







Sector Error-Oriented Durability-Aware Fast Repair in Erasure-Coded Cloud Storage Systems

Yifei Xiao¹ , Shijie Zhou¹ , Linpeng Zhong¹ , and Zhao Zhang² 

¹ School of Information and Software Engineering, University of Electronic Science and Technology of China, Chengdu, China

sjzhou@uestc.edu.cn

² School of Public Affairs and Administration, University of Electronic Science and Technology of China, Chengdu, China

Abstract. There is a variety of erasure-coded data placement schemes that make a great contribution to data repair. To repair data, the operator should replace the failed node with a new node first. However, almost all these schemes assume the node replacement process (NRP) is done quickly, which is not true. Generally, NRP includes failure detection and failure repair, which may take hours or even days. Long delay of replacement may cause the recovered data lost again due to the lack of durability. To improve data durability, we propose a novel scheme called Sector Error-Oriented Durability-Aware Fast Repair (SEDRRepair), which carefully couples data migration and data reconstruction in parallel for data repair. We conduct mathematical analysis and compute the optimal repair in our model. The results show that, compared to the traditional erasure coding methods, SEDRRepair saves the repair time by up to 60% in most cases and improves data durability while keeping minimal storage.

Keywords: Data repair · Node replacement · Sector error · Data durability · Erasure coding

1 Introduction

As failures are the norm in cloud storage systems, improving data reliability while maintaining the system performance during data repair is one of the most important challenges in the literature [12]. To guarantee data reliability in the face of failures, erasure coding techniques are gaining popularity due to their comparable fault tolerance with reduced storage overhead compared to simple replication.

A variety of data placement schemes based on erasure coding make a great contribution for data reliability [1, 3, 6, 8, 15]. To recover data, the operator should replace the failed node with a new node first. However, almost all these schemes assume the node replacement process (NRP) is done quickly, which is not true. Figure 1 shows an abstract model to characterize gray failure [9], which means

the external app observes a failure but the internal observer does not. While the model also reveals the general process of NRP, which is used to handle simple crash and fail-stop (CFS) failures. In general, as shown in Fig. 1, the process of NRP includes 2 steps: ❶ failure detection, in which the observer detect a CFS failure based on the external app’s report or its own probing. If both the app and the observer agree that the system is experiencing a failure, a CFS failure can be confirmed, ❷ failure repair, in which the node replacement should be done and the reactor will do some repair work, such as rebooting or reconstruction.

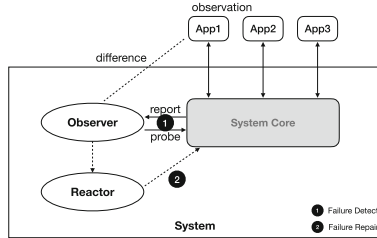


Fig. 1. The abstract model of a data center, in which there are 2 logical entities: a system, which provides a service, and an app, which uses system. Examples of a system include a distributed storage service, a data center network, a web search service, and an IaaS platform. An app could be a web application, a user, or an operator. The observer can check a failure either from the app’s reports or its own probes, and this process is called **failure detection**. Once a failure is confirmed, the observer can inform reactor to handle it, such as rebooting or reconstruction, and this process is called **failure repair**.

NRP is not easy because every part of the process can be time-consuming. In failure detection, the observer asks the error detector to run at a certain period. The interval of detection can not be too short to degrade the performance of the system, or be too long to pick up the failure too late. Besides, nodes can become unavailable for a large number of reasons (e.g., a storage node is overloaded, a node binary may crash, a machine may experience a hardware error [5]). Fortunately, the vast majority of such unavailability events are transient and do not result in permanent data loss, but it still needs 15 min to filter most transient failures since less than 10% of events had node unavailability with a duration under 15 min [5]. Besides, waiting for the app’s observation is also time-consuming, because users may not report or report soon even when they are afflicted by failures. In failure repair, the reboot scheme will take some time and NRP is also time-consuming since the system is online 24/7 while those operators are not. Therefore, NRP cannot be done immediately, the delay can be hours or even days.

The longer the delay NRP takes, the higher the risk of losing recovered data. To the best of our knowledge, there are only memories or caches can be used to save recovered data before NRP. Memory overflow, server malfunction or power outage could take place at any time, thus the updates of recovered data may be

lost permanently due to the lack of durability (updates are very common, more than 90% of write requests are updates [18]).

It has long been recognized that encoding data into its erasure-coded form will incur a much heavier computation load than simple replication [20], thus there have been extensive studies on improving the repair performance of erasure coding, such as proposing theoretically proven erasure codes that minimize the repair traffic or I/Os [8, 18] or proposing methods based on XOR operations to reduce the computation load [16], or to accelerate the computation by better utilizing the resources in modern CPUs [23]. **However, simple replication can offer continuous data durability in the face of failures, which is overlooked in the literature.** Against the above backdrop, we restrict our attention to data repair based on continuous data durability where it is required that any storage nodes can not impede data durability even during data repairing. To this end, we seek to answer the following questions: 1) How to guarantee data durability before NRP? Can we provide continuous data durability without adding extra storage (e.g., store data in other healthy nodes)? 2) Based on 1), how to repair lost blocks quickly after NRP?

To answer the above questions, we did a lot of research and proposed an effective scheme called Sector Error-Oriented Durability-Aware Fast Repair (SEDRRepair) to speed up data repair.

- We first start to issue the problem of continuous data durability, and we mathematically analyze the optimal repair and its conditions to be met.
- We propose Sector Error-Oriented Durability-Aware Fast Repair (SEDRRepair) to provide fast repair based on data durability when the sector error occurs.
- Our work is generic, i.e., it can combine with other erasure codes and tackle multi-failure cases.
- We conduct extensive test results and show that SEDRRepair can effectively reduce the total repair time while maintaining continuous data durability.

The rest of the paper is organized as follows. In Sect. 2, we introduce background and our motivation. In Sect. 3, we conduct mathematical analysis of our model. In Sect. 4, we present the implementation of SEDRRepair. We evaluate SEDRRepair in Sect. 5 and introduce the related work in Sect. 6. Finally, the conclusion of our work is in Sect. 7.

2 Background and Motivation

2.1 Erasure Codes and RS Codes

A leading technique to achieve strong fault-tolerance in cloud storage systems is to utilize erasure codes. Erasure codes are usually specified by two parameters: the number of data symbols k to be encoded, and the number of coded symbols n to be produced [22]. The data symbols and the coded symbols are usually assumed to be in finite field $GF(2^w)$ in computer systems. A (n, k) erasure codes storage system composed of n nodes dedicates k nodes to data, and the remaining $(n - k)$ nodes are dedicated to coding.

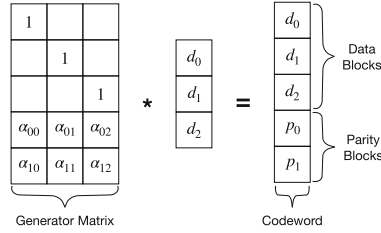


Fig. 2. The encoding process of RS(5, 3). The leftmost matrix is called *generator matrix*, which encodes data blocks (d_0, d_1, d_2) into codeword $(d_0, d_1, d_2, p_0, p_1)$.

RS codes [17] are a well-known erasure code construction and have been widely deployed in production [7, 14, 15, 17, 21]. RS(n, k) encodes k uncoded equal-size blocks into a stripe with n coded equal-size blocks via linear combinations based on $GF(2^w)$. Figure 2 shows the typical encoding process of RS(5, 3), where the leftmost matrix is called *generator matrix*, which can be generated from Vandermonde matrix [17] or Cauchy matrix [2]. The top k rows of the generator matrix compose a $k \times k$ *identity matrix* (here $k = 3$). The remaining m rows are called *coding matrix* [15] (here $m = n - k = 2$). The generator matrix encodes the data blocks (denoted by d_0, d_1, d_2) into a *codeword* $(d_0, d_1, d_2, p_0, p_1)$. Each block can refer to one symbol in the codeword. After encoding, data blocks (d_0, d_1, d_2) will be sent to the corresponding data nodes and the parity blocks (p_0, p_1) will be sent to the corresponding parity nodes. From Fig. 2 we can infer that, in a (n, k) RS-based cloud storage system, each parity block could be represented by the linear combination of the k data blocks with the following equation,

$$p_i = \sum_{j=0}^{k-1} \alpha_{i,j} d_j, i \in [0, m - 1] \tag{1}$$

In this paper, we also use RS codes to generate the parity blocks.

2.2 Motivation

As mentioned in Sect. 1, NRP can be time-consuming, which may lead a lack of data durability for a long time. However, most of data repair schemes overlooked this problem, which increases the risk of losing data. In this paper, we seek to fill this gap in the literature.

Besides, there is a large body of work that overlooked the accessibility of the failed node with sector error, thus we utilize it to accelerate the process of data repair as well as improving data durability.

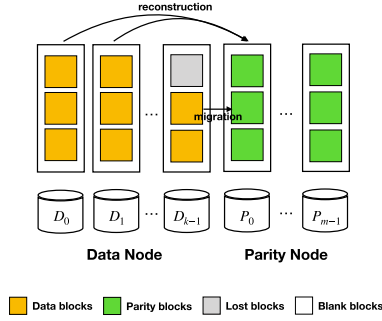


Fig. 3. The repair model for BN. (Color figure online)

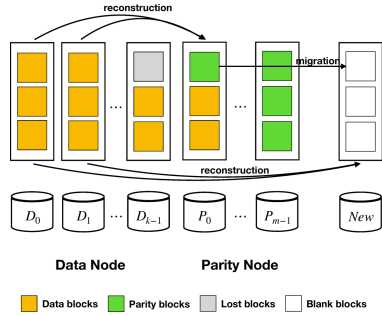


Fig. 4. The repair model for AN (the new node is as a data node).

3 Mathematical Analysis

We conduct simple mathematical analysis to provide preliminary insights into the performance gain of the theoretical optimal repair over the conventional repair in a cloud storage system. For ease of presentation, we assume there is only one failed node.

In our model, we employ RS(n, k), thus we have k data nodes and $n - k$ parity nodes. We label the data nodes as $D_i, i \in [0, k - 1]$ and the parity nodes as $P_j, j \in [0, m - 1], m = n - k$. Generally, every node has identical number of blocks, and we denote it as U . Furthermore, as we only consider the failed node with sector error, i.e., we can use parts of the failed node. Let M denote the accessible number of blocks, thus the number of lost blocks is $U - M$. As shown in Fig. 3, the failed node D_{k-1} has $U - M$ lost blocks colored gray and M good blocks colored yellow.

As mentioned above, according to whether the NRP is done, we divided data repair into 2 phases: ① BN (before NRP), ② AN (after NRP).

For the First Phase (BN): As mentioned earlier, in BN, we can couple migration and reconstruction. For migration, let t_m, M_1 denote the time to migrate a block from one node to another node and the migration number of blocks

in BN, respectively. Similarly, for reconstruction, let t_r, C_1 denote the time to reconstruct a block and the reconstruction number of blocks in BN, respectively. As Eq. (1) shows, the reconstruction of one block needs to receive k blocks from other k healthy nodes, while migration only needs to transmit one block from one node to another, obviously, $t_m < t_r$. As migration and reconstruction can run in parallel (see details in Sect. 4), we have,

$$T_1 = M_1 \cdot t_m = C_1 \cdot t_r, M_1 \in (0, M], C_1 \in [0, U - M] \tag{2}$$

Now, $(C_1 + M_1)$ lost blocks are re-accessible, thus the meta info server should change the addresses of the lost blocks to P_0 , ensuring smooth connections between users and these lost blocks.

For the Second Phase (AN): In this phase, the NRP is done, i.e., a new node is available (as shown in Fig. 4, here we call the new node N). As P_0 stores some data blocks of D_{k-1} for data durability in BN, we should decide the roles of P_0 and N in this phase (e.g., if the NRP is too slow that most blocks of P_0 are data blocks of the failed node D_{k-1} , maybe it is better to let P_0 replace D_{k-1} as a new data node). So we have two choices: ① exchange the roles of P_0 and N , or ② maintain the roles of P_0 and N .

Taking the example of choosing ①, that is, we should fully fill P_0 with data blocks and fill N with parity blocks. As shown in Fig. 4, it is required to do three things:

1. Migration ($P_0 \rightarrow N$), it is required to migrate the parity blocks from P_0 to N , as P_0 receives M_1 data blocks from migration and C_1 data blocks from reconstruction in BN, there are $U - M_1 - C_1$ parity blocks colored green left which can be migrated to N (as $t_m < t_r$, we prefer to employ migration).
2. Reconstruction for N ($ER \rightarrow N$, ER means erasure coding), there are $M_1 + C_1$ blank blocks of N demanding to be filled by reconstruction.
3. Reconstruction for P_0 ($ER \rightarrow P_0$), as P_0 moves $U - M_1 - C_1$ parity blocks to N , these positions should be filled with data blocks by reconstruction.

Let T_2 and T be the repair time of AN and the total repair time, respectively. So we have,

$$\begin{aligned} T_2 &= \max((U - M_1 - C_1) \cdot t_m, \\ &\quad (M_1 + C_1) \cdot t_r, (U - M_1 - C_1) \cdot t_r) \\ &= \max((M_1 + C_1) \cdot t_r, (U - M_1 - C_1) \cdot t_r) \end{aligned} \tag{3}$$

$$T = T_1 + T_2 \tag{4}$$

From Eq. (2) to Eq. (4), we can readily show that T is minimized when $(M_1 + C_1) \cdot t_r = (U - M_1 - C_1) \cdot t_r$, so we get,

$$C_1 = \frac{t_m}{t_r} M_1 \tag{5}$$

$$C_1 + M_1 = \frac{U}{2} \tag{6}$$

If we choose ②, similarly, it is required to do three things:

1. Migration ($P_0 \rightarrow N$), as P_0 receives M_1 data blocks from migration and C_1 data blocks from reconstruction in BN, we have $M_1 + C_1$ data blocks colored yellow which can be migrated to N .
2. Reconstruction for N ($ER \rightarrow N$), N still need $U - M_1 - C_1$ data blocks which are generated by reconstruction.
3. Reconstruction for P_0 ($ER \rightarrow P_0$), as P_0 moves $M_1 + C_1$ parity blocks to N , these positions should be filled with data blocks by reconstruction.

So we have,

$$\begin{aligned} T_2 &= \max((M_1 + C_1) \cdot t_m, \\ &\quad (U - M_1 - C_1) \cdot t_r, (M_1 + C_1) \cdot t_r) \\ &= \max((M_1 + C_1) \cdot t_r, (U - M_1 - C_1) \cdot t_r) \end{aligned} \quad (7)$$

The Interesting Thing is, no Matter we Choose ① or ②, T_2 is the Same. But here is the special case: if NRP is not finished (e.g., the operator notices the failed node too late) until all the migrations and reconstructions have been completed in P_0 , which means, P_0 is totally a ‘DataNode’. Obviously, ① is the better choice. Except for the special case, ① and ② have the same result.

In conclusion, Eq. (5) and Eq. (6) are the conditions of achieving the optimal repair, which is in accordance with the results of our simulation experiments in Sect. 5.

In this section, we only analyze the single-failure case, but if there are m failed data nodes, e.g., $D_i, i \in [0, m - 1]$ is failed, we can set $(D_i, P_i), i \in [0, m - 1]$ as pairs, and use the same method to repair them. Therefore, the scheme can also be used in multi-failure cases.

4 Implementation

To address data durability in data repair, we propose a novel scheme Sector Error-Oriented Durability-Aware Fast Repair (SEDRRepair), which is divided into 2 parts: 1) BN, 2) AN.

Similar to [20], to simplify our analysis, we do not address disk I/O interference, which occurs in the following cases: ❶ a node reads a block for reconstruction and reads another block for reconstruction, and ❷ a node reads a block while writing another block. Meanwhile, we do not consider the computational costs of coding operations, which are negligible compared to disk I/Os and network transmission [10].

4.1 BN

First let us review the first question: 1) How to provide data durability before NRP? As mentioned earlier, updates are common in cloud storage systems.

Before NRP, we can only store the updates of the failed node in caches or memories, however, this is dangerous. To improve the data durability, we set the parity node as the temporary place for the data blocks of the failed node, since it's known that the parity node stores linear combinations of other data blocks instead of primitive data. Therefore, it's acceptable to replace parity blocks with data blocks for data durability.

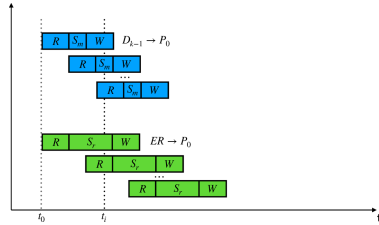


Fig. 5. The pipeline model of BN. (Color figure online)

Choosing the parity node as the temporary storage node to provide data durability is a straightforward way, but it's provably effective. Besides, it's not required to use extra storage to save data.

Feasibility for Parallel: In Sect. 3, t_m and t_r represent the time to migrate a block from one node to another node and the time to reconstruct a block of the failed node, respectively. Now we extend our general formulation to model the values of t_m and t_r in detail.

Figure 5 illuminates the reason why we can couple migration and reconstruction in parallel: for migration ($D_{k-1} \rightarrow P_0$), t_m (the blue bars) consists of three parts:

- R , the read time of a block,
- S_m , the transmission time of a block from one node to another;
- W , the write time of a block;

For ease of presentation, we assume $R = W$. For reconstruction ($ER \rightarrow P_0$), t_r (the green bars) also consists of three parts:

- R , the read time of a block, which occurs in k helpers,
- S_r , the total transmission time of all k helpers for sending k blocks, because $t_m < t_r$, we get $S_m < S_r$;
- W , the write time of a block;

At any moment of BN (e.g., at t_i in Fig. 5), where it is required to do three things for migration:

1. writing a block in P_0 ;
2. transferring a block from D_{k-1} to P_0 ;
3. reading a block in D_{k-1} ;

As mentioned above, we do not consider the disk I/O interference, thus they can run in parallel. On the other hand, we focus on the green part, where it is required to do 2 things for reconstruction: 1) transferring k blocks to P_0 ; 2) reading k blocks from k helpers;

Obviously, we can also perform them in parallel. Therefore, it is feasible to couple migration and reconstruction in BN.

Algorithm 1: BN Algorithm

```

Data:
the number of good blocks in the failed node  $M$ ;
the number of blocks in a node  $U$  ;
the past time before new node is available  $mt$ ;
Result:
the number of lost blocks repaired in BN  $C$ ;
// migration
1 if  $R > S_m$  then
2   |  $t_{max} = R; t_{min} = S_m$ ;
3 else
4   |  $t_{max} = S_m; t_{min} = R$ ;
5 end
6  $mCnt = (mt - t_{min} - W)/t_{max}$ ;
7  $mTime = M \times t_{max} + t_{min} + W$ ;
8 if  $mCnt < M$  then
9   |  $C = mCnt$ ;
10 else
11   |  $C = M$ ;
12 end
// reconstruction
13 if  $W > S_r$  then
14   |  $t_{max} = W; t_{min} = S_r$ ;
15 else
16   |  $t_{max} = S_r; t_{min} = W$ ;
17 end
18  $rCnt = (mt - t_{min} - R)/t_{max}$ ;
19  $rTime = (U - M) \times t_{max} + t_{min} + R$ ;
20 if  $rCnt > (U - M)$  then
21   |  $C = C + rCnt$ ;
22 else
23   |  $C = C + (U - M)$ ;
24 end
25  $T_1 = \max(mTime, rTime)$ ;

```

Algorithm Details: Algorithm 1 presents the main idea of BN. Let C denote the number of blocks in the failed node repaired in BN, and mt denote the past time before new node is available (if $T_1 < mt$, that means the repair is over but

new node has not arrived). As shown in Fig. 5, we should align the blue bars based on the longest time consumer (here $R > S_m$, we align the blue bars based on R), and store it to t_{max} (lines 1–5). According to the number of good blocks M and mt , we can get the number of blocks for migration, and preserve it to C (lines 6–12).

Meanwhile, we can do reconstruction for lost blocks. Similarly, we first get the longest time consumer for pipelined work (lines 13–17), and then we can get the $rCnt$ which preserves the number of blocks for reconstruction (lines 17–24). Finally, we compute the repair time of the first phase stored in T_1 and return the number of blocks repaired C .

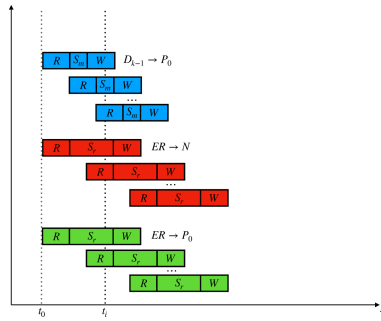


Fig. 6. The pipeline model of AN. (Color figure online)

4.2 An

Feasibility for Parallel: Similarly, Fig. 6 illuminates the reason why we can couple migration and reconstruction in parallel. At t_i , the only difference between BN and AN is that, we should read 2 blocks in every helper, but as mentioned above, we do not address the IO interference, thus we can perform them in parallel.

As the conclusion we make in Sect. 3, the AN algorithm is very easy to implement. Thus, we do not show it in this paper.

This is our answer to the second question: 2) How to fast repair lost blocks after NRP? Similar with BN, we can also couple migration and reconstruction to do the repair in parallel.

In conclusion, SEDRepair consists of two phases: BN and AN, BN is used to offer temporary data durability until the NRP is done, while AN is used to complete the conventional data repair and offer continuous data durability.

5 Performance Evaluation

Experimental Setup: To verify our model, we conduct an number of tests, which focus on the total repair time for data repair and the service time for

requesting one block. All tests in this work are conducted on a workstation with an Intel Core i7 CPU (4 cores) running at 2.2 GHz, 16 GB DDR3 memory, which runs the Ubuntu 18.04 64-bit operating system and the compiler is GCC 7.3.0 which is the default compiler of the OS. Using different compilers and different compiler options may yield slightly different coding throughputs, but will not change the relative relationship among different repair methods, when the same compiler and compiler options are used across them.

In our tests, we remove all the actual operations of disk I/Os and network transmission from the prototype, and simulate the operations by computing their execution times based on the input network and disk bandwidths. We compare SEDRepair with two approaches:

1. ERT (reconstruction-only), which is the conventional method based on RS codes, ERT only use reconstruction operations;
2. ER (reconstruction-only, with durability), which is based on ERT, but ER preserves recovered data in the parity nodes in BN. That is, ER offers data durability.

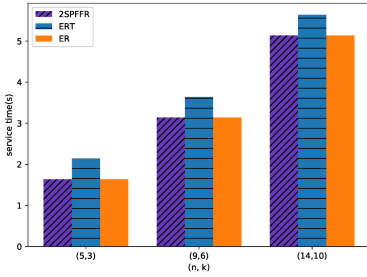
We encode the blocks by RS(5, 3), RS(9, 6) (adopted by QFS [13]) and RS(14, 10) (adopted by Facebook [11]). Our implementation is based on encoding and decoding APIs from Jerasure library 2.0 [15].

We assume the following default configurations. We set the disk bandwidth as 100 MB/s and network bandwidth as 1 Gb/s. We configure both the block size and the packet size as 64 MB. The number of blocks in every node is fixed as 1000 blocks ($U = 1000$) in each experimental run for consistent test. We compare SEDRepair with ER and ERT. We plot the total repair time over ten runs.

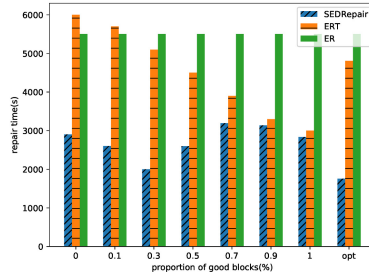
Experiment A.1 (comparison of service time): First we consider the service time per request. As app users can directly get data from memory instead of disk, the service time only consists of read time (R in Fig. 5) and transmission time (S_m and S_r in Fig. 5). As shown in Fig. 7a, the service time of SEDRepair and ER is shorter than ERT, because ERT store recovered data in the memory of the failed node in BN, which needs k blocks from k helpers. But SEDRepair and ER first store data in the memory of the parity node in BN, which only need $k - 1$ blocks from helpers (as the parity node has a parity block for reconstruction), thus the average service time of SEDRepair and ER is 1.64s in RS(5, 3), 3.14s in RS(9, 6), and 5.14s in RS(14, 10).

We next consider five possible factors for total repair time: 1) the proportion of lost blocks, 2) the parameters of n and k , 3) the first phase time T_1 , 4) the block size, 5) the packet size.

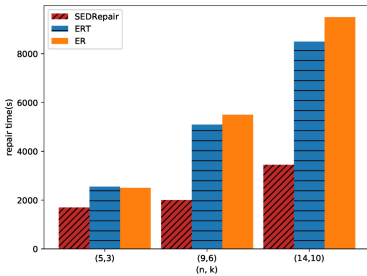
Experiment A.2 (impact of lost block): Figure 7b shows the simulation results of the total repair time in different methods, in which we vary the proportion of good blocks in the failed node from 0% to 100%, among which ER is the worst, since its performance is bottlenecked by the network consumption and I/O of the failed node. ERT is better than ER in most cases, but it can not offer data durability in BN. Overall, SEDRepair reduces the repair time of both ER and ERT, for example, by 68.0% and 63.3% when $P = \text{opt}$, which is computed by Eq. (5) and Eq. (6).



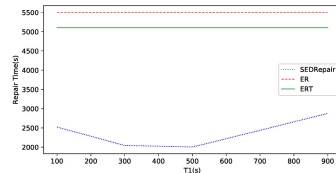
(a) The service time comparison under different (n, k) .



(b) The repair time comparison under different amount of good blocks.



(c) The repair time comparison under different (n, k) .

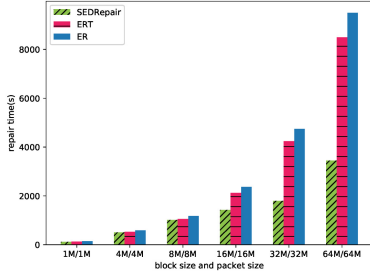


(d) The repair time comparison under different T_1 .

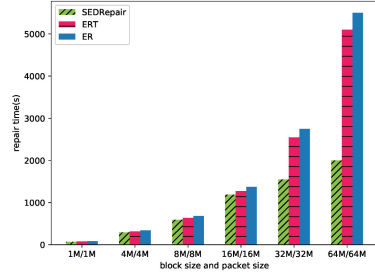
Fig. 7. The comparison under different service time, good blocks, (n, k) and T_1 .

Experiment A.2 (impact of erasure coding): We now evaluate the total repair time for different RS(n, k). Here, we focus on RS(5, 3), RS(9, 6), and RS(14, 10). We assume that $T_1 = 500$ s, the proportion of good blocks is set to 30% (i.e., $P = 300, U = 1000$). Figure 7c shows the results. The repair time of ERT and ER increases significantly in RS(9, 6) and RS(14, 10), compared to RS(5, 3), as it increases the amount of repair traffic. Overall, SEDRepair reduces the repair time of ERT and ER by 60.7% and 63.6% in RS(9, 6), 59.4% and 63.7% in RS(14, 10), and 32.1% and 33.5% in RS(5, 3), respectively.

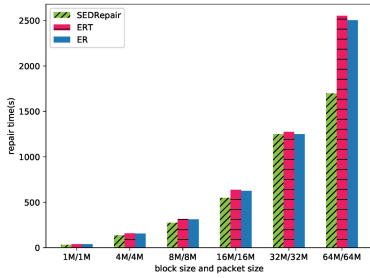
Experiment A.3 (impact of T_1): As mentioned earlier, it may take a long time for NRP, thus we evaluate the impacts of different T_1 . We keep good blocks 70% and select RS(9, 6). We range T_1 from 100 s to 900 s. As shown in Fig. 7d, the performance of ER and ERT remains unaffected by different T_1 , since no matter how long it takes before the new node available, when the requests come, they can only employ reconstruction to repair data, and fill all blanks of the new node by decoding. The blue line shows the impact of T_1 for SEDRepair, where there is a minimum value near 500 s. Our analysis shows that different T_1 makes different amount of migration and computation operations, and produce different proportion of data blocks in AN. According to Eq. (5) to Eq. (6), the



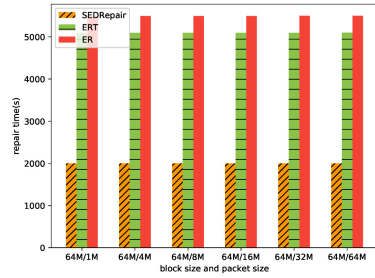
(a) The repair time under different block size and packet size in RS(14, 10).



(b) The repair time under different block size and packet size in RS(9, 6).



(c) The repair time under different block size and packet size in RS(5, 3).



(d) The repair time in different packet size in RS(9, 6).

Fig. 8. The repair time in different packet size.

closer $\frac{M_1}{C_1}$ is to $\frac{t_m}{t_r}$, the shorter the repair time is. Thus, it is consistent with our analysis.

Experiment A.4 (impact of block size and packet size): Figure 8 shows the repair time in the impact of different block size and packet size. We found that when the block size and packet size is small (i.e., from 1M to 8M), the difference among them is not significantly obvious in the performance of repairing data. However, if the block size $> 8M$, our method performs much more greater than ER and ERT, not only because of coupling migration and reconstruction, but also because the network becomes dominant. Meanwhile, we found that, as shown in Fig. 8d, the packet size has little influence on our model.

6 Related Work

We focus on the continuous data durability throughout the whole data repair. The main design of our work is mainly based on FastPR ($G = 1$) [20], which couples migration and reconstruction operations in parallel. But FastPR is

proactive because it conducts migration before the failure occurs. From [4] we can see, some parts of the failed node can be accessible, which gives us an opportunity to do data migration on the failed node. Therefore, different from FastPR, SEDRepair is reactive. Besides, unlike FastPR, we do not need extra storage (called hot-standby nodes in FastPR) for saving data.

CAU [19] is a update scheme which focus on mitigating the rack-across update traffic, and it performs interim replication, which creates a short-lived replication to maintain high data reliability. The idea of interim replication also helps us design SEDRepair.

7 Conclusion

To solve the problem of continuous data durability before node replacement process (NRP), we propose Sector Error-Oriented Durability-Aware Fast Repair (SEDRepair), which carefully couples migration and reconstruction in parallel. To verify our model, we have conducted series of experimental studies on its performance to identify various impacts of different facts (such as block size, packet size, erasure coding). The results of tests show that we can save repair time by over 60% in most cases while maintaining fast service without extra storage. We believe our method also works in a real environment, thus we plan to migrate our model to a local cluster and Amazon EC2. Besides, the multi-failure case is also our future consideration.

Acknowledgment. We thank the anonymous reviewers for their insightful feedback. We also appreciate Jingwei Li, Zhirong Shen and Hu Xiong for their sincere help.

References

1. Blaum, M., Brady, J., Bruck, J., Menon, J.: Evenodd: an efficient scheme for tolerating double disk failures in raid architectures. *IEEE Trans. Comput.* **44**(2), 192–202 (1995)
2. Blömer, J., Kalfane, M., Karp, R., Karpinski, M., Luby, M., Zuckerman, D.: An XOR-based erasure-resilient coding scheme (1995)
3. Chan, J.C., Ding, Q., Lee, P.P., Chan, H.H.: Parity logging with reserved space: towards efficient updates and recovery in erasure-coded clustered storage. In: 12th {USENIX} Conference on File and Storage Technologies ({FAST} 2014), pp. 163–176 (2014)
4. Emami, T.K.: Partial disk failures and improved storage resiliency, November 2011
5. Ford, D., et al.: Availability in globally distributed storage systems (2010)
6. Huang, C., Li, J., Chen, M.: On optimizing XOR-based codes for fault-tolerant storage applications. In: 2007 IEEE Information Theory Workshop, pp. 218–223. IEEE (2007)
7. Huang, C., et al.: Erasure coding in windows azure storage, p. 2 (2012)
8. Huang, C., Xu, L.: STAR: an efficient coding scheme for correcting triple storage node failures. *IEEE Trans. Comput.* **57**, 889–901 (2008)

9. Huang, P., et al.: Gray failure: the achilles' heel of cloud-scale systems. In: Proceedings of the 16th Workshop on Hot Topics in Operating Systems, pp. 150–155 (2017)
10. Khan, O., Burns, R., Plank, J.S., Pierce, W., Huang, C.: Rethinking erasure codes for cloud file systems: minimizing I/O for recovery and degraded reads, p. 20 (2012)
11. Muralidhar, S., et al.: F4: Facebook's warm {BLOB} storage system. In: 11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 2014), pp. 383–398 (2014)
12. Nachiappan, R., Javadi, B., Calheiros, R.N., Matawie, K.M.: Cloud storage reliability for big data applications: a state of the art survey. *J. Netw. Comput. Appl.* **97**, 35–47 (2017)
13. Ovsiannikov, M., Rus, S., Reeves, D., Sutter, P., Rao, S., Kelly, J.: The quantcast file system. *Proc. VLDB Endow.* **6**(11), 1092–1101 (2013)
14. Plank, J.S.: The raid-6 liberation code. *Int. J. High Perform. Comput. Appl.* **23**(3), 242–251 (2009)
15. Plank, J.S., Simmerman, S., Schuman, C.D.: Jerasure: a library in C/C++ facilitating erasure coding for storage applications-version 1.2. University of Tennessee, Technical report, CS-08-627, 23 (2008)
16. Plank, J.S., Xu, L.: Optimizing cauchy reed-solomon codes for fault-tolerant network storage applications. In: Fifth IEEE International Symposium on Network Computing and Applications (NCA 2006), pp. 173–180. IEEE (2006)
17. Reed, I.S., Solomon, G.: Polynomial codes over certain finite fields. *J. Soc. Ind. Appl. Math.* **8**(2), 300–304 (1960)
18. Shen, J., Zhang, K., Gu, J., Zhou, Y., Wang, X.: Efficient scheduling for multi-block updates in erasure coding based storage systems. *IEEE Trans. Comput.* **67**(4), 573–581 (2017)
19. Shen, Z., Lee, P.P.C.: Cross-rack-aware updates in erasure-coded data centers, p. 80 (2018)
20. Shen, Z., Li, X., Lee, P.P.C.: Fast predictive repair in erasure-coded storage, pp. 556–567 (2019)
21. Vajha, M., et al.: Clay codes: Moulding MDS codes to yield an MSR code. In: 16th USENIX Conference on File and Storage Technologies (FAST 2018), Oakland, CA, pp. 139–154. USENIX Association, February 2018
22. Wicker, S.B., Bhargava, V.K.: Reed-Solomon Codes and Their Applications. Wiley, Hoboken (1999)
23. Zhou, T., Tian, C.: Fast erasure coding for data storage: a comprehensive study of the acceleration techniques. *ACM Trans. Storage (TOS)* **16**(1), 1–24 (2020)