A Distributed Block Storage Optimization Mechanism Based on Ceph

Yang Pi[1, 2, a], Zhanglong Wang[1, 2, b]

[1]School of Computer Science and Technology, Chongqing University of Posts and Telecommunications, Chongqing, 400065, China

[2]Chongqing Engineering Research Center of Mobile Internet Data Application, Chongqing, 400065, China

[a]1056864298@qq.com, [b]1129279512@qq.com

Keywords: Ceph, Block storage, Cache Tiering, Storage selection, Load balancing

Abstract: RADOS block device (RBD) is the storage service component in Ceph, which provides an important block storage service for OpenStack and other cloud computing platforms. In response to the challenges faced by enterprises in terms of higher resource utilization and read/write rates on block storage services, Ceph provides Cache Tiering to improve cluster performance in heterogeneous storage environments. However, the least recently used (LRU) algorithm in Cache Tiering would evict more valuable data due to cache pollution which causes higher latency for some requests; Meanwhile, when the data is allocated on the storage node, the Controlled Replication Under Scalable Hashing (CRUSH) algorithm only considers the storage node capacity, which makes Ceph unable to dynamically balance the node's I/O load. To solve these problems, a storage selection strategy based on the prediction model is proposed to increase the hit ratio of object access in the cache pool and improve the overall I/O performance of the cluster; Besides, the cache pool I/O load balancing strategy is also proposed. Compared with the native mechanism, the proposed block storage optimization mechanism can achieve higher I/O throughput and more balanced I/O load.

1. Introduction

Ceph is a highly reliable and scalable distributed storage system that provides object storage, block storage, and file storage services to meet different application needs [1]. With the development of cloud computing technology represented by OpenStack, RBD, a block storage service component of Ceph, has been widely deployed and applied, becoming an indispensable block device storage backend for devices (e.g., virtual machines) [2]. Moreover, the I/O performance and reliability are the key to Ceph block storage mechanism.

In terms of block storage devices, the emerging solid state disk (SSD) has a faster access speed but is more expensive than the hard disk drive (HDD). Therefore, in Ceph, scenes in which SSD and HDD are mixed are common. In this case, Ceph provides a mechanism called Cache Tiering to improve cluster performance. That is, by establishing a layered object storage device (OSD) pool, the SSD OSD pool is used as a cache layer of the backend HDD OSD pool, which provides higher I/O performance for the Ceph RBD client. However, when the data is evicted at the cache layer, LRU algorithm in Cache Tiering only evicts data based on recent access records, which may lead to the eviction of more valuable and hot data due to occasional cold data access [3]. In addition, Ceph provides the CRUSH algorithm, which balances data distribution and load according to the capacity of OSD [4]. At this point, it can well adapt to the dynamic changes of the cluster, but other problems of OSD are not taken into account. For example, Ceph cannot balance the load when the I/O of some OSDs is overloaded and the I/O of others is idle.

The main contributions of this paper are as follows: Based on Ceph's Cache Tiering mechanism, we proposed a storage selection strategy based on predictive model, which determines whether the object request is to access the SSD OSD pool or the backend HDD OSD pool according to the object access frequency. This strategy can reduce the unnecessary overhead caused by cold data processing, so as to increase the overall I/O performance of the cluster. At the same time, we

proposed a load balancing strategy to make the I/O load of OSDs in cache pool more balanced.

2. Related Work

The research on Ceph mainly focuses on heterogeneous storage, read and write performance optimization, and application optimization on Ceph.

In terms of Ceph heterogeneous storage, LinWu et al. [5] proposed the Biased Object Storage Strategy, which reduces the number of write accesses of the SSD Storage device under the environment of heterogeneous Storage medium, and ensures the overall performance improvement of Ceph under the condition of load balance. For the defect that Ceph only uses the node storage capacity as the weight factor when performing data distribution, Duo Wu et al. [6] proposed that the network bandwidth, CPU load, and I/O load should be involved in the calculation of the weight of the storage node in heterogeneous environment, which makes it more reasonable to select the primary node, and finally improves the overall read performance of the cluster.

In terms of performance optimization of reads and writes on Ceph, Myoungwon Oh et al. [7] took all-flash as storage node medium, and proposed optimization technologies such as coarse-grained locking mechanism, non-blocking log and lightweight transaction processing to solve the problems caused by flash memory replacing HDD as storage medium, so as to improve Ceph's performance in random reads and writes of small files. Meanwhile, Ke Zhan et al. [8] used two threaded multi-threaded algorithms to optimize file read and write performance, improving small file upload and download performance.

In the research on application optimization of Ceph, Yutong Liang et al. [9] applied MapReduce to Ceph and proposed a better data placement strategy under the environment of different network bandwidth and node computing capacity, which made the overall data access time shorter and satisfied the load balance condition of storage nodes. Li Wang et al. [10] proposed to use the block device provided by Ceph in combination with the container, and proposed use-rbd as a lighter and more stable solution, due to the defects of qem-rbd and ker-rbd having some problems in the use of the container.

In this paper, a storage selection strategy based on the prediction model is proposed to improv e the hit ratio of the cache pool, and the I/O load balancing strategy of OSDs in the cache pool is also proposed.

3. Problem Analysis

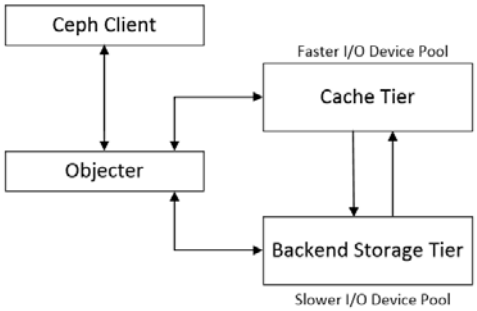

Figure 1. Cache Tiering

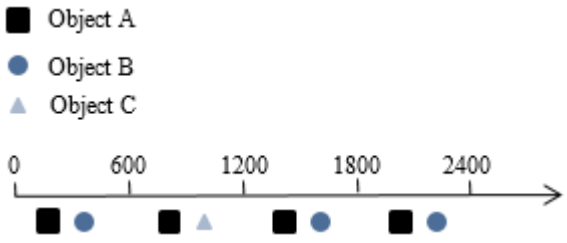

Figure 2. Example of object access in cache pool

To provide clients with better I/O performance, Ceph provides a Cache Tiering mechanism, as shown in Fig. 1. Cache Tiering uses a faster storage device (e.g., SSD) pool as the cache layer for a backend storage pool with slower storage devices (e.g., HDD). In most cases, the write request of the client first reaches the cache layer, and then an ACK is sent to the client through the cache layer, and dirty objects in the cache pool are flushed back to the backend storage pool when appropriate (e.g., the number of dirty objects gets a certain threshold). In addition, when the client makes a read request, it also reaches the cache layer first. If the requested object is not in the cache layer, it will be copied from the backend storage layer to the cache layer and sent to the client by the cache layer

[11]. Meanwhile, Cache Tiering uses the LRU algorithm to evict infrequently accessed objects when the capacity of the cache layer has reached a certain threshold.

To effectively utilize the limited cache pool resources in Cache Tiering, cold data should be stored in the backend storage pool, while hot data should be stored in the cache pool. Therefore, in the context of mass data storage, distinguishing the heat (i.e., the access frequency) of data and adopting different processing strategies can fully utilize the storage resources in Cache Tiering, and reduce the unnecessary overhead caused by cold data processing (e.g., cold data entering the cache pool from the backend storage pool, the LRU evicting the cold data).

As a caching mechanism, Cache Tiering has inherent flaws. First of all, in the massive data storage environment, the LRU strategy based on the locality principle shows its incompatibility. As shown in Fig. 2, we assume that in the object access time series with a period of 600s, the cache pool contains three objects A, B, and C. Object A is a frequently accessed object (4 times in total), object B is a frequently accessed object (3 times in total), and object C is temporarily accessed only in a time period of 600s to 1200s. If the cache pool capacity reaches the threshold in this time period, the object B will be evicted due to the arrival of the object C according to the time locality principle of LRU. Object C is only temporarily accessed, but it is considered likely to be accessed recently by the LRU algorithm. Therefore, The LRU lacks historical access analysis of massive data, thus bringing about the above cache pollution problem. In the actual Ceph environment, in the case of a large number of cold object requests, the above cache pollution problem will be more serious, that is, frequently accessed hot data may be evicted by occasionally accessed cold data [12], causing low hit ratio of hot objects in the cache pool and finally reducing the overall I/O performance of the cluster.

In this paper, we proposed a storage selection strategy based on predictive model for the above problems. The strategy adapts to the characteristics of massive data storage and analyzes the long-term access records of storage objects. Meanwhile, it judges whether to choose to access the SSD cache pool or the backend HDD storage pool according to the object heat in a certain period, so as to reduce the unnecessary overhead caused by cold data processing, and finally improve the cluster performance.

Secondly, Ceph's CRUSH algorithm takes the capacity of storage nodes in the cluster as the weight factor to select the storage nodes of objects, which ensures the balance of storage distribution. However, it also means that nodes with larger capacity have more object access, and such nodes have higher I/O load. In addition, different objects have different heat, and hot objects can bring higher I/O load to the nodes than cold objects. Therefore, in a heterogeneous storage environment, nodes with low I/O performance may be in I/O overload state while those with high I/O performance may be in I/O idle state. In the Cache Tiering mechanism, the client primarily sends I/O requests to the cache pool, so the cache pool needs a solution to the I/O load imbalance above.

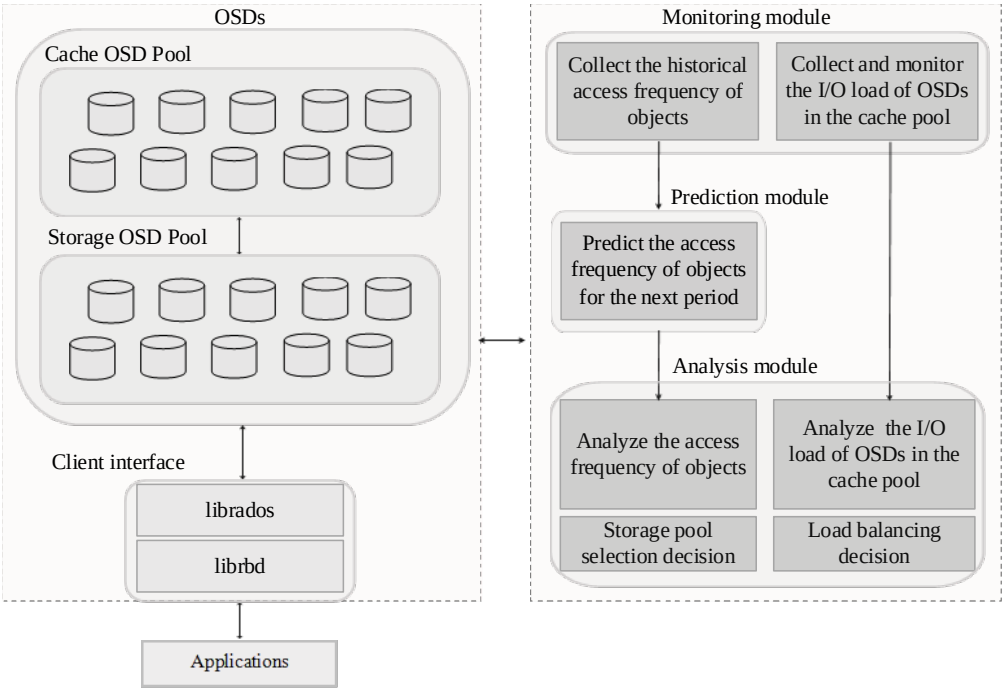

Figure 3. System module architecture

4. Cache Optimization Strategy Based On Cache Tiering

4.1. Cache Structure Design

Based on the problem analysis in the above section, this paper designes the system module shown in Fig. 3, on the basis of the Cache Tiering module structure of native Ceph system. Compared with native system, our designed system adds and modifies the following modules:

4.1.1 Monitoring module

This module is responsible for collecting and monitoring the I/O load information of each OSD in the cache pool, and collecting the historical access frequency of objects in the cluster by analyzing log files.

4.1.2 Prediction module

In distributed storage, users' access to data is not completely random, but predictable. This module uses the exponential smoothing prediction algorithm to analyze the historical object access records collected by the monitoring module, and predicts the access frequency of the next period of objects during the idle time of the system.

4.1.3 Analysis module

This module analyzes the prediction module and monitoring module’s I/O load information of OSDs in the cache pool, and makes corresponding decisions. For the objects whose access frequency exceeds the certain threshold, they will be recorded in the hot object record table. The table will be sent to the client interface module as cluster information, so that the client can judge whether to send the request to the cache pool or the backend storage pool. In addition, when some OSDs is overloaded in the cache pool, this module will analyze the I/O load information of OSDs in the cache pool sent by the monitoring module and send the determined load balancing scheme to the cluster.

4.1.4 Client interface module

We made changes to Ceph's native client interface module. Before accessing the object, the client interface module first checks whether the object is in the hot object record table. Requests for

objects in the hot object record table will arrive at the cache pool, while requests for objects not in the hot object record table will arrive at the backend storage pool.

4.2. Storage Selection Strategy Based on Predictive Model

The LRU algorithm in Cache Tiering only considers the recent access of the object when evicting objects in the cache pool. To eliminate this limitation, this paper uses the prediction model to analyze the historical access frequency of the object and dynamically obtain the object access frequency in the next period. Instead of putting all object requests into the cache pool like the Ceph native strategy does, we control which storage pool the request should arrive at, according to the access frequency of objects.

(1) Prediction model

Exponential smoothing is a commonly used predictive algorithm based on time series analysis [13]. It is easy to perform, and does not abandon the analysis of historical data. The basic idea of it is that different weights are given to the observed values of the past period, and the weight of the observations in the nearer period is greater. Also, the predicted value is the weighted sum of the previous observations. According to different smoothing times, the exponential smoothing algorithm is divided into: one-time exponential smoothing method, second exponential smoothing method and three-dimensional exponential smoothing method. Considering the simplicity and low consumption of the algorithm, this strategy uses one-time exponential smoothing method, as shown in equation (1) :

$$S_t = a * y_{t-1} + (1 - a) * S_{t-1} \quad (1)$$

S_t is the smooth value and the predicted value for period t; y_{t-1} is the actual value for period t-1; S_{t-1} is the smooth value and the predicted value for period t-1; a is the smoothing constant, whose value range is [0,1].

(2) Storage selection strategy

In the prediction module, we use the above prediction model to analyze the object history access frequency for the past 1, 2, 3, ..., t-1 periods, and then obtain the prediction of the object access frequency for period t. When the predictive value exceeds the threshold H, its corresponding object will be recorded in the HTable. Also, H can be flexibly adjusted according to experience.

The HTable generation algorithm is shown in Algorithm 1. At the same time, the algorithm used by the client interface to judge the target storage pool accessed by the request is shown in Algorithm 2. And the relevant description of the notation is as follows: OSet represents the object set: $\{O_1,O_2,\ldots,O_i,\ldots,O_n\}$, F^i_t represents the prediction of the access frequency of the object O_i for period t, H represents the access frequency threshold of objects sent to the cache pool, HTable represents the table for recording hot objects, and requestSet represents the set of requests from clients: $\{R_1,R_2,\ldots,R_i,\ldots,R_n\}$.

Table.1. HTable generation algorithm.

Input: OSet: $\{O_1,O_2,\ldots,O_i,\ldots,O_n\}$.
process:
1: for i=1,2,…,n do
2: Obtain the historical access frequency of the object O_i: F^i_1 , F^i_2 , $F_3{}^i$,... , F^i_{t-1} ;
3: Calculates the F^i_t of O_i for period t by using the prediction model;
4: if $F^i_t > H$
5: if O_i is not in the HTable
6: Record O_i into the HTable;
7: if $F^i_t < H$
8: if O_i is in the HTable
9: Delete the record of O_i from the HTable;
Output: HTable.

Table.2. Storage pool selection algorithm.

Input: requestSet: $\{R_1,R_2,\ldots,R_i,\ldots,R_n\}$.
process:
1: for i=1,2,…,n do
2: if the object to be accessed by R_i is in the HTable
3: Set the id of the target storage pool to be accessed by R_i as the cache pool id;
4: else
5: Set the id of the target storage pool to be accessed by R_i as the backend pool id;
Output: the id of the target storage pool to be accessed.

4.3. Load Balancing Strategy of OSDs in the Cache Pool

This strategy adds the function of monitoring and collecting the I/O load information of the cache pool to the monitoring module, and the load will be analyzed in the analysis module so as to formulate the corresponding load balancing scheme.

(1) The I/O load of the OSD

The cluster node OSD has a specific I/O load range during normal operation, so when the I/O load exceeds the threshold, the OSD is judged to be in the I/O overload state, otherwise it is judged to be in the idle state.

(2) Load balancing strategy

In the cache pool, if the I/O load of some OSDs is overloaded and the I/O of other OSDs is idle, this strategy gives the corresponding load scheme for two different I/O requests of reading and writing respectively. In the case of unbalanced read I/O load of OSDs, this strategy migrates some objects in OSDs with read I/O overload to the target OSDs with higher I/O performance, lower read I/O load and larger capacity, until the read I/O load of each OSD is lower than the threshold. For the unbalanced write I/O load of OSDs, this strategy migrates partial write requests to the target OSDs with higher I/O performance, lower write I/O load and larger capacity, until the write I/O load of each OSD is lower than the threshold. Moreover, this strategy uses the CRUSH algorithm of Ceph to calculate the location of the target OSDs, and at the same time uses the I/O performance, I/O load and capacity of the OSD to calculate the comprehensive weight of the OSD, which ensures that OSDs with higher I/O performance, lower I/O load and larger capacity tend to be selected as the target storage nodes. The comprehensive weight of the OSD is calculated as equation (2) :

$$W_i = perf_i * \omega 1 + load_i * \omega 2 + cap_i * \omega 3 \quad (2)$$

The $perf_i$, $load_i$ and cap_i respectively represent the I/O performance, I/O load and capacity of OSD_i. Also, ω1, ω2, ω3 respectively represent the importance proportion of $perf_i$, $load_i$, cap_i in the calculation of the comprehensive weight, which can be adjusted according to experience, and satisfy the condition of ω1+ω2+ω3=1.

The load balancing strategy is shown in Algorithm 3. And the relevant description of the notation is as follows: OSDSet is denoted as the OSD set: $\{OSD_1,OSD_2,...,OSD_i,...,OSD_n\}$, $load^r_i$ and $load^w_i$ represents the read I/O load and the write I/O load of OSD_i respectively, T^r_i and T^w_i represents the threshold of read I/O load and the threshold of write I/O load of OSD_i respectively, $targetOSD_i$ represents the target OSD which objects or requests are migrated to, $target\_load^r_i$ and $target\_load^w_i$ represents the read I/O load and the write I/O load of $targetOSD_i$ respectively, $target\_T^r_i$ and $target\_T^w_i$ represents the threshold of the read I/O load and the threshold of the write I/O load of $targetOSD_i$ respectively, WSet represents the comprehensive weight set of OSDs: $\{W_1,W_2,...,W_i,...,W_n\}$.

Table.3. load balancing algorithm.

Input: OSDSet: $\{OSD_1,OSD_2,...,OSD_i,...,OSD_n\}$.
process:
1: for i=1, 2,…, n do // traverse OSDSet.
2: Calculate W_i of OSD_i by equation (1); //get WSet: $\{W_1,W_2,...,W_i,...,W_n\}$
3: Obtain the read I/O load set of OSDs: $\{load^r_1,load^r_2\ldots,load^r_i,\ldots,load^r_n\}$;

4: Obtain the write I/O load set of OSDs: {$load^w_1$,$load^w_2$…,$load^w_i$,…,$load^w_n$};
5: for i=1,2,…,n do //traverse OSDSet
6: if $load^r_i > T^r_i$ && $target\_load^r_i < target\_T^r_i$ //the trigger condition for read I/O load balancing
7: Obtain the access frequency predictions of objects in OSD_i in the next period: F^1_t ,F^2_t ,...,F^i_t,...,F^n_t, and sort them from high to low;
8: for j=1,2,…,n do //traverse OSet
9: Use WSet to calculate the target OSD($targetOSD_j$) when migrating the object O_j and the corresponding request in CRUSH algorithm;
10: Preferentially migrate objects with higher access frequency;
11: else if $load^w_i > T^w_i$ && $target\_load^w_i < target\_T^w_i$ //the trigger condition for write I/O load balancing
12: Use WSet to calculate the target OSD($targetOSD_j$) when migrating the request from the client in CRUSH algorithm;
Output: The set of the target OSD: {$targetOSD_1$,$targetOSD_2$,...,$targetOSD_i$,...,$targetOSD_n$}.

5. Experiments

The experiments in this paper implement the related configuration of Cache Tiering in Ceph, which simulates the situation where multiple users randomly access Ceph block devices, and the average size of the accessed files is 40MB. Meanwhile, we compared the native system strategy and the optimization strategy in terms of read-write performance, cache hit ratio, and I/O load variance. The experimental environment is as follows: the Ceph version is 10.2.9, the OSD CPU is 1 GHz, the OSD memory is 1 GB, the number of OSDs of the cache pool and the backend storage pool are all 6, the number of replica in the cache pool and the backend storage pool are all 3, the storage medium for OSDs of the cache pool and the backend storage pool are SSD and HDD respectively, and the total capacity of the backend storage pool is 80GB.

5.1. Read/Write Performance Comparison

In the optimization strategy in this paper, whether the request chooses to access the cache pool or the backend storage pool depends on the predicted frequency value of the accessed object in the current period. And only when this value is greater than the threshold H, the request will be sent to the cache pool. We first carried out two experiments with a period of 1 min, which analyzed the impact of different H on the read-write performance of the optimized Ceph when the cache pool capacity is 4GB, 8GB, 12GB and 16GB, respectively.

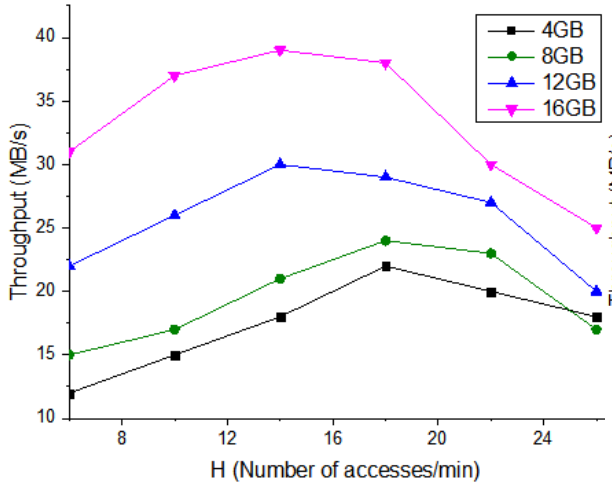

Figure 4. The average read throughput of cluster with different capacities and H

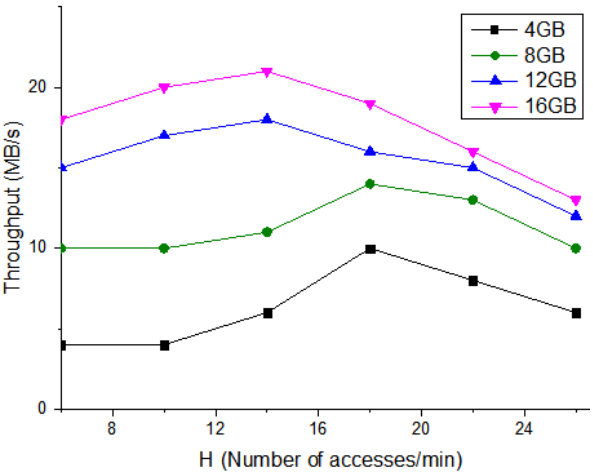

Figure 5. The average write throughput of cluster with different capacities and H.

According to the experimental results in Fig. 4 and Fig. 5, there is a common rule in the case of different capacity of the cache pool, that is, as the value of H increases, the overall throughput of the

cluster increases first and then decreases. The reason is as follows: in the case of a certain capacity of the cache pool, when H increases, the access frequency of objects in the cache pool will increase, and the probability of object eviction in the cache pool is smaller, so the hit ratio will be higher and the performance of the cluster will be higher. When H increases to a certain extent, only a few requests can reach the cache pool, and more requests access the backend storage pool. Also, because the performance of the backend storage pool is lower than that of the cache pool, the overall performance of the cluster becomes lower in this case. Moreover, as a whole, when H is 18, the read and write throughput of the cluster is the highest under the scenarios of different cache pool capacities. Therefore, we use H=18 as the optimal condition for the following comparison experiments with different cache pool capacities.

As shown in Fig. 8, in the case of H=18, with the increase of the total size of the cache pool, the cache pool hit ratio of native system and optimized system all increase gradually, and finally tend to be stable, because the data in the cache pool will not be evicted when the capacity of the cache pool reaches a certain value. Compared with the native system, the optimized system system has a certain improvement in the cache pool hit ratio.

Also, we conducted a comparative experiment on the average read throughput and the average write throughput of the cluster between the native system and the optimized system when H=18. The results are shown in Fig. 6 and Fig. 7. It can be seen that with the increase of the total capacity of the cache pool, the optimized system's average read-write throughput and the native system's average read-write throughput all have a similar trend: first it gradually increases, and finally it tends to be stable. The reason is that, at the beginning, in the cache pool, the cache hit ratio increases as the total capacity increases, therefore, the average read throughput and average write throughput increase, and finally tend to be stable due to the performance limitations of the cache pool. We can also see that the optimized system has a certain increase in read throughput and write throughput compared with the native system.

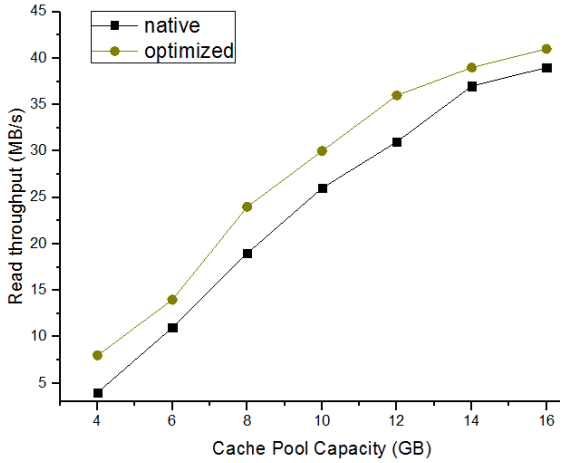

Figure 6. The comparation of average read throughput.

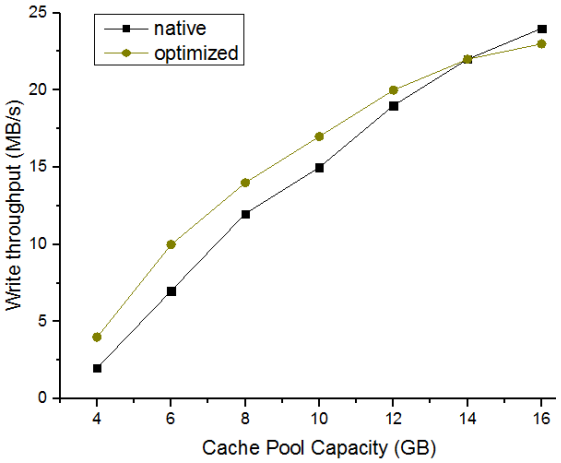

Figure 7. The comparation of average write throughput.

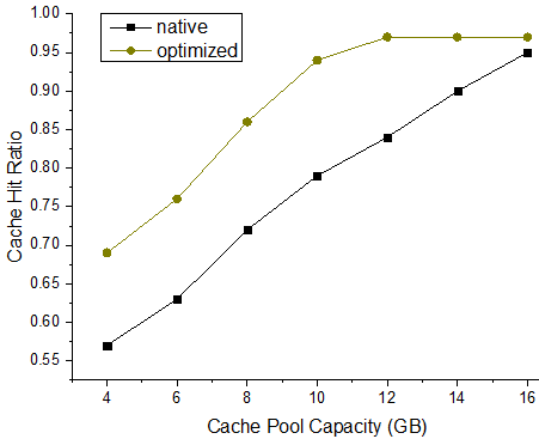

Figure 8. The comparation of the hit ratio in the cache pool

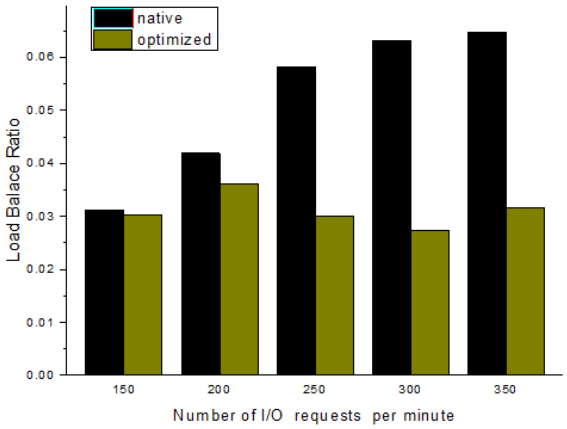

Figure 9. The comparation of load balancing

5.2. Load Balancing Comparison

In this experiment, the variance of the I/O load of OSDs in the cache pool is taken as the coefficient for measuring the overall load balance of the cache pool. The test time is 5 minutes and the number of requests from the client varies per minute. As can be seen from Fig. 9, as the number of the requests per minute increases, the overall I/O load balancing degree of the native cache pool decreases, while the overall I/O load balancing degree of the cache pool using the proposed strategy is better and more stable. The reason is that in the native system, as the number of requests from the client increases, the OSDs with the larger capacity receive more I/O requests, and their loads also increase. Hence, the overall load of the cache pool becomes more unbalanced, while the optimized system can balance the I/O load of OSD according to the capacity, I/O performance and load degree.

6. Conclusion

In this paper, we optimized the Ceph distributed block storage mechanism and proposed a storage selection strategy based on the prediction model under the Cache Tiering mechanism, so as to improve the performance of the system; Besides, for the balance of I/O load of OSDs in the cache pool, we proposed a load balancing strategy based on the capacity, I/O performance, and load degree of the OSD. From the experimental results, we can see that the optimization strategy proposed in this paper can improve the I/O performance of the cluster to some extent and make the I/O load of the cluster more balanced. Furthermore, in the Cache Tiering architecture, less expensive and less time-delayed I/O communication between the cache pool and the backend storage pool is what we can continue to work on in the future.

References

[1] Zaytsev A , Ito H , Hollowell C , et al. Current Status of the Ceph Based Storage Systems at the RACF[J]. Journal of Physics: Conference Series, 2015, 664(4):042027.

[2] Wang L , Wen Y . Design and Implementation of Ceph Block Device in Userspace for Container Scenarios[C]// International Symposium on Computer. IEEE, 2016.

[3] Chen F , Koufaty D A , Zhang X . Hystor:making the best use of solid state drives in high performance storage systems[C]// International Conference on Supercomputing. ACM, 2011.

[4] Weil S A , Brandt S A , Miller E L , et al. Grid resource management - CRUSH: controlled, scalable, decentralized placement of replicated data[C]// Proceedings of the ACM/IEEE SC2006 Conference on High Performance Networking and Computing, November 11-17, 2006, Tampa, FL, USA. ACM, 2006.

[5] Wu L, Zhuge Q , Sha H M , et al. BOSS: An Efficient Data Distribution Strategy for Object

Storage Systems with Hybrid Devices[J]. IEEE Access, 2017:1-1.

[6] Wu D, Wang Y , Feng H , et al. Optimization Design and Realization of Ceph Storage System Based on Software Defined Network[C]// International Conference on Computational Intelligence & Security. IEEE Computer Society, 2017.

[7] Oh M, Eom J , Yoon J , et al. Performance Optimization for All Flash Scale-Out Storage[C]// IEEE International Conference on Cluster Computing. IEEE, 2016.

[8] Zhan K, Piao A H. Optimization of Ceph Reads/Writes Based on Multi-threaded Algorithms[C]// IEEE International Conference on High Performance Computing & Communications; IEEE International Conference on Smart City; IEEE International Conference on Data Science & Systems. IEEE, 2017.

[9] Sha H M, Liang Y, Jiang W, et al. Optimizing Data Placement of MapReduce on Ceph-Based Framework under Load-Balancing Constraint[C]// 2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS). IEEE Computer Society, 2016.

[10] Wang L, Wen Y. Design and Implementation of Ceph Block Device in Userspace for Container Scenarios[C]// International Symposium on Computer. IEEE, 2016.

[11] Information on http://docs.ceph.com/docs/master/rados/operations/cache-tiering/.

[12] Huang S, Wei Q, Chen J, et al. Improving flash-based disk cache with Lazy Adaptive Replacement[C]// 2013 IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST). IEEE, 2013.

[13] Sarigiannidis P, Gkaliouris A, Kakali V, et al. On forecasting the ONU sleep period in XG-PON systems using exponential smoothing techniques[C]// Global Communications Conference. 2015.