Optimizing the Ceph Distributed File System for High Performance Computing

Kisik Jeong
Sungkyunkwan University
Suwon, South Korea
kisik.jeong@csl.skku.edu

Carl Duffy
Seoul National University
Seoul, South Korea
cduffy@snu.ac.kr

Jin-Soo Kim
Seoul National University
Seoul, South Korea
jinsoo.kim@snu.ac.kr

Joonwon Lee
Sungkyunkwan University
Suwon, South Korea
joonwon@skku.edu

***Abstract*—With increasing demand for running big data analytics and machine learning workloads with diverse data types, high performance computing (HPC) systems consequently need to support diverse types of storage services. Ceph is one possible candidate for such HPC environments, as Ceph provides interfaces for object, block, and file storage. Ceph, however, is not designed for HPC environments, thus it needs to be optimized for HPC workloads. In this paper, we find and analyze problems that arise when running HPC workloads on Ceph, and propose a novel optimization technique called F2FS-split, based on the F2FS file system and several other optimizations. We measure the performance of Ceph in HPC environments, and show that F2FS-split outperforms both F2FS and XFS by 39% and 59%, respectively, in a write dominant workload. We also observe that modifying the Ceph RADOS object size can improve read speed further.**

***Index Terms*—Ceph, distributed file system, high performance computing**

I. Introduction

During the past few decades, the high performance computing (HPC) community has mostly focused on compute-intensive scientific applications. However, due to the recent prevalence of big data analytics and machine learning techniques, there is a growing demand for running these workloads on HPC systems [1]–[3]. Traditional HPC systems rely on parallel file systems such as Lustre [4] and IBM GPFS/Spectrum Scale [5], [6] to process I/O quickly. However, as the workloads running on HPC systems become more diverse, a more versatile storage system is needed that can provide not only a distributed file system service, but also a scalable object storage service and a block storage service.

Ceph is a unified, distributed, and scalable storage solution that is widely used in cloud computing environments [7]. Ceph provides interfaces for object, block, and file storage. Furthermore, all components in Ceph are scalable horizontally, and it does not have a single point of failure.

Because Ceph provides all the storage services needed by big data analytics and machine learning workloads, we believe Ceph is a promising candidate for a next-generation storage platform in the HPC environment. Recently, the ARC Center of Excellence for Particle Physics at the Terascale (CoEPP) chose Ceph as a new generation storage platform for Australian high energy physics [9].

Although Ceph has been chosen for use in recent HPC workload environments, Ceph should satisfy the needs for traditional HPC workloads. Traditional HPC workloads perform reads from a large shared file stored in a file system, followed by writes to the file system in a parallel manner. In this case, a storage service should provide good sequential read/write performance. However, Ceph divides large files into a number of chunks and distributes them to several different disks. This feature has two main problems in HPC environments: 1) Ceph translates sequential accesses into random accesses, 2) Ceph needs to manage many files, incurring high journal overheads in the underlying file system.

To deal with this problem, this paper proposes several Ceph optimizations for HPC environments. Our approach is to use the F2FS file system instead of the default XFS file system as an underlying file system for Ceph. Due to the log-structured nature of F2FS, utilizing it as Ceph's underlying file system brings potential performance benefits. Futhermore, we propose a new file system design called F2FS-split, where the metadata region of F2FS is placed into a separate SSD (Solid-State Drive), together with the external Ceph journal. This eliminates unwanted head movement in the HDD (Hard Disk Drive) when updating file system metadata. Additionally, we tune F2FS to further avoid HDD head movement.

Our evaluation results with a parallel I/O workload using MPI-IO show that F2FS-split outperforms both F2FS and XFS by 39% and 59%, respectively, in a write dominant workload. We also observe that modifying the Ceph RADOS object size can improve read performance further.

The rest of this paper is organized as follows. Section II briefly overviews the Ceph architecture, F2FS file system, and MPI-IO. In Section III, we show a motivating example and its challenges. Section IV proposes Ceph optimizations for HPC workloads. We evaluate our new design and analyze the results in Section V. Finally, we conclude the paper in Section VI.

II. Background

A. Ceph Architecture

Ceph provides storage services at the object level (object storage), block level (block storage), and file level (file system). Internally, all storage services in Ceph are built on a unified layer called RADOS (Reliable Autonomic Distributed Object Store) [8]. Ceph stores data as objects via RADOS, where object data can be retrieved directly from an OSD server by calculating the location of an object using the CRUSH

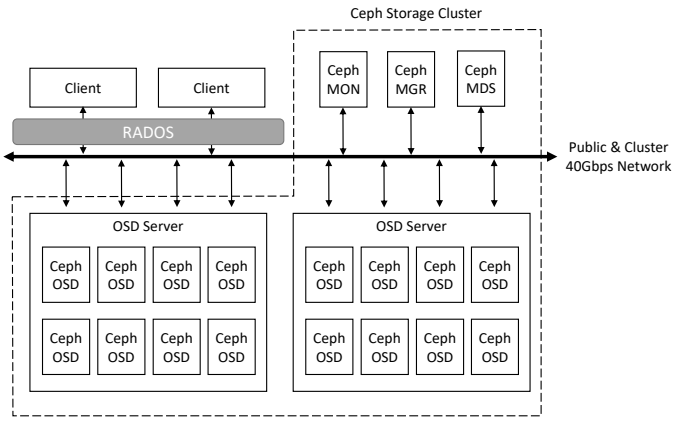


Fig. 1: Ceph storage cluster architecture

algorithm [12]. In comparison, traditional distributed storage systems retrieve data locations from a centralized server. This centralized design can lead to performance bottlenecks, as well as introducing a single point of failure. Using the CRUSH algorithm, Ceph removes the need for a centralized server so that it can scale, recover, and rebalance dynamically.

The Ceph file system (CephFS) is a POSIX-compliant file system that stores data in a Ceph storage cluster. In CephFS, the MDS daemon manages directory map and filename information for the file system. Every file is striped in fixed units (4MB by default) at the RADOS level and stored in several OSD servers so that CephFS achieves high performance and reliability. If required, CephFS can set a different file striping unit for each file or directory by changing its layout attribute.

Ceph consists of several daemons, each of which performs a special task. The Ceph OSD daemon (OSD) stores and retrieves data, manages data replication, and controls recovery and rebalancing. Furthermore, it manages the status of OSDs in a cluster via exchanging heartbeat messages. The Ceph manager daemon (MGR) tracks various status information such as storage utilization, performance metrics, and system load in a Ceph cluster. The Ceph monitor daemon (MON) manages monitor, manager, OSDs, and CRUSH map information in a data structure called a *cluster map*. Additionally, it administers authentication between a Ceph client and other daemons. Finally, the Ceph metadata daemon (MDS) stores directory map and filename information for the Ceph file system. Combined, this deployment of Ceph daemons forms a *Ceph storage cluster*. Figure 1 illustrates the overall architecture of a Ceph storage cluster.

B. Ceph Storage Backends

Ceph OSDs store data in several ways [13]. The latest Ceph OSD version, Luminous (v12.2.x), provides two methods, called BlueStore and FileStore. Although BlueStore is the default option for Ceph OSD backends, FileStore is currently the more stable and safer option, and thus we use FileStore instead of BlueStore throughout this paper. We leave the comparisons with BlueStore as future work.

FileStore stores each object passed from the RADOS layer as an individual file in a local file system such as XFS, BTRFS, and ZFS. XFS is used as the default backend file system in FileStore. When we use FileStore, external journaling is necessary to ensure Ceph-level crash consistency. Ceph provides strong crash consistency for each object, where all the write operations associated with storing an object must be done in an atomic transaction. However, there is no POSIX API that ensures atomicity for multiple write operations to different files. Due to this, Ceph writes a log in the external journal first, in an append-only manner, and then worker threads perform the actual write operations to the file system. Many systems use an SSD (Solid State Drive) as the external journal device to increase Ceph performance further. Using an external journal can improve write performance thanks to append-only logging.

C. F2FS

F2FS is an append-only file system similar to LFS (Log-structured File System) [11], but is specifically designed for NAND flash-based storage devices [10]. Figure 2a illustrates the basic disk layout of F2FS. Each metadata structure in F2FS is located at a fixed location on the disk (CP, SIT, NAT, and SSA in Figure 2a). When a file is written to F2FS, the data is written sequentially in a log-structured manner, but the metadata located at the beginning of the disk is updated in place.

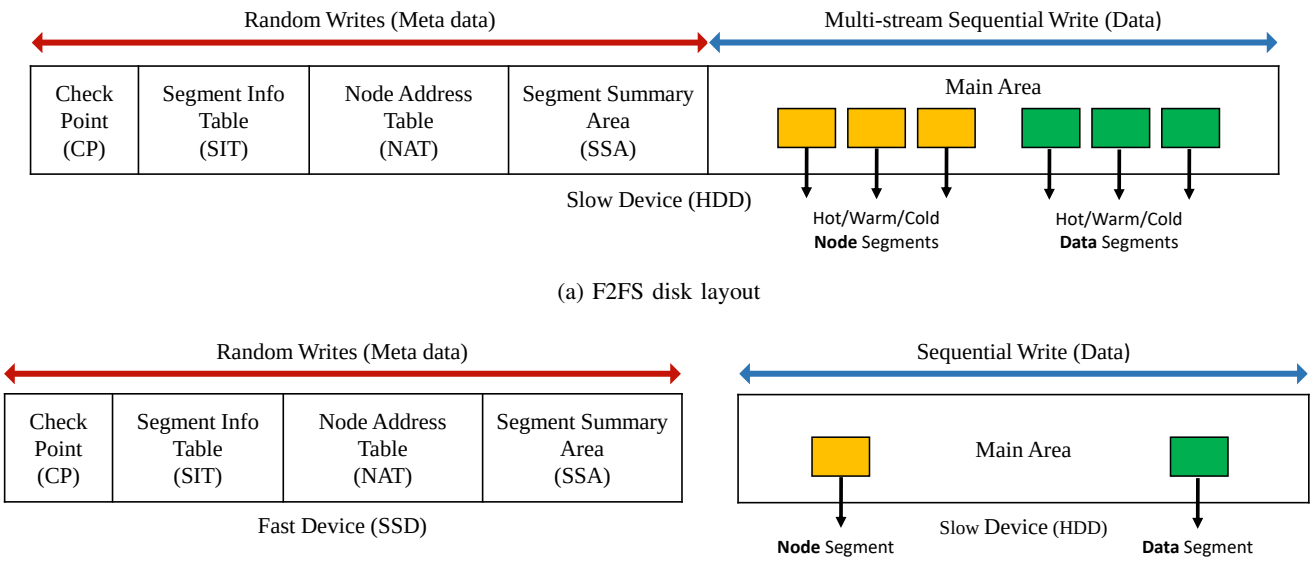
D. MPI-IO

A parallel I/O system for distributed memory architectures needs a mechanism that can specify collective operations with a non-contiguous data layout in memory and files. There is a similarity between reading or writing files to storage, and receiving or sending messages over a network. This implies parallel I/O can be implemented in a similar way that parallel communication is implemented in MPI. MPI-IO has provided parallel I/O functionality since MPI 2.0 in an effort to improve I/O performance for distributed memory applications. ROMIO [15] is a widely-available MPI-IO implementation which uses an abstract-device interface for I/O (ADIO) to support different backend file systems.

III. Motivation

A. Motivating Workload

Typical HPC workloads use MPI-IO, which reads a large input file shared among many processes, and writes to an output file. In this paper, we created a synthetic workload that simulated an HPC workload scenario. The synthetic workload performed 256GB write/read on a single file to a CephFS-mounted directory using the MPI-IO interface. The chunk size of each write/read request was set to 4MB, the default stripe size in CephFS. We implemented a backend interface in ADIO for Ceph which supported both direct and synchronous I/O options. The number of MPI-IO processes was 16, which was the same as the number of Ceph OSDs. Since we used two client servers, each server ran eight MPI-IO processes.



(a) F2FS disk layout

(b) F2FS-split disk layout

Fig. 2: Disk layouts in F2FS and F2FS-split

B. Motivating Anaysis

We ran the synthetic workload for writes using FileStore with the default backend file system, XFS. Additionally, we traced the written sector numbers and their data types during runtime using the **ftrace** utility [14].

Figure 3a shows the positions of written sectors over time in one of the OSDs for the synthetic workload, when the underlying file system is XFS. We can see that data is separated into several regions. Even though a client writes a single file at the CephFS level, Ceph stripes the file into a number of objects and stores them as many files in the local file system. XFS divides the whole disk space into a number of allocation groups, and handles concurrent writes in different allocation groups. This random access pattern leads to poor performance due to frequent mechanical head movements in the HDD. Furthermore, each write to the many small files incurs a significant amount of journaling overhead to ensure crash consistency at the file system level.

IV. Optimizing Ceph for HPC Workloads

In this section, we propose several approaches to optimizing Ceph's FileStore for HPC workloads.

A. Replacing the Backend File System with F2FS

The main reason that FileStore exhibits low performance in HPC workloads is because large file writes to CephFS are translated to many small file writes in the local file system. In order to overcome this problem, we choose a log-structured file system replacement, F2FS. Figure 3b is the HDD ftrace result recorded after changing FileStore's local file system to F2FS. Since F2FS writes data in a log-structured manner, small random writes are translated into large sequential writes to disk. However, F2FS still suffers from an undesirable amount of mechanical head movement because of two reasons: 1) multi-headed logging and 2) fixed locations of metadata.

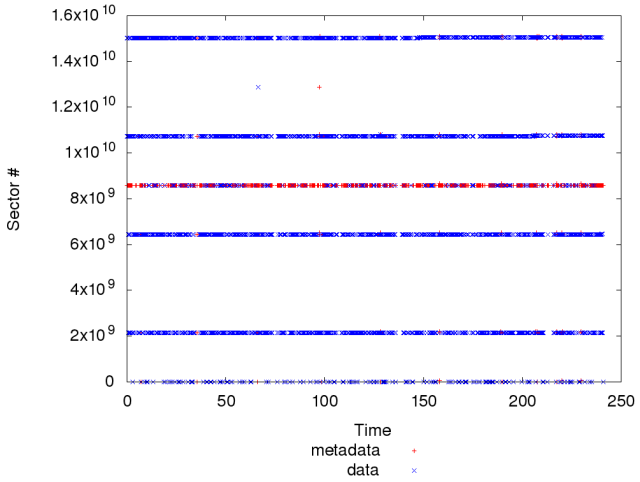
B. Tuning Multi-headed Logging

F2FS uses multi-headed logging to reduce the garbage collection overhead introduced as a result of append-only updates. In multi-headed logging, file system data and metadata are placed into different segments depending on their *hotness*. The number of logs in multi-headed logging can be configured by setting the `active_logs` value at mount time. The `active_logs` value can be two (only nodes and data), four (hot/cold nodes and data), and six (hot/cold/warm nodes and data), where six is the default value.

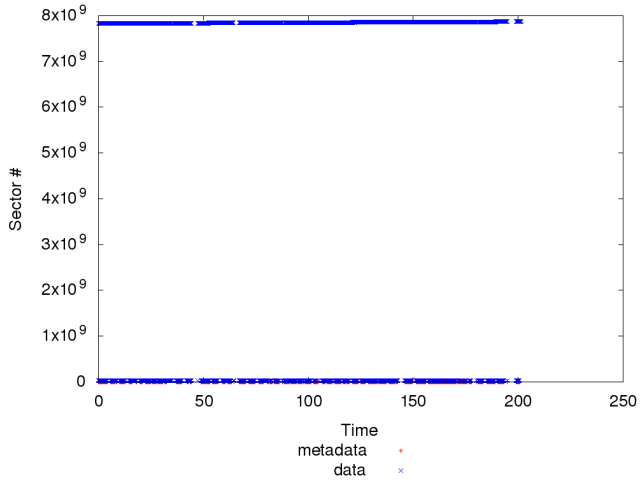
To avoid HDD head movement introduced by multi-headed logging, we adjusted the number of logs from six to two. Figure 3c displays the HDD ftrace result recorded after this change. From Figure 3c, we can see that file system data is written sequentially.

C. F2FS-Split

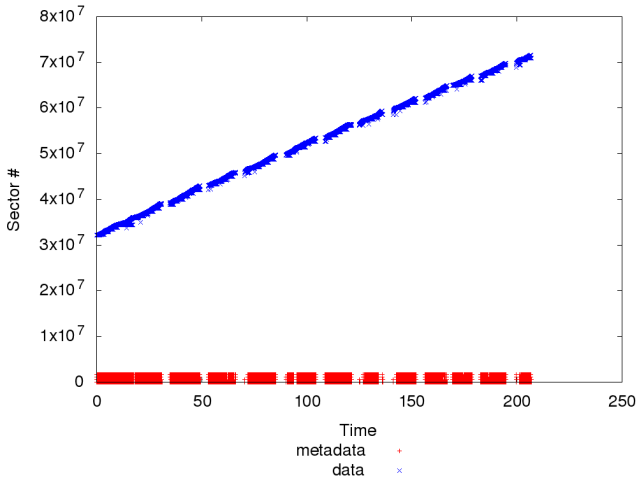
F2FS provides multi-drive functionality that enables a single F2FS file system to span multiple devices. This feature has been supported since f2fs-tools version 1.8.0. Our proposal is to separate F2FS metadata and data into different devices. We call this scheme *F2FS-split*. We have modified the f2fs-tools source code that calculates the size of the metadata region (SIT, NAT, SSA) of a given device, and divided the file system into a metadata region and a file data region. F2FS-Split allocates F2FS's metadata region into a fast SSD, and the file data region into a slow HDD. The size of the metadata region is relatively small compared to the data region. For example, when the size of the HDD is 8TB which will be used as F2FS's data region, the size of the metadata region



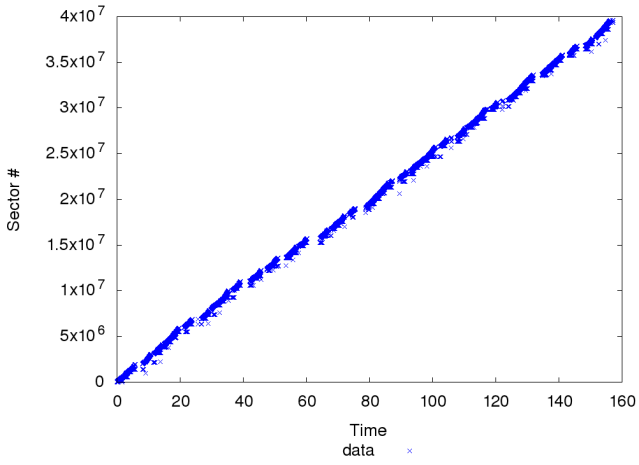
(a) XFS



(b) F2FS with `active_logs`=6



(c) F2FS with `active_logs`=2



(d) F2FS-split with `active_logs`=2

Fig. 3: HDD written sector offset over time for the synthetic workload using ftrace with XFS, F2FS, and F2FS-split

is only 32GB. We placed the SSD metadata region on an already present device containing the external Ceph journal for FileStore, and therefore, we did not need to provide additional disks to support our F2FS-split proposal. Figure 3d is the HDD ftrace of an experiment where we set the local file system to F2FS-split. We can see that the data is written sequentially on an HDD. As the metadata region was moved to another device, metadata writes were absent from this trace.

V. Evaluation

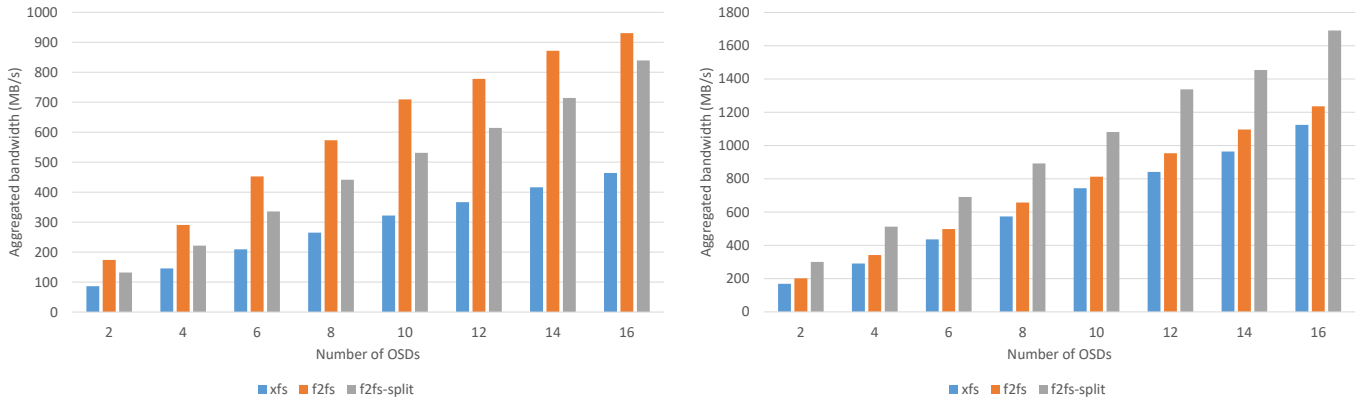
In this section, we evaluate our design, F2FS-split, comparing it with XFS and F2FS. The evaluation workload was the same as the synthetic MPI-IO workload described in section III.

A. Experimental Setup

Table I shows the organization of our experimental Ceph testbed. In our experiments, we used one administration server that ran Ceph MON, MGR, and MDS daemons. We had two client servers that generated I/O requests to CephFS. We used two storage servers to run Ceph OSD daemons. Both client and storage servers were connected to each other via a 40Gbps Ethernet network.

TABLE I: Experimental Ceph Testbed

MON / MGR / MDS / Clients (x2)	
Model	Dell R730
Processor	Intel(R) Xeon(R) CPU E5-2640 v4
Memory	128GB
OSD Servers (x2)	
Model	Supermicro SC829H
Processor	Intel(R) Xeon(R) CPU E5-2640 v4
Memory	128GB
Storage	Seagate ST8000VN0022-2EL112 8TB SATA x8 Intel(R) 750 series 400 GB NVMe x2
Interconnection	
Switch	Mellanox SX6012 40Gbps Ethernet switch
Network	40Gbps Ethernet



(a) Read workload

(b) Write workload

Fig. 4: Aggregated bandwidth of the sythetic workload with varying number of OSDs

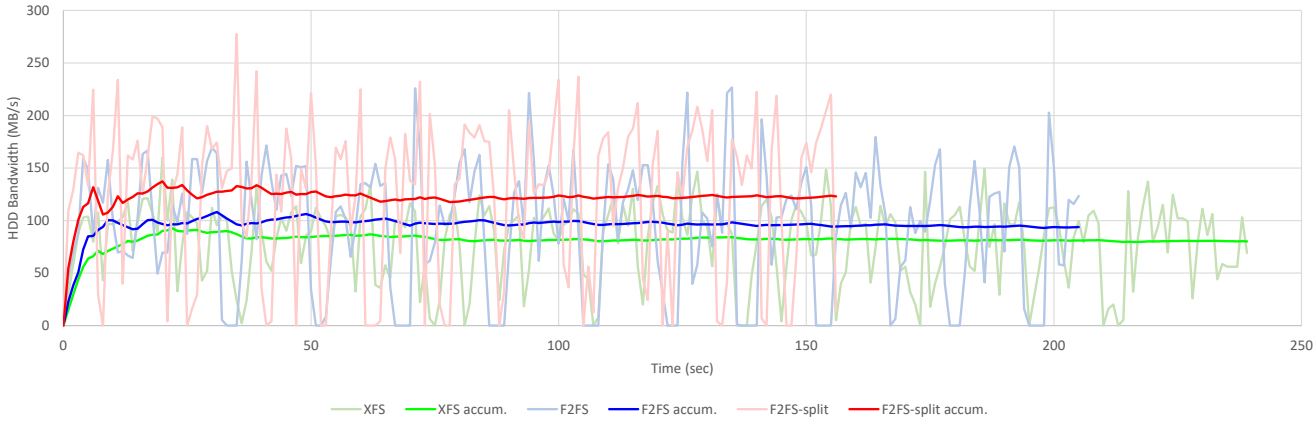


Fig. 5: HDD bandwidth variation over time during write synthetic workload (# OSD: 16)

Each storage server was equipped with eight 8TB Seagate ST8000VN0022-2EL112 SATA HDDs, and two 400GB Intel 750 series NVMe SSDs. A HDD was used as the main storage device for the OSD daemon, and an NVMe SSD was used for the external Ceph journal. In our configuration, a single HDD was dedicated to each OSD daemon so that a total of eight OSD daemons were running on a storage server. Therefore, our Ceph testbed consists of up to 16 OSDs. Note that four OSDs shared a single NVMe SSD for the external Ceph journal. The size of the external journal for each OSD was configured to 1GB. All experiments were done on Linux kernel version 4.14.7 with the latest Ceph Luminous LTS version (v12.2.8).

B. OSD Scalability Test

Figure 4 shows the result of our experiments with XFS, F2FS, and F2FS-split while varying the number of OSDs from 2 to 16. We set the number of MPI-IO processes to 16. Figure 4a shows the result of a read experiment and Figure 4b that of a write experiment. Overall, the aggregated read/write bandwidth increased linearly with the number of OSDs. In the case of the write experiment, F2FS-split outperformed XFS by over 59%, and F2FS by over 39%, on average. This is because F2FS-split translates small random writes into log-structured sequential writes, therefore reducing the HDD's head movement. Furthermore, as F2FS's metadata region was moved to the fast SSD, additional random writes were avoided in the HDD. In the case of the read experiment, however, F2FS-split was better than XFS by 64%, but showed only 78% of the performance of F2FS on average. Analyzing the block trace, we can see that the request size of F2FS was 256KB, while that of F2FS-split was only 128KB. This reduced batching resulted in a performance decrease. We plan to investigate this problem in more detail in our future work.

C. HDD Bandwidth Analysis

We tracked the bandwidth of every OSD disk during the write experiment, with the number of OSDs set to 16. We read sysfs device statistics (number of written sectors) for each OSD disk every second, and all OSD disks showed similar patterns. Figure 5 depicts the changes in the write bandwidth of one OSD disk for XFS, F2FS, and F2FS-split, along with the graphs for the accumulated average bandwidth. All

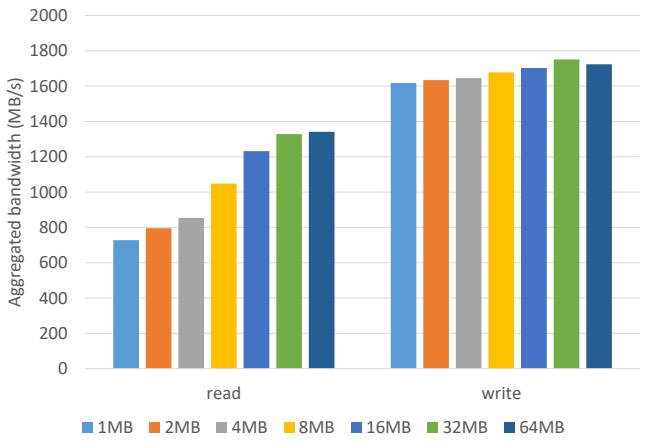


Fig. 6: Performance of read/write synthetic workload with different object size

three file systems suffered from high bandwidth fluctuation. FileStore writes data to the journal area first, and then moves them asynchronously to the file system. However, because the speed of writing data to the file system (HDD) is much slower than that to the external journal (SSD), FileStore frequently performs Ceph journal throttling to match the speed between the SSD and the HDD. This is the main reason why the write bandwidth fluctuated during the experiment. In spite of this, on average, F2FS-split was faster than XFS and F2FS by 53% and 31%, respectively.

D. RADOS Object Scalability Test

We evaluated the effect of the file striping unit size in CephFS, which determines the Ceph RADOS object size. We set the number of OSDs to 16, and the local file system to F2FS-split. We adjusted the MPI-IO write chunk size to be the same as the CephFS striping unit. Figure 6 shows the result of the experiment, varying the striping size from 1MB to 64MB. There was no notable difference in the write case, whereas read demonstrated higher performance when increasing the striping unit size. With the increasing write object size, the file system could not batch write requests exceeding its maximum batch size. In contrast, as the read object size increased, the Linux page cache could prefetch the next data block due to its readahead feature.

VI. Conclusion

In this paper, we analyzed and optimized the performance of the Ceph distributed file system in HPC environments. We changed the backend file system of the Ceph FileStore from XFS to F2FS to reduce file system journaling overhead, as well as to increase sequential access. Additionally, we proposed a modified version of F2FS called F2FS-split, which moved F2FS's metadata region into a separate, fast device. We showed that the overall performance of Ceph was proportional to the number of OSDs. Additionally, F2FS-split showed 39% and 59% higher performance than F2FS and XFS on average, respectively. Finally, we observed that varying the Ceph RADOS object size improves read performance further.

As briefly mentioned in Section V, the current F2FS-split design shows lower read bandwidth compared to the original F2FS. We plan to analyze this idiosyncrasy in more detail. In addition, we are going to perform more comprehensive experiments including a comparison with Ceph BlueStore.

ACKNOWLEDGMENT

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (No. NRF-2016R1A2A1A05005494 and NRF-2016M3C4A7952587). The ICT at Seoul National University provides research facilities for this study.

References

[1] Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., ... & Zhang, X. (2016). End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316.

[2] Dahl, G. E., Yu, D., Deng, L., & Acero, A. (2012). Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. IEEE Transactions on audio, speech, and language processing, 20(1), 30-42.

[3] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., ... & Berg, A. C. (2015). Imagenet large scale visual recognition challenge. International Journal of Computer Vision, 115(3), 211-252.

[4] Braam, P. J., & Zahir, R. (2002). Lustre: A scalable, high performance file system. Cluster File Systems, Inc.

[5] Schmuck, F. B., & Haskin, R. L. (2002, January). GPFS: A Shared-Disk File System for Large Computing Clusters. In FAST (Vol. 2, No. 19).

[6] Quintero, D., Bolinches, L., Chaudhary, P., Davis, W., Duersch, S., Fachim, C. H., ... & Weiser, O. (2017). IBM Spectrum Scale (formerly GPFS). IBM Redbooks.

[7] Weil, S. A., Brandt, S. A., Miller, E. L., Long, D. D., & Maltzahn, C. (2006, November). 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.

[8] Weil, S. A., Leung, A. W., Brandt, S. A., & Maltzahn, C. (2007, November). Rados: a scalable, reliable storage service for petabyte-scale storage clusters. In Proceedings of the 2nd international workshop on Petascale data storage: held in conjunction with Supercomputing'07 (pp. 35-44). ACM.

[9] Borges, G., Crosby, S., & Boland, L. (2017, October). CephFS: a new generation storage platform for Australian high energy physics. In Journal of Physics: Conference Series (Vol. 898, No. 6, p. 062015). IOP Publishing.

[10] Lee, C., Sim, D., Hwang, J. Y., & Cho, S. (2015, February). F2FS: A New File System for Flash Storage. In FAST (pp. 273-286).

[11] Rosenblum, M., & Ousterhout, J. K. (1992). The design and implementation of a log-structured file system. ACM Transactions on Computer Systems (TOCS), 10(1), 26-52.

[12] Weil, S. A., Brandt, S. A., Miller, E. L., & Maltzahn, C. (2006, November). CRUSH: Controlled, scalable, decentralized placement of replicated data. In Proceedings of the 2006 ACM/IEEE conference on Supercomputing (p. 122). ACM.

[13] Lee, D.-Y., Jeong K., Han, S.-H., Kim, J.-S., Hwang, J.-Y., and Cho, S. (2017, May). Understanding Write Behaviors of Storage Backends in Ceph Object Store. In Proceedings of the 33rd International Conference on Massive Storage Systems and Technology.

[14] Rostedt, S. (2009), Debugging the kernel using ftrace. http://lwn.net/Articles/365835/.

[15] Thakur, R., Lusk, E., & Gropp, W. (1997). Users guide for ROMIO: A high-performance, portable MPI-IO implementation (Vol. 176). Technical Report ANL/MCS-TM-234, Mathematics and Computer Science Division, Argonne National Laboratory.