Explore Data Placement Algorithm for Balanced Recovery Load Distribution

Yingdi Shan, *Zhongguancun Laboratory and Tsinghua University;*
Kang Chen and Yongwei Wu, *Tsinghua University*

https://www.usenix.org/conference/atc23/presentation/shan

This paper is included in the Proceedings of the
2023 USENIX Annual Technical Conference.

July 10–12, 2023 • Boston, MA, USA

978-1-939133-35-9

Open access to the Proceedings of the
2023 USENIX Annual Technical Conference
is sponsored by

Explore Data Placement Algorithm for Balanced Recovery Load Distribution

Yingdi Shan[1,2], Kang Chen[2] and Yongwei Wu[2]

[1]Zhongguancun Laboratory
[2]Tsinghua University

Abstract

In distributed storage systems, the ability to recover from failures is critical for ensuring reliability. To improve recovery speed, these systems often distribute the recovery task across multiple disks and recover data units in parallel. However, the use of fine-grained data units for better load balancing can increase the risk of data loss.

This paper systematically analyzes the recovery load distribution problem and proposes a new data placement algorithm that can achieve load balancing without employing fine-grained data units. The problem of finding an optimal data placement for recovery load balancing is formally defined and shown to be NP-hard. A greedy data placement algorithm is presented, and experimental results demonstrate its superior performance compared to conventional techniques, with up to 2.4 times faster recovery. Furthermore, the algorithm supports low-overhead system expansion.

1 Introduction

In distributed storage systems, data is divided into smaller units called data units, which are grouped together in a placement group for reliability. For example, Google File System (GFS) [6] uses 64MB chunks as data units and replicates each chunk three times to form a placement group. The technique of erasure coding [7, 14, 18] can be utilized to calculate the parity of data units that have been grouped together to form a placement group, thereby providing another method for data reliability. If a single node fails in the storage system, the lost data units on that node must be repaired and distributed to other nodes, a process known as recovery. In this context, the term node is used to refer to an entity within the distributed system, which can be either a whole server or a single disk. Since there are many data units on a single node and different data units on the same node may belong to different placement groups, these storage systems can perform recovery in parallel, improving recovery speed [12, 13, 21]. Parallelized recovery has the potential to increase recovery speed so that it is limited by the cluster's bandwidth rather than the bandwidth of an individual node, improving the overall reliability of the storage system.

However, parallelized recovery does not necessarily imply faster recovery. An imbalance in the distribution of recovery load among various nodes can lead to congestion in certain nodes and prolonged recovery times. Imbalanced recovery loads can also negatively impact the performance of certain nodes, as they may have to devote a significant amount of their bandwidth to recovery.

To address these issues, distributed storage systems often employ fine-grained data units to balance the recovery load [6, 20]. By distributing a sufficient number of data units across various nodes, the recovery load can be evenly distributed through randomization [13]. However, this approach to fine-grained recovery, while effective in load balancing, also increases the risk that any placement group in the cluster fails [1–4, 9, 22]. Furthermore, the utilization of fine-grained data units can result in an uptick in overhead for the management of metadata.

This paper presents a novel data placement algorithm designed to achieve a more balanced recovery performance without the need for fine-grained data units. Data placement, which refers to the mapping of data units to disks, is challenging to design as it impacts data distribution and system scaling. Therefore, an effective data placement algorithm that can balance the recovery load should not have any negative impact on these perspectives.

To create a data placement algorithm for balancing the recovery load, this paper begins by formally defining the optimal recovery load distribution problem as selecting a set of nodes to minimize the weight of the edge with the highest weight in the recovery load graph. We then prove that this problem is NP-hard by showing that it can be transformed into a maximum independent set problem in polynomial time.

The paper then proposes a data placement algorithm based on a greedy strategy that is able to compute a more balanced data placement for recovery load distribution. The algorithm also supports low-overhead system expansion. Experiments demonstrate that the use of this data placement algorithm can improve the storage system's recovery performance by 1.7-2.4 times compared to the original data placement algorithm.

The contributions of this paper include:

- We formally define the optimal recovery load distribution problem. Following this, the paper proves that the problem is NP-hard.
- We propose a greedy data placement algorithm for efficient recovery. It is experimentally demonstrated that the algorithm is able to provide a more balanced recovery load distribution. The algorithm can support low-overhead system expansion.

2 Problem Definition and Analysis

2.1 Repair Load Matrix

The utilization of both replication and erasure coding techniques serves as a means to guarantee reliability in data storage systems. Various erasure codes exist, each having distinct read patterns. For instance, LRC (Local Reconstruction Codes) [7] utilize a minimal number of nodes for repair, while regenerating codes [5, 18] necessitate the read of data from a larger number of nodes but only a fraction of the information from each node. To harmonize these techniques, the repair load matrix is employed as a means to describe the repair process for both replication and erasure coding.

The repair load matrix is a matrix that represents the cost required to repair a failed node. In scenarios where I/O is the system bottleneck, the cost typically refers to I/O expenses, and nodes are often interpreted as individual disks. Conversely, when network bandwidth is the system's bottleneck, the cost usually denotes network expenditures, with nodes commonly referring to whole servers.

Let us denote the cardinality of a placement group as n. Correspondingly, the repair load matrix is represented by an $n \times n$ matrix W. The element $W_{i,j}$ within this matrix signifies the cost incurred to retrieve data from the jth node in the placement group when the ith node within the same group encounters a failure. For example, the repair load matrix for a $(k = 3, r = 1)$ Reed-Solomon (RS) code [14] is given below, and it shows that when any node fails, data needs to be read from all other nodes, and the cost of reading data from any other node is 1.

$$W = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

In the context of an RS code where the redundancy parameter r is greater than 1, there exists a multitude of plausible options for the repair load matrix W. This is contingent upon which nodes are selected for data reading during the recovery process. Any such selection can appropriately serve as the repair load matrix for the system.

For a $(k = 4, r = 2, l = 2)$ LRC code, the repair load matrix can be written as

$$W = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \end{pmatrix}$$

In this matrix, when repairing a data node or local parity node, the LRC needs to read data from $\frac{k}{l}$ nodes in the same group in order to reconstruct the original data. When a global parity node requires repair, the LRC must read all data nodes' contents.

2.2 Recovery Load Graph

The repair load matrix defines the repair process for a single placement group within a distributed storage system. In contrast, the recovery load graph (or recovery load matrix) characterizes the recovery load on each node in the event of a failure within the system. This is achieved by summing the repair load for each placement group. The recovery load graph is defined mathematically as follows:

Definition 1 (Recovery Load Graph)**.** *Let W be the repair load matrix defined above. Suppose there are N nodes in the storage system, and the storage system has S placement groups, denoted $P_1, P_2, \ldots, P_S$. Each placement group P_i is an array with length n, where each value, denoted by $P_{i,k}$, represents a node id. If $P_{t,k} = i$, then we define $Index(P_t, i) = k$. The recovery load graph corresponding to this storage system is $G = (V, E)$, where V is the set of points in the graph with cardinality N, representing the nodes in the system, and E is the set of edges in the graph. Let $E_{i,j}$ denote the edge weight between node i and node j, representing the load of reading data from node j when node i fails. Let $[i \in P]$ return 1 if $i \in P$ and 0 otherwise. The edge weight is calculated as follows:*

$$E_{i,j} = \sum_{t=1}^{S} [i \in P_t] \cdot [j \in P_t] \cdot W_{Index(P_t,i),Index(P_t,j)} \qquad (1)$$

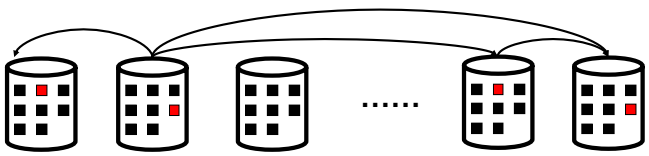


Figure 1: Recover Load Graph

2.3 Optimal Recovery Load Distribution

In this paper, we consider a system in which each placement group comprises an equal quantity of data. Placement groups that are not at capacity, as they consume a minimal amount of storage space, are disregarded. Upon the arrival of new data, the system must determine an appropriate placement group to accommodate it. The objective is to minimize the imbalanced recovery load upon the addition of a new placement group to the system. An alternative approach to this problem would be to determine the optimal recovery load distribution by considering all placement groups simultaneously, as opposed to incrementally identifying a single placement group. However, such an approach may significantly restrict the system's ability to add or remove a node, as it is hard to predict these events in advance.

An optimal distribution of recovery loads should strive to minimize the maximum recovery load after the integration of a new placement group. The definition of the optimal recovery load distribution problem is given below:

Definition 2 (Optimal Recovery Load Distribution Problem)**.** *Given a recovery load graph $G = (V, E)$, the optimal recovery load distribution problem is to construct an array P of length n such that all its elements belong to V, and the corresponding set of edges E', where $E'_{i,j} = E_{i,j} + [i \in P] \cdot [j \in P] \cdot W_{Index(P,i),Index(P,j)}$, such that the maximum value in the edge set E' is minimized. The optimal recovery load corresponding to graph G is:*

$$\min_{P} \max_{i,j \in V} E'_{i,j} \tag{2}$$

As shown in Figure 1, when node i and node j have a common placement group, the edge weight between them needs to be added to the repair cost corresponding to the repair load matrix. The optimal recovery load distribution problem then becomes a problem of selecting n nodes from the N nodes in graph G, such that the highest edge weight in the graph is minimized after the weights are added.

2.4 Complexity Analysis and Proof

In this section, we prove that the optimal recovery load distribution problem is NP-hard by reducing a known NP-complete problem, the maximum independent set problem [8], to it within polynomial complexity.

Definition 3 (Maximum Independent Set Problem)**.** *Given a graph $G = (V, E)$ and an integer n, let $N(i)$ represent the set of neighbor nodes of node i. The decision form of the maximum independent set problem is to determine whether there exists a set $P \subseteq V$ with cardinality not less than n such that $\forall i \in P$, $N(i) \cap P = \emptyset$.*

As shown in Figure 2, an independent set of a graph is a set of nodes such that no two nodes in the set have an edge between them. The red nodes in Figure 2 form an independent set of the graph because there are no edges between any two colored nodes. The maximum independent set problem is to find an independent set of cardinality not less than n in the graph.

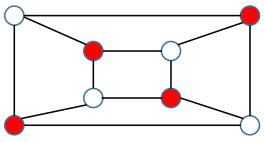


Figure 2: Maximum independent set problem

Lemma 1. *The maximum independent set problem can be reduced to the optimal recovery load distribution problem in polynomial time.*

Proof. We consider the following problem for an arbitrary graph $G = (V, E)$: Given the recovery load graph $G' = G = (V, E)$, where W is a repair load matrix with all entries equal to 1 except for the diagonal entries. The recovery load edge weight $E_{i,j} = 1$ when there is an edge in G from node i to node j. The goal is to find a placement group P that minimizes the maximum value in the updated edge set E' of graph G' after the integration of a new placement group.

We prove that the problem of finding the maximum independent set of graph G can be reduced to solving the optimal recovery load problem. Specifically, we show that the cardinality of the independent set of graph G is not less than n **if and only if** there exists a placement group P with cardinality n such that the optimal recovery load of graph G' is 1.

If there exists an independent set of cardinality not less than n in graph G, we can simply select n nodes from the independent set and place the new placement group on these n nodes in G'. Denote this set of n nodes as p. Since these n nodes are not adjacent to each other, $E_{i,j} = 0$ and $E'_{i,j} = 1$ for any node i and node j that belongs to p. Any other edges in E' remain the same as in E, which are no larger than 1. Therefore, the edge with the largest weight in E' will not exceed 1. As a result, the optimal recovery load corresponding to graph G' is 1.

Conversely, if the optimal recovery load corresponding to graph G' is 1, we assume that there is no independent set of cardinality not less than n in graph G. This means that we can arbitrarily select n nodes from the graph G, and at least two of these n nodes are adjacent. Then, if we select the same set from the graph G', the maximum edge weight of the edge $E_{i,j}$ in this set is 1. After the integration of the new placement, the edge weight of the edge $E'_{i,j}$ is at least 2 because W is a repair load matrix with all entries equal to 1 except for the diagonal entries, which suggests that the optimal recovery load for this graph is at least 2. This contradicts the premise, so it follows that there must exist an independent set of cardinality not less than n in the graph $G = (V, E)$.

Additionally, the reduction of the problem has a complexity of $O(|E|)$, as each edge of the graph G can be transformed into an edge in the recovery load graph by visiting it once. Thus, the problem of finding the maximum independent set can be efficiently reduced to the problem of determining the optimal recovery load distribution in polynomial time. □

Since the maximum independent set problem is an NP-complete problem [8], it follows from Lemma 1 that the optimal recovery load distribution problem is an NP-hard problem.

3 Algorithm Design

3.1 Data Placement Algorithm

This section presents a data placement algorithm based on a greedy strategy for the NP-hard problem of optimal recovery

load distribution. The algorithm aims to select nodes for a placement group of size n from a set of N nodes by choosing, at each step, the node whose sum of recovery costs to other nodes in the current group is smallest. While this method may not necessarily result in the optimal recovery load distribution, it can help to balance the recovery load.

The pseudocode for the algorithm is provided in Algorithm 1. This algorithm is responsible for mapping placement groups to nodes, and it is called to obtain the placement group P whenever a new placement group needs to be added to the storage system. The input to the algorithm is the current recovery load graph $G(V,E)$. The weight $WV(v)$ of a vertex v is defined as the sum of the weights of its adjacent edges, that is, $WV(v) = \sum_{e \in neighbor(v)} E(e)$. $D(v)$ is the number of data units on the node. Capital letters, such as V_0, represent sets, while lowercase letters, such as v_1, represent members of a set. For example, V_0 is a set of vertices and v_0 is a vertex within that set. The function *Pick* is used to randomly select a member from a set. The output of the algorithm is a placement group P.

Algorithm 1: Greedy Data Placement Algorithm

Input : $G(V,E)$
Output : P

1 $v_0 = Pick(\{v \mid WV(v) = min_{v' \in V}(WV(v'))\})$;
2 $P = \{v_0\}$;
3 **while** $|P| < n$ **do**
4 $\quad V_{candidates} = V - P$;
5 $\quad V_0 = V_{candidates} - \{v \mid violates\ criteria\}$;
6 $\quad V_1 = \{v \mid D(v) \leq (min_{v_0 \in V_0} D(v_0)) \cdot (1+\varepsilon) \wedge v \in V_0\}$;
7 $\quad V_2 = \{v \mid \sum_{v' \in P} E_{v,v'} = min_{v_1 \in V_1}(\sum_{v' \in P} E_{v_1,v'}) \wedge v \in V_1\}$;
8 $\quad v_{next} = Pick(V_2)$;
9 $\quad P = P \cup \{v_{next}\}$;
10 Update weight in G;

The proposed data placement algorithm based on the greedy strategy must exclude certain nodes from being placed in the same placement group. For example, nodes in the same placement group should not be on the same server or cabinet in order to reduce the risk of data loss due to systemic failures. Line 5 of the algorithm demonstrates how the candidate set V_0 is constructed based on this rule.

Heuristic rules are used to find the most suitable next node for the current placement group. This algorithm first selects the node with the smallest edge weight sum as the initial node. This process is shown by lines 1 to 2 of the algorithm. In order to achieve a uniform distribution of data, in line 6, V_1 is defined as a vertex whose number of data units does not exceed $(1+\varepsilon)$ times the minimum number of data units in the current storage system. Since the data is difficult to achieve absolute uniform distribution, this algorithm uses an adjustable parameter ε to limit the uniformity of the data placement.

To distribute the recovery load more evenly among nodes, V_2 is defined as the set of nodes with the smallest sum of recovery costs to other nodes in the current replacement group in the event of a failure, as shown in line 7 of the algorithm. Finally, a vertex v_{next} is chosen from V_2 as the next candidate for the placement group and added to the set P. This process continues until n members have been added to the set P, after which the corresponding weights in G can be updated.

When a node represents a server instead of a disk, data units for each node in the group require assignment to a specific disk. This assignment can be efficiently achieved through a round-robin strategy.

3.2 Target Node Selection

When a single node fails and requires recovery, the data stored on it must be redistributed to other nodes in order to maintain the reliability of the system. During this process, it is crucial to select nodes to receive the data units from the failed node in a manner that ensures even distribution of recovered data and maintains balance in the recovery load. The algorithm in Algorithm 2 can be used to select the location of data units to be redistributed to other nodes during the recovery process. This algorithm first excludes nodes that do not meet certain criteria as in Algorithm 1. Next, the algorithm looks for nodes with used storage space within a certain range in order to achieve a balanced distribution of data, as described in line 2. The algorithm then selects nodes that will help maintain recovery load balance after the faulty node is removed, as shown in line 3. Finally, the algorithm aims to evenly distribute the load during recovery; therefore, if the previous conditions are met, it will select the node with the least connection weight to the failed node, as shown in line 4.

Algorithm 2: Target Node Selection for Recovey

Input : $G(V,E)$, placement group P, failure node F
Output : v

1 $V_0 = V - \{v \mid violates\ criteria\}$;
2 $V_1 = \{v \mid D(v) \leq (min_{v_0 \in V_0} D(v_0)) \cdot (1+\varepsilon) \wedge v \in V_0\}$;
3 $V_2 = \{v \mid \sum_{v' \in P} E_{v,v'} = min_{v_1 \in V_1}(\sum_{v' \in P} E_{v_1,v'}) \wedge v \in V_1\}$;
4 $V_3 = \{v \mid E_{v,F} = min_{v_2 \in V_2} E_{v_2,F} \wedge v \in V_2\}$;
5 $v = Pick(V_3)$;
6 Update weight in G;

Since each node may store multiple data units, Algorithm 2 must be run for each placement group to determine the target nodes.

3.3 System Expansion

When adding new nodes to the system, the graph G must be updated to include the nodes corresponding to these new devices. The system can then continue to call the algorithm

in Algorithms 1 to automatically add new placement groups containing the new nodes to the storage system, ensuring that the recovery load remains as small as possible without requiring any data migration.

However, when adding multiple nodes at once, using the algorithm in Algorithms 1 directly can result in all the new placement groups being placed on the same batch of newly added nodes. This can lead to an imbalanced recovery load, with a concentration of recovery load and writing load on the newly added devices, even as more placement groups are added. To address this issue, the algorithm should control the rate at which data is placed on the new nodes during system expansion.

To do this, the new nodes can be placed in a separate collection. Each time the system is expanded, the user can specify a parameter c to control the placement rate of data on the new devices. When calculating data placement, at most c nodes will be selected from this collection, with the remaining $n-c$ devices being selected from other devices according to the algorithm in Algorithm 1. Once the number of data units on a node within the collection matches the number of data units on the node with the least amount of data outside of this collection, the node can be removed from the collection.

4 Evaluation

4.1 Evaluation Setup

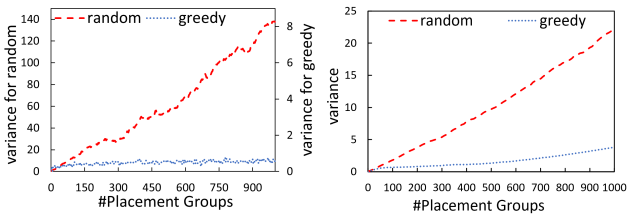
In the context of our evaluation, we designate each individual disk as a node within the system. We carry out both micro-benchmark experiments and overall performance testing to assess performance. The micro-benchmark experiments, performed outside of an actual storage system, specifically evaluate the variability in recovery load distribution and data distribution. Instead of executing actual recovery, micro-benchmark experiments provide an analysis of the recovery load distribution, which are complementary to our overall performance evaluation.

For the overall performance test, we employ a cluster of 16 servers, with each server boasting dual Intel Xeon E5 2643 v4 CPUs, 128GB of 2133 MHz DDR4 memory, a 512GB SATA3 SSD, and six 8TB 7200rpm SAS HDDs. The six HDDs are independent and not grouped by a RAID. All servers run on the CentOS 7.8 operating system. The servers are networked via a 56Gbps Infiniband connection, with an MTU size of 65520, enabling us to better utilize the high-bandwidth network in our evaluation. We set the value of ε to 0.02 for all experiments to maintain a consistent testing environment.

4.2 Micro Benchmark

We conduct micro-benchmark experiments to see how our algorithms can help to improve recovery load balance. In all experiments, N is set to 100, and RS(10,4) code is employed for each placement group.

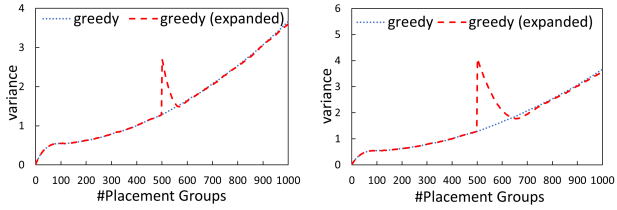
Data Distribution. Figure 3a measures the uniformity of data distribution for different data placement algorithms by calculating the variance of the number of data units. It can be seen that the greedy data placement algorithm is far more uniform than the random data placement algorithm.



(a) Data Distribution (b) Recovery Load Distribution

Figure 3: The variance of data distribution and recovery load distribution.

Recovery Load Distribution Figure 3b presents the recovery load distribution of the system when utilizing different data placement algorithms. The data depicted in the figure demonstrates that the recovery load is more evenly distributed when using a greedy data placement algorithm as compared to random data placement.



(a) Add a disk (b) Add two disks

Figure 4: The variance of the recovery load after system expansion.

Recovery Load Distribution for System Expansion We evaluate the effects of system expansion on recovery load distribution. In this experiment, the variable c is set to 1. The results are illustrated in Figure 4a and Figure 4b. Specifically, Figure 4a represents the scenario where an additional disk is added to the system when there are 500 placement groups stored, while Figure 4b illustrates the situation where two additional disks are added simultaneously.

The data illustrated in Figure 4a demonstrates that while the greedy algorithm may initially create a temporary imbalance in recovery load after system expansion, the recovery load returns to a normal state as more placement groups are integrated. This phenomenon arises due to the recovery load on newly introduced disks being zero initially, resulting in a sharp uptick in the variability of the recovery load distribution. However, as these new disks become populated with data, the recovery load re-establishes its balance.

Conversely, Figure 4b illustrates that simultaneous addition of multiple disks leads to a more pronounced imbalance in the

Table 1: The average recovery time of different data placement algorithms.

Codes	Random	Greedy	Improvement
RS code	554s	273s	2.1x
LRC	460s	192s	2.4x
Clay code	240s	141s	1.7x

recovery load of the storage system and that it takes longer for the system to return to normal.

4.3 Overall Performance

This experiment measures the recovery performance using different data placement algorithms and different erasure codes, including RS code, LRC, and Clay Code [18]. To achieve this, 96 hard disks were distributed across 16 machines, with each data unit set at a size of 10GB and a total of 175 placement groups in the system. The algorithms are integrated into the RCStor storage system [16], as it provides high recovery performance and various erasure codes. Prior to measuring recovery, data of approximately 256GB was placed on each disk, resulting in a total data size of 24.4TB. The recovery process was initiated manually by shutting down a disk, and the time required for recovery completion was recorded. To gauge the maximum recovery bandwidth, we initiated the recovery of all failed placement groups simultaneously. It should be noted that during recovery, the data from the failed disks was reconstructed on other functioning disks. We ensured there was no bandwidth cap for the recovery process. Additionally, the recovery was carried out at a time when the system was not in use, to avoid any operational interruptions. We conducted 10 trials to ascertain the average recovery time. The experimental results are presented in Table 1.

The data in Table 1 illustrates that the use of a greedy data placement algorithm can significantly enhance the recovery performance for various erasure codes. Specifically, the recovery performance is found to be 1.7-2.4 times greater than that of the random data distribution algorithm.

We have also evaluated the influence of randomness on recovery time. Our findings suggest that when utilizing the greedy data placement algorithm, the impact of randomness fluctuates within a range of ±10%. However, with the random data placement algorithm, this variability increases to ±20%.

5 Related Work

Copyset [4] proposes to reduce the probability of data loss by reducing the number of distinct placement groups, which are referred to as copysets. In contrast, the focus of our paper is to ensure recovery load balance given a predetermined small number of placement groups. This number is approximately the same as the number of copysets, contingent upon parameter configurations. In essence, we tackle the issue of data loss by balancing the recovery load given a fixed number of placement groups. This aspect marks a departure from the Copyset paper, which does not focus on recovery load balancing.

PDL [23] is a data placement algorithm that reduces imbalances in inter-cabinet network communication. SelectiveEC [24] maintains load balance during recovery by dynamically selecting nodes for reading and writing through bipartite graph matching. However, they only reduce recovery load imbalance for the network and cannot guarantee the balance of load when accessing disks.

RAID-based data placement algorithms [11, 15, 17, 19, 26] are designed for use with disk arrays, which are not appropriate for use in distributed storage systems. RAID+ [25] and D_3 [10] use orthogonal Latin squares to distribute data and ensure load balance for disk array recovery, but they lack scalability and can only be applied to arrays with tens or hundreds of disks. They also do not support dynamic system expansion.

Overall, existing data placement algorithms struggle to simultaneously provide load balancing for recovery, scalability and low migration overhead.

6 Discussion

In a storage system capable of tolerating multiple failures, resulting in numerous potential repair load matrices, a practical advantage may arise from dynamically selecting recovery sources, taking into account observed loads and stragglers. This dynamic selection aligns with our algorithm and could involve dynamically choosing k nodes from each placement group for data recovery, excluding stragglers, to minimize the maximal recovery load. The future challenge lies in developing an efficient algorithm for this process or devising other methods to account for the impacts of such dynamism.

7 Conclusion

This paper proposes a data placement algorithm based on a greedy strategy that can provide a more balanced recovery load distribution. Experiments show that using the data placement algorithm can improve the recovery performance of the storage system to 1.7-2.4 times compared to the random data placement algorithm.

Acknowledgments

We thank the anonymous reviewers and our shepherd, Nathan Bronson, for their valuable comments and helpful suggestions. The authors from Tsinghua University are all in the Department of Computer Science and Technology, Beijing National Research Center for Information Science and Technology (BNRist), Tsinghua University, China. This work is supported by National Key Research & Development Program of China (2022YFB4502004), Natural Science Foundation of China (62141216, 61877035) and Tsinghua University Initiative Scientific Research Program.

References

[1] Medha Bhadkamkar, Jorge Guerra, Luis Useche, Sam Burnett, Jason Liptak, Raju Rangaswami, and Vagelis Hristidis. BORG: Block-reORGanization for self-optimizing storage systems. In *7th USENIX Conference on File and Storage Technologies (FAST '09)*, pages 183–196, 2009.

[2] André Brinkmann, Kay Salzwedel, and Christian Scheideler. Efficient, distributed data placement strategies for storage area networks. In *Proceedings of the twelfth annual ACM symposium on Parallel algorithms and architectures (SPAA '00)*, pages 119–128, 2000.

[3] Byung-Gon Chun, Frank Dabek, Andreas Haeberlen, Emil Sit, Hakim Weatherspoon, M. Frans Kaashoek, John Kubiatowicz, and Robert Morris. Efficient replica maintenance for distributed storage systems. In *Proceedings of the 3rd Symposium on Networked Systems Design and Implementation (NSDI '06)*, pages 45–58, 2006.

[4] Asaf Cidon, Stephen Rumble, Ryan Stutsman, Sachin Katti, John Ousterhout, and Mendel Rosenblum. Copysets: Reducing the frequency of data loss in cloud storage. In *Proceedings of the 2013 USENIX Conference on Annual Technical Conference (USENIX ATC '13)*, pages 37–48, 2013.

[5] Alexandros G Dimakis, P Brighten Godfrey, Yunnan Wu, Martin J Wainwright, and Kannan Ramchandran. Network coding for distributed storage systems. *IEEE Transactions on Information Theory*, 56(9):4539–4551, 2010.

[6] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles (SOSP '03)*, pages 29–43, 2003.

[7] Cheng Huang, Huseyin Simitci, Yikang Xu, Aaron Ogus, Brad Calder, Parikshit Gopalan, Jin Li, Sergey Yekhanin, et al. Erasure coding in windows azure storage. In *2012 USENIX Annual Technical Conference (USENIX ATC '12)*, pages 15–26, 2012.

[8] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[9] Jack YB Lee and John CS Lui. Automatic recovery from disk failure in continuous-media servers. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 13(5):499–515, 2002.

[10] Zhipeng Li, Min Lv, Yinlong Xu, Yongkun Li, and Liangliang Xu. D3: Deterministic data distribution for efficient data reconstruction in erasure-coded distributed storage systems. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS '19)*, pages 545–556. IEEE, 2019.

[11] Alberto Miranda and Toni Cortes. Raid: Online raid upgrades using dynamic hot data reorganization. In *12th USENIX Conference on File and Storage Technologies (FAST '14)*, pages 133–146, 2014.

[12] Edmund B Nightingale, Jeremy Elson, Jinliang Fan, Owen Hofmann, Jon Howell, and Yutaka Suzue. Flat datacenter storage. In *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI '12)*, pages 1–15, 2012.

[13] John Ousterhout, Arjun Gopalan, Ashish Gupta, Ankita Kejriwal, Collin Lee, Behnam Montazeri, Diego Ongaro, Seo Jin Park, Henry Qin, Mendel Rosenblum, et al. The ramcloud storage system. *ACM Transactions on Computer Systems (TOCS)*, 33(3):1–55, 2015.

[14] Irving S Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, 8(2):300–304, 1960.

[15] Beomjoo Seo and Roger Zimmermann. Efficient disk replacement and data migration algorithms for large disk subsystems. *ACM Transactions on Storage (TOS)*, 1(3):316–345, 2005.

[16] Yingdi Shan, Kang Chen, Tuoyu Gong, Lidong Zhou, Tai Zhou, and Yongwei Wu. Geometric partitioning: Explore the boundary of optimal erasure code repair. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP '21)*, page 457–471, 2021.

[17] Lei Tian, Dan Feng, Hong Jiang, Ke Zhou, Lingfang Zeng, Jianxi Chen, Zhikun Wang, and Zhenlei Song. PRO: A popularity-based multi-threaded reconstruction optimization for RAID-Structured storage systems. In *5th USENIX Conference on File and Storage Technologies (FAST '07)*, pages 301–314, 2007.

[18] Myna Vajha, Vinayak Ramkumar, Bhagyashree Puranik, Ganesh Kini, Elita Lobo, Birenjith Sasidharan, P Vijay Kumar, Alexandar Barg, Min Ye, Srinivasan Narayanamurthy, et al. Clay codes: moulding MDS codes to yield an MSR code. In *16th USENIX Conference on File and Storage Technologies (FAST '18)*, pages 139–154, 2018.

[19] Jiguang Wan, Jibin Wang, Changsheng Xie, and Qing Yang. S^2raid: Parallel raid architecture for fast data recovery. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 25(6):1638–1647, 2013.

[20] Li Wang, Yiming Zhang, Jiawei Xu, and Guangtao Xue. MAPX: Controlled data migration in the expansion of decentralized Object-Based storage systems. In *18th USENIX Conference on File and Storage Technologies (FAST '20)*, pages 1–11, 2020.

[21] Sage A Weil, Scott A Brandt, Ethan L Miller, Darrell DE Long, and Carlos Maltzahn. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th symposium on Operating systems design and implementation (OSDI '06)*, pages 307–320, 2006.

[22] Qin Xin, Ethan L Miller, and SJ Thomas JE Schwarz. Evaluation of distributed recovery in large-scale storage systems. In *13th IEEE International Symposium on High performance Distributed Computing (HPDC '04)*, pages 172–181, 2004.

[23] Liangliang Xu, Min Lv, Zhipeng Li, Cheng Li, and Yinlong Xu. PDL: A data layout towards fast failure recovery for erasure-coded distributed storage systems. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pages 736–745, 2020.

[24] Liangliang Xu, Min Lyu, Qiliang Li, Lingjiang Xie, and Yinlong Xu. SelectiveEC: Selective reconstruction in erasure-coded storage systems. In *12th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage '20)*, 2020.

[25] Guangyan Zhang, Zican Huang, Xiaosong Ma, Songlin Yang, Zhufan Wang, and Weimin Zheng. RAID+: Deterministic and balanced data distribution for large disk enclosures. In *16th USENIX Conference on File and Storage Technologies (FAST '18)*, pages 279–294, 2018.

[26] Weimin Zheng and Guangyan Zhang. FastScale: Accelerate RAID scaling by minimizing data migration. In *9th USENIX Conference on File and Storage Technologies (FAST '11)*, pages 149–161, 2011.