Understanding Write Behaviors of Storage Backends in Ceph Object Store

Dong-Yun Lee*, Kisik Jeong*, Sang-Hoon Han*, Jin-Soo Kim*, Joo-Young Hwang+, and Sangyeun Cho+

Computer Systems Laboratory
**Sungkyunkwan University, South Korea*

Memory Business
+*Samsung Electronics Co., Ltd., South Korea*

{dongyun.lee, kisik, shhan}@csl.skku.edu, jinsookim@skku.edu, {jooyoung.hwang, sangyeun.cho}@samsung.com

***Abstract*—Ceph is a scalable, reliable and high-performance storage solution that is widely used in the cloud computing environment. Internally, Ceph provides three different storage backends: FileStore, KStore and BlueStore. However, little effort has been devoted to identifying the differences in those storage backends and their implications on performance. In this paper, we carry out extensive analysis with a microbenchmark and a long-term workload to compare Ceph storage backends and understand their write behaviors by focusing on WAF (Write Amplification Factor). To accurately analyze WAF, we carefully classify write traffic into several categories for each storage backend.**

We find that writes are amplified by more than 13x, no matter which Ceph storage backend is used. In FileStore, the overhead of Ceph write-ahead journaling triples write traffic compared to the original data size. Also, FileStore has the journaling of journal problem, generating a relatively large amount of file system metadata and journal traffic. KStore suffers severe fluctuations in IOPS (I/O Operations Per Second) and WAF due to large compaction overheads. BlueStore shows the stable performance on both HDDs and SSDs in terms of IOPS, WAF and latency. Overall, FileStore performs the best among all storage backends on SSDs, while BlueStore is also highly promising with good average and tail latency even on HDDs.

I. Introduction

In the cloud computing era, a stable, consistent and high-performance block storage service is essential to run a large number of virtual machines. Ceph is a storage solution that meets all these demanding requirements and has attracted a spotlight in the last decade. Ceph is a scalable, highly reliable software-defined storage solution that provides multiple interfaces for object, block and file level storage [1]. Ceph aims at completely distributed storage without a single point of failure and high fault tolerance with no specific hardware support. Since Ceph provides strong consistency to clients, users can access objects, block devices and files without worrying about consistency. Moreover, because it has a scale-out structure, Ceph can improve its performance gradually by adding additional cluster nodes [2].

Internally, all storage services in Ceph are built upon the Ceph RADOS (Reliable Autonomic Distributed Object Store) layer [3], which manages fixed-size *objects* in a scalable, distributed and reliable manner. Ceph provides three different storage backends in the RADOS layer: FileStore, KStore and BlueStore. FileStore and KStore manage objects on top of traditional file systems and key-value stores (e.g., LevelDB and RocksDB), respectively. On the other hand, BlueStore is a new object store architecture that has been developed actively for the Ceph RADOS layer in recent years. BlueStore saves object data into the raw block device directly, while it manages their metadata on a small key-value store such as RocksDB. Currently, Ceph can be configured to use one of these storage backends freely.

Due to Ceph's popularity in the cloud computing environment, several research efforts have been made to find optimal Ceph configurations under a given Ceph cluster setting [4], [5] or to tune its performance for fast storage like SSD (Solid-State Drive) [6]. However, little attention has been paid to the differences in the storage backends available in Ceph and their implications on the overall performance. In this paper, we compare the write behaviors and performance of Ceph backends with a focus on WAF (Write Amplification Factor). The study on the WAF of various storage backends can be very enlightening to understand the storage access behaviors of Ceph for the following reasons. First, WAF has a major impact not only on the overall performance, but also on device lifetime when Ceph runs on SSDs. Second, the larger WAF, the more limited effective bandwidth given to the underlying storage device. In particular, HDD (Hard Disk Drive) exhibits very low IOPS (I/O Operations Per Second) compared to SSD and it is very important to use raw hardware bandwidth effectively. Finally, as in the previous research with SQLite, there might be issues such as *journaling of journal* [7] problem when implementing distributed storage services on top of a local file system.

We have used a microbenchmark and a long-term workload of 4KB random writes to measure write traffic of various Ceph storage backends on both HDDs and SSDs. Our results with the long-term workload indicate that Ceph amplifies the amount of write traffic by more than 13x under the replication factor of 3, regardless of the storage backend used. In FileStore, we find that write-ahead journaling with separate Ceph journal does not double, but rather *triples* write traffic

compared to the original data size. Also, the *journaling of journal* problem is severe, making file system metadata and journaling traffic as much as the original data size. In the case of KStore, the compaction process takes up almost all write traffic, resulting in poor tail latency and severe fluctuations in IOPS. Finally, BlueStore is free from the *journaling of journal* problem as it stores data directly in the storage device. However, RocksDB traffic to store metadata and object attributes still overwhelms data traffic over a factor of three in BlueStore.

The rest of the paper is organized as follows. Section II presents more detailed background on Ceph. In Section III, we introduce our measurement methodology and experimental configurations. In Section IV, we perform several microbenchmarks and discuss the basic write behaviors of Ceph backends. Section V evaluates various Ceph backends with the long-term workload on HDDs and SSDs. We discuss the related work in Section VI. Finally, Section VII concludes the paper.

II. Background

This section gives a brief overview of the Ceph architecture and its storage backends.

A. Ceph Architecture

Ceph provides multiple storage services at an object level (Ceph object storage), a block level (Ceph block storage) and a file level (Ceph file system). Internally, they are all based on one unified layer called RADOS (Reliable Autonomic Distributed Object Store). Ceph consists of several daemons running in the RADOS layer, each of which performs a specific task. The Ceph monitor (MON) daemon manages the cluster-wide node information called cluster map. The Ceph metadata (MDS) daemon is needed only for the file-level service to maintain the metadata of each file as is done in traditional distributed file systems. Finally, the Ceph object storage device (OSD) daemon is responsible for retrieving and storing objects by interacting with its local disks.

One important feature in the Ceph object and block storage is that clients can directly contact the OSD daemon that has a primary copy of the required data. In traditional distributed storage systems, clients have to make a request to a centralized server first to get metadata (e.g., data locations), which can be a performance bottleneck as well as a critical single point of failure. Ceph eliminates the need for the centralized server by placing data using a pseudo-random distribution algorithm called CRUSH [8].

B. Ceph RADOS Block Device (RBD)

Ceph RADOS block device, also known as RBD, provides a thin-provisioned block device to the clients. A block device represents a consecutive sequence of bytes and Ceph divides it into a set of objects of equal size. When a client modifies a region of the RBD, the corresponding objects are automatically stripped and replicated over the entire Ceph cluster system. The size of objects is set to 4MiB by default.

There are two types of RBD: *librbd* and *krbd*. *librbd* is a user-level library implementation which is widely used as a primary block storage for virtual machines in the cloud computing platforms such as OpenStack. *krbd* is implemented as a kernel module which exports device files directly to the kernel so that clients can mount them just like conventional disks. In this paper, we use the *krbd* module to investigate the performance of the Ceph RADOS block device without any interference from hypervisor or other virtual machines.

C. Ceph Storage Backends

The Ceph OSD daemon consists of many functional modules in order to support software-defined storage services. In the heart of the Ceph OSD daemon, there is a module called *ObjectStore* which is responsible for how objects are stored and managed. In particular, Ceph is designed to support multiple storage engines by registering them as different backends for *ObjectStore*. The stable Ceph Jewel LTS version currently supports three kinds of storage backends: FileStore, KStore and BlueStore. In the following subsections, we briefly present overall architecture and characteristics of each storage backend.

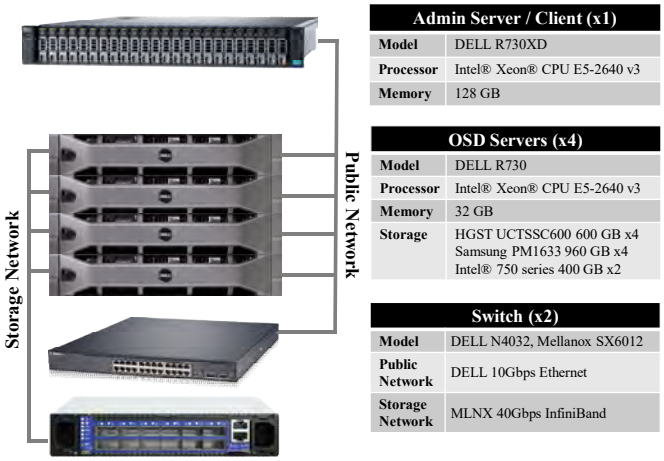
1) FileStore

In FileStore, each object is stored as a separate file in the underlying local file systems such as XFS, BTRFS and ZFS. Using FileStore, Ceph mandates to use an external Ceph journal for ensuring consistency. Since Ceph guarantees strong consistency among data copies, all write operations are treated as atomic transactions. Unfortunately, there is no POSIX API that provides the atomicity of compound write operations. Instead, FileStore first writes incoming transactions to its own journal disk in an append-only manner. After writing to the journal, worker threads in FileStore perform actual write operations to the file system with the `writev()` system call. In every a few seconds up to `filestore_max_sync_interval` (5 seconds by default), FileStore calls `syncfs()` to the disk and then drops the journal entries. In this way, FileStore provides strong consistency.

Having the external journal also brings a performance benefit as the write speed is improved due to the append-only logging mechanism. To enhance the Ceph performance further, many systems employ SSDs as the journal devices. Theoretically, any local file system can be used for FileStore, but due to some issues related to extended attributes (*xattr*), XFS is the only file system officially recommended by Ceph developers.

2) KStore

KStore is an experimental storage backend in the Ceph Jewel version. The basic idea behind KStore is to encapsulate everything from object data to their metadata as key-value pairs and put them into key-value stores since key-value stores are already highly optimized for storing and managing key-value pairs. Any key-value store can be used for KStore by interposing a simple translation layer between *ObjectStore* and the key-value store. The Ceph Jewel version currently supports LevelDB, RocksDB and KineticStorage for KStore. In this



Admin Server / Client (x1)	
Model	DELL R730XD
Processor	Intel® Xeon® CPU E5-2640 v3
Memory	128 GB

OSD Servers (x4)	
Model	DELL R730
Processor	Intel® Xeon® CPU E5-2640 v3
Memory	32 GB
Storage	HGST UCTSSC600 600 GB x4 Samsung PM1633 960 GB x4 Intel® 750 series 400 GB x2

Switch (x2)	
Model	DELL N4032, Mellanox SX6012
Public Network	DELL 10Gbps Ethernet
Storage Network	MLNX 40Gbps InfiniBand

Fig. 1: Experimental Ceph Testbed

paper, we have performed our experiments on LevelDB and RocksDB.

3) BlueStore

BlueStore is another experimental storage backend that is being actively developed by Ceph community. It is expected to be stable in the next upcoming release. The key idea of BlueStore is effective management of objects avoiding limitations in FileStore. One problem in FileStore is a double-write penalty caused by the external journaling. Instead, BlueStore saves object data in a raw block device directly, while managing their metadata with RocksDB. Since BlueStore bypasses the local file system layer, file system overheads such as *journaling of journal* can be avoided. Note that because a file system is still required to run the RocksDB, BlueStore internally has a tiny user-level file system named BlueFS. Moreover, Ceph usually deals with a bunch of objects and it often has to enumerate all objects in an ordered fashion for checking consistency and recovering. However, POSIX does not provide any efficient way to retrieve the objects from multiple directories. Another benefit of using RocksDB is that Ceph can simply retrieve and enumerate all objects stored in the system.

III. Measurement Methodology

A. Evaluation Environment

Figure 1 illustrates an organization of our experimental Ceph testbed. In our experiments, we use one administration server to run the Ceph monitor daemon (MON). The same server is also used as a client which generates I/O requests to the Ceph RBD. We use four storage servers for running the Ceph OSD daemons. There are two private networks in the system; one is a public network that connects all the servers with a 10Gbps Ethernet and the other is a storage network that connects four storage servers with a 40Gbps InfiniBand.

Each storage server is equipped with four 600GB HGST SAS HDDs, four 960GB Samsung PM1633 SAS SSDs and two 400GB Intel 750 NVMe SSDs. We conduct the measurement for the Ceph storage backends using SAS HDDs and SAS SSDs. In our configuration, a storage server runs four OSD daemons either on four HDDs or on four SSDs so that each OSD daemon works on a single storage device. Therefore, there are total 16 OSDs in our Ceph testbed. NVMe SSDs are used for external journaling in FileStore and WAL (Write-Ahead Logging) in BlueStore. All experiments are conducted on the Linux 4.4.43 kernel with the latest Ceph Jewel LTS version (v10.2.5).

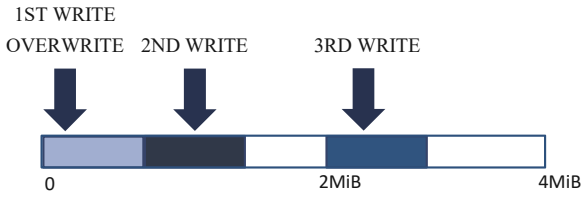


Fig. 2: Workload in Microbenchmark

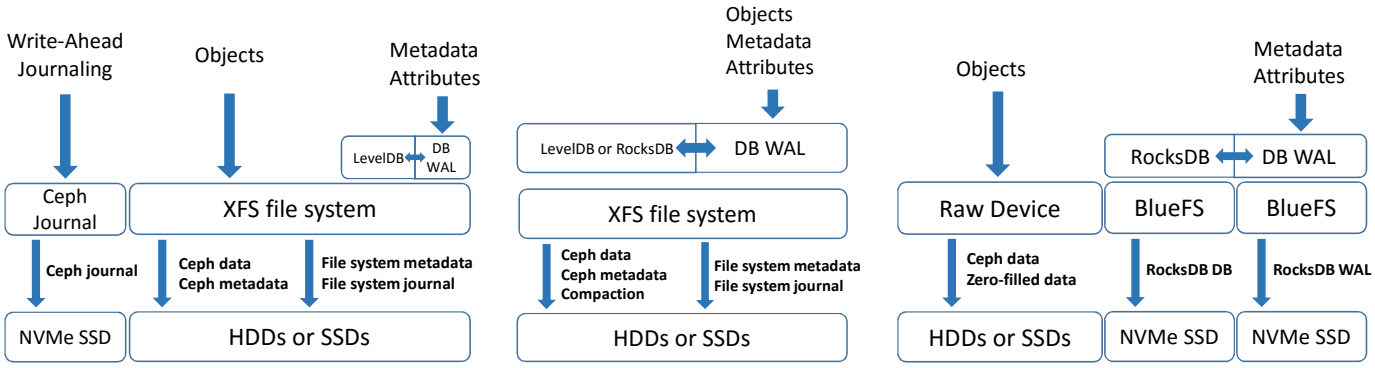
B. Microbenchmark

First, we analyze how WAF changes when we make a single write request to the RBD under different circumstances. Recall that the entire RBD space is divided into a set of 4MiB objects by default. When the replication factor is set to default value of three, total three copies of each object are distributed over the available OSDs.

Our microbenchmark is designed to measure the amount of write traffic in the following cases (cf. Figure 2): (1) when there is a write to an empty object (denoted as 1ST WRITE), (2) when there is a write next to the 1ST WRITE (denoted as 2ND WRITE), (3) when there is a write to the middle of the object leaving a hole between the 2ND WRITE and the current write (denoted as 3RD WRITE) and (4) when there is an overwrite to the location written by the 1ST WRITE (denoted as OVERWRITE). The microbenchmark repeats the same experiment by doubling the request size from 4KiB to 4MiB (i.e., 4KiB, 8KiB, 16KiB, ..., 4MiB). We expect that the amount of metadata traffic generated by Ceph varies in each case. To avoid any interference from the previous experiment, we reinstall Ceph every time for each request size. We use the `ftrace` tool in Linux to collect and classify trace results.

C. Long-term workload

To perform experiments under a long-term workload, we use the `fio` tool in Linux to generate 4KiB random writes to the Ceph RBD, periodically measuring IOPS and WAF from FileStore, KStore (with LevelDB and RocksDB) and BlueStore. For each storage backend, we classify all write requests into several categories and calculate WAF for each category. As mentioned before, the Ceph RBD is widely used to provide large, reliable and high-performance storage for virtual machines. In this VDI (Virtual Desktop Infrastructure) environment, it is well known that the patterns of write requests are mostly random, with their sizes ranging from 4KiB to 8KiB [9][10]. This is why we focus on 4KiB random writes in this paper.



(a) FileStore (b) KStore (c) BlueStore

Fig. 3: Internal Architectures of Ceph Storage Backends

Each long-term experiment is performed in the following order:

1) Install Ceph and create an empty 64GiB *krbd* partition.
2) Drop page cache, call sync and wait for 600 seconds to flush all the dirty data to the disk.
3) Perform 4KiB random writes with the queue depth of 128 (QD=128) using `fio` to the *krbd* partition until the total write amount reaches 90% of the capacity (i.e., 57.6GiB).

All experiments are conducted with 16 HDDs first and they are repeated on 16 SSDs later. According to our experiments, a single write thread with QD=128 is enough to saturate OSD servers on HDDs, but not on SSDs. Therefore, we perform the experiments on SSDs with two write threads (each with QD=128).

As the purpose of this paper is to analyze WAF, we keep track of the amount of written sectors for each disk. We also measure IOPS and latency distribution since they are traditionally important metrics in storage performance analysis. During the long-term experiment, we observe that the `ftrace` overhead affects our experimental results, degrading the overall IOPS by up to 5%. In order to eliminate this overhead, we did not use `ftrace` in the long-term experiment. Instead, we have modified the Linux kernel so that it collects the amount of written sectors for each category and then exports them via the `/proc` interface. At runtime, we periodically read those values from the `/proc` file system. The sampling period is set to 15 seconds for HDDs, but it is shortened to 3 seconds for SSDs as SSDs are much faster than HDDs. The detailed setting and write traffic classification scheme for each storage backend are described below. Unless otherwise specified, we use default configurations of Ceph. In all experiments, the replication factor is set to three.

1) FileStore

We create a single partition in each disk and mount it with the XFS file system. This partition is dedicated to a Ceph OSD daemon as the main data storage. Since FileStore needs an additional Ceph journal partition for its write-ahead journaling, we use two NVMe SSDs in each storage server. An NVMe SSD is divided into two partitions, each of which is assigned to a Ceph OSD daemon as the Ceph journal partition.

As shown in Figure 3(a), we classify write traffic into the following categories in FileStore:

- **Ceph data:** Replicated client data written by Ceph OSD daemons.
- **Ceph metadata:** Data written by Ceph OSD daemons other than **Ceph data**.
- **Ceph journal:** Data written to the Ceph journal partition by Ceph OSD daemons.
- **File system metadata:** File system metadata written by XFS (e.g. inodes, bitmaps, etc.).
- **File system journal:** File system journal written by XFS.

Inside the kernel, it is very difficult to separate **Ceph data** from **Ceph metadata** unless there is an explicit hint from the Ceph layer. Instead, we first calculate the amount of **Ceph data** by multiplying the replication factor to the amount of data written by the client. And then we obtain the amount of **Ceph metadata** by subtracting the amount of **Ceph data** from the total amount of data written by XFS for regular files and directories. Thus, **Ceph metadata** also includes any data written to LevelDB by Ceph OSD daemons.

2) KStore

In KStore, we use only one partition per Ceph OSD daemon which is mounted with the XFS file system. Since the write requests from the client are 4KiB in size, we set the stripe size of key-value pairs (`kstore_default_stripe_size`) to 4096 instead of the default value of 65536.

In KStore, write traffic is classfied into the following categories as shown in Figure 3(b):

- **Ceph data:** Replicated client data written by Ceph OSD daemons.
- **Ceph metadata:** Data written by Ceph OSD daemons other than **Ceph data**.
- **Compaction:** Data written by LevelDB or RocksDB during compaction.

- **File system metadata:** File system metadata written by XFS.
- **File system journal:** File system journal written by XFS.

Note that **Ceph data** and **Ceph metadata** are obtained in the same way as in FileStore. In KStore, **Ceph metadata** includes other key-value pairs in LevelDB or RocksDB written by Ceph OSD daemons. Since LevelDB or RocksDB runs on the XFS file system, KStore also generates **File system metadata** and **File system journal**.

3) BlueStore

For BlueStore, each local disk is separated into two partitions. One is a tiny partition for some notifications such as fsid and keyrings that are not in the data path. The other is for object data that BlueStore directly manages. BlueStore also requires additional partitions for RocksDB and RocksDB WAL. As in FileStore, we make two OSD daemons share one NVMe SSD to run RocksDB.

As depicted in Figure 3(c), write traffic is classified into the following categories in BlueStore:

- **Ceph data:** Replicated client data written to the raw partition directly by Ceph OSD daemons.
- **Ceph metadata:** Data written to RocksDB and RocksDB WAL by Ceph OSD daemons.
- **Compaction:** Data written by RocksDB during compaction.
- **Zero-filled data:** Data filled with zeroes by Ceph OSD daemons.

In BlueStore, the raw partition is allocated and managed in chunks of `bluestore_min_alloc_size` which is 64KiB by default. The unwritten area in each chunk are filled with zeroes when it is written to the raw partition. **Zero-filled data** represents the amount of data writes for filling those *holes*.

IV. Microbenchmark Experiment Analysis

Figure 4 illustrates microbenchmark results on each Ceph storage backend. Generally, the larger the request size, the smaller WAF except for the case of KStore (RocksDB). With the request size of 4KiB, WAF varies from 9.000 to 67.375, which means that when a client writes only a single 4KiB data, Ceph issues 36KiB to 269.5KiB sized data into the storage. A more detailed analysis for each storage backend is discussed below.

A. FileStore

Figure 4(a) displays a microbenchmark result in FileStore. When the request size is 4KiB, WAF is very large ranging from 41.375 to 67.375. To understand this, let us examine what happens when a client writes a single 4KiB data to *krbd*. First, the client knows which OSD server is responsible for the primary copy of the corresponding object in constant time with the help of the CRUSH algorithm and sends a write request to the OSD server. The primary OSD server delivers the write request to the other two secondary OSD servers to make replicas. Internally, the primary and secondary OSD servers encapsulate the incoming write request as a transaction consisting of key-value pairs to be stored in LevelDB, object metadata and object data itself. This transaction is then passed to the underlying FileStore. Receiving the transaction, FileStore conducts write-ahead journaling and this is where Ceph journal traffic comes in. After writing to the journal completes, FileStore re-writes the transaction to the XFS file system and LevelDB, which generates additional file system metadata and file system journal traffic.

As the request size becomes larger, WAF decreases rapidly. This is because the amount of object metadata, key-value pairs, file system metadata and file system journal becomes relatively smaller compared to the request size. Eventually, WAF converges to six as the request size grows to 4MiB; the data replication to three OSD servers triples WAF and another write-ahead journaling in each OSD server doubles it.

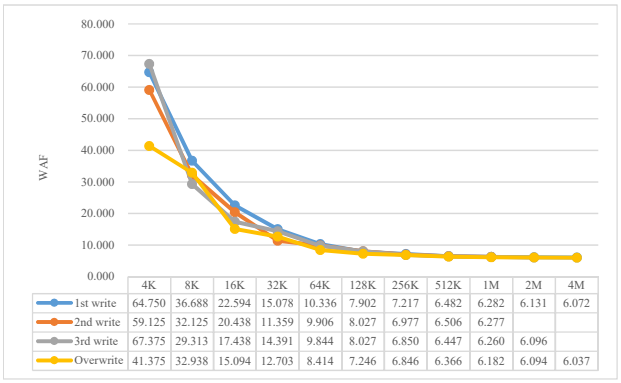
B. KStore

As shown in Figure 4(b) and 4(c), KStore exhibits overall graphs similar to FileStore's. The main difference is that WAF now converges to three as the request size increases. Note that both LevelDB and RocksDB are based on the LSM-tree (Log-Structured Merge-tree). In the LSM-tree, the incoming key-value pairs are first logged in the WAL (write-ahead logging) device. After logging, the key-value pairs are not immediately written to the storage. Instead, they are temporarily put into the memory buffer known as *memtable*. Only when the *memtable* becomes full, the entire contents are written into the storage in bulk. The write-ahead logging in LSM-tree is conceptually similar to the Ceph journal, hence the ideal WAF for KStore also should be six as in FileStore. In most cases, however, the write amount of our microbenchmark is too small to be flushed from the *memtable*. This is why WAF converges not to six, but to three in KStore.

The sudden jumps in RocksDB when the request size is larger than 512KiB are also due to the *memtable* flush. We have repeated same experiments several times, but WAF has changed very irregularly in this region. We suspect that RocksDB has a very sophisticated *memtable* flush policy. In the long run, the data in the *memtable* will be eventually flushed and those flushed data also should be compacted periodically. Hence, WAF in KStore will be increased significantly in the real environment. We take this effect into account in the long-term workload.

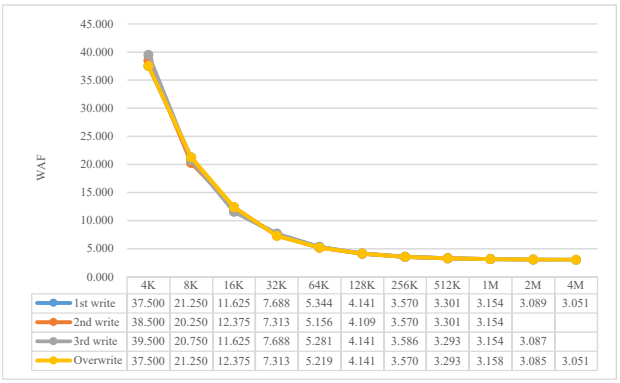
C. BlueStore

The result of BlueStore is somewhat interesting, as it shows a different trend from FileStore's and KStore's. First, WAF of the 3RD WRITE, which creates a hole in the middle of an object, is significantly higher than those of the 1ST WRITE and the 2ND WRITE at the request sizes of 4KiB, 8KiB and 16KiB. This is because the default configuration of the minimum chunk size in BlueStore is set to 64KiB. If a new chunk needs to be allocated, BlueStore fills the unused space in the current chunk with zeroes and then writes the chunk to the storage. When the request size is 32KiB, the entire chunk is filled with



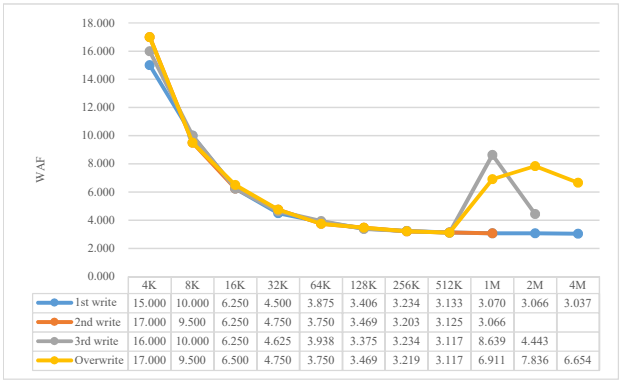
	4K	8K	16K	32K	64K	128K	256K	512K	1M	2M	4M
1st write	64.750	36.688	22.594	15.078	10.336	7.902	7.217	6.482	6.282	6.131	6.072
2nd write	59.125	32.125	20.438	11.359	9.906	8.027	6.977	6.506	6.277		
3rd write	67.375	29.313	17.438	14.391	9.844	8.027	6.850	6.447	6.260	6.096	
Overwrite	41.375	32.938	15.094	12.703	8.414	7.246	6.846	6.366	6.182	6.094	6.037

(a) FileStore



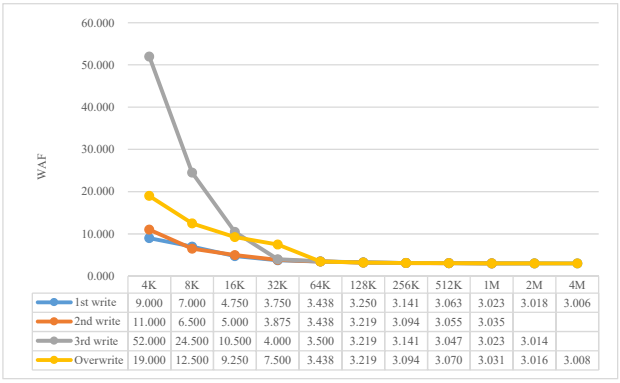
	4K	8K	16K	32K	64K	128K	256K	512K	1M	2M	4M
1st write	37.500	21.250	11.625	7.688	5.344	4.141	3.570	3.301	3.154	3.089	3.051
2nd write	38.500	20.250	12.375	7.313	5.156	4.109	3.570	3.301	3.154		
3rd write	39.500	20.750	11.625	7.688	5.281	4.141	3.586	3.293	3.154	3.087	
Overwrite	37.500	21.250	12.375	7.313	5.219	4.141	3.570	3.293	3.158	3.085	3.051

(b) KStore (LevelDB)



	4K	8K	16K	32K	64K	128K	256K	512K	1M	2M	4M
1st write	15.000	10.000	6.250	4.500	3.875	3.406	3.234	3.133	3.070	3.066	3.037
2nd write	17.000	9.500	6.250	4.750	3.750	3.469	3.203	3.125	3.066		
3rd write	16.000	10.000	6.250	4.625	3.938	3.375	3.234	3.117	8.639	4.443	
Overwrite	17.000	9.500	6.500	4.750	3.750	3.469	3.219	3.117	6.911	7.836	6.654

(c) KStore (RocksDB)



	4K	8K	16K	32K	64K	128K	256K	512K	1M	2M	4M
1st write	9.000	7.000	4.750	3.750	3.438	3.250	3.141	3.063	3.023	3.018	3.006
2nd write	11.000	6.500	5.000	3.875	3.438	3.219	3.094	3.055	3.035		
3rd write	52.000	24.500	10.500	4.000	3.500	3.219	3.141	3.047	3.023	3.014	
Overwrite	19.000	12.500	9.250	7.500	3.438	3.219	3.094	3.070	3.031	3.016	3.008

(d) BlueStore

Fig. 4: Results of Microbenchmark Experiments

data due to the 1ST WRITE and the 2ND WRITE and there is no more need to fill the hole.

Second, when the request size is smaller than 64KiB, WAF caused by the OVERWRITE is higher than that by the 1ST WRITE. This is because BlueStore tries to maintain data consistency against sudden power-off by writing the data into the WAL device if the request needs to overwrite the existing chunk partially. If the partial overwrite were performed to the existing chunk directly, we would not recover the original data in case the write operation were interrupted.

Finally, BlueStore shows much better WAF compared to the other storage backends. Since no local file system is used to store object data, BlueStore can make write traffic slim. In addition, unlike FileStore and KStore, BlueStore has no double-write issue as long as the request is not a partial overwrite. These make WAF converge to three in BlueStore when the request size is larger than or equal to the chunk size (e.g., 64KiB).

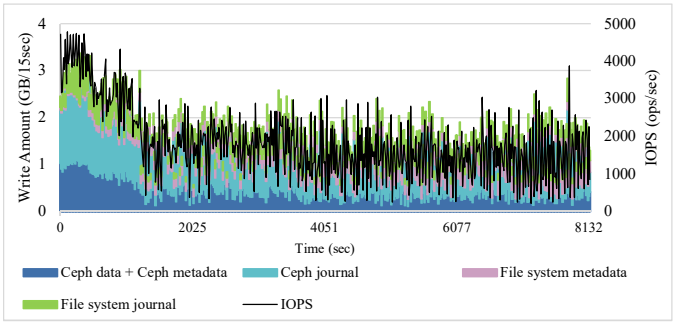
V. LONG-TERM EXPERIMENT ANALYSIS

Figure 5, 6, 7 and 8 present the results of our long-term experiments in FileStore, KStore with LevelDB, KStore with RocksDB and BlueStore, respectively. Each graph shows changes in the amount of data written to the storage for each category with IOPS measured on the client side. Note that, in FileStore, we did not distinguish the amount of Ceph data from that of Ceph metadata clearly, as it is difficult to track whether the data written to the Ceph journal has been also written to the original location at a certain sampling point.

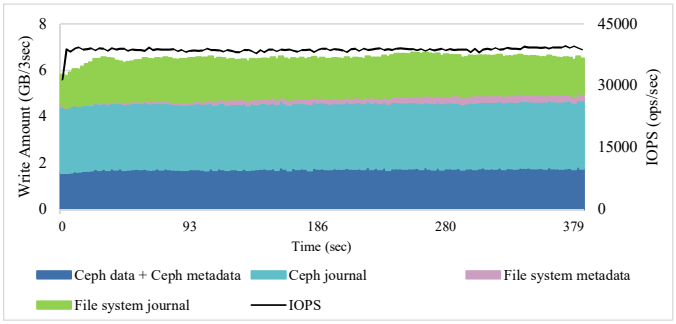
At first glance, we can see that each Ceph storage backend has unique patterns and the performance varies significantly among the backends. Table I summarizes the contribution of each category to the overall WAF in each storage backend. Note that the values in Table I are obtained by measuring the amount of write traffic in all Ceph OSD servers. In the following subsections, we first analyze the results of the long-term experiments on two different disk configurations (e.g., HDDs and SSDs) for each Ceph storage backend. In Section V-D, we summarize our findings and discuss the lessons learned from the long-term experiments.

A. FileStore

From 5(a) and 5(b), we can observe an interesting fact in common; the IOPS curves (denoted as a solid line) closely follow write traffic to the Ceph journal. This is because the client receives acknowledgments of the write requests right after three replicated transactions are journaled by FileStore in each Ceph OSD server. If we assume that the CRUSH algorithm uniformly distributes the write requests over the entire OSD servers, the number of completed write requests

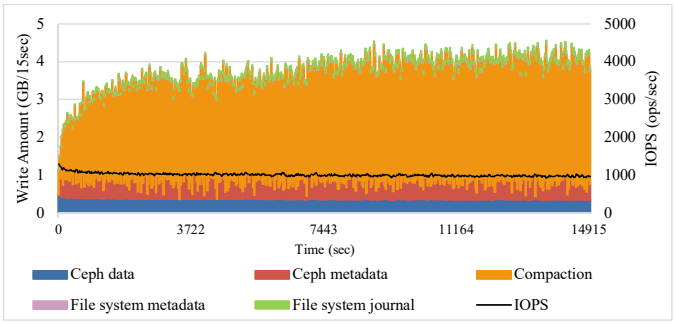


(a) FileStore - HDD + NVMe SSD

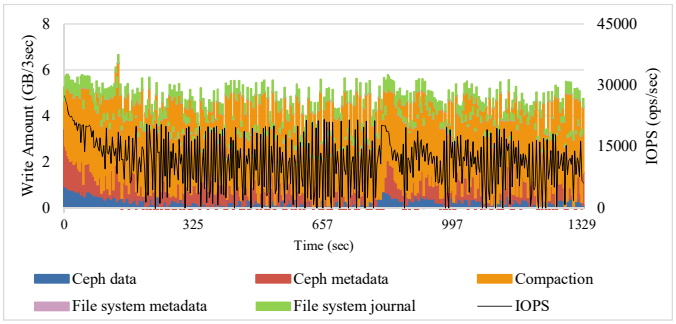


(b) FileStore - SSD + NVMe SSD

Fig. 5: Results of Long-Term Experiment in FileStore

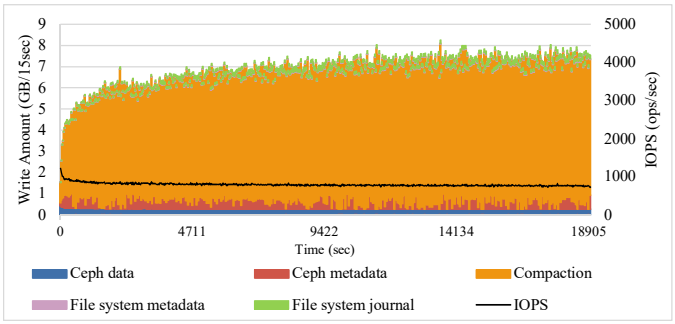


(a) KStore (LevelDB) - HDD

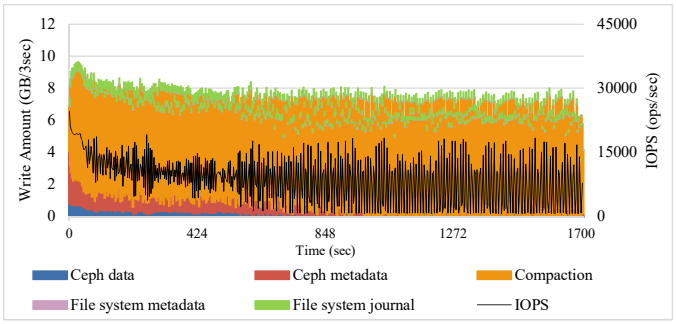


(b) KStore (LevelDB) - SSD

Fig. 6: Results of Long-Term Experiment in KStore (LevelDB)

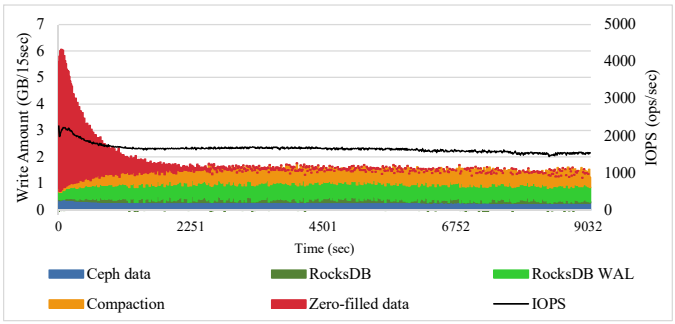


(a) KStore (RocksDB) - HDD

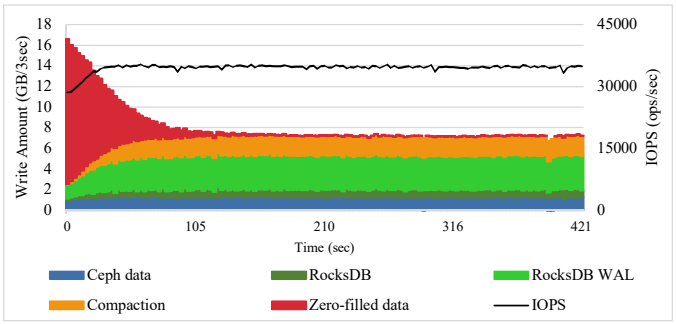


(b) KStore (RocksDB) - SSD

Fig. 7: Results of Long-Term Experiment in KStore (RocksDB)



(a) BlueStore - HDD + NVMe SSD



(b) BlueStore - SSD + NVMe SSD

Fig. 8: Results of Long-Term Experiment in BlueStore

TABLE I: Results of WAF for Long-Term Experiments

		Ceph data	Ceph metadata			Ceph journal	Compaction	Zero-filled data	File system metadata	File system journal	Sum
			Files	RocksDB	RocksDB WAL						
FileStore	HDD	3.000	1.332	-	-	6.026	-	-	1.944	2.256	**14.558**
	SSD	3.000	0.702	-	-	5.994	-	-	0.419	3.635	**13.750**
KStore (LevelDB)	HDD	3.000	4.206	-	-	-	23.301	-	0.098	0.645	**31.250**
	SSD	3.000	5.012	-	-	-	19.362	-	0.040	2.236	**29.650**
KStore (RocksDB)	HDD	3.000	6.307	-	-	-	60.240	-	0.258	1.229	**71.034**
	SSD	3.000	7.006	-	-	-	52.572	-	0.111	3.036	**65.725**
BlueStore	HDD	3.000	-	1.656	5.869	-	4.192	3.127	-	-	**17.844**
	SSD	3.000	-	1.615	7.436	-	4.165	2.865	-	-	**19.081**

on the client side will be proportional to the amount of write traffic in the Ceph journal.

However, there is a notable difference between the two cases; a pattern of write traffic in HDDs is totally different from that in SSDs. In the case of HDDs, IOPS stays at nearly 4000 ops/sec for about 1000 seconds, but afterward, it is decreased to below 3000 ops/sec until the experiment ends. This performance drop is mainly due to the slow speed of HDDs and throttling mechanisms used in FileStore. At the beginning of the experiment when FileStore is fresh, the client can get acknowledgments quickly as the Ceph journal first absorbs the incoming write transactions. As FileStore keeps writing the Ceph journal to NVMe SSDs in an append-only manner, the speed of re-write to HDDs cannot catch up the speed of the Ceph journaling. To prevent latency hiccups, FileStore checks the number of journal entries that have not yet flushed to HDDs and throttles other transactions from the upper layer if the number of unflushed journal entries hits certain thresholds (`journal_throttle_low_threshold` and `journal_throttle_high_threshold`). After the sufficient number of transactions are written to HDDs, throttling is released and the high-performance Ceph journal quickly absorbs write requests again until it reaches the throttling thresholds. As these routines are repeated, the amount of write traffic continues to oscillate within a short time range. Because write transactions are frequently throttled, the IOPS curve also fluctuates.

Unlike in HDDs, the amount of write traffic in SSDs seems to be completely stable as depicted in Figure 5(b). The amount of the Ceph journal written at the beginning of the experiment is almost the same as that written at the end of the experiment and there is quantitatively little change during the whole experiment. It means that the underlying SSDs on which the XFS file system is mounted can process queued transactions as fast as the Ceph journal writes.

The above two rows in Table I are the results for FileStore. Since FileStore relies on the external Ceph journaling, the conventional wisdom is that it doubles WAF due to the redundant write of data to the Ceph journal. However, Table I shows that the Ceph journal contributes to WAF by about six on both HDDs and SSDs, indicating that the Ceph journal actually does not double, but *triples* write traffic in each Ceph OSD server. This is because FileStore writes not only data but also other metadata and attributes to the Ceph journal.

Also, we can see that the *journaling of journal* problem exists in FileStore. Apart from Ceph metadata and Ceph journal, another file system metadata and file system journal increase the total WAF by almost four. This implies that the amount of file system metadata and file system journal is even larger than the actual data size in each OSD server.

B. KStore

As we can see in Figure 6 and 7, the huge amount of compaction traffic (denoted by yellow color) is responsible for most of WAF. When we focus on the cases with HDDs, KStore results based on two different key-value stores are almost similar. In both cases, the amount of write traffic due to compaction increases as time goes on, while IOPS is decreasing slightly.

If we look into the cases with SSDs, we can observe the continuous oscillation in WAF and IOPS during the whole experiment. Especially in Figure 6(b), the amount of Ceph data is measured to zero, meaning that the client is not receiving any acknowledgments during the sampling period (i.e., 3 seconds).

The results of KStore shown in Table I can be summarized as follows. First, the total WAF in KStore ranges from 29.650 to 71.034. This high level of WAF is unusual and infeasible in practice. From Table I, we can see that compaction is responsible for 65.3% $\sim$ 84.8% of the total WAF. Second, comparing the two key-value stores, RocksDB performs compaction more aggressively; RocksDB increases the amount of writes issued during compaction by more than 2.6x compared to LevelDB.

C. BlueStore

Figure 8(a) and 8(b) show the results when BlueStore is used as a storage backend. The overall trends in both graphs are similar. One notable difference in BlueStore from FileStore and KStore is that there is a large amount of zero-filled data traffic as soon as the experiment starts. As we explained in Section IV-C, the requests for zero-filled data are triggered when BlueStore needs to fill unused space of the current chunk with zeroes. The amount of zero-filled data decreases over time since the zero-filling operation is performed only once for each chunk.

Since BlueStore allocates chunks sequentially to the raw block device, writing zero-filled data at the beginning of the

TABLE II: Overall Results of Experiments Based on HDDs

	FileStore	KStore (LevelDB)	KStore (RocksDB)	BlueStore
IOPS (ops/sec)	1851	1008	796	1657
WAF	14.56	31.25	71.03	17.84
Avg latency (ms)	69.20	126.88	160.72	77.21
99.99th latency (ms)	7176.19	3523.00	3621.00	334.00

TABLE III: Overall Results of Experiments Based on SSDs

	FileStore	KStore (LevelDB)	KStore (RocksDB)	BlueStore
IOPS (ops/sec)	38739	11324	8861	34556
WAF	13.75	29.65	65.73	19.08
Avg latency (ms)	6.61	22.60	28.88	7.42
99.99th latency (ms)	50.99	5997.00	2868.00	51.46

experiment can be done quickly even on slow HDDs. However, continuous 4KiB random writes to already allocated chunks incur fully random accesses to HDDs and this makes IOPS suffer on HDDs. Unlike the case of FileStore, the IOPS curve is relatively more stable when using HDDs.

When BlueStore runs on SSDs, it is interesting that IOPS slightly increases after the initial phase of the experiment. This is because, unlike in the case of HDDs, the random writes after the initial phase do not become a bottleneck as SSDs have superior random write performance to HDDs. Instead, IOPS increases a little as the amount of additional writes due to zero-filled data diminishes.

When SAS SSDs are used as main storage for BlueStore, we find that NVMe SSD used as RocksDB and RocksDB WAL devices can be bottlenecked, as the performance difference between SAS SSDs and NVMe SSDs is not significant. Especially, our additional experiments show that IOPS drops by about 15% when a single NVMe SSD is shared by all four OSD servers for RocksDB and RocksDB WAL in a storage server.

Table I shows a breakdown of WAF for each category in BlueStore. In both HDDs and SSDs, the total traffic caused by RocksDB (RocksDB + RocksDB WAL + Compaction) is huge which is responsible for 65.7% $\sim$ 69.3% of the total WAF. In particular, we can see that the write traffic caused by RocksDB WAL is relatively dominant. The reason is that most of 4KiB random writes will be converted to partial overwrites to the existing chunks which require BlueStore to log data in the RocksDB WAL.

D. Overall Results

Table II summarizes the overall results of four tests when using HDDs as main storage. First, we can see that no matter which Ceph storage backend we choose, Ceph has the high WAF ranging from 14.56 to 71.03. Considering that the replication factor is three, the total amount of data written to the storage device is amplified by 4.85x $\sim$ 23.68x in each storage server when a single 4KiB data is written. Second, FileStore performs the best among four storage backends in terms of IOPS and the average latency. However, the 99.99th tail latency of FileStore is extremely bad; the time for the client to receive an acknowledgment for a single 4KiB data can be prolonged up to 7.3 seconds. Third, although IOPS and the average latency in BlueStore are slightly worse than those of FileStore, the 99.99th tail latency of BlueStore is much better (i.e., 334.0 ms). Fourth, despite one of the motivations of BlueStore is to avoid separate Ceph journaling in FileStore, it has the larger WAF than FileStore's. It is because BlueStore still performs write-ahead logging in the RocksDB WAL to prevent data loss if the size of the write request is smaller than the minimum chunk size. Finally, KStore based on key-value stores such as LevelDB and RocksDB suffers from compaction overhead, showing the worst IOPS and WAF.

The results on SSDs are summarized in Table III. Comparing them each other, FileStore seems to be the best storage backend on SSD-based Ceph systems. Unlike the case of HDDs, FileStore outperforms not only in IOPS and the average latency, but also in the 99.99th tail latency. As can be seen in Figure 5, this is due to that the write speed of the main SSD is fast enough to catch up the speed of the Ceph journal write. In most cases, the client receives acknowledgments as soon as writing to the Ceph journal completes, which also helps in reducing latency in FileStore. BlueStore performs a little behind FileStore, and KStore again performs the worst, resulting in very long tail latencies even on SSDs.

According to our results, BlueStore seems to be the most promising storage backend under the latency-critical situation especially when HDDs are used as the main storage media. When running Ceph on SSDs, FileStore is still the storage backend of choice with BlueStore being a close competitor to FileStore.

VI. Related Work

Since Ceph is publicly announced in 2006 by Weil et al. [1], the initial work has been centered around Ceph file system which provides file-level service to clients [11], [12], [13]. Gudu et al. investigate performance and scalability of Ceph RADOS RBD [14]. However, their work is conducted on an earlier version of Ceph (Emperor version) and the major evaluation metric is the client-side throughput.

As SSD has been popular as high-performance storage devices, several approaches have been proposed to optimize the performance of Ceph block storage on SSDs. Samsung compares the performance of Ceph block storage using their own NVMe SSDs with varying the number of OSDs per SSD and the number of SSDs per OSD [15]. Oh et al. point out performance bottlenecks in FileStore of the Ceph Hammer version and propose several optimizations under the all-flash environment consisting of NVRAM and SSDs [6]. Moreover, there has been an attempt to get the maximum performance from the Ceph by tuning some kernel parameters such as MTU (Maximum Transmission Unit) and file system mount options as well as Ceph configurations [4].

However, none of the previous work has focused on the characteristics of the Ceph storage backends and their implications on performance and WAF. We believe this paper is

the first study to compare write behaviors of the Ceph storage backends with a focus on WAF.

VII. Conclusion

This paper mainly analyzes write behaviors and the performance of Ceph storage backends focusing on WAF. Through extensive experiments with a microbenchmark and a long-term workload, we have shown the particular write patterns that each Ceph storage backend generates and compared them with each other on two types of storage devices, HDDs and SSDs.

We find that IOPS is closely related to WAF and writes are amplified by more than 13x in Ceph regardless of the storage backend used. FileStore actually triples the write traffic due to its own external Ceph journaling. In addition, there exists the *journaling of journal* problem with the amount of file system metadata and file system journal exceeding the original data size. KStore suffers the overhead of compaction performed in the underlying key-value stores, showing the worst IOPS and WAF among all storage backends. BlueStore does not have the *journaling of journal* problem since data are stored in the raw block device directly bypassing the file system layer. BlueStore also avoids external Ceph journaling, although small-sized requests are still logged in the RocksDB WAL. Overall, FileStore performs the best among all storage backends on SSDs, but BlueStore is still very promising as it performs reasonably well on both HDDs and SSDs, showing good average and tail latency even on HDDs.

Acknowledgment

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea Government (MSIP) (No. NRF2016R1A2A1A05005494). This work was also funded by Samsung Electronics, Co.

References

[1] 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*. USENIX Association, 2006, pp. 307–320.

[2] "Ceph storage introduction," https://ceph.com/ceph-storage/.

[3] "Ceph architecture," http://docs.ceph.com/docs/master/architecture/.

[4] J. Zhang, "Ceph: Open source storage software optimizations on intel architecture for cloud workloads," Intel, pp. 13–18, 2015.

[5] C. Xue, "Accelerating ceph performance profiling and tuning with cetune," Intel, 2016.

[6] M. Oh, J. Eom, J. Yoon, J. Y. Yun, S. Kim, and H. Y. Yeom, "Performance optimization for all flash scale-out storage," in *Cluster Computing (CLUSTER), 2016 IEEE International Conference on*. IEEE, 2016, pp. 316–325.

[7] K. Shen, S. Park, and M. Zhu, "Journaling of journal is (almost) free." in *FAST*, 2014, pp. 287–293.

[8] 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*. ACM, 2006, p. 122.

[9] I. Ahmad, "Easy and efficient disk i/o workload characterization in vmware esx server," in *Workload Characterization, 2007. IISWC 2007. IEEE 10th International Symposium on*. IEEE, 2007, pp. 149–158.

[10] E. Kim, "Enterprise Applications, How to create a synthetic workload test." SNIA Education, 2014, p. 16.

[11] 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 *Proceedings of the 8th Parallel Data Storage Workshop*. ACM, 2013, pp. 14–19.

[12] X. Zhang, S. Gaddam, and A. Chronopoulos, "Ceph distributed file system benchmarks on an openstack cloud," in *Cloud Computing in Emerging Markets (CCEM), 2015 IEEE International Conference on*. IEEE, 2015, pp. 113–120.

[13] C.-T. Yang, W.-H. Lien, Y.-C. Shen, and F.-Y. Leu, "Implementation of a software-defined storage service with heterogeneous storage technologies," in *Advanced Information Networking and Applications Workshops (WAINA), 2015 IEEE 29th International Conference on*. IEEE, 2015, pp. 102–107.

[14] D. Gudu, M. Hardt, and A. Streit, "Evaluating the performance and scalability of the ceph distributed storage system," in *Big Data (Big Data), 2014 IEEE International Conference on*. IEEE, 2014, pp. 177–182.

[15] "High-performance cluster storage for iops-intensive workloads/optimize ceph cluster performance by combining red hat ceph storage on samsung nvme ssds," Tech. Rep., 2016.