SelectiveEC: Selective Reconstruction in Erasure-coded Storage Systems

Liangliang Xu, Min Lyu, Qiliang Li, Lingjiang Xie, Yinlong Xu
University of Science and Technology of China
Anhui Province Key Laboratory of High Performance Computing, USTC

Abstract

Erasure coding has been a commonly used approach to provide high reliability with low storage cost. But the skewed load in a recovery batch severely slows down the failure recovery process in storage systems. To this end, we propose a balanced scheduling module, SelectiveEC, which schedules reconstruction tasks out of order by dynamically selecting some stripes to be reconstructed into a batch and selecting source nodes and replacement nodes for each reconstruction task. So it achieves balanced network recovery traffic, computing resources and disk I/Os against single node failure in erasure-coded storage systems. Compared with conventional random reconstruction, SelectiveEC increases the parallelism of recovery process up to 106% and averagely bigger than 97% in our simulation. Therefore, SelectiveEC not only speeds up recovery process, but also reduces the interference of failure recovery on the front-end applications.

1 Introduction

A distributed storage system (DSS) consists of many devices to provide massive storage capacity, such as GFS [6], HDFS [16], Ceph [19] and Azure [4]. The large amount of commercial devices used in large-scale DSSes are prone to failures. To keep data availability in case of failures, a usual approach is to provide data redundancy in the form of replication or erasure coding (EC). Compared with replication, EC can achieve the same fault tolerance with much less redundancy [18], and therefore is widely used for better storage efficiency. Nevertheless, implementing EC is less efficient in the reconstruction of lost data because of decoding and network transmission. Nowadays, the storage capacity of nodes in DSSes becomes larger and larger and "fat node" comes in, e.g. Pangu [17] is comprised of more than 10K nodes and up to 72 TBs (about 1.5M chunks) per node. By investigating failure traces, single node failure dominates failure cases, which accounts for over 90% of all failure events in real-world deployments [5]. So how to reconstruct the large volume of lost data in case of single node failure with minimal interference to foreground traffic in a relatively short time is very meaningful and pretty challenging.

In common DSSes, network is 1Gb or 10Gb Ethernet [17], which is evidently less than aggregated disk bandwidth in a node. Still they must reserve sufficient network bandwidth to provide good online data services, such as MapReduce and data query. So they usually configure limited network bandwidth for lost data reconstruction, e.g. 30MB/s, 90MB/s or 150MB/s in Pangu [17]. The reconstruction of lost data with EC needs connect multiple source nodes, which makes the network transmission commonly becomes the bottleneck for failure recovery. In addition, DSSes execute the reconstruction in batches for a long reconstruction queue due to limited available system resources. It commonly induces imbalanced recovery traffic in a batch with limited number of stripes. Moreover, random selections of source and replacement nodes further aggravate imbalanced upstream and downstream traffic. Though the deployment of high-speed network, such as Infiniband, will speed up the recovery process [8, 11], it induces much higher cost. Even so, the imbalanced workload on different nodes for failure recovery still slows down the recovery process.

In this paper, we will analyze the failure recovery process and emphasize that imbalanced workload on different nodes is an important factor contributing to the time cost of failure recovery. We propose a graph model, based on which, the degree of recovery parallelism (DRP) is defined and used to reflect the load balance in the reconstruction tasks of a batch. Based on DRP, we propose a balanced scheduling module, SelectiveEC, to improve the load balance for single failure recovery with two optimization techniques. One is dynamically selecting some stripes to be reconstructed into a batch and the other is elaborately selecting source nodes and replacement nodes, to balance upstream and downstream network usage of nodes, moreover, also the disk I/O, CPU and memory usage. A balanced recovery tasks schedule of SelectiveEC not only speeds up recovery process, but also reduces interference of recovery tasks with the front-end requests. We simulate SelectiveEC and compare with random reconstruction. SelectiveEC increases DRP up to 106% for tasks in the first batch, and averagely, normalized DRP bigger than 0.97.

2 Background and Motivation

Reed Solomon (RS) codes [13] are the most popular erasure codes widely deployed in real DSSes [4,6,16,19]. A (k,m)-RS code encodes k data blocks into m additional *parity* blocks. All the $k+m$ data/parity blocks form a *stripe* of *size* $k+m$. A (k,m)-RS code tolerates any m blocks being lost in a stripe, satisfying the so-called *maximum distance separable* (MDS) property [12]. DSSes commonly adopt random data placement to guarantee the load balance of storage among all nodes with sufficient stripes. Erasure coding saves storage space while keeps the same level of fault tolerance compared with replication. However, it also introduces some problems in DSSes, especially in failure recovery. In the following, we present three main problems based on our measurements.

2.1 Problem 1: Network becomes the bottleneck in reconstruction

The typical process of reconstructing a lost data chunk is as follows. Take a (k,m)-RS code as an example, the DSS first chooses a replacement node to execute the pending reconstruction task, then reads k chunks from any k out of $k+m-1$ source nodes, reconstructs the lost chunk by decoding, and at last writes the decoded chunk into its local secondary storage. So the time cost of reconstructing a chunk consists of four parts, reading time from k source nodes T_1, network transmission time T_2, decoding time T_3 and writing time to the replacement node T_4. Let the chunk size be B, the read bandwidth of a source node be $B^s_{I/O}$, the write bandwidth of the replacement node be $B^r_{I/O}$ and the bandwidth of network be B_w. The reconstruction time will be $T = T_1 + T_2 + T_3 + T_4 = max(\frac{B}{B^s_{I/O}}) + \frac{kB}{B_w} + T_{decode} + \frac{B}{B^r_{I/O}}$. It is found that T_2 is amplified k times because the replacement node should receive k data chunks from source nodes to decode a lost chunk. We monitor the time in a local storage

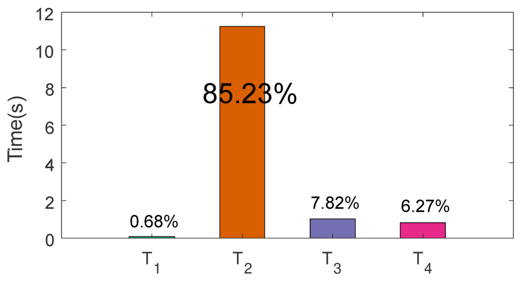


Figure 1: Reconstructing a $(3,2)$-RS chunk in HDFS with 1Gbps Ethernet. Adopting zero-copy strategy in T_1.

system consisting of 28 nodes with HDFS 3, each of which is configured with quad-core 3.4 GHz Intel Core i5-7500 CPU, 8GB RAM, 1T HDD, set 128MB default chunk size and crash a node with 1000 chunks. It is worthwhile to point out that providing services to front-end applications has higher priority than failure recovery. Considering the interference to foreground traffic, it usually sets a limited network bandwidth for EC reconstruction in DSSes. As far as we know, the bandwidth configured for data reconstruction in DSSes usually does not exceed 1Gbps, e.g. 30MB/s, 90MB/s or 150MB/s in Pangu, default Pangu-slow with 30MB/s [17]. So, in the experiments, we use 1Gbps Ethernet to simulate the network bandwidth for EC reconstruction.

Fig. 1 shows the average results of 1000 reconstruction tasks, which shows that chunk transmission time contributes the most part of reconstructing time, 85.23% of the total time cost, even with very small setting of $k=3$. Along with the increase of k, the proportion of T_2 in the total reconstructing time will increase and become more severe bottleneck in recovery.

2.2 Problem 2: How many stripes in a batch are reasonable

Due to the limited system source and the requirement of fast response to the front-end requests, a DSS realizes the data reconstruction of a failed node in batches of data chunks. By default in HDFS [2], the number of failed chunks reconstructed in a batch equals to twice the number of live DateNodes. Though random distribution of data chunks achieves almost uniform distribution among all nodes, it is hard to reach load balance within a batch with limited number of chunks. We propose a bipartite graph, G_{s-r}, to model the load of all nodes for lost data recovery. The vertices of G_{s-r} are divided into two groups, replacement nodes and chunk nodes. Each vertex in G_{s-r} corresponds to a role of live nodes in the DSS. Each live node can be a chunk node of a reconstruction task to provide an available chunk and also be a replacement node of another reconstruction task. When reconstructing a lost chunk in a replacement node needs to read a data chunk from a chunk node, there is an edge connecting the corresponding vertices in G_{s-r}, referring to Fig. 2 as an example.

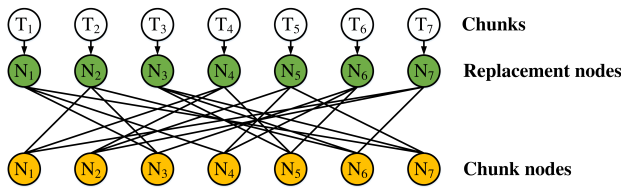

T_1 T_2 T_3 T_4 T_5 T_6 T_7
Chunks
N_1 N_2 N_3 N_4 N_5 N_6 N_7
Replacement nodes
N_1 N_2 N_3 N_4 N_5 N_6 N_7
Chunk nodes

Figure 2: The optimal DRP based on G_{s-r} for a $(3,2)$-RS coded DSS.

Suppose that the upstream and downstream bandwidth of all nodes are the same, denoted as B_w. We first define the *size* *of a recovery timeslot* as the time cost of transmitting k data chunks for reconstructing a lost chunk from source nodes to a replacement node. Let the chunk size be B with a (k,m)-RS code. Then the size of recovery timeslot is $\frac{kB}{B_w}$, e.g., $\frac{3B}{B_w}$ in Fig. 2. We next define *the degree of recovery parallelism (abbr. as DRP)* as the number of failed chunks processed in a recovery timeslot. We find that, if the degrees of all vertices in G_{s-r} are the same, i.e., k, and the network bandwidth of each link connecting a source node and a replacement node is no smaller than $\frac{B_w}{k}$, each live node in the DSS can reconstruct a lost data chunk in a timeslot. So the DSS reaches the complete load balance for failure recovery in the timeslot. In this case, it is equivalent to finding k perfect matchings in G_{s-r}. So if there are N live nodes in a DSS and there exists a k-regular bipartite graph in the corresponding G_{s-r}, we can reach the maximum DRP, N.

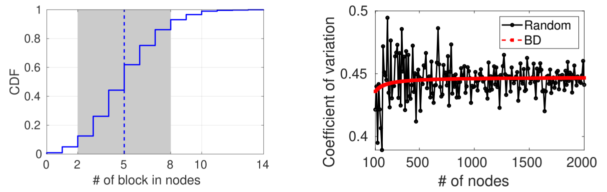
But for a large-scale DSS, although it is easy to find a replacement node for each reconstruction task, i.e. a perfect matching between reconstruction tasks and replacement vertices, it is still difficult to find k source nodes for each reconstruction task and distribute the upstream traffic, which is a k-regular subgraph of the G_{s-r}. Table. 1 shows that it can not even find a k-regular subgraph in a DSS scaled up to 19 nodes. It implies that the maximum DRP cannot be achieved if we assume that each live node reconstructs a data chunk in a timeslot (i.e., in a batch) and the DRP in production is indeed only half of the maximum (see Fig. 7 and Fig. 8). We can significantly increase the DRP with our balanced scheduling module SelectiveEC.

# of Nodes	7	10	13	16	19
Hall	85%	100%	100%	100%	100%
f-factor	96%	43%	13%	3%	0%

Table 1: Probability of perfect match for replacement and k-regular links with source nodes for $(3,2)$-RS. We check perfect match for replacement nodes by Hall theorem [1] and k-regular links for source nodes by f-factor theorem [1]. Each case runs 100 times.

2.3 Problem 3: Nonuniform data layout in a batch

In order to keep uniform distribution of data chunks in a DSS, random data layout [7,16,20] is commonly adopted. From the analysis above, reconstructing N failed chunks can not reach load balance in a batch. In Fig. 3(a), we write 2000 $(3,2)$-RS coded stripes into a DSS with 2000 nodes and count the number of data chunks on each node. Fig. 3(a) shows that more than 80% nodes holds $2-8$ chunks, and the maximum number of chunks on a node is 14, $2.8\times$ of the average. We are to analyze the degree of imbalanced random data layout. The binomial distribution (denoted by *BD*) of n events with the



(a) Data distribution (b) Coefficient of variation

Figure 3: Simulation of random data layout based on $(3,2)$-RS.

same probability p [21] is multiple independent experiments. In our scene, the probability of there being a chunk on a node when writing a data stripe nicely satisfies a event of BD, where the parameters n and p correspond to N stripes in a batch and $(k+m)/N$. The coefficient of variation (denoted by *CV*) [22] is a general variate to measure uniform of data sets with different scale. In Fig. 3(b), we use CV to compare the random data layout and the simulation of BD, and find that BD approximates well to random data layout and becomes more closer to random data layout with larger scale of DSSes. The CV of BD can be formulated as $\frac{\sqrt{Np(1-p)}}{Np} = \sqrt{\frac{1-\frac{k+m}{N}}{k+m}} \approx \sqrt{\frac{1}{k+m}}$, which is a constant, independent to the configurations of DSSes. As an example, the CV of BD for $(3,2)$-RS is $\sqrt{1/5} \approx 0.4472$, which will induce serious load imbalance for failure recovery, for a sample with CV greater than 0.15 is considered as seriously skewed in probability theory and statistics. However, since there are usually lots of chunks to be reconstructed in the pending queue in a node with large storage capacity, we can adjust the order of reconstruction stripes to get more balanced load for data reconstruction in a batch.

3 SelectiveEC

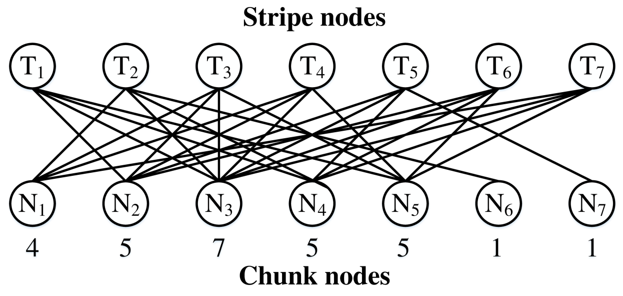
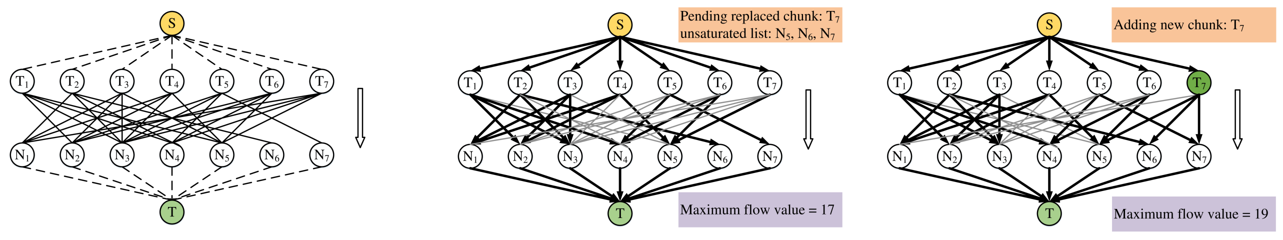


Figure 4: Bipartite graph for selection of source nodes in a $(3,2)$-RS coded DSS. Each task has $k+m-1=4$ connections with chunk nodes. The numbers below each chunk nodes are the number of source chunks in each stripe node.

SelectiveEC realizes balanced scheduling of reconstruction



(a) Flow graph of G_s (b) Found maximum flow (c) Updating maximum flow

Figure 5: Scheduling chunks for source nodes. For (a), the capacity of dotted lines is k and that of each solid line is 1. For (b), the flow value of black lines is 1, which means tasks will choose the corresponding chunk nodes as source. For (c), new T_7 satisfies whose number of chunk nodes are more than flow value 1.

tasks in two steps. It dynamically selects reconstruction tasks into a batch to achieve high parallelism and load balance among source nodes at the first step, and at the second step, selects the replacement nodes to achieve load balance among replacement nodes. It balances network traffic across nodes during the failure recovery, meanwhile nearly balances the memory usage, CPU usage and disk I/Os. So it achieves better trade off between recovery performance and interference on front-end requests than the commonly used recovery process in DSSes.

3.1 Scheduling stripes in batch and selecting source nodes

The random data distribution achieves uniform distribution of data chunks among all nodes with huge number of stripes, but it results in serious load imbalance in a batch with limited number of successive stripes, which is the commonly used scheduling of data chunks reconstruction in DSSes. Thanks to the very long pending queue of stripes to be reconstructed, we can adjust the reconstruction order of stripes, i.e., dynamically select some stripes to be reconstructed into a batch to realize load balance in a batch.

We start from constructing a bipartite graph G_s for the selection of source nodes. Assume there are N live nodes in a DSS. We take the N live nodes as a group of vertices in G_s, named as *chunk nodes*, and take N stripes, each of which having a lost chunk to be reconstructed, as another group of vertices, named as *stripe nodes*. If there is a chunk of stripe T_i in a live node N_j, there is an edge (T_i, N_j) in G_s. Refer to Fig. 4 as an example of G_s in a $(3,2)$-RS coded DSS, which shows nonuniform distribution of chunks among nodes in a batch of stripes.

To reconstruct a chunk, we need to read k chunks in the same stripe, each from a different chunk node. So for each stripe node, we should choose k out of $k+m-1$ edges in G_s incident to it to reconstruct the lost chunk. The optimal state is that all live nodes are *saturated*, which means that each stripe node connects to k chunk nodes and vise versa. In case of all nodes being saturated, we achieves load balance in a batch of chunks to be reconstructed. So it becomes to find a k-regular subgraph in a large bipartite graph G_s with a large amount of stripes, where there is a lost chunk in each stripe. But as a matter of fact, it is NP-hard to deal with it for a large-scale DSS [3].

We use maximum flow to select a batch of lost chunks achieving the saturated or nearly saturated state to maximize the load balance within a batch. We construct a flow graph FG_s based on G_s, by adding a source vertex s and connecting it to all stripe nodes, adding a sink vertex t and connecting it to all chunk nodes. We assign the capacity of each edge connecting a chunk node and t to be k, the capacity of each edge connecting a stripe node and s be k, and the capacity of other edges connecting a stripe node and a chunk node to be 1. The flow corresponding to the optimal state is called a saturated flow and its value is Nk. Given a flow f of FG_s, a stripe node T_i (chunk node N_j) is called saturated if the edge (s, T_i) ((N_j, t)) is saturated, i.e. k edges incident to T_i (N_j) are with flow value 1. Refer to Fig. 5(a), the constructed flow graph from Fig. 4.

With Ford-Fulkerson algorithm, we get a maximum flow f_{FG_s} of FG_s. If f_{FG_s} is saturated, we obtain a k-regular subgraph of G_s by choosing the edges with value 1 in f_{FG_s}. So we get a batch of chunks to be constructed achieving load balance of disk read among all live nodes. Otherwise, some nodes are unsaturated. We then update the flow graph FG_s by replacing the stripe node with the minimum incident edges with flow value 1 by a stripe from the remaining pending reconstruction tasks queue, to get a flow with a bigger value. In particular, the updating first finds the stripe node with the minimum connections with chunk nodes whose flow values are 1 in f_{FG_s} and deletes it (the responding chunk reconstruction will be postponed to a coming batch), then lists the unsaturated chunk nodes, and finally finds a new task in the queue with more unsaturated chunk nodes than the unsaturated task and replaces it. By repeating the steps above, we can obtain a bigger flow which is a saturated flow or a nearly saturated flow. Fig. 5(b) shows a maximum flow of Fig. 5(a), a stripe to be replaced is T_7 and the unsaturated nodes are N_5, N_6 and N_7. Fig. 5(c) finds a new stripe with chunk nodes N_1, N_5, N_6, N_7 and replaces T_7, where we get an updated flow with a bigger value.

3.2 Selecting replacement nodes

The selection of chunk nodes above achieves balanced upstream network bandwidth of nodes, while the downstream bandwidth depends on the selection of replacement nodes.

We also start from constructing a bipartite graph G_r for replacement nodes. To keep reliability in DSSes, no more than two chunks of the same stripe are stored in a node, so each lost chunk should be reconstructed in a node different from its chunk nodes. Thus, G_r in fact is the complement of G_s. To maximize the degree of recovery parallelism, it is optimal if each live node executes a reconstruction task in a batch. That is, it needs to find a perfect match in G_r. Since there are lots of nodes in DSSes, the bipartite graph G_r is usually dense and thus a perfect matching is easily obtained. In fact, we find that Hall theorem holds for almost all cases in practice. So a perfect matching for replacement nodes is common.

3.3 Analysis on the DRP

After scheduling stripes and selecting chunks and replacement nodes, we get a bipartite graph related to chunks and replacement nodes G_{r-s}, a graph model representing the relation of pending reconstruction tasks and live nodes. For example, Fig. 6 is G_{r-s} for Fig. 5(c). Based on G_{r-s}, it is easy to show the improvement of SelectiveEC on recovery performance.

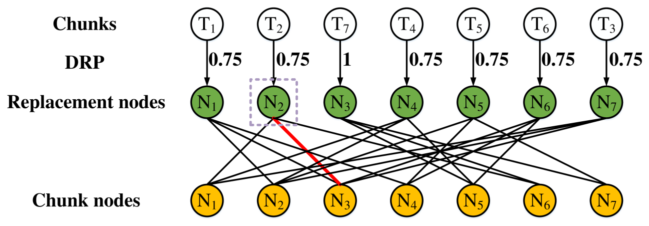


Figure 6: An example of G_{r-s} based on $(3,2)$-RS. The red edge is the slowest in all three connections of N_2.

We compute the DRP by summing up the completion ratio of each reconstruction task in the recovery timeslot. Note that the real network bandwidth of each task depends on the slowest connection in all of its chunk nodes and replacement nodes. Taking T_2 in Fig. 6 as an example, the red edge is slowest because N_3 acts as a chunk node for 4 tasks, so DRP of T_2 is $\frac{B_w}{4} * \frac{3B}{B_w} / B = 0.75$. Similarly we can compute DRPs of other tasks and finally get the value of DRP is 5.5.

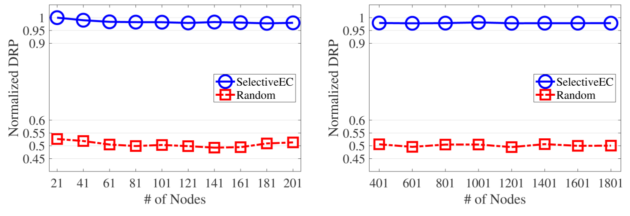
4 Performance Evaluation

4.1 Methodology

We implemented a simulative prototype of SeletiveEC. The simulations run in a server with two 12-core Intel Xeon E5-2650 processors, 64GB DDR4 memory, and Linux 3.10.0. We conduct our evaluation using the $(3,2)$-RS codes in different node scales. In order to simulate a single "fat node" failure, we set the number of chunks on the failed node as 100 times of the number of surviving nodes. We evaluate SeletiveEC with the normalized DRP of the first batch and all batches, which is defined by the ratio of DRP and the optimal DRP N. The updating of SeletiveEC is finding new chunks to be reconstructed in the total queue. We compared SelectiveEC with random reconstruction, a default recovery scheduling implemented in HDFS [2], which selects N successive stripes from pending reconstruction queue in each batch and randomly selecting chunk and replacement nodes.

4.2 Results

The first batch. We evaluated the normalized DRPs of the first batch in small-scale DSSes presented in Fig. 7(a), where the number of nodes varies from 21 to 201, and large-scale DSSes presented in Fig. 7(b), where the number of nodes varies from 401 to 1801. For small scale, we find the normalized DRPs of SelectiveEC are all bigger than 0.975, even reach 1 in some cases, while those of random reconstruction are around 0.5. SelectiveEC improves the DRP up to 106%. The results of large scale are similar to that of small scale, with the improvement up to 97.6%. SelectiveEC almost achieves the optimal result for maximum flow (with value kN) about the first batch of tasks due to traversing the total reconstruction queue in simulation, moreover a perfect matching almost exists for choosing replacement nodes. So normalized DRPs for the first batch of SelectiveEC are steady near 1. As for random reconstruction, we have proved that the non-uniformity of random data layout is determined by the parameters k,m of RS codes in § 2.3 and independent of the number of nodes in DSSes. Because DRP depends on the selection of source and replacement nodes, which relies on the data layout, the DRPs of random reconstruction are almost steady. Because the selection of source and replacement nodes hardly achieves balance within a limited number of stripes, the DRPs stay low for random reconstruction in our experiments.



(a) Regular case (b) Big case

Figure 7: The DRP of first batch in recovery.

Full batches. Fig. 8 shows the CDF of the normalized DRPs in each batch with the number of nodes varies from 101 to

1001, which is around 0.97 for SelectiveEC, while it is only around 0.50 for random reconstruction. We find that there are a few batches with the normalized DRPs below 0.90 and these batches account for only 5% to 10% and occur in the last few batches. This is mainly because there are only fewer stripes remaining in the reconstruction queue to be chosen when updating the maximum flow. But even so, it is still bigger than 0.80.

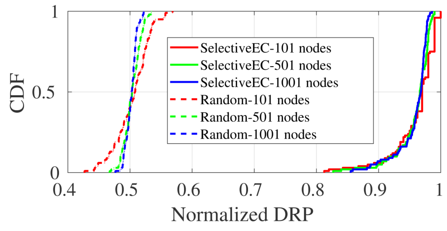


Figure 8: The DRP of full batches in recovery.

5 Related work

Dayu [17] is a greedy scheduling algorithm for recovery of lost data with replicas, while the reconstruction of a chunk in a (k,m)-code needs to retrieve k blocks selectively from $k+m-1$ chunk nodes. S3 [9,10] is an online algorithm taking into account deadline of reconstruction tasks. SelectiveEC can be extended by better maintenance of the reconstructed queue considering the deadline of reconstruction task in future. CAR [14, 15] balances the amount of cross-rack repair traffic about source nodes across multiple racks by a greedy algorithm, while SelectiveEC better balances traffic from source and replacement nodes. SelectiveEC is based on maximum flow to optimize traffic, which can be extended with weight at nodes and on edges to deal with the common straggler problem in heterogeneous environments.

6 Conclusion

This paper proposes SelectiveEC, a balanced scheduling module, for single failure recovery by selecting pending reconstruction tasks in batch, targetedly choosing source nodes and replacement nodes. Compared with random reconstruction, SelectiveEC improves the degree of recovery parallelism up to 106% and averagely bigger than 97%. Moreover, SelectiveEC is orthogonal to the existing works of speeding up failure recovery in EC storage systems. So it can be commonly deployed in erasured-code storage systems.

Acknowledgments

We thank the anonymous reviewers and Ramnatthan Alagappan (our shepherd) for their feedback. This work was supported by the National Key R&D Program of China under Grant No. 2018YFB1003204, and National Nature Science Foundation of China under Grant No. 61832011 and 61772486.

7 Discussion

This paper tries to design a key module SelectiveEC to achieve balanced scheduling for reconstruction tasks in erasure-coded storage systems. Although we have proposed the modeling and the scheduling algorithm, there are still a lot of works to be done to efficiently deploy SelectiveEC in practical systems.

I **Updating speed of maximum flows.** On the one hand, we update the value of the maximum flow for scheduling stripes and source nodes in batch by finding a task with more unsaturated source nodes, but the increment of the value of the maximum flow may be small, which induces a long delay of the scheduling algorithm. So how to find a new indicator towards optimal update for selecting a new task with maximum increment of the value of the maximum flow? On the other hand, every updating initializes a network flow on each edge as zero, which also induces a long delay of the scheduling algorithm. How to optimize the updating speed of maximum flows?

II **Trade-offs of SelectiveEC.** SelectiveEC updates a maximum flow by scanning the reconstruction queue whose length is hundreds times of the number of surviving nodes, which induces a long delay of scheduling algorithm. To efficiently implement SelectiveEC in DSSes, we have to handle such a trade off between the executing time of the scheduling algorithm and the completion time for reconstructing a batch, especially for large heterogeneous DSSes with stragglers. So we need to design an efficient heuristic algorithm to find the near optimal maximum flow.

III **Multiple failures model.** SelectiveEC also supports multiple failures recovery. When concurrent failures occur in a DSS, a primary node is dedicated to execute the recovery task of each stripe, and transfers the blocks reconstructed by it to other backup nodes. Thus, in case of concurrent failures, distributing the reconstructed blocks from primary nodes to backup nodes will also be an important part of recovery process, which should be balanced.

We will next carefully analyze the complexity of the scheduling algorithm, dynamically adjust SelectiveEC by adding important parameters in face of diversified DSSes and generalize SelectiveEC to support multiple failures model in algorithms and implementation. Until now, our program still has plenty of room to be improved. We expect to share our experiences with more researchers and look forward to constructive suggestions to optimize our module.

References

[1] Jin Akiyama and Mikio Kano. *Factors and factorizations of graphs: Proof techniques in factor theory*, volume 2031. Springer, 2011.

[2] Apache. HDFS. https://hadoop.apache.org/docs/r3.1.3/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html, 2019.

[3] J. A. Bondy and Usr Murty. Graph theory (graduate texts in mathematics). 2008.

[4] Brad Calder, Ju Wang, Aaron Ogus, Niranjan Nilakantan, Arild Skjolsvold, Sam McKelvie, Yikang Xu, Shashwat Srivastav, Jiesheng Wu, Huseyin Simitci, et al. Windows azure storage: a highly available cloud storage service with strong consistency. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 143–157, 2011.

[5] Daniel Ford, François Labelle, Florentina Popovici, Murray Stokely, Van-Anh Truong, Luiz Barroso, Carrie Grimes, and Sean Quinlan. Availability in globally distributed storage systems. 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*, pages 29–43, 2003.

[7] RJ Honicky and Ethan L Miller. Replication under scalable hashing: A family of algorithms for scalable decentralized data distribution. In *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.*, page 96. IEEE, 2004.

[8] Nusrat S Islam, Mohammad Wahidur Rahman, Jithin Jose, Raghunath Rajachandrasekar, Hao Wang, Hari Subramoni, Chet Murthy, and Dhabaleswar K Panda. High performance rdma-based design of hdfs over infiniband. In *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–12. IEEE, 2012.

[9] Shijing Li, Tian Lan, Moo-Ryong Ra, and Rajesh Krishna Panta. S3: joint scheduling and source selection for background traffic in erasure-coded storage. In *37th IEEE International Conference on Distributed Computing Systems, ICDCS 2017, Atlanta, GA, USA, June 5-8, 2017*, pages 2025–2030, 2017.

[10] Shijing Li, Tian Lan, Moo-Ryong Ra, and Rajesh Krishna Panta. Joint scheduling and source selection for background traffic in erasure-coded storage. *IEEE Trans. Parallel Distrib. Syst.*, 29(12):2826–2837, 2018.

[11] Jiuxing Liu, Jiesheng Wu, and Dhabaleswar K Panda. High performance rdma-based mpi implementation over infiniband. *International Journal of Parallel Programming*, 32(3):167–198, 2004.

[12] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The theory of error-correcting codes*, volume 16. Elsevier, 1977.

[13] 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.

[14] Zhirong Shen, Patrick PC Lee, Jiwu Shu, and Wenzhong Guo. Cross-rack-aware single failure recovery for clustered file systems. *IEEE Transactions on Dependable and Secure Computing*, 2017.

[15] Zhirong Shen, Jiwu Shu, and Patrick PC Lee. Reconsidering single failure recovery in clustered file systems. In *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 323–334. IEEE, 2016.

[16] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*, pages 1–10. Ieee, 2010.

[17] Zhufan Wang, Guangyan Zhang, Yang Wang, Qinglin Yang, and Jiaji Zhu. Dayu: fast and low-interference data recovery in very-large storage systems. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 993–1008, 2019.

[18] Hakim Weatherspoon and John D Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *International Workshop on Peer-to-Peer Systems*, pages 328–337. Springer, 2002.

[19] 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*, pages 307–320, 2006.

[20] Sage A Weil, Scott A Brandt, Ethan L Miller, and Carlos Maltzahn. Crush: Controlled, scalable, decentralized placement of replicated data. In *SC'06: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, pages 31–31. IEEE, 2006.

[21] Wikipedia. Binomial distribution. https://en.wikipedia.org/wiki/Binomial_distribution, 2020.

[22] Wikipedia. Coefficient of variation. https://en.wikipedia.org/wiki/Coefficient_of_variation, 2020.