Taming Performance Hotspots In Cloud Storage With Dynamic Load Redistribution

Ridwan Rashid Noel
Dept. of Computer Science
University of Texas at San Antonio
San Antonio, Texas-78249
Email: ridwanrashid.noel@utsa.edu

Palden Lama
Dept. of Computer Science
University of Texas at San Antonio
San Antonio, Texas-78249
Email: palden.lama@utsa.edu

***Abstract*—Cloud storage services are associated with high latency variance, and degraded throughput which is problematic when users are fetching and storing content for interactive applications. This can be attributed to performance hotspots created by slow nodes in a storage cluster, and performance interference caused by multi-tenancy, and background tasks such as data scrubbing, backfilling, recovery, etc. In this paper, we present DLR, a system that improves the performance of cloud storage services in the presence of hardware heterogeneity, and performance interference through a dynamic load redistribution technique. We designed DLR to dynamically adjust the load serving ratio of storage servers based on the system-level performance measurements from the storage cluster. We implemented DLR using Ceph, a popular distributed object storage system, and evaluated its performance on NSF-Cloud's Chameleon testbed using Ceph's Rados benchmark. Experimental results show that DLR improves the average throughput and latency of Ceph storage by up to 65%, and 41% respectively compared to the default case. Compared to Ceph's in-built load balancing technique, DLR improves the throughput by up to 98%, and latency by 96%.**

***Keywords*-Hardware Heterogeneity; Performance Interference; Ceph; Cloud Storage;**

I. Introduction

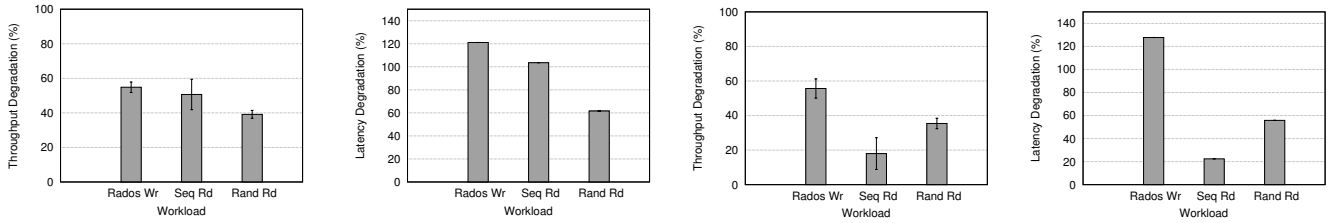
Cloud storage services (both public and private) are increasingly used as cost-effective platforms for storing large-scale enterprise data due to the flexibility, availability, and scalability provided by the underlying object-based storage technology (e.g OpenStack Swift [3], Ceph [18], Amazon S3, etc.). However, on today's cloud services, both fetching and storing content are associated with high latency variance [16, 21], and degraded throughput. This can be attributed to performance hotspots created by slow nodes in a heterogeneous storage cluster, and performance interference caused by multi-tenancy [13] as well as background tasks such as data scrubbing, backfilling, recovery, etc.

Unlike traditional application and organization-specific clusters, consolidated cloud environments are likely to be constructed from a variety of machine classes [11, 14, 15]. Furthermore, with the advent of software-defined storage system such as Ceph, which supports rolling hardware and software upgrades, as well as the ability to run mixed hardware configurations, the storage nodes are even more likely to be heterogeneous. Our motivational case study in Section III shows that the presence of four slower nodes in a Ceph storage cluster of eight nodes degrades the object store throughput by up to 54% and causes 2.2X increase in the latency. As cloud storage is shared by multiple tenants, the contention of shared resources can cause significant performance degradation [13]. Furthermore, storage related background tasks such as data scrubbing, backfilling, recovery, etc. are potential sources of performance interference. Unlike hardware heterogeneity, performance interference can cause various nodes to become performance hotspots at different times. Our motivational case study shows that inducing performance hotspots on randomly selected nodes in a Ceph storage cluster degrades the object store throughput by 55% and increases the latency by 2.3X.

Existing cloud storage services implement very simple schemes that have little or no ability to react quickly to performance hotspots. Recent efforts [16] propose to reduce latencies in cloud data store by adaptively selecting one out of multiple replica servers to serve a request based on a continuous stream of in-band feedback about a server's load. However, such approach is only suitable for low-latency data stores (e.g key-value store) where the service times of individual requests are small enough so that sufficient feedback can be collected to accurately rank the replica servers in the face of performance fluctuations, and changing system dynamics. However, large-scale cloud storage systems such as object-based storage have to deal with a wide range of data object sizes, and correspondingly varying service times.

In this paper we present DLR, a system that improves the performance of cloud storage services in the presence of hardware heterogeneity, and performance interference through a dynamic load redistribution technique. DLR dynamically adjusts the load serving ratio of storage servers based on the system-level performance feedback from the storage cluster. It leverages the fact that system-level performance metrics such %iowait is highly correlated with the performance of storage nodes, and at the same time such metrics can be collected quickly irrespective of individual request service times. DLR is applicable to any cloud storage system where data is replicated, and a mechanism to adjust the load serving ratio of storage servers is available. We implemented DLR in Ceph, a popular software-defined

2159-6190/17 $31.00 © 2017 IEEE
DOI 10.1109/CLOUD.2017.15



(a) Throughput Degradation due to Hardware Heterogeneity (b) Latency Degradation due to Hardware Heterogeneity (c) Throughput Degradation due to Performance Interference (d) Latency Degradation due to Performance Interference

Figure 1: Performance Degradation due to Hardware Heterogeneity and Performance Interference

storage (SDS) solution based on a fully distributed, reliable and autonomous object store. Performance evaluation was conducted on a Ceph cluster built using NSFCloud's Chameleon testbed. Similar to others [10, 17] we ran Rados Bench, an in-built benchmarking program, to measure the performance of Ceph object store. As a baseline for performance comparison, we used Ceph in the default mode, and also applied Ceph's in-built load balancing technique. Experimental results show that DLR improves the average throughput and latency of Ceph storage by up to 65%, and 41% respectively compared to the default case. Compared to Ceph's in-built load balancing technique, DLR improves the throughput by up to 98%, and latency by 96%.

The rest of the paper is organized as follows. Section II provides background studies, and Section III describes about the motivations and challenges of this work. Section IV discusses the DLR system architecture and design. Section V evaluates the proposed system using Ceph's Rados benchmarks. Section VI outlines related work. Section VII summarizes our contribution and discusses future work.

II. Background

Ceph is an open source software-defined storage (SDS) solution, which is gaining increasing popularity due to its robust design and scaling capabilities. It provides all data access methods (file, object, block) and appeals to IT administrators with its unified storage approach. Its core, RADOS, is a fully distributed, reliable and autonomous object store [20]. Ceph's building blocks are called *OSDs Object Storage Daemons*, which are responsible for storing objects on local filesystems, as well as working together to replicate data, detect and recover from failures, or migrate data when OSDs join or leave the cluster. Ceph OSD Daemons create object replicas on other Ceph Nodes to ensure data safety and high availability.

Ceph uses a pseudo-random placement algorithm called CRUSH (Controlled Replication Under Scalable Hashing)[19], which allows OSDs and clients to compute object locations instead of looking them up in a centralized table. Then clients can directly interact with the OSDs for I/O. The CRUSH map also allows for defining hierarchies of failure domains for placing object replicas at different levels, e.g. disk, host, rack and room. Based on these hierarchies and on CRUSH rules, CRUSH maps objects to placement groups (logical aggregations of objects inside one pool) and then maps each placement group to one primary OSD, and some replica OSDs. A Ceph client always reads data from the *primary OSD*. While writing, the client writes data to primary OSD, and then replicates the data to the other OSDs.

Ceph monitors are responsible for maintaining the cluster map, which includes information on the cluster topology, a list of OSDs, pools, placement groups, and mapping of placement groups to OSDs. Clients contact one of the monitors to obtain the most recent cluster map. Ceph exposes different interfaces to storage: a POSIX-compliant filesystem (CephFS), block storage (Rados Block Device/RBD) and a RESTful API (Rados Gateway). Clients can directly access RADOS through librados, with support for several programming languages, including: C/C++ and Python.

III. Motivation

A. Effect of Hardware Heterogeneity

To study the impact of hardware heterogeneity on cloud storage performance, we setup an eight node Ceph cluster on the NSF Cloud's Chameleon testbed, and simulate low I/O bandwidth on four of the nodes by running fio [2] benchmark in the background. We run the Rados Bench in write, sequential read, and random read modes on the Ceph cluster with and without heterogeneity. Each experiment is repeated three times, and the average performance is reported. As shown in Figures 1a and 1b, the presence of four slower nodes in a Ceph storage cluster of eight nodes degrades the object store throughput by up to 54% and causes 2.2X increase in the latency.

B. Performance Interference

Unlike hardware heterogeneity, performance interference from various sources are likely to cause time-varying performance hotspots across various cloud storage nodes. To study the impact of performance interference on cloud storage performance, we run fio random read benchmark in four randomly selected nodes in the Ceph cluster, and repeated

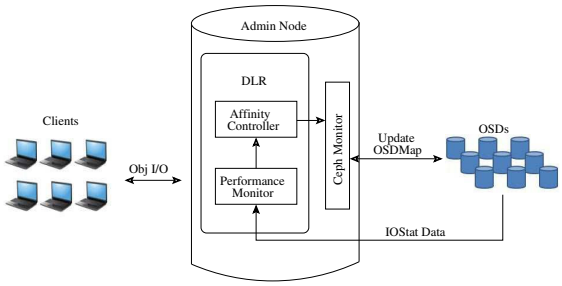


Figure 2: System Architecture

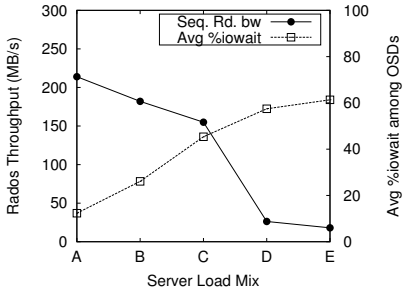


Figure 3: Effect of average %iowait values on Rados Performance

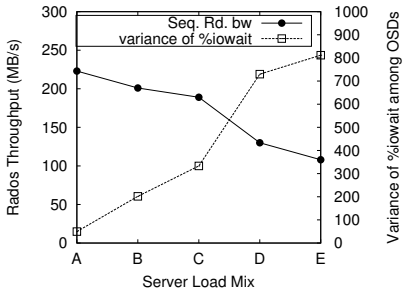


Figure 4: Effect of the variance of %iowait values on Rados Performance

the process after 90 second interval. We measure the average throughput, and latency of Rados Bench with and without performance interference. Figures 1c and 1d, show that inducing performance hotspots on randomly selected nodes in a Ceph storage cluster degrades the object store throughput by up to 55% and increases the latency by 2.3X.

IV. Design

A. System Architecture

The DLR system architecture consists of two main modules: the Performance Monitor and Affinity Controller. The modules run as daemon processes on Ceph admin node. The Performance Monitor periodically measures the system-level performance metrics of the OSDs, and sends the performance data to the Affinity Controller module. The Affinity Controller processes the performance data and updates the primary affinity values of the OSDs according to a dynamic affinity control algorithm. In Ceph, each OSD is associated with a primary affinity value between 0 and 1, which determines the probability that the OSD will act as the primary OSD. Hence, primary affinity can be used as a mechanism to control the load serving ratio of an OSD. The Affinity Controller invokes the Ceph monitor to update the OSDmap of the Ceph cluster so that the new primary-affinity values of the OSDs take effect. Figure 2 shows the high level overview of our proposed architecture.

B. Performance Monitor

The Performance Monitor relies on system level performance metrics as an indicator for detecting performance hotspots in a cloud storage cluster. Since disk I/O is a major source of performance bottleneck for data-intensive applications, we study I/O related performance metric, %iowait, and its correlation with the observed object store performance. %iowait metric provides the percentage of time that the CPU or CPUs are idle during which the system has an outstanding disk I/O request. Figure 3 shows the average %iowait values among the OSDs, and the performance of Rados Bench Sequential Read for various server load mixes. A,B,C,D,E denote the server load mixes in which we run the fio benchmark with increasing job sizes (A=250MB, B=500MB, C=1GB, D=2GB, E=4GB) as background jobs to introduce disk I/O contention on all the OSDs. The configuration details of fio is given in Section V-B. As shown in Figure 3, as the average %iowait values among the OSDs increases, the Rados throughput decreases. We got similar results for other Rados benchmarks. We further study the relationship between the variance of %iowait values among the OSDs, and the observed object store performance by running various server load mixes. Figure 4 shows that increase in the variance of %iowait values among the OSDs is correlated with the decrease in Rados throughput.

C. Affinity Controller

The Affinity Controller periodically adjusts the primary-affinity of the OSDs according to the dynamic affinity control algorithm given in Algorithm 1. In the algorithm, we apply three threshold values to control the change in the primary-affinity values of the OSDs. The first threshold, θ_h is used to determine if the %iowait value of an OSD is high enough to consider a decrease in its primary-affinity value. If the %iowait value is larger than θ_h, we check if (δ_2) the relative difference between the OSD's %iowait value, and the average %iowait among all OSDs is greater than the second threshold, θ_i. This relative difference represents the extent of load imbalance in the cluster. If the load imbalance is high enough, we decrease the primary affinity of the corresponding OSD by the sum of δ_1, and δ_2. Here, δ_1 is the normalized difference between the OSD's %iowait value,

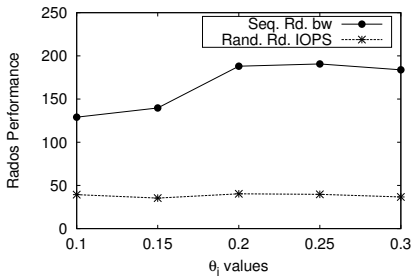


Figure 5: Effect of threshold values on Rados Performance

and the upper threshold θ_h. We set the lower limit on the primary-affinity value as 0.1. We apply a third threshold, θ_l to determine if the %iowait value of an OSD is too low, indicating that the OSD is no longer a performance hotspot. If the condition is met then we increase the primary-affinity aggressively to the maximum limit of 1. The control interval of the Affinity Controller is set to be 20 seconds. If this duration is too small, the measurement of %iowait value can be noisy, and if the duration is too large then the controller will be less sensitive to the performance hotspots.

Algorithm 1 Dynamic Affinity Control Algorithm

1: **Variable:** n: total number of OSDs, θ_h: threshold1, θ_i: threshold2, θ_l: threshold3, iow: %iowait values for an OSD, ϕ: percentage difference from average %iowait value for an OSD, α: current primary-affinity value of an OSD
2: **while** $True$ **do**
3: $avg_{iow} = \sum_{i=1}^{n}(iow_i/n)$
4: **for** i in n **do**
5: **if** $(iow_i > \theta_h)$ **then**
6: $\delta_1 \leftarrow (iow_i - \theta_h)/100$
7: $\phi_i \leftarrow (iow_i - avg_{iow})/iow_i$
8: **if** $(\phi_i > \theta_i)$ **then**
9: $\delta_2 \leftarrow (\phi_i - \theta_i)$
10: $\alpha_{i+1} \leftarrow max(0.1, (\alpha_i - (\delta_1 + \delta_2)))$
11: **end if**
12: **else**
13: **if** $(iow_i < \theta_l)\ and\ (\alpha_i < 1)$ **then**
14: $\alpha_{i+1} \leftarrow 1$
15: **end if**
16: **end if**
17: **end for**
18: **end while**

D. Threshold Parameter Tuning

It is important to tune the threshold values used in Algorithm 1 to optimize the performance of DLR approach. From our observation and also from the results shown in Figure 3, we found that if the %iowait value of an OSD exceeds 50, the overall performance degrades drastically. So we set the value of θ_h to be 50. θ_l is set to be 20. For tuning the value of θ_i, we ran Rados bench with various thresholds while inducing performance hotspots on random nodes. In all cases, we found that setting θ_i to be 0.2 gives the best Rados performance overall, e.g. Figure 5 shows the throughput of Rados Sequential read for various θ_i.

V. Evaluation

A. Testbed

For the experiments we setup an 8-node Ceph storage cluster on the NSFCloud's Chameleon testbed. One extra node is used as the Ceph Admin. Each node is equipped with 2.3GHz Intel Xeon dual-core processor E312xx (Sandy Bridge). The nodes run Ubuntu Linux 14.04.1 LTS with kernel 3.13.0 and each has 40GB of hard disk space and 4GB of memory. Ceph version 9.2.1 (Infernalis-stable) is used. The Ceph admin node is also configured as the Ceph monitor node of the cluster. The Ceph cluster has three pools with 512 placement groups and a replication factor of 2.

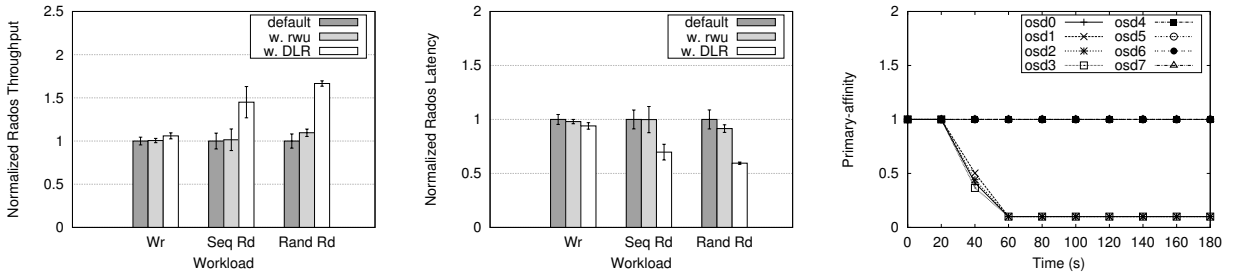
B. Benchmarks

Ceph comes with an inbuilt benchmark known as RADOS bench which is used to measure the performance of a Ceph object store. It has a command line interface, where one can pass different parameters, such as: I/O operation (Write, Sequential Read, Random Read), number of concurrent operations, duration of run or object size. RADOS Write bench simply writes out objects to the underlying object storage as fast as possible, and Rados Sequential Read and Random Read reads the objects in sequential and random order respectively. The performance metric used for Rados Write and Sequential Read is the throughput (MB/s), and the metric for the Rados Random Read is the IOPs. The latency for the Rados bench is also measured. In our experiments, we run fio (version 2.1.3) as background processes on the OSDs to induce performance hotspots. We use the default fio blocksize (4KB), libaio ioengine, iodepth of 16 and ran 4 parallel randread jobs of size 2GB each.

As a baseline for performance comparison, we used Ceph in the default mode, and Ceph with its in-built load balancing technique called Ceph osd-reweight-by-utilization[1]. This technique reduces the weight of the heavily overused OSDs to load balance the cluster. Reducing the weight of over-loaded OSDs is expected to improve the overall performance as data is moved from the over utilized OSDs to the OSDs that have average utilizations below a given threshold.

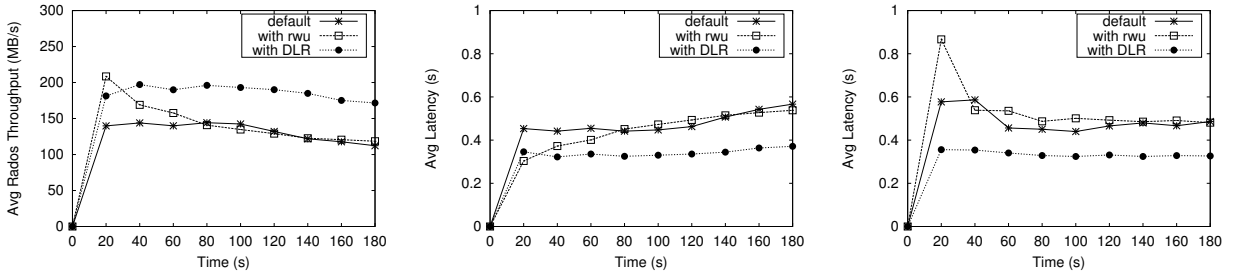
C. Addressing the Impact of Hardware Heterogeneity

For simulating a heterogeneous storage cluster, we run fio random read benchmark as background processes on half of the OSDs in the Ceph cluster. In this setting, we run the three Rados Benchmark (Write, Sequential Read and Random Read) for a duration of 180 seconds each with default object



(a) Rados Throughput (b) Rados Latency (c) Change in the Primary-affinity values in the OSDs using DLR

Figure 6: Comparison of Normalized Rados Performance among default approach, reweight-by-utilization and DLR in the case of Hardware Heterogeneity



(a) Rados Seq Rd Avg Throughput (b) Rados Seq Rd Avg Latency (c) Rados Random Rd Avg Latency

Figure 7: Comparison of Average Rados Performance among default approach, reweight-by-utilization and DLR during different time interval in the case of Hardware Heterogeneity

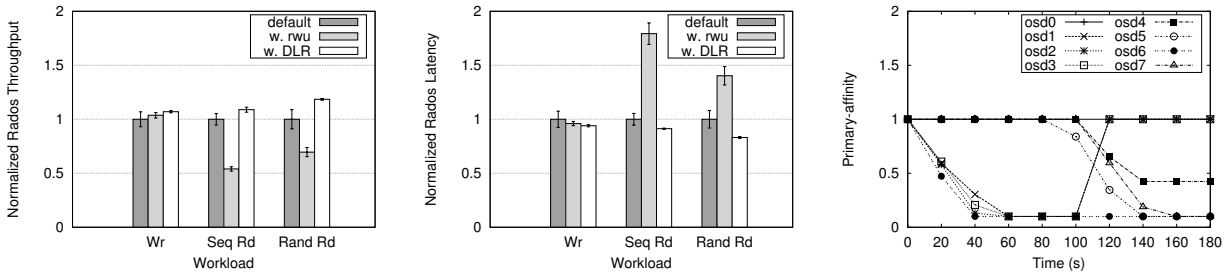
size of 4MB and 16 concurrent I/O operations. All of the experiments are run 3 times and the average performance and standard error is measured. The OS cache was cleared after each run to remove any effects of caching.

Figure 6a and Figure 6b compare the normalized Rados throughput and latency among the default mode, reweight-by-utilization, and DLR. The average performance is normalized according to the performance of the default Ceph mode. As shown in Figures 6a, and 6b, DLR significantly outperforms the Ceph default, and reweight-by-utilization case. Compared to default Ceph, DLR improves the Sequential Read throughput by 45%, and the Random Read IOPs by 65%. DLR achieves 30% improvement in Sequential Read latency and 41% improvement in Random Read latency. The superior performance of DLR is due to its ability to mitigate performance hotspots through dynamic affinity control. Figure 6c shows the change in the primary-affinity of the OSDs due to DLR. Figure 7 compares the average Rados performance among the default approach, reweight-by-utilization and DLR during different time intervals.

The Ceph OSD reweight-by-utilization approach does not show much performance improvement due to the associated data movement overhead. We observe that in case of the Rados Write benchmark, DLR achieves a small performance improvement over other approaches. DLR's primary affinity adjustment does not have much impact on Ceph write operation since every write operation on a primary OSD is followed by subsequent writes on the replica OSDs.

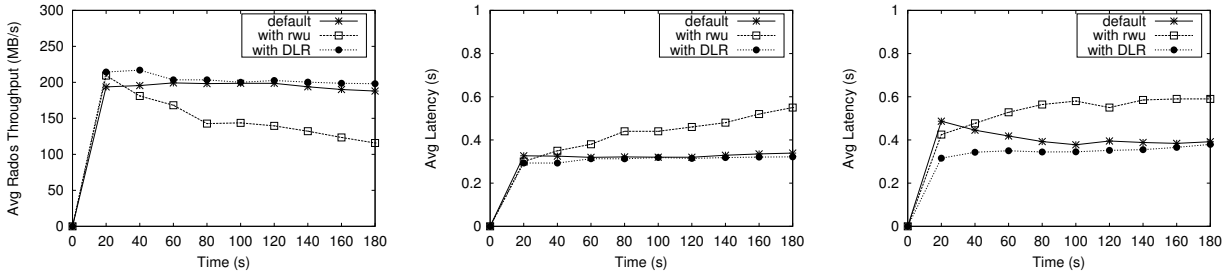
D. Mitigating Performance Interference

For simulating time-varying performance hotspots caused by interference, we ran fio random read for 90 seconds on four random OSDs, and then after killing the initial fio random reads we repeated the process on four other random OSDs. We ran each Rados Bench for 180 seconds with 4MB object size and 16 concurrent I/O operations. Figures 8a and 8b compare the normalized Rados throughput and latency among the default approach, reweight-by-utilization and DLR. We observe that compared to default Ceph, DLR improves the throughput of Rados Sequential and Random Read benchmarks by 9% and 19% respectively. The latency improvement is 8% and 16% respectively. Compared to the hardware heterogeneity case, the performance difference between DLR, and default Ceph is less here. This is because the performance degradation of Rados read benchmarks due to performance interference is relatively less than the degradation caused by hardware heterogeneity in our experiments. Hence, there is less opportunity for performance



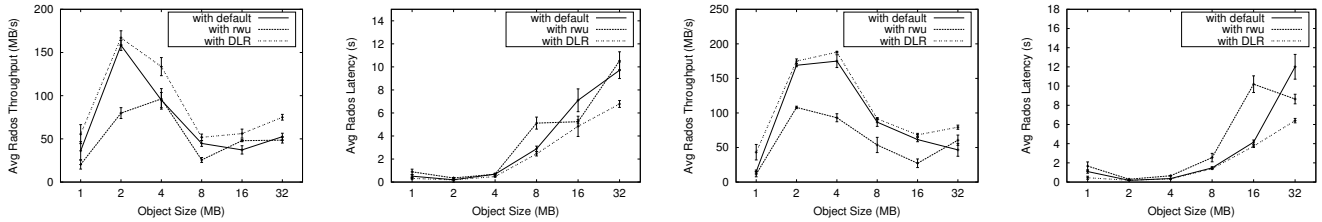
(a) Rados Throughput (b) Rados Latency (c) Change in the Primary-affinity values in the OSDs using DLR

Figure 8: Comparison of Normalized Rados Performance among default approach, reweight-by-utilization and DLR in the case of Performance Interference



(a) Rados Seq Rd Avg Throughput (b) Rados Seq Rd Avg Latency (c) Rados Random Rd Avg Latency

Figure 9: Comparison of Average Rados Performance among default approach, reweight-by-utilization and DLR during different time interval in the case of Performance Interference



(a) Rados Sequential Read Throughput in Hardware Heterogeneity (b) Rados Sequential Read Latency in Hardware Heterogeneity (c) Rados Sequential Read Throughput in Performance Interference (d) Rados Sequential Read Latency in Performance Interference

Figure 10: Comparison of Average Rados Sequential Read Performance among default approach, reweight-by-utilization and DLR over different object sizes in the case of Hardware Heterogeneity and Performance Interference

improvement by DLR. We also observe that OSD reweight-by-utilization shows much worse performance than the default case, since it incurs frequent data movement overheads in the face of dynamically changing performance hotspots. Figure 8c shows the change in the OSD primary-affinity of the OSDs due to DLR. In this case, the primary-affinity of the OSDs suffering from background interference decreases in steps and then increases at the end of interference. Figure 9 compares the average Rados performance among the default approach, reweight-by-utilization and DLR.

E. Evaluation with Various Object Sizes

Figure 10 compares Rados sequential read throughput, and latency achieved by the default approach, reweight-by-utilization and DLR over different object sizes in the case of hardware heterogeneity and performance interference. Similar to [10], we observe that the throughput was low for smaller object sizes, which is a common scenario for storage systems. The bandwidth reaches the network limit for medium sized objects, and when the object size is larger than a saturation point, throughput drops and the

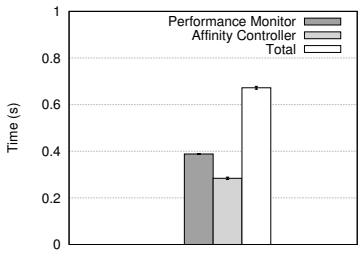


Figure 11: Performance Overhead Breakdown of DLR

latency increases drastically. In all of the scenarios, DLR outperforms the default Ceph and reweight-by-utilization approach. We got similar performance improvements when we experimented with other Rados benchmarks, and with various thread counts with fixed object size.

F. Overhead Analysis of DLR

Figure 11 shows the runtime of the Performance Monitor, and Affinity Controller of DLR in our testbed. We observe that the total time taken by the components is minimal, and significantly less than the control interval of 20 seconds. Since the Performance Monitor can collect the %iowait values of the OSDs in parallel, the overhead of performance monitoring will be negligible even for large clusters. The Affinity Controller also has minimal overhead since changing primary affinity only requires an update on the OSD map and unlike the OSD reweight-by-utilization technique, it does not involve data movement among the OSDs.

VI. Related Work

There are several studies focusing on mitigating the impact of hardware heterogeneity, and performance interference in the Cloud environment. In [6, 8] heterogeneity aware workload placement strategies were proposed to greedily place tenant jobs onto good performing VMs. Harmony [22], is a heterogeneity-aware framework that dynamically adjusts the number of machines to achieve a balance between energy savings and scheduling delay, while considering the reconfiguration cost. Lee et al. [12] proposed a system architecture to allocate resource in a cost-effective manner, and discussed a scheduling scheme that provides good performance and fairness simultaneously in heterogeneous cluster, by adopting progress share as a share metric. Paragon [7] proposes an online and scalable data center scheduler that is heterogeneity and interference-aware. It uses collaborative filtering techniques to classify incoming workload and then greedily schedules the applications minimizing interference and maximizing server utilization. Bu et al. [5] also considered data locality in their interference aware task scheduler. In contrast to these works, we focus mitigating the performance hotspots caused by hardware heterogeneity and performance interference in the Cloud storage clusters.

Several other studies have proposed different solutions on improving performance by balancing storage loads in a cloud or data center, mostly in the context of achieving fault-tolerance through data replication, e.g. Hadoop Distributed File System (HDFS), Google File System (GFS) [9], DepSky storage system [4], etc. In the context of Microsoft Azure and Amazon S3, CosTLO [21] presented the efficacy of using request duplication in coping with performance variability. In contrast to these works, our approach does not rely on redundant requests and is complementary to the above.

In a recent effort, Suresh et al. [16] propose to reduce latencies in cloud data store by adaptively selecting one out of multiple replica servers to serve a request based on a continuous stream of in-band feedback about a server's load. However, such approach is only suitable for low-latency data stores (e.g key-value store) where the service times of individual requests are small enough so that sufficient feedback can be collected to accurately rank the replica servers in the face of performance fluctuations, and changing system dynamics. However, large-scale cloud storage systems such as object-based storage have to deal with a wide range of data object sizes, and correspondingly varying service times. In this paper, we propose to dynamically adjust the load serving ratio of storage servers based on the system-level performance feedback from the storage cluster. We leverage the fact that system-level performance metrics such %iowait is highly correlated with the performance of storage nodes, and at the same time such metrics can be collected quickly irrespective of individual request service times.

In [17], Wang et al. presented file and block I/O performance and scalability evaluation of Ceph for scientific high-performance computing (HPC) environments. Through systematic experiments and tuning efforts, they observed that Ceph can perform close to 70% of raw hardware bandwidth at object level and about 62% of at file system level. Gudu et al. [10] presented a multidimensional scalability evaluation of Ceph with the aim to achieve better understanding of how each dimension (number of OSDs, number of clients, object size) affects performance. In contrast to these works, this paper focuses on improving the performance of a Ceph cluster in the presence of performance hotspots through dynamic load redistribution. To the best of our knowledge, this is one of the first works to address the issue of performance hotspots in a Ceph cluster.

VII. Conclusions And Future Work

In this paper, we presented a dynamic load redistribution technique to tame the performance hotspots caused by hardware heterogeneity and performance interference in cloud storage. The proposed technique periodically adjusts the load serving ratios of storage nodes according to a dynamic affinity control algorithm. We implemented the proposed technique in Ceph, an open source software-defined storage system, and evaluated its performance on NSFCloud's

Chameleon testbed using Ceph's Rados benchmark. Experimental results show that our approach improves the average throughput and latency of Ceph storage by up to 65%, and 41% respectively compared to the default case. Compared to Ceph's in-built load balancing technique, DLR improves the throughput by 98%, and latency by 96%. In future, we will work on incorporating dynamic tuning of the thresholds and will evaluate our technique at a larger scale using high-level application benchmarks, e.g Cosbench [23], etc.

ACKNOWLEDGMENT

Results presented in this paper were obtained using the Chameleon testbed supported by the National Science Foundation.

REFERENCES

[1] Ceph Control Commands. "http://docs.ceph.com/docs/master/rados/operations/control/".

[2] fio(1). "https://linux.die.net/man/1/fio/".

[3] Openstack swift. http://docs.openstack.org/developer/swift.

[4] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa. Depsky: dependable and secure storage in a cloud-of-clouds. *ACM Transactions on Storage (TOS)*, 9(4):12, 2013.

[5] X. Bu, J. Rao, and C. Z. Xu. Interference and locality-aware task scheduling for mapreduce applications in virtual clusters. In *Proc. of the 22nd ACM International Symposium on High-performance Parallel and Distributed Computing (HPDC)*, 2013.

[6] J. Dejun, G. Pierre, and C.-H. Chi. Resource provisioning of web applications in heterogeneous clouds. In *Proc. of the 2nd USENIX conference on Web application development*, 2011.

[7] C. Delimitrou and C. Kozyrakis. Paragon: Qos-aware scheduling for heterogeneous datacenters. In *ACM SIGPLAN Notices*, volume 48, pages 77–88, 2013.

[8] B. Farley, A. Juels, V. Varadarajan, T. Ristenpart, K. D. Bowers, and M. M. Swift. More for your money: exploiting performance heterogeneity in public clouds. In *Proc. of the 3rd ACM Symposium on Cloud Computing*, 2012.

[9] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. In *ACM SIGOPS operating systems review*, volume 37, pages 29–43, 2003.

[10] D. Gudu, M. Hardt, and A. Streit. Evaluating the performance and scalability of the ceph distributed storage system. In *Proc. of the IEEE International Conference on Big Data*, 2014.

[11] A. Gupta, O. Sarood, L. V. Kale, and D. Milojicic. Improving hpc application performance in cloud through dynamic load balancing. In *Proc. of the 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2013.

[12] G. Lee and R. H. Katz. Heterogeneity-aware resource allocation and scheduling in the cloud. In *HotCloud*, 2011.

[13] G. Liu, H. Shen, and H. Wang. Deadline guaranteed service for multi-tenant cloud storage. *IEEE Transactions on Parallel and Distributed Systems*, 27(10):2851–2865, 2016.

[14] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the 3rd ACM Symposium on Cloud Computing*, 2012.

[15] B. Sharma, V. Chudnovsky, J. L. Hellerstein, R. Rifaat, and C. R. Das. Modeling and synthesizing task placement constraints in google compute clusters. In *Proc. of the 2nd ACM Symposium on Cloud Computing*, 2011.

[16] L. Suresh, M. Canini, S. Schmid, and A. Feldmann. C3: Cutting tail latency in cloud data stores via adaptive replica selection. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, 2015.

[17] F. Wang, M. Nelson, S. Oral, S. Atchley, S. Weil, B. W. Settlemyer, B. Caldwell, and J. Hill. Performance and scalability evaluation of the ceph parallel file system. In *Proc. of the 8th ACM Parallel Data Storage Workshop*, 2013.

[18] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn. Ceph: A scalable, high-performance distributed file system. In *Proc. of the 7th USENIX symposium on Operating systems design and implementation (OSDI)*, 2006.

[19] S. A. Weil, S. A. Brandt, E. L. Miller, and C. Maltzahn. Crush: Controlled, scalable, decentralized placement of replicated data. In *Proc. of the ACM/IEEE conference on Supercomputing*, 2006.

[20] S. A. Weil, A. W. Leung, S. A. Brandt, and C. Maltzahn. Rados: a scalable, reliable storage service for petabyte-scale storage clusters. In *Proc. of the 2nd ACM international workshop on Petascale data storage*, 2007.

[21] Z. Wu, C. Yu, and H. V. Madhyastha. Costlo: Cost-effective redundancy for lower latency variance on cloud storage services. In *Proc. of USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, 2015.

[22] Q. Zhang, M. F. Zhani, R. Boutaba, and J. L. Hellerstein. Harmony: Dynamic heterogeneity-aware resource provisioning in the cloud. In *Proc. of the 33rd IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2013.

[23] Q. Zheng, H. Chen, Y. Wang, J. Duan, and Z. Huang. Cosbench: A benchmark tool for cloud object storage services. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 998–999. IEEE, 2012.