A New Collaborative Scheduling Mechanism Based on Grading Mapping for Resource Balance in Distributed Object Cloud Storage System

Yu Lu[1], Ningjiang Chen[1,2(✉)], Wenjuan Pu[1], and Ruifeng Wang[1]

[1] School of Computer and Electronic Information, Guangxi University, Nanning 530004, China
chnjgxu@edu.cn

[2] Guangxi Key Laboratory of Multimedia Communications and Network Technology, Nanning 530004, China

Abstract. An algorithmic mapping of storage locations brings high storage efficiency to the storage system, but the loss of efficient scheduling makes systems prone to crashing at low usage. This paper uses the Ceph storage system as a research sample to analyze these issues and proposes a grading mapping adaptive storage resource collaborative optimization mechanism. This approach grading both the storage device and the storage content, and introduced random factors and influence factors as two-factors to quantify the grading mapping relationship between the two of them. This relation coordinates the storage systems' performance and reliability. In addition, a collaborative storage algorithm is proposed to realize balanced storage efficiency and control data migration. The experimental results show that in comparison with the inherent mechanism in the traditional Ceph system, the proposed cooperative storage adaptation mechanism for data balancing has increased the average system usage by 17% and reduces data migration by 50% compared to the traditional research approach.

Keywords: Cloud storage · Storage balance · Grading mapping · Ceph · Collaborative scheduling

1 Introduction

In the internet-based collaborative computing [1] mode, there are similarities between the operation and management of collaborative applications and the scheduling of nodes in distributed systems [2]. The performance of applications is affected by the construction and management of system infrastructure. In cloud storage, the storage mapping of data is related to the performance and the stability of the storage system. Distributed object storage systems with hash

H. Gao et al. (Eds.): CollaborateCom 2020, LNICST 350, pp. 533–549, 2021.
https://doi.org/10.1007/978-3-030-67540-0_35

algorithm mapping storage locations are widely used, have shown good performance. However, the lack of a central scheduling service makes the mapping of data to storage devices uncontrollable, so for balanced storage of data, collaborative scheduling between devices is required spontaneously.

Ceph [3] is a representative object-based storage system [4] (OBSS), which hasn't a dedicated metadata server [5] (MDS) to record the object-based storage device (OSD) locations of segmented objects. The OSD location uses a specific mapping algorithm CRUSH [6] to determine the locations of the objects and the replicas. When the data are searched or modified again, the mapping process can be done independently on each client. When the device is replaced or a new one is added, it is necessary for the storage system to adaptively calculate the storage location of the object to realize the recovery and balance, so easily scaling storage requirements for a huge scale of data.

In today's medium and large data centers, this feature reduces the data risks and lowers operating costs when replacing equipment. But the mapping algorithm [7] can't sense the state of the device where the data will be written, and if the device is overloaded, the data writes are denied, at which point the upper-level systems with Ceph as the base storage (such as Openstack [8], which provides block storage [15] primarily for applications such as its virtual machines) will crash.

In view of the above problems, existing researches mainly optimize CRUSH or adopt frequent migration to balance the system [9]. These are either too conservative and have a poor balance effect or are too aggressive and cause more performance problems. In view of these deficiencies, this paper analyzes the key factors of such problems using Ceph as the research platform, which differs from existing researches by using the collaborative work between storage nodes to control the process of mapping and migration of storage data. A graded mapping adaptation mechanism is proposed to make the data relevant to the device. This paper first devises influencing and random factors to capture the relationship between PGs(Placement Group) and OSDs(Object Storage Device), as well as a collaborative algorithm to enable a single OSD to use these factors for collaborative scheduling with other OSDs. The classification of OSD and PG also increases computational efficiency, manages migration direction, and reduces the operational difficulty. At the same time, the stability of the system can be greatly improved through effective migration.

The structure of this paper is as follows. In Sect. 2, we will introduce the Ceph storage process, analyze the problems and the factors that have been missed by existing researches, and then a new mechanism is proposed in Sect. 3. The experiments to verify the proposed mechanism are given in Sect. 4, and Sect. 5 conclude the paper.

2 Problem Analysis and Motivation

In the basic framework of Ceph, the Rados is responsible for the mapping process between the file to the storage device OSD [10]. This process has three basic steps: from file to object, from object to PG, and from PG to OSD. The relationship is shown in Fig. 1.

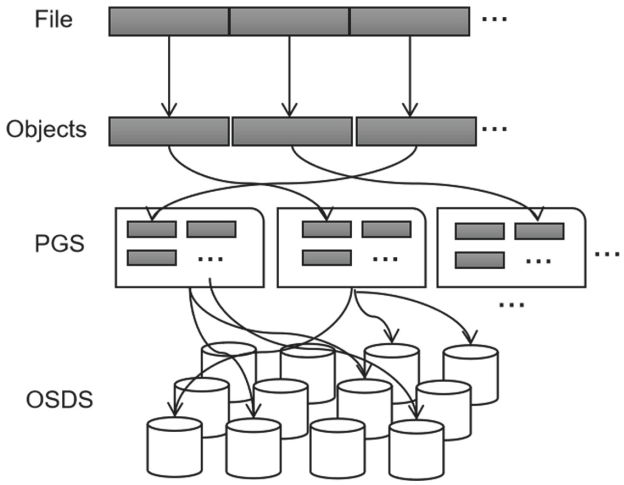


Fig. 1. Ceph storage mapping process

Step 1. $(ino, ono) \rightarrow oid$: This step splits the files. ino is the id of the file, ono is the unique identifier of the file slices, oid is the unique id of the object. Almost all object storage systems and slicing storage systems split files and store them in a similar mechanism.

Step 2. $hash(oid)\&mask \rightarrow pgid$: This step computes the PGs for the object. The object's unique identifier is used as the seed. In Ceph, this identifier is usually composed of the file name and object identifier. Then, a specific hash algorithm is used to generate the random number, and then the number of PGs is used to make the remainder operation. This allows objects to be randomly distributed to individual PGs. The number of PGs is confirmed when the system cluster is initialized. The conventional method is to use the number of redundancy as the reference when the number of PGs is initialized [11], as calculated by Eq. (1):

$$TotalPGs = (Number\_of\_OSD * 100)/replication \tag{1}$$

Object mapping to PG is a completely random process in both research and production environments. Although the remainder operation is approximately evenly distributed, the difference between PGs is actually not as small as imagined as the randomness varies from the average value. We conducted an experimental analysis to explore this problem. This experiment uses the Ceph source code as the object of analysis to calculate the object distribution of 38TB of file data in 6,666 PGs(300 OSD according to Eq. 1 and the number of PGs in the 2 replica [12] mode). The file size is simulated as 1 GB. The OID is composed of a random file name and a numeric number, which is consistent with the system numbering. Figure 2 shows the PG number distribution of the theoretical deviation between the actual number of objects and the number of objects in the PG for the simulation environment, the horizontal coordinate is the difference

between the size of a single PG and the average size of all PGs, and the vertical coordinate is the number of PGs. From the figure, the largest PG and the smallest PG differ by approximately 200 objects (4 MB for a single object). PG is the minimum storage unit considered in the Ceph system. Therefore, neither balancing efficiency nor data migration control can be improved if PG differences are not focused on in the research of balanced storage.

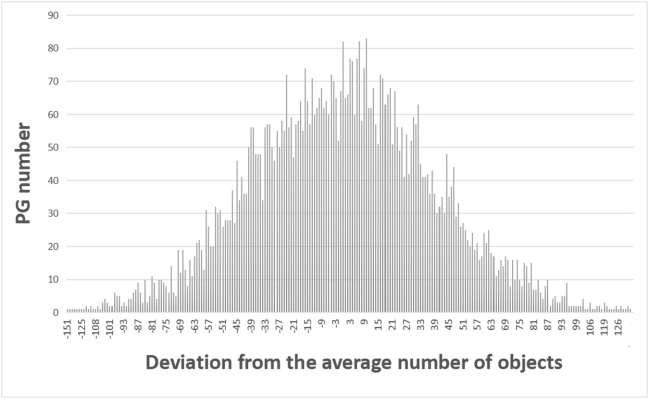


Fig. 2. Object deviation statistics in PG

Step 3. $crush(pgid) \rightarrow (osd1, osd2)$: This step is the OSD selection. It is implemented by using Ceph's default algorithm CRUSH which is a pseudo-random reproducible process. At present, most of researches work about storage balance focus on optimizing this step, including two typical methods, the first type is based on the size of the OSD capacity so that the average distribution of PG on each OSD, the other type is to dynamically adjust the number of PG on the OSD in the system runtime. Both approaches consider the PG to be uniform in size, the difference is that the former is based on the equalization of PG [13] the latter is based on the use of the classical MDS storage system [14] for equalization. However, as can be seen from the second step above, PGs do not have the same size, so these two methods can only perform relatively balanced work to some extent, without further considering the difference of PG. The size of the PG changes dynamically in real-time and is random.

Based on these conclusions, we know that distribution of average PG on OSD does not mean an average distribution of data and that the central scheduling of PG loses the advantage of mapping storage locations by algorithms. However, the mapping and parameter changes lead to data migration, which results in further performance problems and stability risks. Because the PG size difference is not considered, the existing researches are limited in their ability to solve the problem of storage balance; they incur performance costs. Therefore, this paper proposes a collaborative scheduling mechanism based on grading mapping, which gives the PG the ability to migrate autonomously. The balanced tasks can be coordinated

among a number of OSDs, it can also provide accurate data migration solutions for PGs, ensuring that the amount of migrated data is controllable. Secondly, a model is established to evaluate the relationship between the PG and the OSD to avoid the scenarios in which a data migration triggers a cascade of further data migration. The main grading model guides the migration direction and conforms to the changes in the cluster's high performance and reliability for each period of the cluster's overall use. As a result, the proposed approach ensures high performance and high reliability of the system.

3 Collaborative Scheduling Mechanism Based on Grading Mapping

For the problems in Ceph analyzed in the previous section, the situation is that when an OSD node usage exceeds the average usage rate, the balancing mechanism can be triggered by the node itself, and the PG that needs to be migrated is evaluated according to its own size, usage rate, and the average cluster usage rate. Each PG that needs to be migrated can get the feedback from all the new OSD sets and find the best set to migrate, as shown in the figures below.

For the classic Ceph system, there are unresolved issues at each step:

(a) PG does not have the ability to migrate independently, even if a PG is selected, it cannot be migrated.
(b) Even with the ability to move, a random choice without direction may never find a choice that can be migrated.
(c) There is no evaluation standard that reflects whether the OSD combination can accept the PG.

To address the above problems, this paper proposes a novel solution that is introduced below.

3.1 Random Factor and Impact Factor

In the traditional Ceph storage system, the PG does not have the ability to redirect. The Reweight artificial modification mechanism is an external enhancement available in the BlueStore [16]. Therefore, in order to solve the balanced storage problem, the PG redirection capability is first given, so this capability is contained within the PG. This is accomplished by adding a random factor to the parameters. The addition of random factors interferes with the CRUSH selection results, and the changes in the incoming parameters allow the pseudo-random process to have more solutions. The principle of the random factor r^i is as follows:

$$R^i < OSDs >= CRUSH(pgid, r^i) \tag{2}$$

Where pgid is the unique identifier of the PG, r^i is the random factor, i is the number of selections, and R^i is the OSD combination selected by the r^i (the

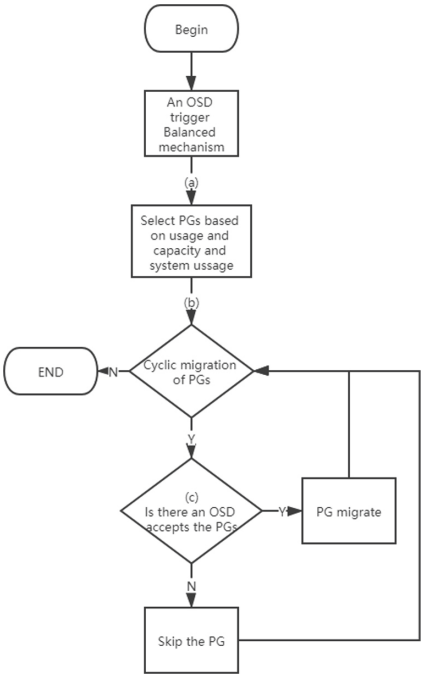


Fig. 3. The process of achieving storage balance in the Ceph system

master node and multiple copy storage nodes). For r^i, it can be randomly generated as a seed according to parameters such as time or memory space according to the underlying implementation of each system. Because of the randomness of CRUSH, the resulting set R^i also changes when r^i changes. The purpose of the balanced storage that this paper focuses on is to make the usage rate of each OSD of the whole system as close as possible to the average usage rate. If the average usage rate of a system is assumed to be M, the index equalization rate of a balanced storage condition of a system is evaluated. β can be calculated as follows:

$$\beta = \frac{\sum_{j=1}^{n}(x_j - M)^2}{n} \tag{3}$$

where x_j is the usage rate of a single node in the cluster. The equalization rate reflects the variance of cluster usage; it is a reflection of the difference in usage of each node in the cluster. For the impact of a PG redirection on the whole system, the calculation of the impact factor can be compared with the β before the PG migration and the β after the PG migration. The system state after the PG migration can also be judged, so the impact factor θ can be defined as follows:

$$\theta = \begin{cases} -1 & x_r > 1 \quad or \quad x_r - M > \alpha \\ \dfrac{\sum_{j=1}^{n}(x_r - M)^2}{\sum_{j=1}^{n}(x_j - M)^2} & else \end{cases} \tag{4}$$

Here r is an OSD in a selection result set R, x_r is the usage rate of a single node in the migrated cluster, α is a set threshold. If the usage rate exceeds the threshold value compared with the average usage rate, the equalization is triggered. If the usage rate of an OSD in a group of OSDs exceeds 1 after the PG migration or the usage rate exceeds the average usage rate, the impact factor of this group is -1. This judgment prevents two undesirable outcomes. The first is the migration making other OSDs unavailable. The second is that the migration will not make the new node unavailable, but it will trigger another migration of the new OSD.

3.2 Grading Mapping Mechanism

The definition of the random factor gives the PG redirection ability. However, if there is no corresponding control and optimization method, when a PG needs to be migrated only the unknown random calculation can be performed and the most suitable combination can be found. This process is uncontrollable, and the calculation of the impact factor can also result in a substantial computational performance cost. Therefore, on the basis of random factors, a novel logical division OSD method is proposed, and the random number of random factors is used as the grading division method to ensure that the migration of PGs has direction and level. The advantage of OSD grading is that each pool can customize the rules for PG migration in and out. Because PG migration determines the amount of data migration, it is also possible to indirectly limit data migration through OSD grading. Further, we propose a grading approach for PGs, which combines PG grading with OSD grading. This approach converts the selection process of objects from PG→OSD to PG→grading→OSD. The grading of PG is matched with the grading of OSD. The OSDs grading is based on the number of selections of the random factors. Therefore, the PG level change must start from the lowest level of the OSD; it can be determined according to the random factor and the impact factor whether it can be migrated to a group of OSDs in this grading pool. If the low-level grading pool does not accept the PG according to the policy, the upgrade selects a higher-level OSD grading pool. The OSD grading pool is shown in Fig. 4.

In this way, when the average cluster usage is low, the PG can perform the equalization with less calculation. When the average cluster usage is high, more choices can be made. High performance is ensured at low system usage and high reliability at high usage rates. According to the above design concepts, the basic process of the OSD and PG dual grading strategy is as follows. The first step is the initialization phase, which consists of two parts: one is to pre-statically set the initial level of the OSD; the other is to initialize the PG level. Compared with the traditional random selection strategy of the CRUSH algorithm, the OSD

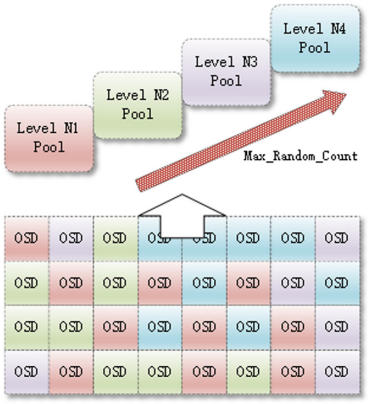


Fig. 4. Grading pool diagram

classification mechanism constrains the selectable range of PGs to one level. Considering the real-time changes of the PG size, it should be distributed in grading pools as evenly as possible in the PG level initialization. Therefore, this paper constructs a uniform hash ring according to the total size of each grading pool. The uniform hash is derived from the consistency hash. The maximum value of the hash ring is an integer, and the length of every hash segment is the same. The number of nodes on the hash ring is the same as the number of pools, and PG determines the grade by taking the hash value by pgid and then taking the surplus in relation to the number of pools. The principle is shown in Fig. 5.

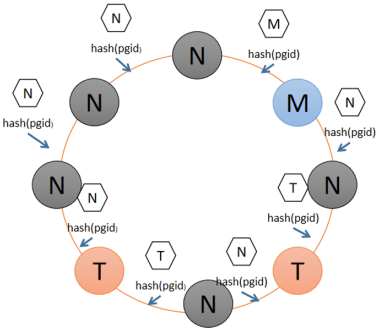


Fig. 5. PG initialization level with hashring

The example shown in the Figures is a grading method of three levels: M, N, and T. The size ratio of each logical pool is M:N:T = 1:5:2, so 8 nodes can be constructed according to the ratio. The entire hash ring is divided into 8 equal segments, 8 nodes are randomly scattered in 8 positions. Then, each PG performs a remainder operation on the ring maximum value according to the

hash value calculated by its pgid. The remaining value falls into a point on the ring. Using a clockwise search, the first node level encountered is the PG level. The initialization algorithm of the PG is as follows Algorithm 1:

Algorithm 1. PG Initialization algorithm

Require: ;
The set of all $OSDs\_Levels\_size$;
The set of all PGs; Integer MAX_RING
integer nodes_count;
integer[level][ratio] level_array;
level_array=Get_the_minimum_ratio($OSDs\_Levels$);
for each $level \in level\_array$ **do**
  nodes_count+=level[ratio];
end for
integer[nodes_count] nodes_array;
for each $level \in level\_array$ **do**
  for $i = 0$ to $level[ratio]$ **do**
    i=randomInt(nodes_count);
    while nodes_array[i] **do**
      i++;
    end while
    nodes_array[i]=level[level]
  end for
end for
initialize a hash ring by levels;
MAX_RING=MAX_RING÷nodes_count;
for each $pg \in all\ PGs$ **do**
  interval=(integer)hash(pgid)÷(MAX_RING)
  pg.level=nodes_array[interval] ;
end for

The second step is the collaborative balancing process, which is mainly composed of a PG selection, a PG redirection calculation, and a PG redirection to complete the migration. An example of the working process of this mechanism is shown in Fig. 6.

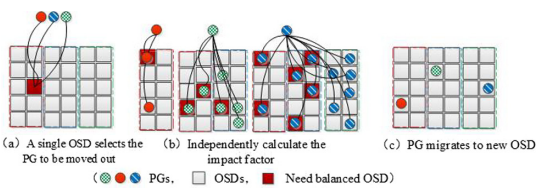


Fig. 6. PGs migration process

In the figure, the classification of the cluster is divided into three levels: red, blue, and green. The maximum random number is set to 2, 4, and 5. In (a), one node usage rate is higher than the average usage α, the mechanism is triggered in the OSD, and n PGs in the OSD are selected according to the mechanism.

(a) Select 3 PGs for migration according to the rules.
(b) Starts with the red PG, because the maximum random number of the first level is 2, so the red PG performs two random calculations to obtain 2 random factors. At the same time, two sets of OSD combinations in the first stage are obtained. The group OSD sends a migration request, assuming a reply of $[-1, a]$ ($a != -1$). A returned value of -1 indicates that if the red PG migrates to the combination, a new node migration is triggered or the node is unavailable. A returned value of non-1 indicates that the combination accepted the migration request and the red PG completed the migration. Similarly, the green PG obtains $[-1, -1]$ in the combination of the two random factors of the first level, and the migration cannot be completed at the first level. As the result, the level rises to the second level. The maximum random number of the second level is 4, so 4 sets of OSD combinations are obtained with 4 random factors. Assuming that the four return values are $[-1, a1, a2, a3]$ ($-1<a1<a2<a3$), according to the calculation formula of the influence factor, we can know that the combined balance degree of a1 is greater than a3, and the migration of the smallest one of the three non-1s is taken. Finally, the blue PG also follows this approach.
(c) Set the new level and the new random factor to the new parameters of the three PGs, and the PG starts to migrate. After the migration, all of the node usage rates are less than the threshold α compared with the average usage rate, and no single point usage rate is prominent; therefore, the balance is achieved.

The algorithm is described as Algorithm 2:

Algorithm 1 is to initialize the parameters, the number of executions of this algorithmic process is only related to the number of PG, so the time complexity is O(n). Algorithm 2 is a balanced storage algorithm, the number of executions of which is a pre-set parameter, and the time complexity of the algorithm is O(n) too.

The mechanism uses two factors to plan and constrain the balancing, avoiding the uncertainty that occurs in collaborative balancing. Based on the grading mechanism, each migration of data is a joint decision of multiple nodes. The management model of a two-way migration strategy avoids the requirement to balance more nodes that are triggered by data migration. Since the balancing mechanism is decided first between nodes and then data is migrated, each migration is an efficient migration, reducing IO performance losses. The key performance of the mechanism lies in the calculation. When the cluster has a low usage rate, the balance can be completed in the low-level grading pool. In this case, the number of operations is small, and the performance is guaranteed.

Algorithm 2. Adaptive grading data balanced mechanism algorithm

```
Require: ;
    OSD;
    Average usage of this level M;
 1: PG[] pgs,i=0;
 2: List *Levels;
 3: balance=abs((M-OSD.ratio))*OSD.size;
 4: allPGs=sort_by_size_asc(OSD.PGs);
 5: while balance>0 do
 6:    pgs.push(allPGs[i]);
 7:    i++;
 8:    balance-=allPGs[i];
 9: end while
10: for each pg ∈ pgs do
11:    map(r,θ) result;
12:    while Levels→hasNext() do
13:       for i = 0 to Levels.max_random_count do
14:          r=Random();
15:          OSDs=CRUSH(pg,r);
16:          for each osd ∈ OSDs do
17:             result.add(r,result.get(r)+θ)
18:          end for
19:       end for
20:       sort_by_θ_asc(result);
21:       repeat
22:          if *result.θ>0 then
23:             pg.level=Level.level;
24:             pg.r=*result.r;
25:          end if
26:       until !*result.hasNext()
27:    end while
28: end for
```

When the cluster has a high usage rate, it needs to consider higher levels. Finding the right combination decreases the performance, but ensures the validity of the calculation results, thus improving the reliability of the system.

4 Experiments and Evaluation

Experiments are conducted to compare the performance of the mechanism proposed in this paper, the Ceph system, and related research [9,12] examples in terms of data balancing and migration control by means of simulation. For the simulation system, this paper uses the CRUSH in the Ceph source code to re-implement a simulation system for writing and recording Ceph data based on the BS architecture. The architecture of the system is shown in Fig. 7.

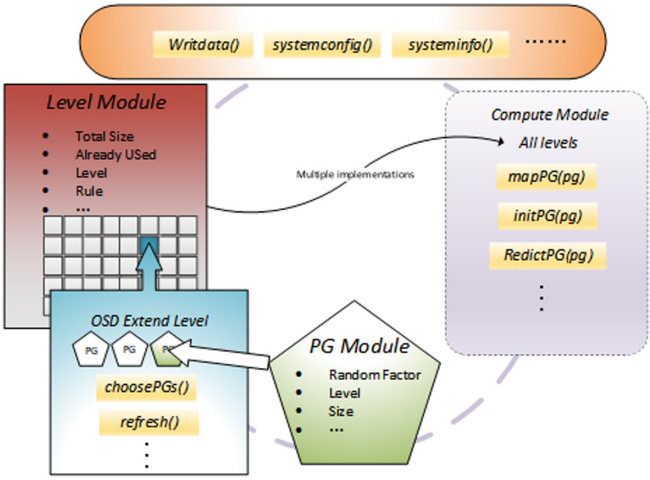


Fig. 7. Architecture of simulation system

The implementation of the simulation system is mainly divided into five modules. The uppermost layer is the interface module, which is used to configure the system, write data, obtain storage information, etc. Level Module is a grading pool class, which defines the capacity, used capacity, level, etc. OSD Module extends from Level Module, so OSD has the parent class's properties and independent PG operation methods. All calculations are performed in a single module that is completely decoupled to ensure the implementation of the algorithm for each control group.

4.1 Evaluation of the Effect of Storage Balance

The experimental comparison object of the balanced storage is mainly the research work of uniform PG and the traditional Ceph storage system. The experiments make the PG uniformly distributed across OSDs to simulate the native Ceph balancing method. Three control simulation environments are configured respectively: Normal, Partially small (single OSD capacity is half of the size), and Partially large (half the OSD size is 2 times the size). The average overall usage when testing the maximum writes R for the system is defined as:

$$R = max\_input/total\_size \tag{5}$$

The variance of the OSD usage is defined as:

$$V = \frac{\sum_{j=1}^{n}(u_j - R)^2}{n} \tag{6}$$

Where u_j is the single node usage rate.

Because it is difficult to use 100% of the algorithm map storage locations, the following approach is used as a reasonable alternative. When the usage rate of a single-point OSD reaches 100%, the system first returns the usage rate.

Experiments will be performed many times with different level distributions, different OSD counts, different OSD sizes, and different per-write volumes. Compared the mechanism proposed in this paper, the classic Ceph, and the simulation of the uniform PG distribution on OSDs(Simulated Ceph open source tool up-map). All experiments are in 2 replicate mode. Each time the system writes 3 times, the amount of testing data (until one of the nodes reaches 100%), the variance, and the usage rate at runtime are recorded. The experimental results are shown in Fig. 8. The vertical coordinates on the left are the variance values for each experimental control group, represented by a histogram. The right vertical coordinate is the system usage rate of each control group in each experiment, and the horizontal coordinate is the number of experiments.

From the figure, we can see that in all of the experiments, the comparison group using the mechanism proposed in this paper has an average utilization rate of more than 15% compared with the other two comparison objects. In some comparison groups, such as Partially small, it can reach 20%. In addition, the experiment using this mechanism maintains a very stable state in usage, and there is not much fluctuation. This is because each writes data is random, and the mechanism of this paper can be used to circumvent this randomness. In comparison, the other two approaches do not. Especially in the Partially small experimental environment, the other two approaches collapse when the overall usage rate is low. This is because the PGs in these approaches are not aware of the OSD's usage. The system using tool up-map is not completely better than the classic system. The reason is that the analyzed distribution of the PG is not equal to the data distribution. In terms of variance, the control group using the proposed strategy is lower than the other two approaches. The storage balance of the system is improved after using the new balancing mechanism.

4.2 Evaluation of Data Migration

In the previous section, we pointed out that although the method of using equalized write frequency has completed the function of equalized storage, it incurred substantial data migration. This causes the loss of system performance and made the systems unreliable. Compared with previous research, the algorithm of this paper focused on the transfer of a balanced object based on a single PG, rather than an entire OSD. In this experiment, this paper uses a quantitative writing method to write a fixed size of data each time [9]. The offline calculation to obtain the data migration will be used in this section. The evaluation compared the amount of data migration that occurred when writing the same amount of data due to Reweight in the classic Ceph system. The Reweight mechanism was not a well-balanced solution, so the amount of data migration was not compared. The results are shown in Fig. 9. The histogram shows the comparison of the migration of data between the 2 mechanisms in equalized storage at each

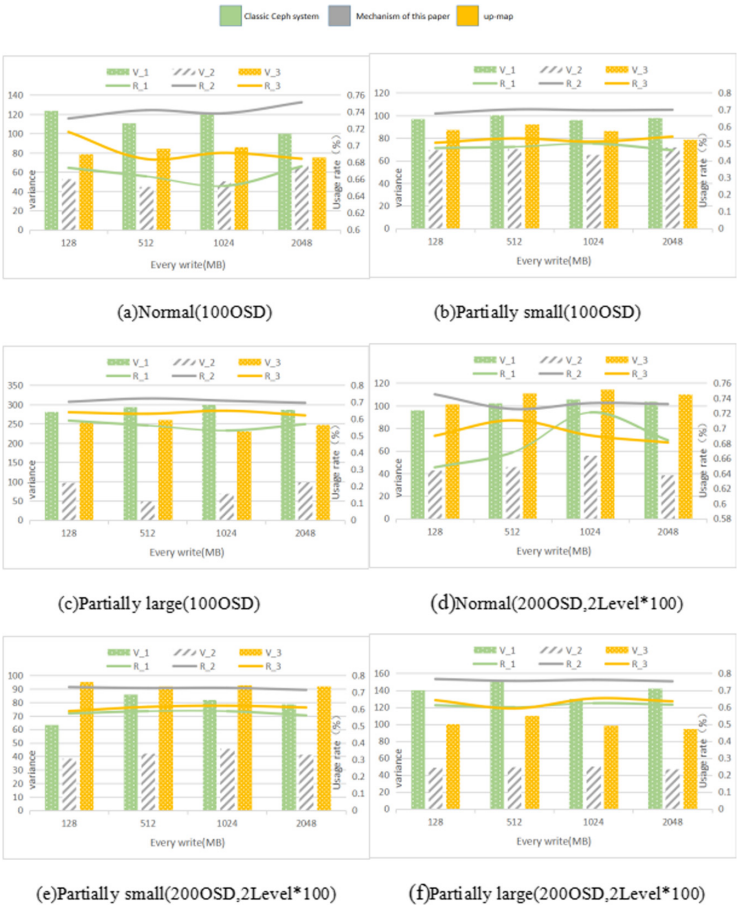
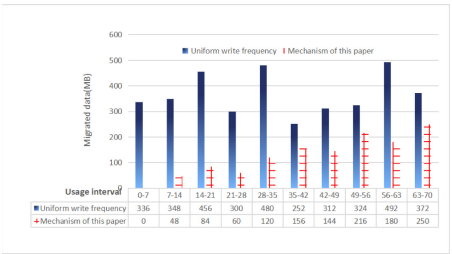


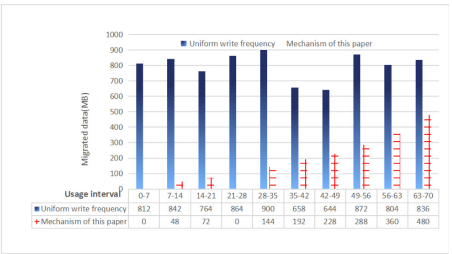
Fig. 8. Balanced storage capacity experiment

interval as usage increases, the table below shows the detailed migration data sizes.

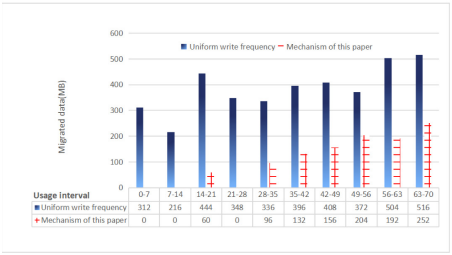
As can be seen from the above figures, using the balanced storage strategy based on the write frequency, when the data are written into the system, no matter whether the system usage rate is high or low, a substantial amount of data migration occurs. The main reason is the differences among the PGs: this method seeks to migrate objects to achieve balanced storage, but the real migration is PGs. Consequently, data migration increases each time the data are written. In contrast, the algorithm strategy of this paper increases with the usage rate, because each node may trigger a new node to be balanced after each writes. However, at low usage rates, it does not intervene in the system to trigger data migration, ensuring equalization. While storing, it also ensures that the performance of the system is not affected. The migration of data increases at



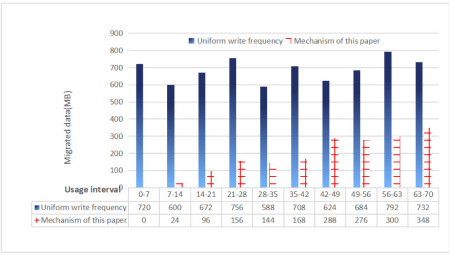
(a) 100OSD(1024M Per Write)



(b) 100OSD(2048MB Per Write)



(c) 200OSD(1024MB Per Write)



(d) 200OSD(2048MB Per Write)

Fig. 9. Comparisons of data migration

high usage rates; the migration amount across levels increases because the initial migration leaves many PGs in a low level. The strategy in this paper migrates 50% fewer data compared to the typical research approach and the reduced data migration ensures the reliability of the system.

4.3 Summary of the Experiments

It can be seen from the above experiments that the mechanism proposed in this paper has good performance in balanced storage, and at the same time, it has obvious advantages in data migration control compared to some typical previous researches. The mode of using collaborative work makes balancing spontaneous and efficient; it also makes the system more reliable. It is suitable for large-scale clusters. This is because the use of fewer OSD clusters means there are fewer OSDs per level after grading, so there are a limited number of different combinations for every PG.

5 Conclusions

In view of the storage imbalance problem in cloud storage systems, in this paper, a collaborative scheduling balanced storage mechanism based on a grading mapping model is proposed with Ceph as the research platform. The main contribution is to propose a novel collaboration mechanism to enable collaborative data

balancing among storage nodes, which ensures balanced storage performance while reducing data migration costs. The proposed mechanism is independent and can be used with other algorithms mapping storage locations in the storage system to comprehensively solve the balanced storage and data migration problems. At the same time, many application scenarios such as data migration, cold data precipitation, etc. can be derived in the future. These are data balanced storage strategies with high application prospects. However, the work in this paper also has limitations, which is a goal for future work. First, the algorithm designed in this paper is very inefficient when the system is at high usage, because more attempts are needed and no good constraint can be found at present. Secondly, the structure of the storage system is simple and does not take into account structures such as Bucket in the algorithm, which is also the focus of future work.

Acknowledgment. This work is supported by the Natural Science Foundation of China(No. 61762008), the National Key Research and Development Project of China (No. 2018YFB1404404), the Major special project of science and technology of Guangxi(No.AA18118047-7), and the Guangxi Natural Science Foundation Project (No. 2017GXNSFAA198141).

References

1. Zhang, W., Flores, H., Pan, H.U.I.: Towards collaborative multi-device computing. In: 2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops). IEEE (2018)
2. Rump, F., Timm, B., Raphael, E.: Distributed and collaborative malware analysis with MASS. In: 2017 IEEE 42nd Conference on Local Computer Networks (LCN). IEEE (2017)
3. Aghayev, A., et al.: File systems unfit as distributed storage backends: lessons from 10 years of Ceph evolution. In: Proceedings of the 27th ACM Symposium on Operating Systems Principles. ACM (2019)
4. Zhou, J., et al.: Pattern-directed replication scheme for heterogeneous object-based storage. In: 17th IEEE/ACM International Symposium on Cluster 2017, Cloud and Grid Computing (CCGRID). IEEE (2017)
5. Kisley, R.V., Philip, D.K.: Distributed file serving architecture with metadata storage and data access at the data server connection speed. U.S. Patent No. 9,262,094. 16 February 2016
6. Huang, M., et al.: Research on data migration optimization of Ceph. In: 2017 14th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP). IEEE (2017)
7. Zhao, N., et al.: A reliable power management scheme for consistent hashing based distributed key value storage systems. Front. Inf. Technol. Electron. Eng. **17**(10), 994–1007 (2016)
8. Zhang, X., Gaddam, S., Chronopoulos, A.T.: Ceph distributed file system benchmarks on an openstack cloud. In: 2015 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM). IEEE (2015)
9. Wang, L.: Optimizations on Ceph Cache Tiering. KylinCloud, Ceph day (2015)

10. Weil, S.A., et al.: RADOS: a scalable, reliable storage service for petabyte-scale storage clusters. In: Proceedings of the 2nd International Workshop on Petascale Data Storage: Held in Conjunction with Supercomputing 2007. ACM (2007)
11. Zhou, J., et al.: Pattern-directed replication scheme for heterogeneous object-based storage. In: 17th IEEE/ACM International Symposium on Cluster 2017, Cloud and Grid Computing (CCGRID). IEEE (2017)
12. Mseddi, A., Salahuddin, M.A., Zhani, M.F., et al.: Efficient replica migration scheme for distributed cloud storage systems. IEEE Trans. Cloud Comput. (2018)
13. D'atri, A., Bhembre, V., Singh, K.: Learning Ceph: unifed, scalable, and reliable open source storage solution. Packt Publishing Ltd. (2017)
14. Ou, J., et al.: EDM: an endurance-aware data migration scheme for load balancing in SSD storage clusters. In: 2014 IEEE 28th International Parallel and Distributed Processing Symposium. IEEE (2014)
15. Zhang, X., Wang, Y., Wang, Q., et al.: A new approach to double I/O performance for Ceph distributed file system in cloud computing. In: 2019 2nd International Conference on Data Intelligence and Security (ICDIS), pp. 68–75. IEEE (2019)
16. Aghayev, A., Weil, S., Kuchnik, M., et al.: File systems unfit as distributed storage backends: lessons from 10 years of Ceph evolution. In: Proceedings of the 27th ACM Symposium on Operating Systems Principles, pp. 353–369 (2019)