FastStore: Optimization of Distributed Block Storage Services for Cloud Computing

1st Xiao Zhang
School of Computer Science
Northwestern Polytechnical University
Xi'an, China
zhangxiao@nwpu.edu.cn

2nd Huiru Xie
School of Computer Science
Northwestern Polytechnical University
Xi'an, China
xiefyy9911@mail.nwpu.edu.cn

3rd Zhe Wang
School of Computer Science
Northwestern Polytechnical University
Xi'an, China
1226327496@qq.com

4th Shujie Han
School of Computer Science
Peking University
Beijing, China
shujiehan@pku.edu.cn

5th Leijie Zeng
School of Computer Science
Northwestern Polytechnical University
Xi'an, China
zenglj@nwpu.edu.cn

6th Wendi Cheng
School of Computer Science
Northwestern Polytechnical University
Xi'an, China
347895316@qq.com

***Abstract*—Block service is an essential storage component in cloud computing systems, widely utilized for virtual machine disks and snapshots. Most block service virtualizations, including Ceph and Cinder, typically partition physical storage media into small units. Through dynamic allocation and mapping of these resource units, they are combined to form a virtual hard disk. Ceph abstracts the underlying storage into objects, performing space allocation and address mapping to deliver a highly scalable, unified storage service. However, the multi-layer mapping from blocks to objects and back to blocks imposes significant performance penalties. Additionally, Ceph employs several mechanisms to ensure object atomicity, which is unnecessary for block services, leading to writing amplification and decreased performance. In this paper, we present FastStore, a novel storage backend optimized for block services. FastStore eliminates unnecessary mechanisms and optimizes the allocation method for objects used in block services. We implement FastStore within Ceph, achieving a sequential reading performance 10 times faster than the current state-of-the-art Bluestore. Furthermore, it doubles sequential writing performance and enhances random writing performance by 40%. Additionally, FastStore reduces performance fluctuations and latency by 70%-80%.**

***Index Terms*—block storage service, object atomicity, storage backend, distributed system**

I. Introduction

The performance of virtual disks in cloud computing has a significant impact on the overall latency, throughput, and scalability of cloud-based applications and services. I/O virtualization technology divides physical disks into smaller units, combining them into virtual disks through address mapping. This process involves multiple rounds of address mapping and conversion, resulting in lower access efficiency. Early studies indicate that limitations in commodity I/O virtualization technology restrict the out-of-the-box storage bandwidth to 51% and 77% of a non-virtualized disk for writes and reads, respectively [1]. Even with the use of the latest storage media, such as NVMe SSDs, the original performance of the distributed storage system is only 30% compared to the raw disk's performance [2]. Applying faster storage media in a distributed storage system can yield better performance. However, it will result in a greater performance loss than the raw device since the inefficient software stack. The I/O stack of a distributed storage system is redundant and complex, shifting the performance bottleneck from storage devices to software [3]–[5].

Many unified distributed systems, such as Luster, Swift and Ceph, use objects to manage the underlying space, which simplifies metadata management compared to traditional local file systems, thus reducing metadata overhead. They encapsulate storage media as objects and then implement the virtualization of files and blocks based on these objects. Finally, they provide various services to upper-level applications. Ceph, developed in 2003 by Sage Weil at the University of California, Santa Cruz, creatively proposes the Controlled Replication Under Scalable Hashing (CRUSH) algorithm for data placement, as outlined in [6]. In comparison with traditional centralized architectures, Ceph offers high performance, scalability, and availability in dynamically changing and heterogeneous clusters. As shown in Fig. 1, Ceph supports block storage (RADOS Block Device, RBD), file storage (Ceph File System, CephFS), and object storage (Rados Gateway, RGW). RBD is a commonly used storage type in Ceph and is widely employed in open-source cloud computing platforms to provide virtual machine storage and object access capabilities [7]. After more than a decade of development and performance optimization, Ceph has designed various storage backends to continually enhance performance [8]. However, due to multiple layers of abstraction and redundancy mechanisms, different storage backends still exhibit significant write amplification [9], resulting in a substantial performance loss compared to raw disks.

To meet the modularity requirements of clusters and provide scalability and consistency, distributed systems typically

Xiao Zhang and Leijie Zeng are the corresponding authors. zhangxiao, zenglj@nwpu.edu.cn

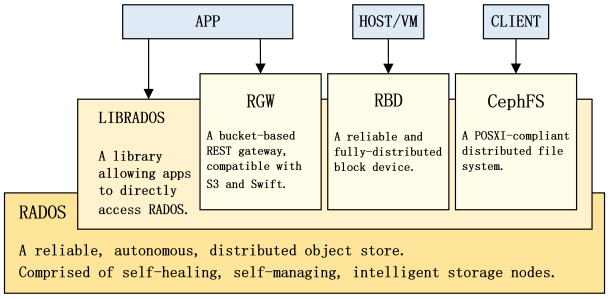


Fig. 1. The architecture of Ceph.

incorporate multiple layers of internal abstraction. While the hierarchical design of software can shield the heterogeneity of clusters, it also introduces unnecessary transformations. Consequently, Ceph RBD degrades performance significantly compared to the raw block device. The main issues are:

- The object storage system needs to ensure the integrity and atomicity of the data within a single object. However, block storage stores data in multiple objects, making it unnecessary to ensure integrity within individual objects. As a result, the system cannot fully exploit the performance advantages of underlying block devices due to object atomicity.
- In block mapping virtualization based on object storage, the underlying block device is encapsulated into objects, and these objects are then mapped to virtual blocks. The double process of encapsulation and mapping generates additional metadata, leading to redundancy in space management, data mapping, and garbage collection.
- The continuous logical space in the virtual disks is mapped onto discontinuous physical disks. Due to redirected writes, the more times users write, the more fragmented the data distribution becomes. Therefore, the sequential access performance of virtual disks is much lower than that of physical disks.

In light of these issues, we analyze the write mechanism for block virtualization based on object storage. We also discuss the performance overhead and fluctuation in different write mechanisms. Finally, we propose a new distributed storage backend, FastStore, for block storage services. Our main contributions are as follows:

- Abandoning redundant object atomicity for block storage and flexibly controlling data placement and I/O operations.
- Utilizing In-Place updating to reduce unnecessary read/write operations. Rebuilding the storage backend to eliminate isolation between different write schemes.
- Optimizing space allocation and recycling mechanisms and enabling the preallocation of fixed-size space.

The experimental results show that FastStore doubles sequential writing performance, and its random write performance is 40% higher than that of BlueStore. The performance of sequential reads is up to 10 times that of BlueStore. Furthermore, the tail latency is reduced by 70%-80%, resulting in decreased performance fluctuation.

The rest of the paper is organized as follows:

Section II introduces the background of Ceph and the implmentation of its block service. Section III identifies the performance overhead of the object storage, and the problem of performance fluctuation caused by different write schemes. Section IV introduces our optimization and the implementation of FastStore. Section V tests and analyzes the performance of the block device and our storage backends. Section VI describes research related to Ceph optimization. Section VII summarizes our work and makes prospects for future optimizations.

II. Background

In this paper, our aim is to build a high performance distibuted storage service by redesigning the backend of the Ceph [6]. This section provides a brief introduction to Ceph and the I/O path of current storage backend, introducing terms that will be used in the rest of the paper.

A. Block Service of Ceph

Ceph is a robust and highly scalable open-source distributed file system designed for efficient storage and retrieval of large volumes of data. It provides a unified solution for object storage, block storage, and file storage within a single, cohesive system. Ceph's architecture is organized into multiple components, including Object Storage Daemons (OSDs), Monitor nodes, and Metadata Servers (MDS). OSDs manage the actual data storage, while Monitor nodes oversee the cluster's state and health. MDS facilitates file system operations, enabling Ceph to offer distributed file storage capabilities.

Ceph implements the block service (RBD, RADOS Block Device) based on the distributed object store named RADOS (Reliable Autonomic Distributed Object Store). RBD devices are used as virtual disks for virtual machines or snapshots of virtual disks. An RBD image is split into objects with the same size, with the default object size set to 4MB. For a 4GB block image, Ceph will create 1024 objects named rbd_{name}_{indexno}, while the indexno ranges from 0000 to 1024. The object will not be created before a write operation on the corresponding range occurs. RADOS manages the position of objects using the CRUSH (Controlled Replication Under Scalable Hashing) algorithm [10]. I/O operations on RBD devices are transformed into operations on objects on the client side. The client uses CRUSH to determine the primary OSD that contains the given object and sends the object I/O request to the OSD. The primary OSD handles the object operations and stores the data on underlying storage devices such as HDDs and SSDs.

B. Ceph storage backend

An OSD server processes incoming object I/O requests and performs I/O on the storage backend. The performance of ObjectStore plays an important role in overall system performance. The first implementation of the ObjectStore

interface was, in fact, a user-space file system called Extent and a B-Tree-based Object File System (EBOFS) [8]. FileStore saves objects in a local file system such as Btrfs and XFS, which was the default production backend from 2009 to 2017. The current default storage backend of Ceph is BlueStore. It saves metadata in RocksDB and stores data on raw disks. The overhead of running an object store workload on a journaling file system is evident. Object creation throughput is 80% higher on a raw HDD and 70% higher on a raw NVMe SSD [8]. Crimson is the next-generation object store optimized for fast NVMe storage [11].

Ceph provides block service, file service, and object services based on the object store, with objects stored on block devices such as HDDs and SSDs. The conversion between block service and objects is not free. The write amplification by Ceph storage backends is more than 13x [9].

C. Two-layer mapping of object storage system

Ceph's architecture completely separates storage and application services. As shown in Fig. 2, RADOS and its derived librados provide a unified object storage service to the upper layer by encapsulating the operations of the underlying storage. It is possible to develop any storage application based on object storage theoretically. At the same time, although upper-layer applications can use the RBD interface to access data on the devices, they still need to encapsulate data as objects inside.

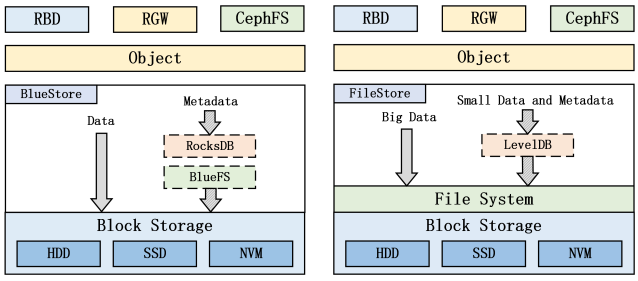


Fig. 2. The architecture of the storage backend.

Large and complex software systems often adopt a hierarchical architecture, providing different levels of data abstraction at each layer to achieve high cohesion. Each layer is assigned specific responsibilities, promoting reasonable division of labor and maintaining low coupling between layers. Ceph's various levels of implementation may have certain performance bottlenecks. However, this section primarily focuses on the macro architecture level, addressing redundant mapping relationships between different levels. Ceph relies on ObjectStore to manage different types of storage media and provide storage services that conform to the object transaction. The ObjectStore encapsulates the underlying I/O processing and sets up a set of operations with a strong consistency of objects. Currently, the main ObjectStores in the production environments are FileStore and BlueStore. Instead of directly accessing the storage device, FileStore manages it through the file system of the operating system. As a result, it can utilize the natural mapping between files and objects. However, this architecture has drawbacks, such as unguaranteed transaction consistency, inefficient metadata management, and difficulty supporting the new hardware. Given this, Ceph introduced BlueStore as the storage backend in the Jewel version. Unlike FileStore, BlueStore abandons the traditional local file system and accesses data directly from the device at the specified position. At the same time, BlueStore strictly separates metadata from user data, introducing RocksDB to store metadata, attribute information, and small data. As a result, it improves the efficiency of metadata operation and solves the transaction consistency problem by RocksDB.

BlueStore simplifies the I/O path and significantly improves performance by eliminating transformations between objects and files. However, as shown in Fig. 2, Ceph still needs to map the block storage service from the block device to the object storage system and from the objects to the RBD interface. This two-layer mapping not only increases the overhead of metadata management, leading to functional redundancy, but also makes it difficult to fully leverage the device's advantages. According to the experiment in III-A, block storage based on the Ceph cluster has severe performance degradation due to the multi-layer abstraction between software and hardware.

In summary, the dual mapping architecture of the Ceph RBD interface, while providing a more convenient data management approach for block storage in a distributed environment, also presents new challenges for system performance and optimization. Addressing the performance degradation and fluctuations caused by dual mapping, this paper analyzes and resolves these issues while maintaining system flexibility and feature completeness to fully leverage the performance potential of underlying storage media.

D. Mechanism to Ensure Object Atomicity

As a distributed object storage system, Ceph needs to ensure object atomicity. Because the local file system cannot support atomicity transactions, Filestore uses Write-Ahead Logging(WAL). We can use the journal to recover unpersisted data and accelerate writing operations without compromising reliability. However, it causes severe write amplification and sacrifices half of the throughput of the disk. At the same time, the local file system also uses the WAL and causes the "Journaling of Journal" problems [12]. With the redundant writes due to multiple copies, FileStore produces up to 14. 588 times write magnification [9].

As a transactional storage system designed for all-flash arrays, BlueStore needs to minimize the negative impact of journals while ensuring reliability and atomicity [8]. To do this, in the process of persisting the user data to the device, BlueStore creatively combines Read Modify Write (RMW) and Copy on Write (COW) and proposes the "incremental journal" strategy.

If no data exists in the writing range, this write operation is called the new write (NEW); otherwise, it is called the overwrite (OW). When writing to a newly allocated space,

BlueStore can persist data directly and update the corresponding metadata through RocksDB. NEW will not overwrite the existing data on the disk and guarantee metadata consistency through RocksDB. Therefore, we are not concerned about the failed write and missing data of NEW. Below we focus on the reliability of OW.

The block size (BS) refers to the atomic granularity when performing operations at the storage device. For HDD, this is typically 512 bytes; SSD uses larger BS, such as 4KB. For any write operation, BlueStore splits its range into three parts based on the BS: the non-aligned part in the front (Front), the non-aligned part in the tail (Tail), and the aligned part in the middle (Mid). Fig. 3 shows some specific examples.

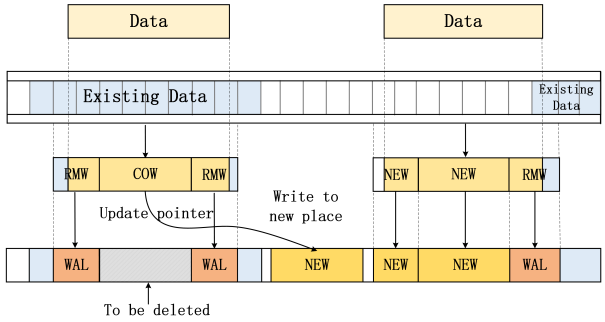


Fig. 3. Splitting write operations based on the BS.

BlueStore adopts RMW in the Front and Tail when OW occurs. First, read the block of existing data. Then merge the modified content with the original block. Lastly, write back the updated block to the original position. Considering the potential risk of power failure during the OW, it introduces WAL for this process. In WAL, the updated block is written into RocksDB in the form of journal, which will then be updated In-Place asynchronously. The whole process is completed through the "simple write" method.

BlueStore adopts the COW in the Mid when OW occurs. Instead of In-Place updating the existing data, it allocates a new space to store the data to write. First, complete the NEW, then update the corresponding address pointer. Last but not least, the original space is released. In BlueStore, implement this writing through the "deferred write" method.

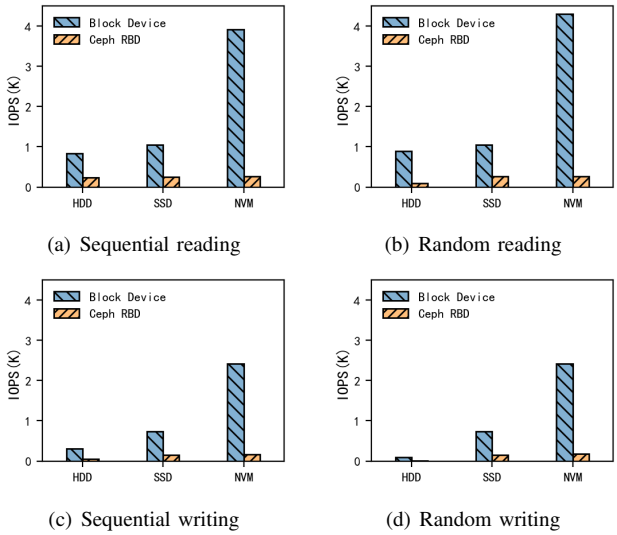
BlueStore uses different write modes due to data range and times to balance atomicity with system performance. As the experiment shown in III-B, the complex processing of the storage backend also brings severe performance fluctuations to the upper block storage service.

III. Motivation

The block service is a core offering in cloud computing platforms, providing virtual disks for each virtual machine and storage space for snapshots. Ceph offers three types of storage services, making it a crucial cloud storage platform. Ceph has replaced Swift as the default storage backend for OpenStack, holding a 57% share in the OpenStack storage field, significantly surpassing other storage solutions. Subsequently, it adapted to the public cloud interface represented by Amazon S3 and entered the next frontier of virtualization technology - Docker [13]. In this section, we identify the performance overhead caused by virtualization and the performance fluctuations resulting from the complex I/O stack. Given that HDDs remain the dominant storage medium in cloud computing and big data applications [14], our proposed solution can provide better performance in many scenarios.

A. The performance penalty of virtualization

Virtualization requires additional metadata, and the adoption of multiple replicas in distributed file systems can lead to write amplification. These instances of write amplification can result in significant degradation of write performance. We compare and test the I/O performance of the underlying block device and Ceph block storage. We use IO testing tool FIO on the Linux system for experimental verification. The data block size is set to 512K. Experimental environment and configurations are detailed in section V. The test results are shown in Fig. 4.



(a) Sequential reading (b) Random reading

(c) Sequential writing (d) Random writing

Fig. 4. The performance comparison of Ceph RBD and block storage device.

Firstly, although the selection of the storage hardware has an essential impact on the performance of the storage system, the full performance of block storage still cannot be unleashed by merely replacing underlying devices. With the continuous performance improvement of hardware, the performance deterioration caused by software accounts for an increasing proportion.

Secondly, the write performance of Ceph is generally lower than the read performance under the same condition. To ensure data reliability, Ceph adopts the strong-consistent copy strategy, which requires storing redundant data on multiple nodes. For reading, data can be obtained by communicating with a single node. For writing, data first needs to be written to the main node, with the calculation of the location of the other copies by Crush Map. The client will not receive the response until multiple copies of data are successfully written to their corresponding nodes [15].

Finally, the performance provided by Ceph RBD is far worse than raw block storage. In sequential reading, the IOPS of disks are 3.56, 4.19, and 14.83 times that of Ceph RBD under HDD, SSD, and NVM, respectively. In random reading, the IOPS performance difference between NVM and Ceph RBD is up to 15 times. Under the 3-copy mechanism, the write performance of block storage cannot reach 1/3 of the performance of the raw device. With the performance improvement of the underlying hardware, the performance difference between Ceph RBD and underlying devices becomes more significant. In the process of I/O, compared to the difference caused by hardware, the latency caused by software occupies a larger proportion. The block storage service generated by Ceph cannot extensively leverage raw devices' performance. Hence, there is still a performance bottleneck in it.

B. The performance fluctuations of different writing schemes

In addition to benchmarks such as IOPS and BW, performance fluctuations of storage systems will also have a significant impact on the performance of upper-layer applications [16]. To verify the I/O stability of the Ceph RBD interface, FIO is used for the I/O tests of the Ceph cluster and underlying devices. Tests are only carried out for HDD, and the other software and hardware conditions are the same as in III-A.

In order to verify the impact of different writing schemes on the performance of RBD, we design different writing scenarios with varying offsets and lengths for testing. As shown in Tab. I, we design three groups of writing schemes and test each scheme five times. For all schemes, Offset_1 is used for the first write, and Offset_2 is used for the next four writes. The five writes of scheme one and scheme two are all performed with the same offset, while in scheme three, there is a 1K difference in offset between the first write and the following four writes. Since the default size of Ceph objects is 4M, to avoid the additional overhead caused by cross-object access, the length of data is 64k each time. One hundred objects are written for each test, each with the same writing scheme inside.

TABLE I
OFFSETS AND LENGTHS

Scheme	Offset_1	Offset_2	Length
1	0	0	64K
2	32K	32K	64K
3	32K	31K	64K

As shown in Fig. 5, the IOPS performance of the first writes for the three groups of writing schemes is basically the same. However, the differences mainly occur in the following four writes. For schemes one and two, the IOPS of the following four writes are higher than the first. For scheme one, the IOPS of the second write is 186% of the first. However, in scheme three, the IOPS of the following write is lower than the first write, and the IOPS of the second write is only 1/3 of the first. Though writing the same amount of data, the performance fluctuation caused by different writing schemes can vary between 33%-66%.

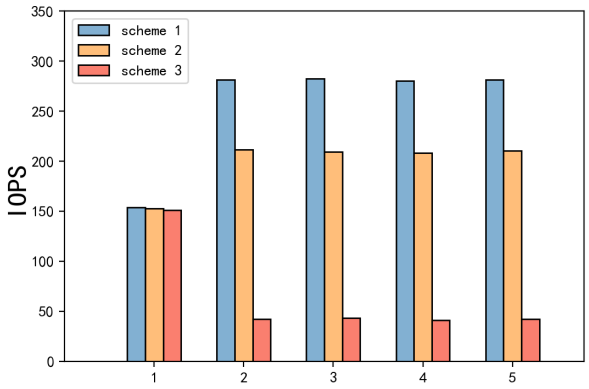


Fig. 5. The performance fluctuation in BlueStore.

The results show significant performance differences contributed by different offsets and write times. The complex writing schemes inevitably influence the block storage service's stability. Therefore, it is reflected from the performance fluctuations that there is still a very large improvement space for Ceph RBD.

IV. DESIGN AND IMPLEMENT OF FASTSTORE

In this section, we present the design and implementation of a new storage backend, FastStore. It was developed based on Bluestore, the following design choices are made: 1) In-place update to reduce metadata update. 2) Preallocate to make more logically adjacent data save nearby. 3) Remove redundancy atomicity writing mechanism for RBD.

A. In-place update

BlueStore defines min_alloc_size (MAS) as the minimum allocable space to reduce space fragmentation and improve data index efficiency. As shown in Fig. 6, according to the logical range in the object, data can be divided into non-MAS aligned parts on both sides and MAS-aligned parts in the middle, respectively executing Small Write (do_write_small) and Big Write (do_write_big) processes in BlueStore.

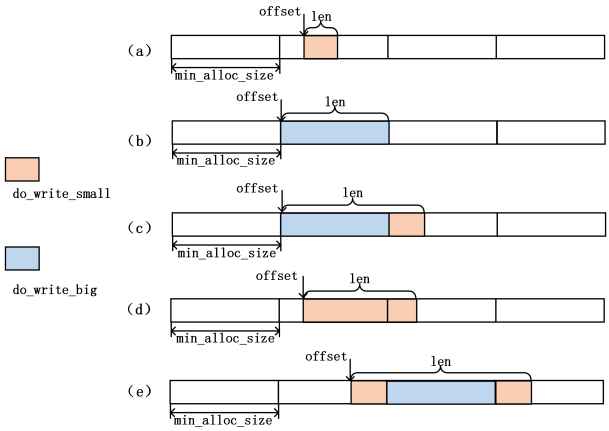


Fig. 6. The division of the Small Write and Big Write.

In general, MAS is equal to BS. However, it can be set as an integer multiple of BS for efficiency. In this case, an Extent may have some "holes" at the granularity of the block. For example, if BS is 4KB, MAS is 16KB, and the user writes 1KB of data, 16KB of disk space must be allocated. However, only 4KB of block space is used, and the system will mark the other three blocks unused.

Since MAS is an integer multiple of BS, the range of Big Write must align MAS and BS. Therefore, the execution process of Big Write is relatively simple: First, release the Extent that conflicts with the existing data and corresponding Pextent, and determine the data length in each write loop. Second, try to find an existing reusable Blob for each write loop and create a new Blob if it does not exist. Finally, store the critical information for each write loop in the queue of writing items. Following processing for the write items in the queue is performed by _do_alloc_write (DOW), which includes assigning and associating the Pextent, setting and marking the Extent, and performing asynchronous data writing on the device.

On the one hand, considering the data may not align with the BS, two types of BS alignment paddings are added before writing. If the padding part does not have existing data (unused), fill it with zeros. If it has existing data (used), RMW is inevitable. In this case, this part needs to be read, merged, and filled in with data to obtain new content and write it to the original position.

On the other hand, considering the data may not meet the MAS, adding two writing methods without queuing. Because BS is the atomic unit of disk operations and the contents of the "hole" are invalid. Writing on the unused "hole" does not affect the rest of the Extent, even though a power failure occurs or writing some incorrect data. Therefore, the unused part of the allocated space can perform "simple write" directly, and the used part can perform "deferred write" directly.

The strategies implemented by different write schemes in III-B are shown in Tab. II. For the first time, three write schemes perform NEW. Because RBD uses thin-provisioned configurations, it does not allocate the space until writing data. Therefore, the IOPS of NEW is lower than OW. For the other times, three schemes perform COW once, COW twice, and RMW twice, respectively. Because RMW needs to process read, merge, and fill in sequence and the content of the padding part needs WAL. Therefore, the OW performance of scheme 3 is much lower than other schemes.

The writing strategies to ensure atomicity of objects complicate the I/O process, and as mentioned in IV-C, we found this strategy needs to be revised for block storage. When executing WAL, data needs to be written to logs, while for COW, data is written elsewhere, requiring metadata updates. Both strategies result in write amplification. Adopting In-Place updates can significantly reduce the amount of data written. Therefore, we propose an optimization storage backend that uses In-Place OW to solve the problems of frequent unnecessary read/write, the isolation between Small Write and Big Write, and the latency in the distributed storage systems. Fig. 7 shows the writing strategy of FastStore.

TABLE II
PROCESSING STRATEGIES FOR DIFFERENT WRITING SCHEMES

No	Range	BS	MAS	OW	Processing Strategies		
1-1	0-64K	✓	✓	×	0-64K	Big	NEW
1-2	0-64K	✓	✓	✓	0-64K	Big	COW
2-1	32-96K	✓	×	×	32-64K	Small	NEW
					64-96K	Small	NEW
2-2	32-96K	✓	×	✓	32-64K	Small	COW
					64-96K	Small	COW
3-1	32-96K	✓	×	×	32-64K	Small	NEW
					64-96K	Small	NEW
3-2	32-96K	×	×	✓	31-32K	Small	RMW
					32-95K	Small	RMW

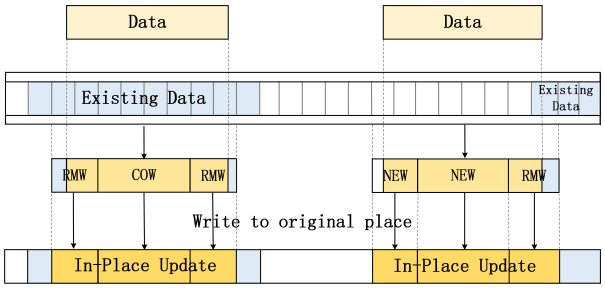


Fig. 7. Writing strategies of FastStore.

Like BlueStore, FastStore divides the data into Front, Tail, and Mid, based on MAS. Then execute the Small Write and Big Write, respectively. However, compared with BlueStore, FastStore eliminates almost all redundant atomicity protection for RBD.

The data range is still divided based on BS. On the one hand, for the aligned parts in the middle, FastStore will not perform COW but directly update the data in the original place, just like NEW. On the other hand, for the non-aligned parts in the head and tail, FastStore still performs RMW. However, when this content is written to the block device, FastStore no longer executes WAL.

In a word, FastStore allows block storage service to update data at the original position directly while removing the WAL mechanism. Compared with the original scheme, FastStore could avoid the write amplification and "double write" problems caused by WAL and optimize a series of redundant metadata modification operations caused by COW.

B. Re-design physical space management

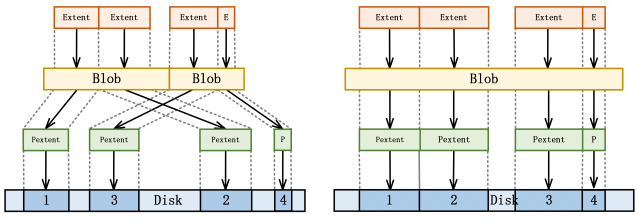
Multi-layer abstraction also leads to the redundancy of space management, data mapping, and garbage collection. Ceph uses an allocator module to allocate available space and a garbage collection module to recycle invalid space. In IV-A, the RBD writing in FastStore will directly overwrite the original place of the block device. Therefore, optimizing the mechanism of space allocation and recycling involved in the I/O process is necessary.

The Ceph community hoped BlueStore to double the write performance while avoiding FlieStore's flaws. This goal has

been partially achieved based on NVMe SSD [17], but the performance of BlueStore in traditional storage devices like HDD is still not good. Compared with FileStore, its small-sized random I/O performance is limited. Compared with In-Place updates, COW can effectively avoid additional read operations and the potential risk of data corruption, but it also has drawbacks.

First, COW breaks the physical continuity of data distribution on the disks. After multiple COW, a long sequential read will likely become multiple short random reads. Read performance is critical, and it indirectly affects the performance of RMW. Therefore, storage systems that use COW are prone to performance bottlenecks. In addition, because COW requires frequent space reallocation and address pointer redirection, it will change more metadata [18]. The performance will be compromised if metadata cannot reside in the cache. On this basis, we improve the space allocation and recycling for block storage as follows:

- We fix the space mapping relationship and pre-allocate the fixed-size space before writing. As shown in Fig. 8, Ceph calculates and allocates the Extent strictly based on the offset and length of data and then maps it to the corresponding Pextent. FastStore allows RBD to allocate a fixed size of physical space to a new Blob as soon as it is created and associates the new Blob with the whole Pextent. The size of Pextent is set by the value of parameter target_blob_size, of which the default value is 512K. Thus, the total size of space allocation in DAW will no longer be the sum of the data lengths of all write items in the queue but the product of the number of write items that use a new Blob and 512K.
- The space management overhead is reduced by reusing the original Pextent of the Blob instead of reallocating the physical space when OW. Thus, there is one and only one Pextent of constant size and position for each Blob, no matter how often OW is performed.
- The garbage collection mechanism is improved to eliminate the need for redundancy determination and space release in advance for the Pextent that conflicts with existing data.



(a) BlueStore (b) FastStore

Fig. 8. Comparison of space allocation between BlueStore and FastStore.

C. Atomic Redundancy of Objects

Ceph stores data as objects on the underlying device. For the convenience of the discussion below, let's briefly introduce the metadata management and storage process of objects. Each object consists of data (stored as Bufferlist) and metadata (stored as Onode). As illustrated in Fig. 9, a logically contiguous data segment (Extent) serves as the basic data management unit. An Extent may correspond to multiple non-contiguous physical data segments (Pextent). The Onode contains the object id and an ExtentMap structure. The ExtentMap contains several Extents and identifies the mapping relationship between objects and Extents. The Blob serves as an intermediate structure for managing the mapping relationship between physical and logical data segments. Ultimately, the data is mapped to the underlying disk by Pextents. In the original COW mechanism of Ceph, data is not directly updated to the Pextent in the original location of the disk, but is reallocated to a free Pextent for writing. After data is written, the corresponding address pointer is updated and the old Pextent is released for data recycling.

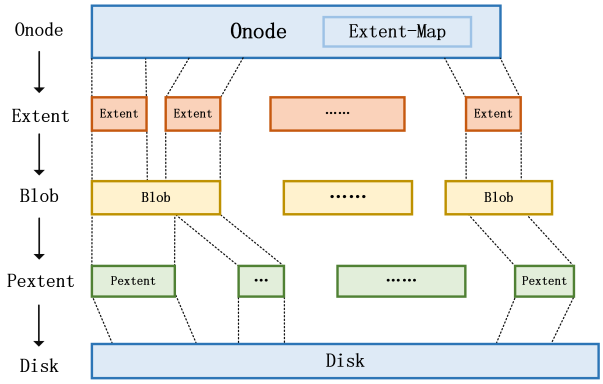


Fig. 9. Object storage structure of BlueStore.

The atomicity of transactions requires comprehensive consideration of data modification operations, ensuring either complete execution or no change at all, adhering to the principle of "All or Nothing" [19]. Ceph stores all data in the form of objects in a distributed cluster. Upper-level clients partition block data into fixed-sized objects, which are then mapped to underlying OSDs through the CRUSH algorithm. To ensure the accessibility and consistency of data updates, Ceph relies on the underlying RADOS for data replication, recovery, and dynamic scaling.

Ceph's data recovery involves two phases: replica negotiation consensus and data consistency. The first phase is the peering of placement groups (pg), bringing them to a peered state, focusing on replica consistency coordination within pg. The second phase involves pg recovery and backfill, aiming for a clean state and performing actual data recovery within pg. Ceph utilizes logs for data recovery, including two types: Pg log records all update operations on objects within the pg, capturing only the metadata information without the specific data. Simultaneously, the log of the underlying object storage backend ensures the atomicity of object writes, recording both metadata and data information. Ceph's data recovery relies on coarse-grained metadata log records based on pg.

In Ceph, objects are the basic unit of data, and block and file storage are built on top of objects. Ceph ensures data consistency by providing atomic operations at the object level. For RBD and CephFS, they are both built on Ceph Object Storage and therefore inherit the atomicity guarantee of Ceph Object Storage. Analyzing Ceph's dual mapping and atomicity write mechanism reveals that object atomicity assurance has no practical significance for block interfaces in distributed storage systems. This is due to the following reasons:

- During the mapping of block interfaces to object storage, Ceph's internal transaction atomicity requires read and write operations to be completed within the same object. Thus, it cannot support transactions involving cross-object reads and writes, contradicting the original intent of Ceph's block storage design. The goal of Ceph block storage is to provide a top-level interface consistent with traditional disks, allowing upper-layer applications to be unaware of the restrictions between IO scope and object scope.
- In the stage where object storage is mapped to a block device, the underlying hardware itself does not guarantee atomicity. Since the underlying storage backend needs to provide independent transaction atomicity for each object, the lower-level block device passively inherits the data protection mechanism and atomicity strategy of the upper-layer RADOS objects, which is redundant for block devices.
- In cases of data anomalies such as inconsistent replicas, Rados does not use object storage backend logs for data recovery but relies on pg logs. Changes to object storage logs do not affect the existing data recovery process in Ceph.

In summary, the object atomicity guarantee mentioned above is redundant for the Ceph RBD interface. Due to Ceph's internal multi-layer abstraction and dual mapping mechanism, its object-based transaction mechanism struggles to fully leverage the performance advantages of underlying block storage devices. To address this issue, this article proposes a new Ceph storage backend called FastStore by eliminating the redundant atomicity guarantee for data writes. The main challenge of this solution lies in handling object atomicity while ensuring transaction support and better adapting to the requirements of the Ceph block interface to enhance overall performance.

Ceph block devices are called images in the storage pool. An image consists of several objects, including metadata objects and data objects. The metadata object is similar to the file system's root directory. If it is corrupted, the corresponding data object cannot be accessible. Because metadata objects occupy tiny space, their writing processes have little impact on performance. Therefore, we only identify and optimize the writing of data objects. FastStore identifies the object type by decoding the information received from OSD. If the keyword "rbd_data" is contained, the written object is RBD interface data. Thus, the process can be optimized without affecting the writing of CephFS and RGW interfaces. Compared to BlueStore, FastStore can accurately identify the RBD write in all operations and flexibly control data persistence.

V. Evaluation

We assess the performance of FastStore on various hardware configurations and validate its effectiveness through measurements of IOPS, latency, and stability.

A. Environmental setup

We implement the proposed design in Ceph 15.2.4. We modify around 1,700 lines of code to prototype our proposal. To ensure the 3-copy mechanism, we use five physical machines to build a cluster. One machine is set as both the Monitor node and Object-based Storage Device (OSD) node, with the other four machines as merely OSD nodes. All nodes are interconnected with each other by Gigabit Ethernet. The detailed configuration is shown in Tab. III.

TABLE III
Configuration of TestBed

Component	Configuration
CPU	Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz
Memory	Samsung M393A2K43BB1-CTD 32GB
Ethernet	Intel Corporation Ethernet Connection X722 for 1GbE
HDD	ST1000LM049-2GH172 1TB
SSD	Samsung SSD 860 250GB
NVM	Intel Optane DC persistent memory
OS	Ubuntu 20.04.1 LTS
Kernel	GNU/Linux 5.4.0-91-generic x86_64
Ceph	15.2.4 Octopus (stable)
FIO	3.16
Local Filesystem	ext4

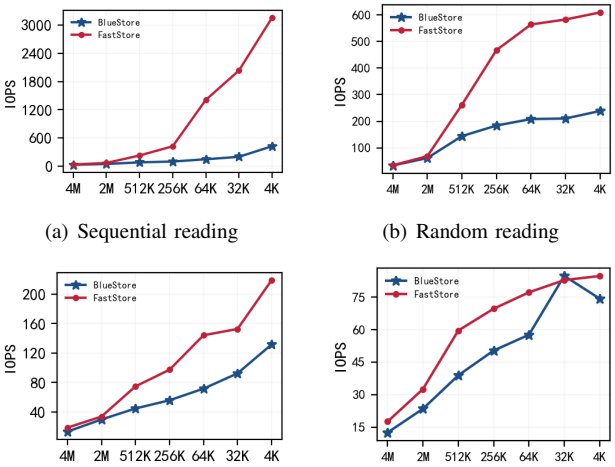
This article employs the widely used IO testing tool FIO on the Linux system for experimental verification. FIO is a workload generation and performance testing tool Jens Axboe developed for simulating block reads and writes. The tool can generate different types of test pressures based on various test engines and provide detailed performance data results. Ceph block storage, also known as RADOS block device, is typically directly mounted on the operating system as a raw disk for testing. This approach is closer to the client, and it allows the use of benchmarking tools in the operating system, providing more accurate performance characterization data.

Because FIO has multiple configurable parameters and high flexibility, its usage is relatively complex. Here's a brief introduction to the configuration parameters used in this article: the total data written in a single test is 100G. IO modes are configured as sequential read, sequential write, random read, and random write. The test engine selected is libaio. As a commonly used test engine for operating system block devices, libaio generates high parallel workloads on kernel block devices with relatively small overhead by invoking Linux kernel's asynchronous read/write interface. The queue depth is set to 32. To avoid the influence of the operating system cache, Direct IO mode is used, and the kernel's Buffer Cache is cleared during initialization.

B. IOPS and Throughput

In order to confirm the performance difference caused by our optimization, this section verifies the performance of FastStore in the distributed cluster. We compare it with BlueStore, a widely used storage backend. The experiment tests the performance under different data sizes, covering four scenarios: sequential read, sequential write, random read, and random write. Since the default size of the Ceph object in Ceph is 4M, the data size is less than 4M.

Fig. 10 shows the IOPS results on HDD. Fig. 16 shows the throughput and 99.99th latency results. FastStore outperforms BlueStore in almost all cases. As the I/O size decreases, the IOPS improves, and optimization becomes more and more significant.



(a) Sequential reading (b) Random reading

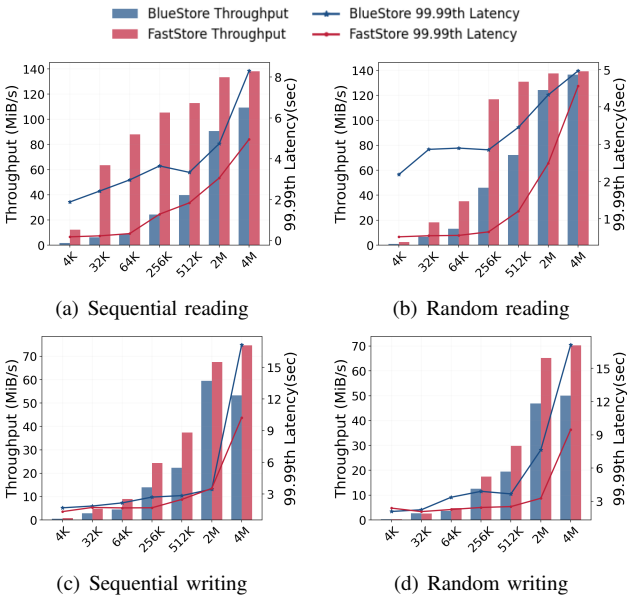
(c) Sequential writing (d) Random writing

Fig. 10. Comparison of IOPS between BlueStore and FastStore (HDD).

Compared with the BlueStore, FastStore pre-allocates space in advance, and In-Place updates data to avoid complex metadata processing and redundant journals. As a result, it reduces the time required for allocating space and indexing addresses and eliminates the write amplification caused by WAL, thus effectively improving the write performance. In the case of random write, the IOPS increase is about 40%. In the case of sequential write, FastStore doubles the performance.

Moreover, BlueStore changes the sequential access into random by performing multiple COW. FastStore eliminates the data location redirection when OW. It ensures data continuity in the physical space, improving reading performance, especially sequential reading. FastStore's random read performance is 2.7 times better than BlueStore's. When the data size is 32K, the sequential read IOPS of FastStore is up to 10 times that of BlueStore.

Unlike HDD, SSD must erase the existing data before OW. Frequent erasures generate many additional I/O, causing write amplification and shortening SSD's lifetime. To solve the problem of performance loss and device wear, SSD generally internally converts and processes SATA commands by the Flash Translation Layer (FTL). When OW occurs, read the existing data and update it to the internal DRAM. Then, set the original place to invalid and update new data in the new position [20]. Unlike SSDs, HDDs require many mechanical processes during random I/O operations [21].

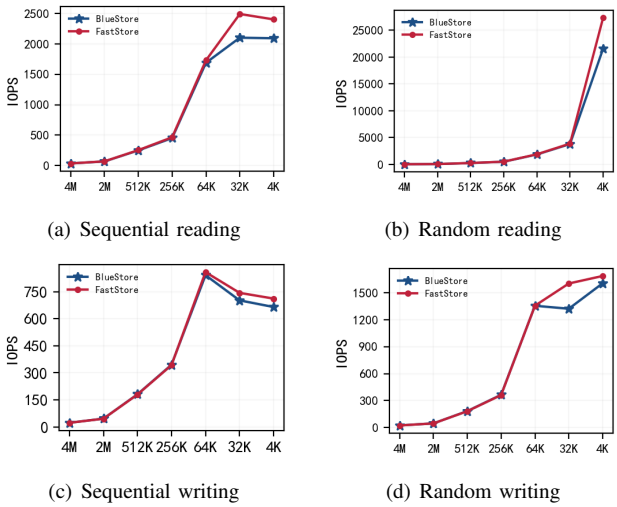


(a) Sequential reading (b) Random reading

(c) Sequential writing (d) Random writing

Fig. 11. Comparison of Throughput and 99.99th Latency between BlueStore and FastStore (HDD).

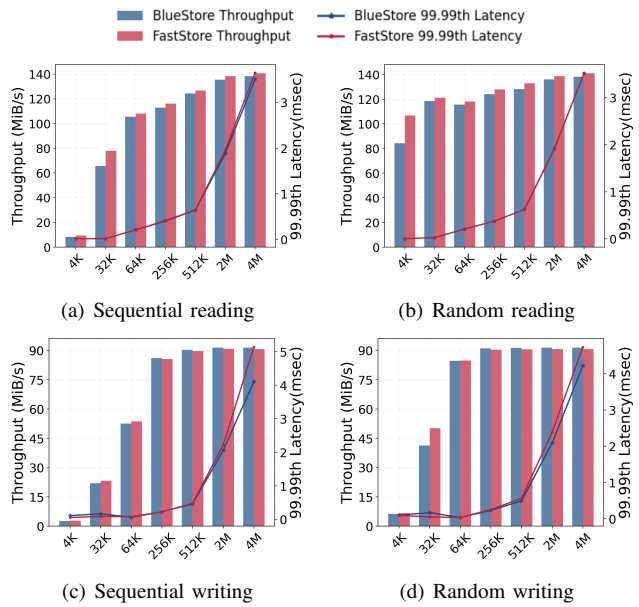
BlueStore cannot guarantee the continuity of continuous data when it is persistent. FastStore produces a lot of performance improvement on HDD by changing the COW to the In-Place update. However, as shown in Fig. 12 and Fig. 13, because the FTL redirects the write range to a new place, the optimization of our method is insignificant on SSD. Because sequential and random access do not significantly impact NVM's performance, the same as SSD, FastStore only slightly improves on NVM.



(a) Sequential reading (b) Random reading

(c) Sequential writing (d) Random writing

Fig. 12. Comparison of IOPS between BlueStore and FastStore (SSD).



(a) Sequential reading (b) Random reading

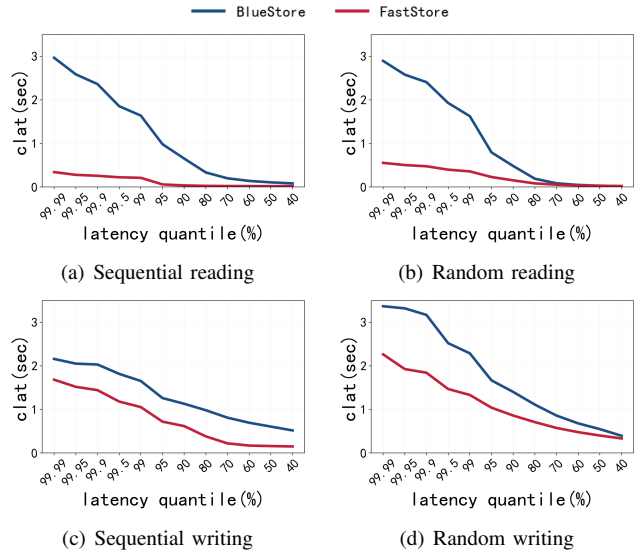
(c) Sequential writing (d) Random writing

Fig. 13. Comparison of Throughput and 99.99th Latency between BlueStore and FastStore (SSD).

However, the significant demand for storage capacity leads to HDD's irreplaceable position. It is still the mainstream storage device in cloud computing [14]. FastStore ensures data reliability and simplifies the redundancy atomicity between objects and blocks, effectively improving the I/O performance of block storage service in large-scale distributed clusters.

C. Latency and fluctuation

In the experiment described in V-B, we also test the latency of two storage backends. Fig. 14 shows our results, and its abscissa represents the latency percentiles.



(a) Sequential reading (b) Random reading

(c) Sequential writing (d) Random writing

Fig. 14. Comparison of latency between BlueStore and FastStore (HDD).

Compared to BlueStore, FastStore can reduce sequential write latency by 73% and random write latency by 71%. Additionally, FastStore markedly alleviates tail latency in reading, decreasing the 99th percentile latency for sequential and random reads by 11% and 19%, respectively.

In order to verify the performance fluctuations caused by different write schemes, we conducted a repeat of the second experiment presented in Section III-B using the RBD images provided by FastStore and BlueStore, respectively. The results are depicted in Fig. 15. Each time a write operation is performed, the standard deviation of IOPS values for different schemes is presented in Tab. IV. Since FastStore undergoes the same NEW process as BlueStore initially, the first write closely resembles the experiment in Section III-B. However, in the subsequent four iterations, the optimized backend simplifies the process of OW. Consequently, different write schemes implement similar In-Place update processes, and the values of IOPS remain stable. This indicates that FastStore not only brings about significant performance improvements but also effectively mitigates the performance fluctuations caused by multi-layer abstraction.

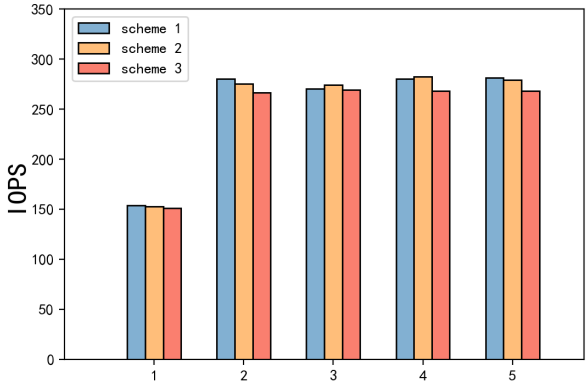


Fig. 15. The performance fluctuation in FastStore.

TABLE IV
STANDARD DEVIATION OF PERFORMANCE FLUCTUATIONS

Test Time	1	2	3	4	5
BlueStore	1.42	123.06	122.67	122.79	122.92
FastStore	1.42	7.09	2.65	7.57	7.00

D. Real workload

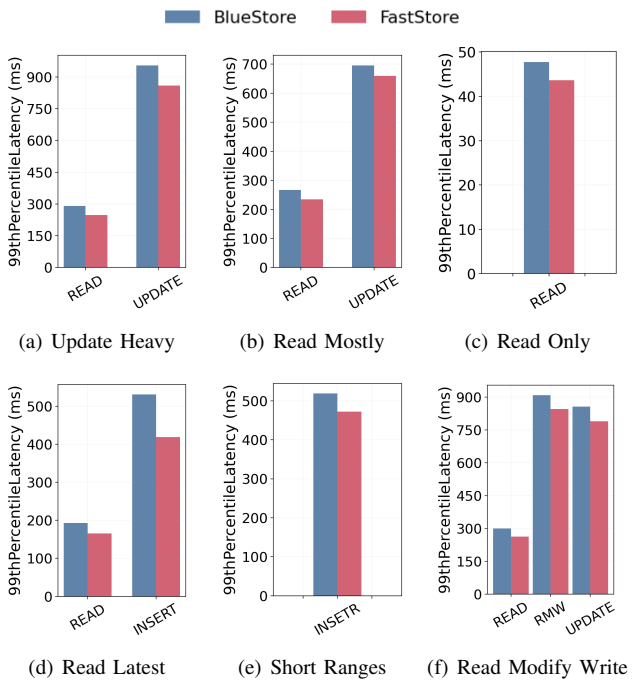
To evaluate the proposed design under the real workload, we run YCSB on the client nodes using block device. Each client uses increased record count (10000) and thread number (10 threads) as a YCSB configuration parameter. Fig. 16 shows the result of YCSB.

Workload a is a mixed workload with an even distribution between read and write operations (50:50 ratio). Workload b, also a mixed workload, leans heavily towards reading, with a ratio of 95:5 (write once, read many). Workload c is designed as a read-only workload, exclusively involving

read operations. Workload d shares similarities with Workload b, but it specifically concentrates on accessing recent data. Additionally, Workload e involves short-range queries, simulating scenarios where data retrieval within specific ranges is crucial. Finally, Workload f revolves around the read-modify-write paradigm, combining read, modify, and write operations.

When examining the results in Fig. 16, under the six aforementioned workloads, FastStore exhibits a 99.99th percentile latency significantly lower than BlueStore. This indicates that the In-Place update strategy proposed in this paper can enhance system performance in various application scenarios. However, the achieved latency by FastStore is not as optimal as depicted in V-B. This discrepancy can be attributed to two main factors.

YCSB issues small I/O update to the cluster. In BlueStore, it needs to write the data besides original place and update the metadata, the amplification is high. While in FastStore, it uses pre-allocate and In-Place update to accelerate small I/O updates. Consequently, FastStore necessitates overwriting data before writing it to conform to the block size, leading to the occurrence of RMW in the object store. Notably, FastStore's primary optimization focus does not encompass RMW scenarios. Secondly, the discrepancy is attributed to read/write control issues. YCSB tests primarily involve mixed workloads. A write operation in FastStore must wait for the completion of numerous read requests.



(a) Update Heavy (b) Read Mostly (c) Read Only

(d) Read Latest (e) Short Ranges (f) Read Modify Write

Fig. 16. Comparison of YCSB between BlueStore and FastStore (HDD).

VI. Related Work

Ceph provides block service, file service, and object storage service, meeting all storage requirements of the cloud computing platform. It is widely used in cloud computing platforms [13]. Performance optimization is a crucial consideration for applications running on cloud computing systems. Most related works focus on improving its data concurrency capability, reducing the overall access latency of the system, and ensuring the scalability and high availability of the system [22]. The community keeps optimizing by developing more efficient storage backends in the last decade [8]. The performance of virtualized storage services is much slower than that of physical devices. Optimization methods can be divided into four categories: 1) reduce write amplification. 2) optimize for new devices. 3) optimize for special workloads. 4) schedule architecture.

Reduce write amplification: Write amplification is a common issue in distributed storage systems, caused by replicas, metadata, and Write-Ahead Logging (WAL). Lee investigated the write amplification of several Ceph storage backends, including FileStore and Bluestore; the write amplification of all storage backends was more than 13x [9]. To alleviate the write dependency overhead for high-performance drives, [23] introduced a new IO stack called HORAE, which improved performance by 2.1 times compared to BlueStore. However, HORAE primarily focuses on optimizing SSDs and does not involve the optimization of HDDs. [24] enhanced FileStore by disabling WAL in the RBD, reducing write amplification by half and bringing 3-5 times improvement in IOPS and 4 times in bandwidth. However, this method was implemented for FileStore and had limitations. [25] reduced access latency of metadata in MDS by using cache. Although it improved the write performance of metadata, the backup strategy led to additional write amplification, and it could cause consistency problems between different MDSs.

Optimize for new device: Focusing on the insufficient utilization of persistent memory (PMEM) caused by aligned and non-aligned writes in BlueStore, MixStore stored small non-aligned data in PMEM and large aligned data on SSD, eliminating write amplification caused by the Write-Ahead Log (WAL) [26]. However, for large data reads, more data was read from SSD than PMEM, and hence, the throughput was not significantly improved. OCStore was a storage backend designed for Open-channel SSD [27]. By reducing the redundancy mechanism of RGW and Ceph FS, it could reduce 70% redundant writes in high-workload situations. However, since it was designed based on special hardware, only a few commercial products could be supported, making this solution not universal. [2] optimized thread control and proposed a CPU-efficient storage backend. Through efficient utilization of NVMe SSD, it provided more than three times performance improvement for small random writes.

Optimize for sepical workloads: Tirthak Patel clarified the proportion of three types of intensive files through data collection and analysis, optimizing the data placement strategy [28]. [29] optimized the writing of large files by splitting and using multiple threads. However, this modification affected the original data organization of Ceph. Sapphire found the best parameter configuration for specific applications through machine learning and black-box techniques, improving Ceph's

performance up to 2.2 times [30]. [31] collected various metric information in the Ceph FS, used as a standard to adjust the workload of each active thread, thereby improving system performance. To solve the concurrent access of threads, [32] designed a lock contention messenger (Async-LCAM), which prevents message corruption, connection interruption, and thread deadlock in scenarios of multi-thread sharing. Kisik Jeong found and analyzed the problems of running HPC workloads in Ceph and solved them based on F2FS and some other file systems [33]. However, Ceph uses BlueStore by default now, which no longer relies on the local file system to manage data.

Schedule architecture: To solve the unbalanced reading problem under replica mode, [34] designed a new selection strategy that effectively reduces the response time for sequential reads. However, performance bottlenecks occurred in large-scale clusters due to network limitations. The MapX algorithm [35] addressed the problem of cluster expansion by marking timestamps for each node and virtualizing different node subsets. Although MapX could reduce the performance penalty of data migration when adding a large number of nodes, for multiple additions of a small number of nodes, it caused cluster segmentation, reducing the performance and availability of the entire system. Crimson is a new generation OSD backend storage designed for fast storage media such as SSDs and NVMe [11]. It rewrites the IO path using SeaStar to decrease the overhead of traditional multi-threaded programming models. Since the optimization environments and methods are different, FastStore can achieve better performance on HDDs than Crimson. However, Crimson can achieve better performance on SSDs/NVMs than FastStore.

VII. Conclusion

In this paper, we have analyzed the performance overhead and fluctuation of distributed block storage systems in Ceph. Based on our observations, we present a new storage backend, FastStore. It is realized by preallocating sequential space, reducing redundant I/O operations and metadata operations. Extensive evaluation results show that our solution significantly improves I/O performance compared to Ceph. The sequential reading performance of FastStore is up to 10 times faster than Ceph. The sequential writing performance is twice as fast as before. The random writing performance increases by 40%. Furthermore, latency and fluctuation are also effectively reduced with FastStore.

Acknowledgment

We sincerely thank our shepherd Qirui Yang and the anonymous reviewers for their valuable feedback. We also thank Simeng Zhang for the discussion on this work. This work is supported by National Key Research & Development Program of China (Grant No. 2022YFB2702101), Shaanxi Key Industrial Province Projects (2021ZDLGY03-02, 2021ZDLGY03-08),the National Natural Science Foundation of China (Grant No.92152301,62272394).

References

[1] J. Shafer, "I/o virtualization bottlenecks in cloud computing today," in *Proceedings of the 2nd conference on I/O virtualization*. USENIX Association, 2010, pp. 5–5.

[2] M. Oh, J. Park, S. K. Park, A. Choi, J. Lee, J.-H. Choi, and H. Y. Yeom, "Re-architecting distributed block storage system for improving random write performance," in *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2021, pp. 104–114.

[3] M. Oh, S. Park, J. Eom, S. Kim, S. Kim, K.-w. Lee, and H. Y. Yeom, "Lalca: Locality-aware lock contention avoidance for nvme-based scale-out storage system," in *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2018, pp. 1143–1152.

[4] J. Yang, J. Izraelevitz, and S. Swanson, "Orion: A distributed file system for {Non-Volatile} main memory and {RDMA-Capable} networks," in *17th USENIX Conference on File and Storage Technologies (FAST 19)*, 2019, pp. 221–234.

[5] J. Zhang, M. Kwon, D. Gouk, S. Koh, C. Lee, M. Alian, M. Chun, M. T. Kandemir, N. S. Kim, J. Kim *et al.*, "{FlashShare}: Punching through server storage stack from kernel to firmware for {Ultra-Low} latency {SSDs}," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 477–492.

[6] 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*, 2006, pp. 307–320.

[7] OpenStack Survey Report. https://www.openstack.org/analytics.

[8] A. Aghayev, S. Weil, M. Kuchnik, M. Nelson, G. R. Ganger, and G. Amvrosiadis, "File systems unfit as distributed storage backends: lessons from 10 years of ceph evolution," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 2019, pp. 353–369.

[9] D.-Y. Lee, K. Jeong, S.-H. Han, J.-S. Kim, J.-Y. Hwang, and S. Cho, "Understanding write behaviors of storage backends in ceph object store," in *Proceedings of the 2017 IEEE International Conference on Massive Storage Systems and Technology*, vol. 10, 2017.

[10] S. A. Weil, S. A. Brandt, E. L. Miller, and C. Maltzahn, "Crush: Controlled, scalable, decentralized placement of replicated data," in *SC'06: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*. IEEE, 2006, pp. 31–31.

[11] S. Just, "Crimson: A new ceph OSD for the age of persistent memory and fast NVMe storage." Santa Clara, CA: USENIX Association, Feb. 2020.

[12] S. Kai, S. Park, and Z. Meng, "Journaling of journal is (almost) free," in *Proceedings of the 12th USENIX conference on File and Storage Technologies*, 2014.

[13] S. Wu, S. Tao, X. Ling, H. Fan, H. Jin, and S. Ibrahim, "ishare: Balancing i/o performance isolation and disk i/o efficiency in virtualized environments," *Concurrency and Computation: Practice and Experience*, vol. 28, no. 2, pp. 386–399, 2016.

[14] D. Reinsel, J. Gantz, and J. Rydning, "Data age 2025: the evolution of data to life-critical," *International Data Corporation (IDC) White Paper*, vol. 1, no. 1, pp. 1–25, 2017.

[15] A. Aghayev, S. Weil, M. Kuchnik, M. Nelson, G. R. Ganger, and G. Amvrosiadis, "The case for custom storage backends in distributed storage systems," *ACM Transactions on Storage (TOS)*, vol. 16, no. 2, pp. 1–31, 2020.

[16] W. Li and B. Tremblay, "Storage benchmarking for workload aware storage platform," in *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*. IEEE, 2016, pp. 835–838.

[17] E. Lee, Y. Han, S. Yang, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "How to teach an old file system dog new object store tricks," in *10th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 18)*, 2018.

[18] W. Lee, K. Lee, H. Son, W.-H. Kim, B. Nam, and Y. Won, "{WALDIO}: Eliminating the filesystem journaling in resolving the journaling of journal anomaly," in *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, 2015, pp. 235–247.

[19] D. B. Terry, V. Prabhakaran, R. R. Kotla, M. Balakrishnan, M. K. Aguilera, and H. Abu-Libdeh, "Consistency-based service level agreements for cloud storage," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, 2013.

[20] S. Maneas, K. Mahdaviani, T. Emami, and B. Schroeder, "A study of {SSD} reliability in large scale enterprise storage deployments," in *18th*

USENIX Conference on File and Storage Technologies (FAST 20), 2020, pp. 137–149.

[21] H. H. Huang, S. Li, A. Szalay, and A. Terzis, "Performance modeling and analysis of flash-based storage devices," in *2011 IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST)*, 2011, pp. 1–11.

[22] X. Zhang, S. Zhang, J. Shi, C. Dong, and Z. Li, "Ceph distributed storage system performance optimization review," *Computer Science*, vol. 48, no. 2, pp. 1–12, 2021.

[23] X. Liao, Y. Lu, E. Xu, and J. Shu, "Write dependency disentanglement with {HORAE}," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 549–565.

[24] X. Zhang, Y. Wang, Q. Wang, and X. Zhao, "A new approach to double i/o performance for ceph distributed file system in cloud computing," in *2019 2nd International Conference on Data Intelligence and Security (ICDIS)*. IEEE, 2019, pp. 68–75.

[25] Z. Ling, X. Fang, and D. Li, "The research and implementation of metadata cache backup technology based on ceph file system," in *IEEE International Conference on Cloud Computing & Big Data Analysis*, 2016.

[26] Y. Tu, Z. Chen, Y. Han, B. Chen, and d. Guan, "Mixstore: persistent memory and ssd-based backend storage," *Computer Research and Development*, 2021.

[27] Y. Lu, J. Zhang, Z. Yang, L. Pan, and J. Shu, "Ocstore: Accelerating distributed object storage with open-channel ssds," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2019, pp. 271–281.

[28] T. Patel, S. Byna, G. K. Lockwood, N. J. Wright, P. Carns, R. Ross, and D. Tiwari, "Uncovering access, reuse, and sharing characteristics of {I/O-Intensive} files on {Large-Scale} production {HPC} systems," in *18th USENIX Conference on File and Storage Technologies (FAST 20)*, 2020, pp. 91–101.

[29] K. Zhan, L. Xu, Z. Yuan, and W. Zhang, "Performance optimization of large files writes to ceph based on multiple pipelines algorithm," in *2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*. IEEE, 2018, pp. 525–532.

[30] W. Lyu, Y. Lu, J. Shu, and W. Zhao, "Sapphire: Automatic configuration recommendation for distributed storage systems," *arXiv preprint arXiv:2007.03220*, 2020.

[31] Y. Han, K. Lee, and S. Park, "A dynamic message-aware communication scheduler for ceph storage system," in *Foundations & Applications of Self Systems, IEEE International Workshops on*, 2016.

[32] B. Jeong, A. Khan, and S. Park, "Async-lcam: a lock contention aware messenger for ceph distributed storage system," *Cluster Computing*, vol. 22, no. 2, pp. 1–12, 2019.

[33] K. Jeong, C. Duffy, J.-S. Kim, and J. Lee, "Optimizing the ceph distributed file system for high performance computing," in *2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. IEEE, 2019, pp. 446–451.

[34] Y. Wang, M. Ye, Q. He, Y. M. Huan, and W. J. Kang, "A new node selecting approach in ceph storage system based on software defined network and multi-attributes decision-making model," *Chinese Journal of Computers*, 2019.

[35] L. Wang, Y. Zhang, J. Xu, and G. Xue, "{MAPX}: Controlled data migration in the expansion of decentralized {Object-Based} storage systems," in *18th USENIX Conference on File and Storage Technologies (FAST 20)*, 2020, pp. 1–11.