OStoreBench: Benchmarking Distributed Object Storage Systems Using Real-world Application Scenarios

Guoxin Kang[1], Defei Kong[2], Lei Wang[1], and Jianfeng Zhan[1]

[1] State Key Laboratory of Computer Architecture, Institute of Computing Technology Chinese Academy of Sciences
{kangguoxin, wanglei, zhanjianfeng}@ict.ac.cn
[2] ByteDance
{kongdefei}@bytedance.com

Abstract. The scalability of the traditional file storage systems is limited by their hierarchical organization and meta-data management. Distributed Object Storage Systems are widely used because they not only keep the advantages of traditional file storage systems, e.g., data sharing, but also alleviate the scalability issue that is benefited from the flat namespaces and integrating the meta-data in the object. However, evaluating and comparing distributed object storage systems remains a great challenges. The existing benchmarks for DOSS provide simple read/write operations without considering the complex workload characteristics in terms of request arrival patterns and distribution of request sizes. In this paper, we present a Object Storage Benchmark suite, named OStoreBench, which characterize critical paths of three real-world application scenarios, including online service, big data analysis and file backup. We evaluate three state-of-the-practice object storage systems with OStoreBench, including Ceph, Openstack Swift, and Haystack. The benchmark suite is publicly available from `https://github.com/EVERYGO111/OStoreBench`.

Keywords: Object Storage · Benchmark · Real-world application scenarios · QoS.

1 Introduction

Object storage systems [1] alleviate the scalability issue that file storage systems [2] face. Specifically, object storage systems adopt a flat namespace organization [3] and an expandable metadata functionality [4], which enables them to deal with massive unstructured data. Object storage systems have been applied in more and more scenarios and become an important service of major cloud service providers, such as Amazon S3 [5], Alibaba cloud Object Storage Service (OSS) [6], and Qiniu cloud storage [7].

To promote the development of distributed object storage systems (in short, DOSS), it is necessary to design and implement a benchmark suite for evaluating

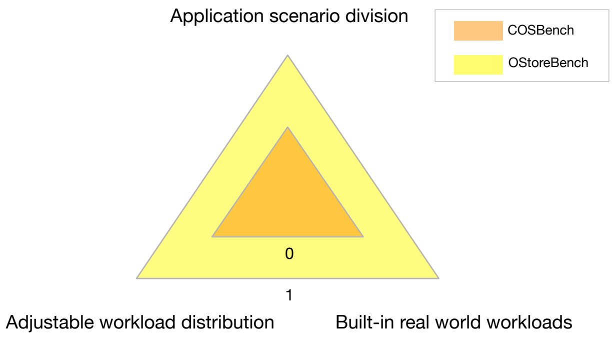


Fig. 1. Why we need OStoreBench? “0” means do not include the feature on the corner,“1” is the opposite.

state-of-the-art and state-of-the-practise DOSS. However, there are two main challenges.

First, no standard interfaces are established among various DOSS, which makes the comparison between different DOSS more difficult. Existing benchmarks are principally designed for file storage systems and are incompatible with DOSS because the former’s POSIX-based workloads include operations on meta-data directories while the workloads of the latter do not have.

Second, COSBench [8], proposed by Intel, is widely used to evaluate DOSS, but it just provides simple read/write operations without considering the complex workload in terms of request arrival patterns and distribution of request sizes. So the benchmarks like COSBench can’t fully represent essential characteristics of real-world workloads, and the evaluation results can not reflect the performances of the storage systems, that is why we need a new object storage benchmark suite shown in Figure 1. As there are numerous application scenarios, it is non-trivial to characterize typical workloads. We propose a benchmark suite for distributed object storage systems, named OStoreBench, which simulates the critical paths [9] of object storage services from query delivering to response receiving. Instead of only focusing on partial stages, e.g., network transferring or storage performance, our benchmarks abstracts the essential tasks of of cloud object storage services [9], which comprehensively discloses the potential performance bottlenecks, and hence they are of great significance for cloud service quality.

Three typical workloads among real-world object storage scenarios are chosen, which are online service workloads, big data analysis workloads and file backup workloads. These workloads are built into OStoreBench and characterized by request distribution, request size and request type.

To suit for incompatible interfaces of different DOSS, we wrap different interfaces of DOSS with the adapter pattern. Meanwhile, the workloads could be customized by user-specified configurations, and thus users can evaluate DOSS in the application scenarios they address.

We use OStoreBench to evaluate three state-of-the-practise object storage systems: Ceph [10], Openstack Swift [11] and Haystack [12] (using its open source implementation Seaweedfs). We find that Seaweedfs, which uses a small file merging strategy, outperforms the rest two systems in terms of both latency and throughput. Unfortunately, the performances of all these three object storage systems fluctuate sharply, which may lower the customer satisfaction and productivity. To address this issue, we first analyze such phenomenon from the perspectives of design principles and system architectures and find that the root cause is the imbalanced distribution of workloads among data nodes.

Our contributions are summarized as follows:

1. We implement OStoreBench, a scenario benchmark suite for distributed object storage systems. OStoreBench wraps different interfaces of DOSS with the adapter pattern and provides users with scenario benchmarks that characterize critical paths of object storage services. Through using the real workloads, the evaluation results of OStoreBench could represent the performances of DOSS in real-world applications.
2. We evaluate three state-of-the-practice object storage systems with OStoreBench and find that the performances of all these three systems fluctuate sharply.
3. Through extensive experiments, we find that the root cause of the performance fluctuation of the three state-of-the-practise system is the unbalanced distribution of requests among data nodes.

2 Related work

In this section, we introduce the related work about distributed object storage systems and benchmarks for them.

2.1 Mainstream distributed object storage system

The mainstream DOSS includes general distributed storage systems like Ceph [10], Lustre [13], and dedicated storage systems optimized for specific scenarios, such as Haystack [12] and Ambry [14]. The main advantages of DOSS are as follows:

Scalability Not like traditional file storage systems, of which the lookup operations become a bottleneck when dealing with massive files due to the hierarchical directory structure, DOSS adopt flat address spaces, which makes it easier to locate and retrieve the data across regions. Moreover, DOSS also provide scalable metadata management. Openstack Swift adopts consistency hash [15] and Ceph uses CRUSH which obtains locations of files by calculation [16]. By simply adding nodes, Ceph can scale to petabytes and beyond.

Table 1. Benchmarks for distributed storage system

Benchmark	**Configurable request distribution**	**Built-in real-world workloads**	**Target system**	
			File-oriented system	***Object-oriented storage system***
Fio			✓	
Filebench	✓		✓	
Iozone			✓	
Postmark	✓	✓	✓	
COSBench	✓			✓
OStoreBench	✓	✓		✓

Application-friendly Most distributed file systems, which use POSIX protocol, provide interfaces such as open, close, read, write, and lseek. However, the interfaces provided by object storage systems are usually RESTful-style APIs based on HTTP protocol. Specifically, PUT, GET and DELETE of HTTP requests are used to upload, download and delete files respectively. Such RESTful-style interfaces are more friendly to the web and mobile application developers.

High availability Traditional file systems often use RAID to backup data. DOSS adopt an object-level multiple replication strategy that is much simpler than block-level redundancy in traditional file systems.

2.2 Benchmarks for distributed storage systems

We list existing benchmarks for storage systems from the perspectives of IO R/W supports, workloads, and target systems in Table I. File storage systems adopt portable operating system interface (POSIX) [17], thus, the benchmarks for file storage systems are designed with POSIX semantics. Flexible IO Tester (fio) [18] is mainly used for block devices or file systems while it doesn't consider the IO access pattern. IOZone [19] is a file system benchmark similar to fio. Filebench [20] can simulate the IO loads of the application layer according to user configurations and currently only supports POSIX based file systems. Unlike Filebench, Postmark [21] provides real traces of workloads such as email services and online news services. However, traditional POSIX-based benchmarks are incompatible with distributed object storage systems, because the POSIX-based workloads include operations on metadata directories while the workloads of object storage systems don't. Industry primarily uses Cosbench [8] proposed by Intel to evaluate DOSS. It can evaluate Amazon S3, Ceph and Openstack Swift but only provides simple read and write operations, so it can't describe the access pattern of real workloads of the object storage systems. Besides, YCSB (Yahoo! Cloud Serving Benchmark) [22] and its extented versions [23] [24] are designed for nosql databases.

2.3 Workloads behavior and storage access patterns

In order to guarantee the response time, the online service workloads are usually allocated with the resources(CPU, Memory, and Disk) required by the peak load. But in fact the resource usage of the online service workloads are very low [25]. And big data analysis workloads are mostly throughput-oriented [3]. Therefore, our app-level workloads are not only of great significance for co-located workload research [25] [26], but also for real-time data analysis research directly in the object storage system [27] [28]. Grant's backup workloads characterization study [29] focuses on comparison the different characteristics between backup and primary storage systems. The size and access pattern of Store and Retrieve data streams received by Dropbox has been explored in this study [30]. Glauber [31] analyzes the characteristics of the workload from the perspective of the Dropbox client, and models the behavior of the client.

3 Benchmark building

In this section, we first list the requirements for building the benchmark of DOSS, then introduce how we design and implement the benchmark.

3.1 Requirements

Unlike the benchmarks for traditional file storage systems, there are three different requirements when building a general benchmark for DOSS:

Scalability As distributed object storage systems are applied on large-scale storage clusters, the evaluation tool should be scalable enough to generate large-scale workloads, such that the bottlenecks of DOSS could be recognised.

Compatibility Unlike file systems which have a standard POSIX interface, there is no unified interface standard for object storage systems. To compare the performances between different object storage systems, the benchmark should be compatible with different interfaces.

Typical workloads The workloads of the benchmark for traditional file storage system contain operations on metadata directories, so these workloads are incompatible with object storage systems. To evaluate the performance of DOSS precisely, the benchmark should provide typical open-loop workloads which are abstracted from real-world application scenarios of DOSS.

3.2 The design of OStoreBench

We propose a methodology for constructing the DOSS benchmark which consists of 5 steps: (1) Investigate the application scenarios and future development

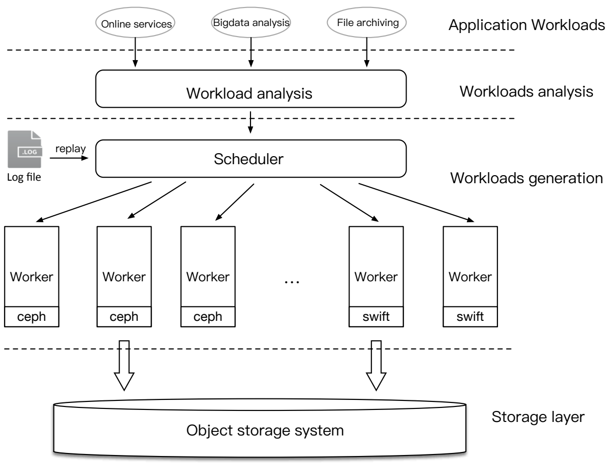


Fig. 2. OStoreBench design.

trends of the object storage systems, (2) Classify the application scenarios according to the characteristics of the application workloads, (3) Analyze the workloads of different application scenarios and build the load model, (4) Simulate the workloads of object storage systems based on the established load model, (5) Build object storage system evaluation tools based on the simulations and different software stacks.

Under the guidance of our proposed methodology, we design and implement OStoreBench. Figure 2 shows the structure of OStoreBench which consists of four layers: application workloads, workloads modeling, workloads generation and storage systems. In the first two layers, we classify the workloads based on the investigation of cloud services and model the typical workloads. In the third layer, we generate the workloads and deliver them to the storage systems. OStoreBench uses the Master/Slave architecture to ensure the scalability. The scheduler (master) generates specific loads according to the user configurations and allocates them to the workers (slave), then the workers deliver requests to DOSS. The number of workers is specified by the user and the workers are created by the scheduler. If the loads become heavier, the scheduler will create workers on the new physical nodes. Besides, we leverage coroutine technology in Golang [32] to make full use of the multi-core resources. We also adopt the adapter pattern to make OStoreBench compatible with different interfaces of DOSS.

The workloads of DOSS have different features in different scenarios. Through our investigation, we find that the online service (such as Google) and the big data analysis (such as Hadoop) are two kinds of the most important applications in the computing cloud, the file backup (such as Dropbox) is one of the most important applications in the storage cloud. Object storage is suitable for applications that require scale and flexibility, and also for importing existing data stores for analytics or backup [33]. Meanwhile, data analysis and backup are two important application scenarios for IBM object storage. So three typical real workloads including online service workloads, big data analysis workloads and file backup workloads are built into OStoreBench. The features of the three typical workloads are introduced as follows.

Online service workloads Online service workloads are mainly abstracted from web search engines and online database in ACloud [34], which require real-time access to objects or files, thus the corresponding workloads are sensitive to latency and fast response time of DOSS is required. The arrival time of requests follows logarithmic normal distribution rather than Poisson distribution [35]. The size of read requests subjects to exponential Gaussian distribution and 98% of the sizes of those files are less than 400KB [34]. The size of write requests subjects to Zipf distribution and 93% of the sizes of those files are less than 400KB. These observations are similar to [36]. There are more read requests than write requests.

Big data analysis workloads Big data analysis workloads are abstracted from the Hadoop clusters at Yahoo!, which dedicates to supporting a mix of data-intensive MapReduce jobs like processing advertisement targeting information [37]. Big data analysis applications aim to analyze the data stored in object storage systems and then store the results in the storage systems. The analyzing procedure usually consists of sequential phases like data preprocessing, statistics and mining operations, with the output of current phase is taken as the input of the next phase. Temporary files generated in the intermediate phases will be deleted and only the results of the last phase will be stored. The arrival time of requests is random. The size of 20% files are zero and 80% of the remaining files are more than 10G in size [37]. The read, write and delete operations are frequent and periodic.

File backup workloads File backup workloads are abstracted from Dropbox system [30]. A typical file backup scenario is the cloud disks such as Dropbox, where users back up files through the client and rarely download files from it. The client periodically detects file changes and continuously uploads new files to the cloud storage service [31]. The size of 40% − 80% requests is less than 100KB. There are more read requests than write requests.

3.3 The implementation of OStoreBench

The workloads generation layer in Figure 1 mainly consists of two parts: the scheduler and the worker. The scheduler receives user configuration2 and replays log files, then allocates the workloads to workers. The worker is a key part of our workloads generation, which generates requests to DOSS based on the loads allocated by the scheduler. Figure 3 shows the key components of the worker which are introduced as follows.

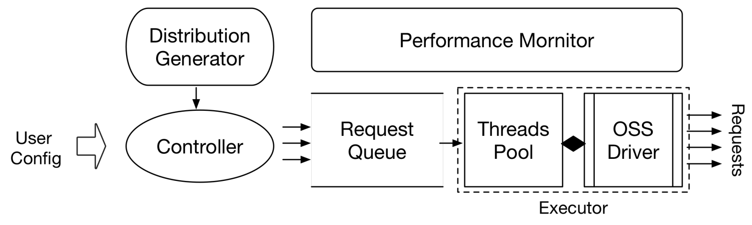


Fig. 3. OStoreBench Components.

Distribution generator The loads are characterized by request distribution, request size and request type. The distribution generator includes a variety of distribution models such as Poisson, Zipf, logarithmic normal.

Controller The controller is the most important module in the worker. Controller receives user configurations, generates requests and puts requests into request queues.

Request queue The request queue uses the "producer-consumer model". The producer (controller) places the requests into the request queue. The requests in request queues are delivered to the destination by the consumer (worker).

Executor The executor wraps requests in request queues into HTTP requests and sends them to the storage system. We implement drivers of different **O**bject **S**torage **S**ystem (OSS) using the adapter pattern. *Performance monitor* The performance monitor exhibits minimum, maximum, average, and 99^{th} percentile latencies as well as throughput.

4 Evaluation

In this section, we evaluate Ceph, Openstack Swift, and Seaweedfs with OStoreBench and analyze the evaluation results from the perspectives of design principles and system architectures. We also find the performance fluctuation problem of these three storage systems and present an approach to address it.

4.1 Approach and metrics

OStoreBench provides three typical open-loop workloads including online service workloads, big data analysis workloads and file backup workloads to Ceph, Openstack Swift, and Seaweedfs. These distributed object storage systems accept the remote procedure call sent by the client through the TCP/IP socket. We use two metrics to measure the performance of systems: (1) latency, including minimum, average, maximum, and 99^{th} percentile latency; (2) throughput.

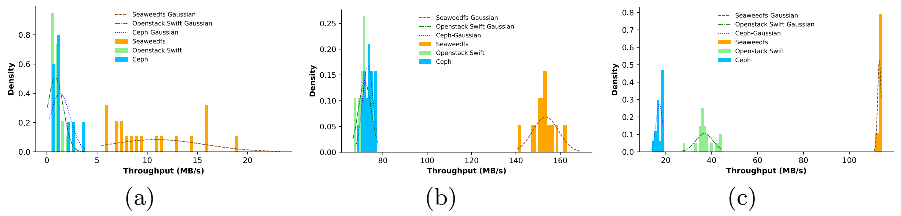


Fig. 4. (a) The variation of throughput under the online service workloads; (b) The variation of throughput under the big data analysis workloads; (c) The variation of throughput under the file archiving workloads.

4.2 Experimental setup

Our experimental environment is a cluster consisting of 4 storage nodes and a master/proxy node with three data replication. The server includes one Intel Xeon E5310 @ 1.6 GHz CPU with 4 core (8 hyper threads), a DRAM of 4 GB and a disk of 8TB. The machines run an Ubuntu 16.04 distribution with the Linux kernel version 4.4. All machines are configured with Intel 82573E 1GbE NICs. We use Seaweedfs version 1.87, OpenStack Swift version 3.0 and Ceph version 13.2.10. We remain the default configurations of these three storage systems. Though the performance of the three object storage systems mentioned above will increase or change in their future releases, our results of experiments are still meaningful because they demonstrate the value of OStoreBench in facilitating performance comparison and system optimization.

4.3 Experimental results

The evaluation for online service workloads The arrival time of requests in online service workloads satisfies the log-normal distribution. The request size satisfies Zipf distribution and is less than 64MB. Figure 4(a) shows the probability density function of the throughput under online service workloads. Seaweedfs achieves the highest average throughput which is 10.8MB per second. Ceph's average throughput is 1.3MB per second and Swift's average throughput is 0.9MB per second. The dashed line is a curve of Gaussian distribution fitting to the histogram. Figure 4(a) shows that the throughput of online service workloads fluctuates sharply. The latency of Seaweedfs is regarded as a baseline and the latency of the other systems is depicted based on their multiples of the baseline. The unit of latency is milliseconds. In this paper, such processing is used for all latency evaluations. Figure 5 shows the read and write latencies under the online service workloads. It is apparent that the average latency of Openstack Swift is more 30 times than Seaweedfs. Although Ceph is much better than Openstack, its latency is still more 20 times than Seaweedfs in the worst case. The latency distributions of Seaweedfs and Openstack Swift are relatively plain. But the latency of Ceph fluctuates sharply, and the read latency increases abruptly at some points.

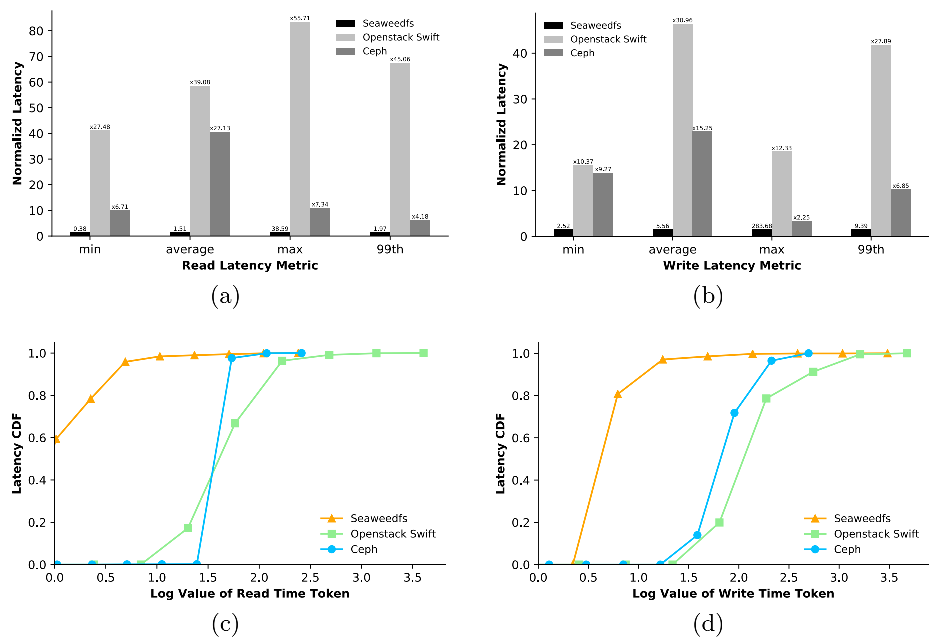


Fig. 5. Read and write latencies under the online service workloads. (a) Read request latency, (b) Write request latency, (c) Logarithmic read latency CDF, (d) Logarithmic write latency CDF.

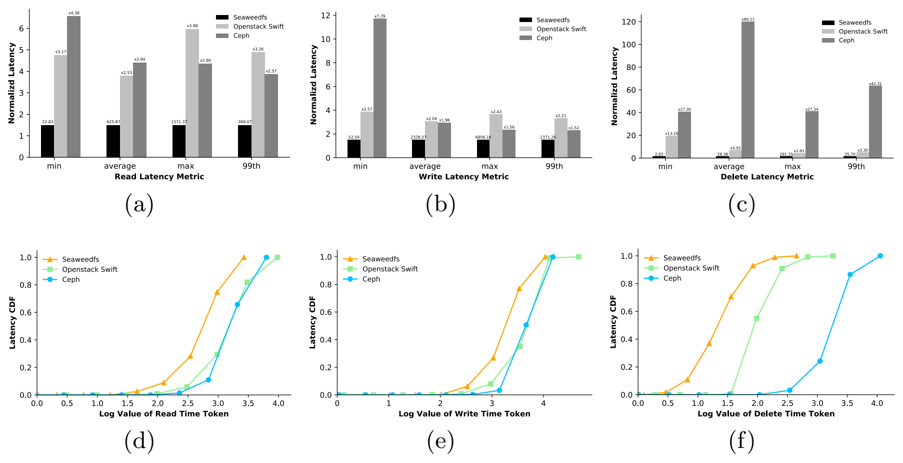
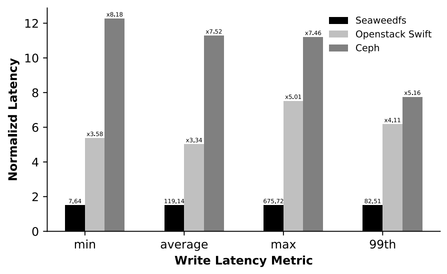
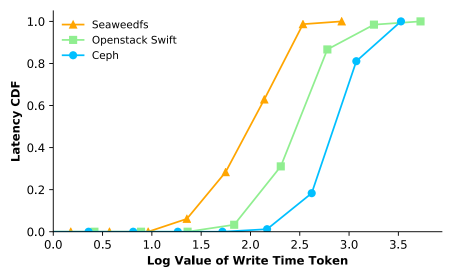


Fig. 6. Read, write and delete latencies under the big data analysis workloads. (a) Read latency; (b) Write latency; (c) Delete latency; (d) Logarithmic read latency CDF; (e) Logarithmic write latency CDF; (f) Logarithmic delete latency CDF.



(a)



(b)

Fig. 7. Write latency under the file backup workloads. (a) Write latency; (b) Logarithmic write latency CDF.

The evaluation of big data analysis workloads The OStoreBench's built-in big data analysis workloads simulate multi-stage loads including a lot of read, write and delete operations. Figure 4(b) shows the probability density function of the throughput under the big data analysis workloads. What stands out in Figure 4(b) is that the throughput of Seaweedfs is higher than others. And the average throughput of Seaweedfs reaches 153.4MB/s. Ceph and Swift have a similar distribution of throughput. Their average throughput is 73.2MB/s and 71.8MB/s respectively. However, the throughput fluctuations of these three systems are significant.

Figure 6 shows the read, write and delete latencies under the big data analysis workloads. As we can see, Seaweedfs achieves the lowest minimum, maximum, average and 99^{th} percentile latencies. The minimum and average read latency of Openstack Swift is lower than Ceph, but the maximum and 99^{th} percentile are higher than Ceph. Openstack Swift's minimum write latency is much smaller than Ceph, and the average, maximum and 99^{th} percentile are close to Ceph. It is obvious that Ceph and Swift have similar performance. Openstack Swift's delete latency is better than Ceph in minimum average, maximum and 99^{th} percentile. The most interesting aspect of Figure 6(d) is that there are two intersection of Ceph and Swift. For 20% of the read requests, Ceph's delay is lower than Swift. The same phenomenon also exists in Figure 6(e).

The evaluation of file backup workloads OStoreBench simulates the operations of the client and periodically writes files to the cloud storage system. In each period, 10 to 100 requests are generated randomly and the upload interval is 5 seconds. The size of generated requests is 64MB at most. Figure 4(c) shows the probability density function of the throughput under the file backup workloads. It can be seen that the throughput of Seaweedfs is significantly higher than those of the other two systems. On average, the throughput of Seaweedfs reaches 113MB/s, but only 17MB/s and 37MB/s for Ceph and Swift respectively. Figure 7 shows the write latency under the file backup workloads. The latency of Seaweedfs is significantly better than the other two systems. From the eval-

Table 2. Volumes on datanodes

Datanode	**Volume**
gd86	3,4,12,19
gd87	8,9,10,22
gd88	14,15,20,23
gd89	16,17,18,21
gd92	2,5,13,24
gd94	1,6,7,11

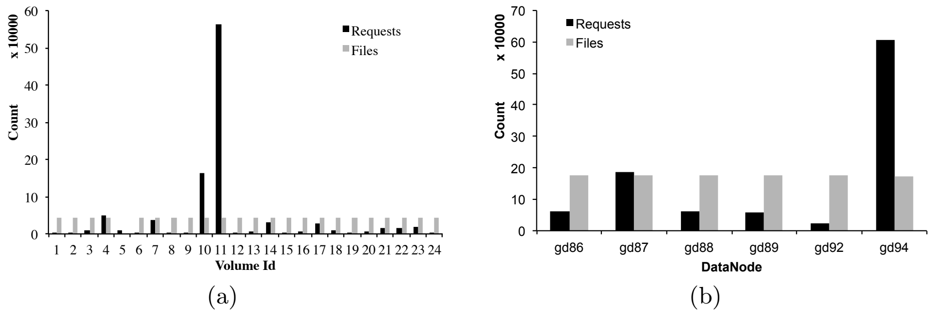


Fig. 8. The distribution of workloads among different data nodes. (a) Files and Requests received by each volume, (b) Files and Requests received by each data node.

uation results, we can see that under the file backup workloads Seaweedfs still has better performance than Ceph and Openstack Swift, and the performance of Ceph is slightly worse than Openstack Swift.

Performance fluctuation analysis From above experimental results, we find that the performance of all these three object storage systems fluctuates violently, so we further conduct comprehensive experiments to analyze the performance fluctuation problem. The experiments are conducted on a cluster consisting of five data nodes and one master node. Relationships between volumes and data nodes are displayed in Table 2. We evaluate Seaweedfs with the online service workloads and make a statistic of the received requests of each data node to analyze the performance fluctuation problem. The request size is set to be no more than 100KB and 2 million files are written to Seaweedfs at first. Then we send 1 million read requests to the cluster. Figure 8 shows the distribution of workloads among different data nodes. We can see that the numbers of requests received by different nodes are seriously unbalanced even though the static files are fairly well-distributed. So we conclude that the main cause of the performance fluctuation is the unbalanced distribution of loads among the data nodes which is the result of the existence of hotspots in data access. Overall, from the views of the design principles and system architectures, these evaluation results indicate:

1. The performance of Seaweedfs is the best in three typical scenarios compared to Ceph and Swift. Seaweedfs adopts a "small file merging" strategy so that the performance of Seaweedfs is most stable than the others. The authentication services of both Ceph and Openstack Swift use third-party services. Two times of network transmissions are required for each authentication. In a production environment, swift's authentication uses Keystone. Ceph's authentication is provided by radosgw [38]. Seaweedfs integrates authentication internally so that the requests latency is less.
2. Ceph's performance has a slight advantage compared to Swift. The performance of Swift is slightly worse than Ceph almost in all scenarios. This difference may be because Ceph is implemented in C++ and Swift is implemented in Python.
3. The performance of all DOSS fluctuates greatly. The source of performance fluctuations is that the load is unbalanced among nodes of the cluster. The essential reason for the imbalance of load is that there are hotspots in data access.

5 Conclusion

This paper presents a scenario benchmark suite(OStoreBench) for distributed object storage systems. OStoreBench provides users with scenario benchmarks that characterize critical paths of cloud object storage services. Through using the real workloads, the evaluation results of OStoreBench could represent the performances of DOSS in real-world applications. Using OStoreBench, We evaluate three representative object storage systems, i.e., Ceph, Openstack Swift and Seaweedfs, in various scenarios. We find that Seaweedfs performs better in all three scenarios and Oepnstack Swift has the worst performance. However, for all of these systems, the performance is unstable in every scenario. We conduct a more detailed evaluation of the object storage systems with the online service workloads and find that the distribution of loads among different data nodes is unbalanced even though the distribution of static files is uniform at the beginning.

6 Future work

This paper explores the evaluation and optimization of object storage systems. In the future, we will conduct the research from two aspects:

1. With the increment of application scenarios of cloud services, we plan to add more workloads in OStoreBench to comprehensively compare the performance between different DOSS. We will also conduct the experiments on a larger cluster.
2. We plan to predict the hotspots by machine learning algorithms such that the data migration could be performed before the nodes become overloaded.

References

1. M. Factor, K. Meth, D. Naor, O. Rodeh, and J. Satran, "Object storage: The future building block for storage systems," in *2005 IEEE International Symposium on Mass Storage Systems and Technology*, pp. 119–123, IEEE, 2005.
2. M. Satyanarayanan, "A survey of distributed file systems," *Annual Review of Computer Science*, vol. 4, no. 1, pp. 73–104, 1990.
3. H. Tang, S. Byna, B. Dong, J. Liu, and Q. Koziol, "Someta: Scalable object-centric metadata management for high performance computing," in *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 359–369, IEEE, 2017.
4. M. Mesnier, G. R. Ganger, and E. Riedel, "Object-based storage," *IEEE Communications Magazine*, vol. 41, no. 8, pp. 84–90, 2003.
5. Amazon, "Amazon s3." `http:/s3.amazonaws.com`,.
6. Alibaba, "Alibaba oss." `https://www.alibabacloud.com/zh/product/oss`.
7. Qiniu, "Qiniu cloud storage." `https://www.qiniu.com`.
8. Q. Zheng, H. Chen, Y. Wang, J. Zhang, and J. Duan, "COSBench," in *the ACM/SPEC international conference*, (New York, New York, USA), pp. 199–210, ACM Press, 2013.
9. W. Gao, F. Tang, J. Zhan, X. Wen, L. Wang, Z. Cao, C. Lan, C. Luo, and Z. Jiang, "Aibench: Scenario-distilling ai benchmarking," *arXiv preprint arXiv:2005.03459*, 2020.
10. S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proceedings of the 7th symposium on Operating systems design and implementation*, pp. 307–320, USENIX Association, 2006.
11. Openstack, "Openstack swift." `https://www.openstack.org`.
12. D. Beaver, S. Kumar, H. C. Li, J. Sobel, P. Vajgel, *et al.*, "Finding a needle in haystack: Facebook's photo storage," in *OSDI*, vol. 10, pp. 1–8, 2010.
13. P. Schwan *et al.*, "Lustre: Building a file system for 1000-node clusters," in *Proceedings of the 2003 Linux symposium*, vol. 2003, pp. 380–386, 2003.
14. S. A. Noghabi, S. Subramanian, P. Narayanan, S. Narayanan, G. Holla, M. Zadeh, T. Li, I. Gupta, and R. H. Campbell, "Ambry: Linkedin's scalable geo-distributed object store," in *Proceedings of the 2016 International Conference on Management of Data*, pp. 253–265, ACM, 2016.
15. Swift, "Consistent hashing." `https://docs.openstack.org/swift/latest/ring_background.html`.
16. S. A. Weil, S. A. Brandt, E. L. Miller, and C. Maltzahn, "Crush: Controlled, scalable, decentralized placement of replicated data," in *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, p. 122, ACM, 2006.
17. Wikipedia, "Posix." `https://zh.wikipedia.org/wiki/POSIX`.
18. Linux, "Flexible i/o tester." `https://linux.die.net/man/1/fio`.
19. iozone, "Iozone filesystem benchmark." `http://www.iozone.org`.
20. V. Tarasov, E. Zadok, and S. Shepler, "Filebench: A flexible framework for file system benchmarking," *login: The USENIX Magazine*, vol. 41, no. 1, 2016.
21. J. Katcher, "Postmark: A new file system benchmark," tech. rep., Technical Report TR3022, Network Appliance, 1997.
22. B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *Proceedings of the 1st ACM symposium on Cloud computing*, pp. 143–154, ACM, 2010.

23. S. Patil, M. Polte, K. Ren, W. Tantisiriroj, L. Xiao, J. López, G. Gibson, A. Fuchs, and B. Rinaldi, "Ycsb++: benchmarking and performance debugging advanced features in scalable table stores," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*, p. 9, ACM, 2011.
24. A. Dey, A. Fekete, R. Nambiar, and U. Röhm, "Ycsb+ t: Benchmarking web-scale transactional databases," in *2014 IEEE 30th International Conference on Data Engineering Workshops*, pp. 223–230, IEEE, 2014.
25. C. Iorgulescu, R. Azimi, Y. Kwon, S. Elnikety, M. Syamala, V. Narasayya, H. Herodotou, P. Tomita, A. Chen, J. Zhang, *et al.*, "Perfiso: Performance isolation for commercial latency-sensitive services," in *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, pp. 519–532, 2018.
26. A. Ousterhout, J. Fried, J. Behrens, A. Belay, and H. Balakrishnan, "Shenango: Achieving high {CPU} efficiency for latency-sensitive datacenter workloads," in *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*, pp. 361–378, 2019.
27. Alibaba, "Hadoop-aliyun module: Integration with aliyun web services," 2017.
28. A. Athena, "Amazon athena." `https://aws.amazon.com/cn/athena`.
29. G. Wallace, F. Douglis, H. Qian, P. Shilane, S. Smaldone, M. Chamness, and W. Hsu, "Characteristics of backup workloads in production systems.," in *FAST*, vol. 12, pp. 4–4, 2012.
30. I. Drago, M. Mellia, M. M. Munafo, A. Sperotto, R. Sadre, and A. Pras, "Inside dropbox: Understanding personal cloud storage services," in *Proceedings of the 2012 Internet Measurement Conference*, IMC '12, (New York, NY, USA), pp. 481–494, ACM, 2012.
31. G. Gonçalves, I. Drago, A. P. C. Da Silva, A. B. Vieira, and J. M. Almeida, "Modeling the dropbox client behavior," in *Communications (ICC), 2014 IEEE International Conference on*, pp. 1332–1337, IEEE, 2014.
32. Golang, "Golang." `https://golang.org`.
33. S. Obrutsky, "Cloud storage: Advantages, disadvantages and enterprise solutions for business," in *Conference: EIT New Zealand*, 2016.
34. Z. Ren, W. Shi, and J. Wan, "Towards realistic benchmarking for cloud file systems: Early experiences," in *Workload Characterization (IISWC), 2014 IEEE International Symposium on*, pp. 88–98, IEEE, 2014.
35. S. M. Ross *et al.*, "Stochastic processes (vol. 2)," 1996.
36. L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distributions: Evidence and implications," in *INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 1, pp. 126–134, IEEE, 1999.
37. C. L. Abad, N. Roberts, Y. Lu, and R. H. Campbell, "A storage-centric analysis of mapreduce workloads: File popularity, temporal locality and arrival patterns," in *Workload Characterization (IISWC), 2012 IEEE International Symposium on*, pp. 100–109, IEEE, 2012.
38. Ceph, "Ceph rados gateway." `http://docs.ceph.com/docs/master/radosgw/`.