement strategies.

Acknowledgements

This work is supported by National Key Research and Development Program of China (Grant No.2016YFB1000304), National Natural Science Foundation of China (Grant No. 61502019), and the Open Project Program of the State Key Laboratory of Mathematical Engineering and Advanced Computing (Grant No. 2019A12). Hailong Yang is the corresponding author.

References

1. Aggarwal, V., Chen, Y.F.R., Lan, T., Xiang, Y.: Sprout: A functional caching approach to minimize service latency in erasure-coded storage. IEEE/ACM Transactions on Networking **25**(6), 3683–3694 (2017)
2. Aghayev, A., Weil, S., Kuchnik, M., Nelson, M., Ganger, G.R., Amvrosiadis, G.: File systems unfit as distributed storage backends: lessons from 10 years of ceph evolution. In: Proceedings of the 27th ACM Symposium on Operating Systems Principles. pp. 353–369 (2019)
3. Bao, H., Wang, Y., Xu, F.: An adaptive erasure code for jointcloud storage of internet of things big data. IEEE Internet of Things Journal **7**(3), 1613–1624 (2019)
4. Ciritoglu, H.E., Batista de Almeida, L., Cunha de Almeida, E., Buda, T.S., Murphy, J., Thorpe, C.: Investigation of replication factor for performance enhancement in the hadoop distributed file system. In: Companion of the 2018 ACM/SPEC International Conference on Performance Engineering. pp. 135–140 (2018)
5. Ding, C., Tang, C.: Infinite families of near mds codes holding t-designs. IEEE Transactions on Information Theory (2020)
6. Li, Z., Lv, M., Xu, Y., Li, Y., Xu, L.: D3: Deterministic data distribution for efficient data reconstruction in erasure-coded distributed storage systems. In: 2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS). pp. 545–556. IEEE (2019)
7. Liao, J., Cai, Z., Trahay, F., Peng, X.: Block placement in distributed file systems based on block access frequency. IEEE Access **6**, 38411–38420 (2018)
8. Mazumdar, S., Seybold, D., Kritikos, K., Verginadis, Y.: A survey on data storage and placement methodologies for cloud-big data ecosystem. Journal of Big Data **6**(1), 15 (2019)
9. Mohan, L.J., Rajawat, K., Parampalli, U., Harwood, A.: Optimal placement for repair-efficient erasure codes in geo-diverse storage centres. Journal of Parallel and Distributed Computing **135**, 101–113 (2020)
10. Nicolaou, N., Cadambe, V., Prakash, N., Konwar, K., Medard, M., Lynch, N.: Ares: Adaptive, reconfigurable, erasure coded, atomic storage. In: 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS). pp. 2195–2205. IEEE (2019)
11. Rashmi, K., Chowdhury, M., Kosaian, J., Stoica, I., Ramchandran, K.: Ec-cache: Load-balanced, low-latency cluster caching with online erasure coding. In: 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16). pp. 401–417 (2016)
12. Veeraiah, D., Rao, J.N.: An efficient data duplication system based on hadoop distributed file system. In: 2020 International Conference on Inventive Computation Technologies (ICICT). pp. 197–200. IEEE (2020)
13. Xiang, Y., Lan, T., Aggarwal, V., Chen, Y.F.R.: Joint latency and cost optimization for erasure-coded data center storage. IEEE/ACM Transactions on Networking **24**(4), 2443–2457 (2015)
14. Yu, Y., Huang, R., Wang, W., Zhang, J., Letaief, K.B.: Sp-cache: load-balanced, redundancy-free cluster caching with selective partition. In: SC18: International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 1–13. IEEE (2018)
15. Zhang, X., Cai, Y., Liu, Y., Xu, Z., Dong, X.: Nade: nodes performance awareness and accurate distance evaluation for degraded read in heterogeneous distributed erasure code-based storage. The Journal of Supercomputing pp. 1–30 (2019)