Optimization of recovery bandwidth utilization in erasure code storage systems

Zhezhe Xing*
Department of Software Engineering, Jilin University, Changchun, Jilin, China

ABSTRACT

In the erasure code storage systems, failure is very common, so a great amount of data needs to be recovered in the system. For each recovery task, we need to select some nodes from the nodes of a stripe to read data to recover the failed data. However, due to the limitation of the bandwidth of each node, the number of simultaneous recovery tasks is limited, and the inappropriate task selection causes the waste of bandwidth, resulting in slow recovery. This paper proposes two greedy approaches Gre-N and Gre-T to solve the above problems. Gre-N assigns tasks to nodes with high bandwidth first. Gre-T sorts tasks in weighted order. The node with the lowest bandwidth utilization is selected each time to avoid the bandwidth exhaustion of a large number of nodes. In addition, the bandwidth utilization of the system is balanced by the low bandwidth node compensation policy. According to the experimental results, compared with the original approach, Gre-N and Gre-T increase the bandwidth utilization by up to 20.2% and 29.7%.

Keywords: Erasure code, cloud system, failure recovery, greedy, bandwidth utilization

1. INTRODUCTION

As cloud storage systems become more and more widely used, it is a good choice to use erasure codes with low space overhead for data fault tolerance. However, erasure codes involve multiple storage nodes, they significantly increase network traffic for failure recovery.

Due to the frequent processing of massive data, failure recovery becomes an unavoidable problem for the erasure code storage system. When there are a large number of tasks that need to be recovered, we need to take full advantage of parallelism to recover as many tasks as possible with limited bandwidth. The blocks of an erasure code stripe are distributed in independent nodes. When recovering data, some blocks need to be selected. When there are a large number of tasks, bandwidth utilization will be very low if our selection causes some nodes to take on too many tasks, low bandwidth utilization is meaning that low recovery speed. The actual situation is that the bandwidth of the nodes in the system is unbalanced. Instead of simply distributing tasks randomly to all nodes in the system, the bandwidth of the nodes should be considered to achieve maximum bandwidth utilization.

Some linear programming methods and graph algorithms can accurately solve this problem, but the time complexity is too high, so it is not suitable for scenes requiring high efficiency. In the existing distributed storage systems, most of them use the method of random extraction to randomly extract enough blocks from the surviving blocks of a strip for recovery, but due to the uneven bandwidth of nodes, this method has not achieved good results. based on the above problems, this paper proposes two greedy methods, Gre-N and Gre-T. Gre-N firstly sorts nodes according to their bandwidth size and selects k blocks with the highest bandwidth among nodes required for each task to participate in recovery, so as to make full use of system bandwidth. Gre-T is further optimizing to solve the problem of bandwidth imbalance. The main technical points are as follows:

- For the recovery tasks in the wait queue, the order is weighted by bandwidth.
- For each recovery task, we select the lowest bandwidth utilization nodes within the nodes of the task, the bandwidth of all nodes is not fully occupied.
- Dynamically monitor the utilization of nodes with high bandwidth. If the utilization reaches a certain level, we select tasks with lower weighted order in the task queue to improve the utilization of nodes with low bandwidth.

* xingzz19@mails.jlu.edu.cn

Third International Conference on Computer Science and Communication Technology (ICCSCT 2022)
edited by Yingfa Lu, Changbo Cheng, Proc. of SPIE Vol. 12506, 1250657
 · 0277-786X · doi: 10.1117/12.2662530

According to our results, compared with the original approach, the Gre-N and Gre-T improve the total bandwidth utilization by up to 20.2% and 29.7%.

2. RELATED WORKS

2.1 Erasure code

Erasure code has lower storage overhead than replication, which are widely used in cloud systems[1-3], there is various kind of erasure codes, RS code[4] is a traditional code and widely deployed in the storage system, it is because it can tolerate any m blocks are lost, m is the number of parity blocks of a stripe there. After the Google disclosed Google File System[5], which used triple replication with the diffusion of commodity disks. Since the data generated by multiple services started to grow explosively, the erasure code's space efficiency was revalued again.

2.2 Failure recovery

In recent years, recovery overhead is important, it has been regarded as a serious problem in storage, especially in large storage systems[6, 7]. Many methods aim at failure recovery in Erasure code, including local parity techniques to decrease the recovery overhead. WEAVER Codes[8] suggested a generic method that contains almost all of the possible placements. Followed by Azure's Local Reconstruction Codes[6] and Facebook Xorbas[7], Microsoft Pyramid Codes[9] studies the durability of local parities.

2.3 Greedy algorithm

The greedy algorithm is widely studied and applied in different fields because of its simple design, easy implementation, and low complexity compared with other methods. Reference[10] uses powerful sampling methods to apply greedy algorithms to the MapReduce paradigm, including maximum cover and submodular maximization subject to p-system constraint problems. Based on the greedy strategy, Reference[11] proposed a new method to obtain good space to solve a parameter family partial differential equation. Reference[12] proposed an improved the greedy algorithm, Carousel Greedy, aiming at the defects of the greedy algorithm, and applied it to a variety of combinational optimization problems.

3. THE GREEDY APPROACHES GRE-N AND GRE-T

3.1 Problem definition

Here we define the problem to be solved in the form of mathematical symbols. The node-set in the system is $N = \{N_1, N_2, \ldots, N_n\}$, the corresponding bandwidth of each node is $B = \{B_{N_1}, B_{N_2}, \ldots, B_{N_n}\}$, where B_{N_i} is measured in the size of one data block. We define the set of tasks awaiting recovery as $R = \{R_1, R_2, \ldots, R_l\}$, l represents the number of tasks, corresponding to the task R_i, the optional node-set to participate in the recovery of the task is $S_i = \{S_{i_1}, S_{i_2}, \ldots, S_{i_p}\}, S_{i_j} \in N$, where p represents the number of nodes and is the same for all tasks, the actual number of nodes required by the task is defined as $q(q < p)$. Then the problem is defined as, on the premise that the bandwidth used for each node $N_i(N_i \in N)$ does not exceed $B_{N_i}(B_{N_i} \in B)$, $t(t \leq l)$ tasks are selected from the set R, and each task selects a fixed number of nodes from the optional set S_i. And our goal is to maximize t, that is, to process as many recovery tasks as possible with limited bandwidth.

3.2 Gre-N

Gre-N method considering the characteristics of the node bandwidth imbalance, priority is assigned to nodes with high bandwidth, we first sort the nodes by bandwidth from high to low, and first assign tasks to the nodes with high bandwidth. For each node, if there are enough tasks involving the node, select the same number of tasks as the node bandwidth; otherwise, select all tasks until there are no qualified tasks to end the allocation process.

As shown in Figure 1, circles represent the nodes used to perform tasks, and they are arranged in descending order of bandwidth. Squares represent tasks, and the gray grid on the right of each square represents the list of nodes needed to perform the task. Gre-N's policy is to assign tasks to nodes in descending order of bandwidth. As shown in Figure 1, since the nodes involved in task 0 and task 3 have the highest bandwidth, they are preferentially processed, so task 1 and task 4 are subsequently processed.

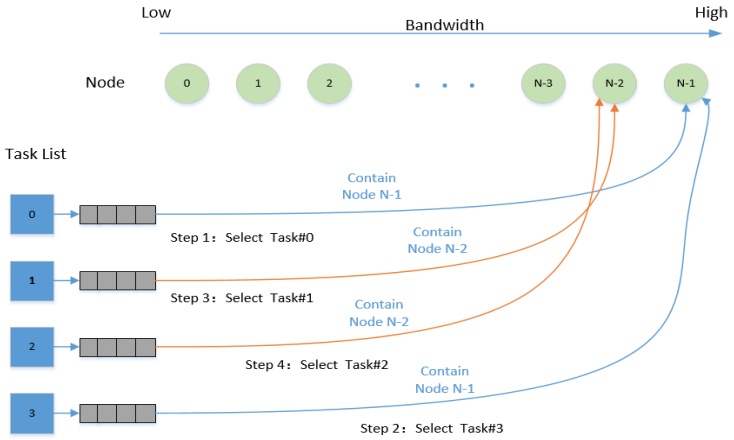

Figure 1. The strategy of Gre-N.

3.3 Gre-T

Compared with the random allocation method, Gre-N method takes into account the problem of bandwidth imbalance. However, since the bandwidth of the top nodes is always inevitably occupied, it is difficult to consider the subsequent better schemes and tasks. Therefore, we propose a better strategy based on the greedy algorithm: Gre-T. Gre-T contains three technical points, which we will introduce separately.

3.3.1 Tasks Sorting. In the process of selecting tasks to be processed, how to select task combinations is very important. Choosing the right task combination may improve the bandwidth utilization of nodes, so we take this into account in Gre-T. Gre-T computes a bandwidth score for each task R_i as the sum of the initial bandwidth of the nodes involved in the task. The calculation formula is as follows:

$$F_{R_i} = \sum_{j=1}^{w} e_{h_j}, \forall h_j \in S_i, \forall e_{h_j} \in B \tag{1}$$

where S_i represents the node-set that the processing task R_i needs to participate in, w represents the number of elements in the corresponding node-set, all elements in S_i belong to the set N, e_{h_j} represents the initial bandwidth of node h_j.

Then, the tasks are sorted in ascending order according to the score, and the next tasks are selected in this order. This is similar to Gre-N's Greedy strategy, in which the tasks are preferentially allocated to nodes with high bandwidth for processing. However, Gre-T adopts a more flexible policy to balance the bandwidth utilization of each node, rather than exhausting the bandwidth of a node.

3.3.2 Assign Tasks Based on Bandwidth Utilization. We calculate the used bandwidth of all nodes in the system in real-time, and then divide the used bandwidth by the initial bandwidth to get the bandwidth utilization of the current node. For each selected task R_i, we first select p nodes corresponding to the task according to S_i, and then select q ($q<p$) nodes with the lowest bandwidth utilization as the final nodes to participate in the task, after the current task is complete, the bandwidth utilization of the q nodes is updated in real-time. As shown in Figure 2, we first calculate the score of tasks according to equation (1), and then sort the tasks in order of high score to a low score, and process the tasks in this order. As shown in Figure 2, for task 0, among the four nodes (N-1, N-2, N-3, N-4) that can participate in the task, the two nodes (N-1, N-2) with low bandwidth utilization are selected to participate in the task. At the end of task 0, we update the bandwidth utilization of N-1 and N-2, and then repeat the above operations for task 1 and task 2. We avoid using a greedy strategy based on the remaining available bandwidth (RAGS) to select nodes. This is because the task sequencing method in Section 3.3.1 is based on the initial bandwidth of the corresponding node, we take the initial bandwidth order to represent the bandwidth order at any time. If RAGS is used, the ranking of the available bandwidth of the node will change greatly after completing part of the task, at this moment need the corresponding adjustment in order to adapt to the current bandwidth scheduling state, because there are a large number of the task, sort of delay cost is very big, the Gre-T according to the node is assigned to a task can guarantee the bandwidth of the node order bandwidth utilization is always consistent with the initial basic bandwidth order. In this way, task reordering is not required.

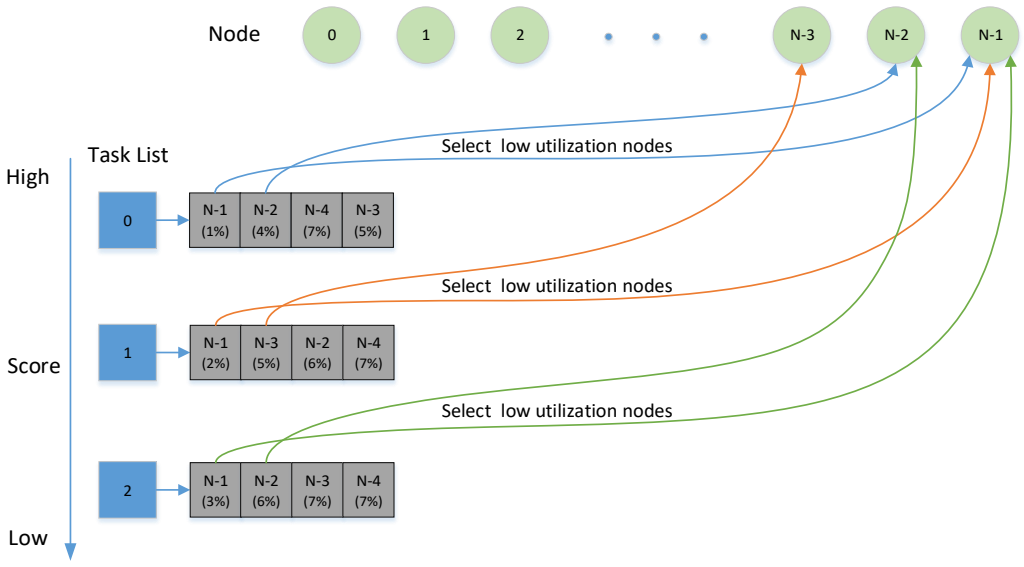

Figure 2. The strategy of Gre-T.

3.3.3 Low Bandwidth Node Compensation Policy. Since tasks are sorted according to the sum of initial bandwidth of nodes involved in tasks, nodes with low initial bandwidth will not be frequently used in the whole process. The Gre-T method uses low bandwidth node compensation policy (LBC) to alleviate the problem of the greedy policy over selecting nodes with high initial bandwidth. In Gre-T, we define a constant *f*. When the node number is smaller than *f*, it is a low-bandwidth node; when the node number is larger than *f*, it is a high-bandwidth node. As shown in Figure 3, we use an on/off logic valve to control whether to trigger the LBC, specifically: LBC is disabled by default. When the bandwidth usage of high-bandwidth nodes is low, LBC is disabled. In this case, the task selection sequence is front to back. When the bandwidth usage of high-bandwidth nodes is too high, LBC is triggered to select tasks from the end of the task sequence. The default LBC trigger conditions are as follows:

$$C = \sum_{i=f}^{n} D_{i_e} - \sum_{i=0}^{f} D_{i_u}, \forall D_i \in N, f < n \tag{2}$$

where D_i represents the node, and N is the set of nodes arranged in descending order of bandwidth, D_{i_e} represents the remaining bandwidth of node D_i, D_{i_u} represents the used bandwidth of node D_i, n represents the total number of nodes. In the beginning, obviously C is greater than 0, LBC is not triggered, and LBC is triggered when c is less than or equal to 0. This indicates that LBC is triggered when the sum of the bandwidth used by all high-bandwidth nodes is greater than or equal to the sum of the unused bandwidth of all low-bandwidth nodes.

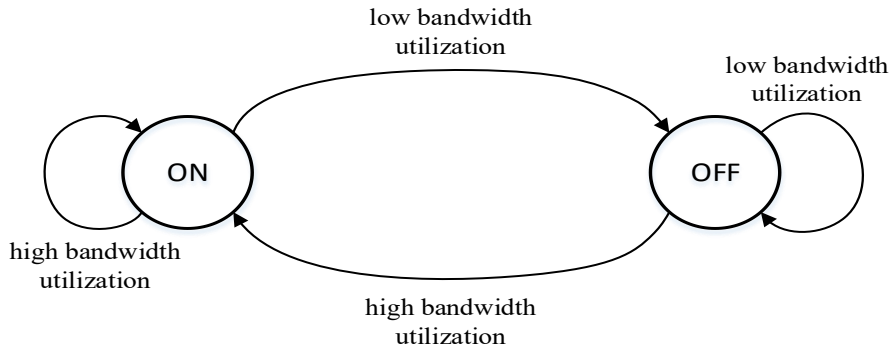

Figure 3. Description of the conversion graph under bandwidth utilization control.

4. EVALUATION

4.1 Test method

We use C++ language to simulate the implementation of our proposed two methods, Gre-N and Gre-T, in a Linux environment. We simulate 1000 nodes in the implementation and use a random function to randomly generate bandwidth for these 1000 nodes. During the experiment, our erasure code strip was composed of nine data blocks and three check blocks. Therefore, for each recovery task, we randomly selected 11 different nodes as the optional node-set, and compared our proposed method with the existing original method to test the effect of our proposed method.

The LBC policy is used when the valve is open and suspended when the valve is closed.

4.2 Results

First, for a total of one million tasks, we tested the total bandwidth utilization of the original versus Gre-N and Gre-T. The duration of executing a recovery task reflects the advantages and disadvantages of a recovery policy. If the total number of recovery tasks is constant, the higher the bandwidth usage, the faster the task execution speed and the shorter the task execution time, indicating that the better the recovery policy is. According to Figure 4, when the total number of tasks is 1 million, the total bandwidth utilization of the original method is 74%, Gre-N is 89%, which is 20.2% higher than that of the original method, Gre-T is 96%, which is 29.7% higher. When the total number of tasks is 3 million, the bandwidth utilization of the original method is 77%, 87% for Gre-N, and 94% for Gre-T, Gre-N improves by 13.0% and Gre-T by 22.1% compared with the original method, which proves that our method is effective under the different total number of tasks.

We did a further experiment with a total of one million tasks to compare Gre-N and Gre-T. After multiple rounds of task selection, all tasks are executed, and the bandwidth utilization after each round is calculated, as shown in Figure 5. As can be seen from the figure, Gre-T has higher bandwidth utilization than Gre-N in most of the iterations, and Gre-T needs fewer rounds of iteration (about 2200) to complete all tasks than Gren (about 2500).

Figure 4. Performance of the three approaches in the case of tasks of 1 million and 3 million.

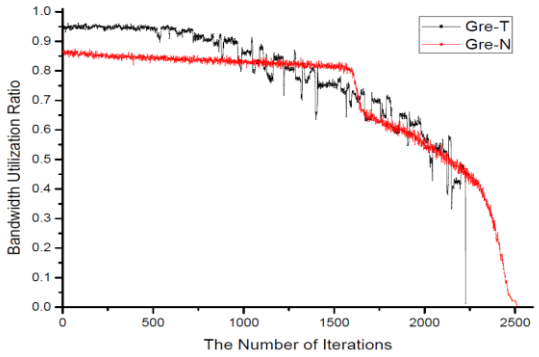

Figure 5. The bandwidth utilization ratio curve of Gre-N and Gre-T.

5. CONCLUSION

Erasure code is widely used in Cloud systems, and recovery is an important part, as for accelerating the speed of recovery, the bandwidth utilization overall system should be as high as good. In order to improve the bandwidth utilization of recovery, we designed and implemented two greedy approaches Gre-N and Gre-T, Gre-N assigns the tasks that invole high bandwidth nodes, and Gre-T makes a further optimization by ordering tasks and selecting the lowest bandwidth utilization nodes, in addition, Gre-T utilizes a dynamical strategy to balance the utilization between low bandwidth nodes and high bandwidth nodes. By comparing with the original approach, the Gre-N and Gre-T improve the total bandwidth utilization by up to 20.2% and 29.7%.

ACKNOWLEDGMENTS

This research is supported by the National Natural Science Foundation of China (62077027), the Ministry of Science and Technology of the People's Republic of China (2018YFC2002500).

REFERENCES

[1] Nachiappan, R., Javadi, B., Calheiros, R. N. and Matawie, K. M., "Adaptive bandwidth-efficient recovery techniques in erasure-coded cloud storage," Lecture Notes in Computer Science, 11014(2018).
[2] Li, X. L., Li, R. H., Hu, Y. C., et al., "Toward unified and configurable erasure coding management in distributed storage systems," 17th USENIX Conference on File and Storage Technologies (FAST 19), 331-344(2019).
[3] Hu, Y. C., Cheng, L. F. and Yao, Q. R., "Exploiting combined locality for wide-stripe erasure coding in distributed storage," 19th USENIX Conference on File and Storage Technologies (FAST 21), 233-248(2021).
[4] Wicker, S. B. and Bhargava, V. K., [*Reed-Solomon Codes and Their Applications],* John W, Sons, (1999).
[5] Passing, J., The Google file system and its application in MapReduce, (2012). Available at: http://int3.de/res/GfsMapReduce/GfsMapReducePaper.pdf.57
[6] Huang, C., Simitci, H., Xu, Y., et al., "Erasure coding in windows azure storage," *2012 USENIX Annual Technical Conf. (USENIX ATC 12),* 15-26(2012).
[7] Caers, R., De Feyter, T., De Couck, M., et al., "Facebook: A literature review," *New Media & Society,* 15(6), 982-1002(2013).
[8] Hafner, J. L., "WEAVER Codes: Highly fault tolerant erasure codes for storage systems," *Fast 05 Conf. on File & Storage Technologies,* 5, 16(2005).
[9] Huang, C., Chen, M. and Li, J., "Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems," *ACM Transactions on Storage,* 9(1), 1-28(2013).
[10] Kumar, R., Moseley, B., Vassilvitskii, S., et al., "Fast greedy algorithms in mapreduce and streaming," *ACM Transactions on Parallel Computing,* 2(3), 1-22(2015).
[11] DeVore, R., Petrova, G. and Wojtaszczyk, P., "Greedy algorithms for reduced bases in Banach spaces," *Constructive Approximation,* 37(3), 455-466(2013).
[12] Cerrone, C., Cerulli, R. and Golden, B., "Carousel greedy: A generalized greedy algorithm with applications in optimization," *Computers & Operations Research,* 85, 97-112(2017).