Performance Optimization for All Flash Scale-out Storage

Myoungwon Oh*, Jugwan Eom*, Jungyeon Yoon*, Jae Yeun Yun†, Seungmin Kim*
*SDS Tech. Lab, †Storage Tech. Lab,
SK Telecom
Email: {omwmw,jugwan.eom,jungyeon.yoon,jaeyeun.yun,seungminkim}@sk.com

Heon Y. Yeom
School of Computer Science & Engineering,
Seoul National University
Email: yeom@snu.ac.kr

***Abstract*—The proliferation of the big data analysis and the wide spread usage of public/private cloud services make it important to expand the storage capacity as the demand is increased. The scale-out storage is gaining more attention since it can inherently provide scalable storage capacity. The flash SSD, on the other hand, is getting popular as the drop-in replacement of the slow HDD, which seems to boost the system performance somewhat at least. However, the performance of traditional scale-out storage system does not get much better even though its HDD is replaced with the flash based high performance SSD since the whole system is designed based on HDD as its underlying storage device. In this paper, we identify performance problems of a representative scale-out storage system, Ceph, and analyze that these problems are caused by 1) Coarse-grained lock, 2) Throttling logic, 3) Batching based operation latency and 4) Transaction Overhead. We propose some optimization techniques for flash-based Ceph. First, we minimize coarse-grained locking. Second, we introduce throttle policy and system tuning. Third, we develop non-blocking logging and light-weight transaction processing. We found that our optimized Ceph shows up to 20 times improvement in the case of small random writes and it also shows more than two times better performance in the case of small random read through our experiments. We also show that the system exhibits linear performance increase as we add more nodes.**

1. Introduction

As we meet the big data era, the need for data capacity is increasing exponentially and the demand for efficient storage system is getting out-of-bound. Hence the scale-out storage is getting more attention compared to the traditional scale-up storage. Generally the scale-up storage system is composed of two data managing controllers and an assembling structure joining storage array and SAN. However, this structure has limitation that its bandwidth is limited by the capacity of the controllers. If the demanding bandwidth is more than what the controllers can provide, the whole system has to be re-built from the scratch. Unlike the scale-up structure, the storage system of scale-out structure can expand its capacity via commodity server expansion as the demand is increased.

As Amazon and Google successfully operate private/public cloud service and other successful application cases based on these services are increasing, many companies either construct and use cloud infrastructure including scale-out storage or provide these services. The typical examples of the scale-out storage system are Swift [4], Ceph [3] and Glusterfs [5]. Ceph gets the most spotlight nowadays [2] because Ceph can provide various storage types as needed by the current storage system. For instance, Ceph can provide block storage service when it supports VM based infrastructure, it can provide object storage service when it supports simple storage service, and it can also provide traditional POSIX based file system interface.

However, these current scale-out system such as Ceph is designed with HDD as its basis, which is good for large-sequential data access [1], [6]. Therefore, their performance can be scalable when the data access pattern is sequential. For random access pattern, their performance have a lot to be desired. This is not suitable for the scale-out structure because scale-out system needs to be scalable regardless of the underlying workload pattern and it does not work well with the present data processing systems [7], [8], [18] which require handling a lot of small data. These performance problem is due to not only the scale-out storage system design which is based on HDD [9], but also the limitation of the HDD itself, which is its high latency due to seek overhead and inherent sequential nature. Hence the strategy of replacing HDD with SSD has increasingly been adopted in order to improve random performance. However, the drop-in replacement strategy does not work well in reality. For example, previous studies brought up performance reduction problems and suggested solutions when DAS (direct attached storage) format was used [11], [12]. However, different structure such as NAS (network attached storage) introduces different problems and a new solution might be required. According to our experiments, the sequential I/O performance of all flash Ceph reached its capacity but its random I/O performance was less than 10K IOPS for write

2168-9253/16 $31.00 © 2016 IEEE
DOI 10.1109/CLUSTER.2016.11

only workload. The other current scale-out system, Gluster filesystem has the same problem. If all flash scale-out system can provide good sequential I/O performance only, adding more HDDs would be a better solution without the need to use SSD at all.

In this paper, we have identified the performance problem of the representative open source scale-out system, Ceph, with all flash media and proposed solutions to mitigate those problems. First of all, we minimize coarse-grained locking to exploit the parallelism of the SSD. Ceph employs many locks for consistency. These locks have virtually no adverse effect with HDD since HDD has a very large response time. SSD, on the other hand, has faster response time and these locks prevent Ceph to fully utilize the SSD. We alleviate these heavy locks to guarantee better performance. Second, we optimize internal throttle logic. Throttle logic is needed for rate limiting of dispersed file system components. When this part is not unified, many problems can occur. Therefore, we found optimization method for this part and improved performance. Third, we tackle the logging system. Ceph mainly uses logging for debugging purposes. However, these logging poses severe overhead when HDD is replaced with SSD since it is now in the critical path. We solved this problem by making the logging non-blocking. Fourth, we lightened transactions. We were able to reduce I/O overhead and make read and write do not occur concurrently. With these optimizations, we have obtained 20 times better performance with small random write (4K) workload. We also observed two times better performance in the case of random reads (4k). In addition there is now a positive linear relationship between the number of nodes and the performance. There is a commercial all flash scale-out storage which solved these performance issues to provide intrinsic performance via similar study with ours. SolidFire [10] is a representative example of a commercial all-flash scale-out storage and it provides deduplication/compression, QoS with the performance of random read 200K and 100K random write. However, SolidFire gives us low sequential performance due to fragmentation. We have developed a new Ceph which has similar random I/O performance as SolidFire and superior sequential performance via this study.

Our optimizations are based on existing approaches [26] [27] [28] [29]. However, besides using these ideas, we enhance the pending queue to minimize coarse-grained locks. Also, in light-weight transaction, we employ write through caching and remove additional software overheads (system call, lock). We integrate all these approaches and evaluate the impact of each of them on a common test bed. Based from our experience, we also discovered other overheads in the case of throttling and system tuning and fixed these as well.

2. Problem analysis

2.1. Performance

Figure 1 shows the result of 4K Random write/read performance using a Ceph cluster with all flash SSDs without

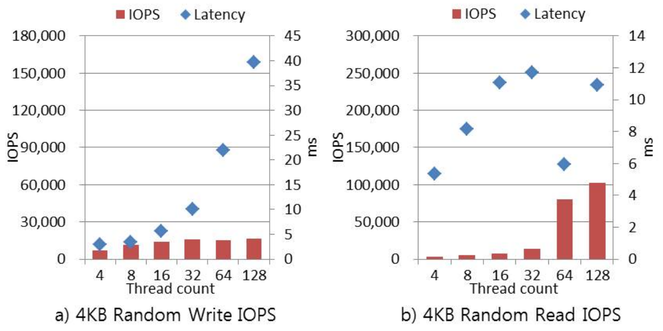


Figure 1: Performance of Ceph using SSDs

any modification in Ceph. We used 4 server nodes equipped with 40 SATA3 SSDs (80% of a whole disk are filled by the data). The details of the experimental environment can be found in Section 4. To test the performance of RBD (RADOS block service), clients are connected to OSDs (Object storage device) via KRBD (Kernel Rados Block Device, Linux kernel module for Ceph block device) module which can export block device to clients.

We observed the following from the result of random I/O performance. First, the random write performance only reaches up to 16K IOPS, even though clients generate enough load by increasing the number of threads. When the number of threads is increased more than 32, the latency is increased sharply with the IOPS remaining virtually the same. Second, In random read case, the number of threads affects the performance. With less than 32 threads, Ceph showed low IOPS and high latency. However, with 64 thread, latency is getting better even through contention is increased. We believe this phenomenon is caused by the structure of Ceph which employs batching based design to fully utilize the HDDs.

2.2. I/O flow on Ceph

Figure 2.(a) shows overall I/O structure in Ceph. Ceph is an object based scalable file system. The objects are grouped by PG (placement group) and distributed using CRUSH algorithm [3]. Ceph guarantees strong consistency. Therefore, read operation is handled by primary OSD and write operation is handled sequentially. Strong consistency has a weak point that it has lower performance than eventual consistency [23] [24]. However, strict consistency is needed in order to support block storage service which requires reliability. Ceph uses splay replication [13] where ack is sent to client when all of the journal writing, both for the primary and replicas, is completed.

Figure 2.(b) shows detailed I/O structure in OSD (step 1 - 7). Since Ceph does not need communication between metadata server and client when processing data I/O by using CRUSH algorithm, we will look into only the communication which is occurred between OSD and client while

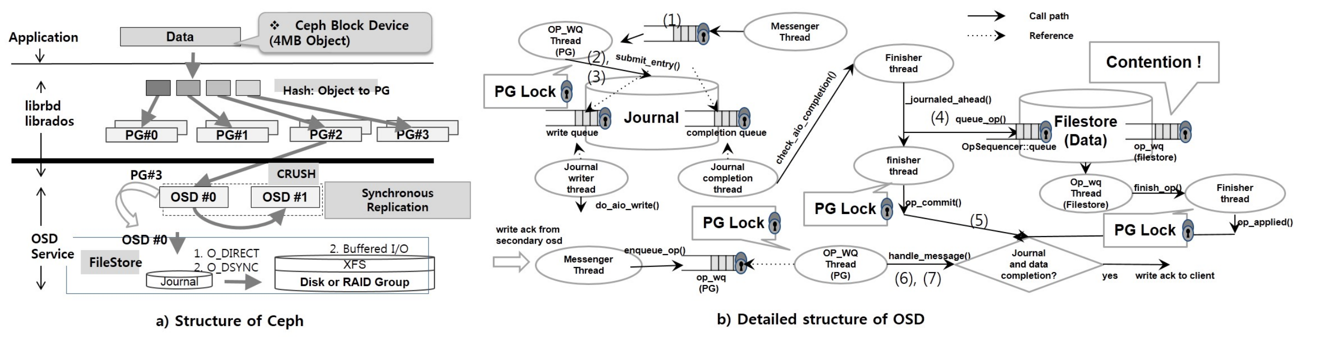


Figure 2: Ceph I/O Flow

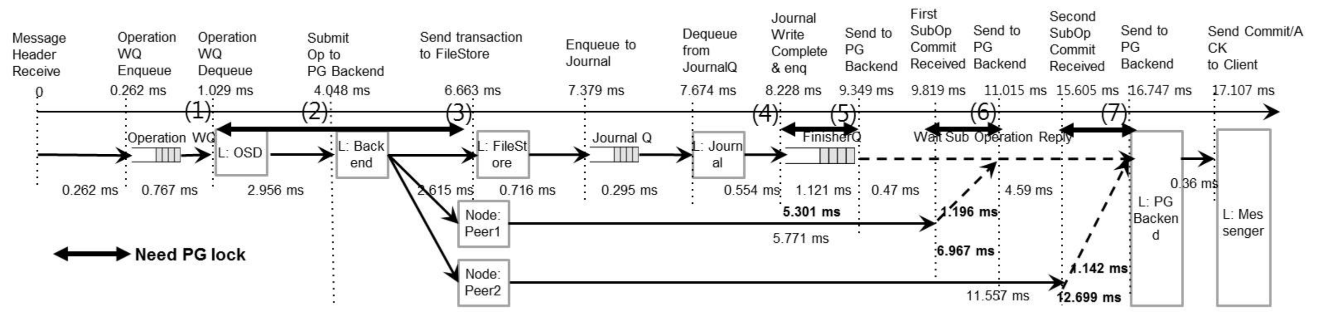


Figure 3: Ceph latency analysis for write path

processing write and read, not overall I/O path in the cluster. When OSD receives a write request from client, messenger thread which is in charge of processing network packets sends data requests to PG queues. PG is object group that composed of the objects (1). If a PG queue is filled up with enough requests, OP_WQ thread is woken up and processes data request after acquiring PG Lock. For a write request, OP_WQ thread writes the log, sends replication request to replicas and submits the write request to journal while holding the PG lock (2),(3). On the other hand, a read request is processed by issuing the read request to the filestore and relaying the reply back to the client without journaling. In the case of write, filestore makes a transaction which includes writing data, medata related with object and PG. The journal completion thread submits a write request to filestore after writing journal data (by direct I/O) (4), then data is written to filesystem asynchronously (by buffered I/O) obeying the write ahead logging principle. After writing is completed in the filestore (5), the primary OSD waits until it receives acks from replicas before sending the client an ack message (6),(7).

After careful analysis of Ceph I/O structure, we have found the following characteristics. 1) Each I/O request needs PG based lock and holds the lock for quite a long time. 2) Many operations such as completion processing for journal and writing data as well as ack processing also need PG lock resulting in considerable overheads. Therefore, the writing journal and completion processing can be delayed due to lock waiting which makes negative impact on performance.

2.3. Latency Measurement

Figure 3 shows the result of latency analysis in the write path (step 1 - 7). This figure depicts the control flow from the point when the message head is received to the point when the ack message is sent to the client. First, message processing in messenger thread takes about 1 ms. We believe that it is reasonable time to process a message (from start point to Operation WQ Dequeue, (1)) After processing the messages, the request is enqueued to the PG queue and it takes about 3 ms which is quite a long time. (from Operation WQ Dequeue (1) to Submit Op to PG Backend, (2)) This delay is mainly due to operations such as sending secondary OSD replication, making logs and reading metadata. Important thing is that these operations are processed while holding a PG Lock. (3) will be explained in Section 2.4. After completing journal writing (Journal Write complete & enq, (4)) which takes about 8.228ms, it again spends extra 1.1 ms to send the event of journal completion to PG backend (Send to PG Backend, (5)). This is because a single thread(finisher) handles all of the completion works for journal and it also needs PG Lock.

In addition, another 1.1 ms is needed when receiving commit event from replicas (from FirstSubOp & SecondSubOp Commit Received to Send to PG Backend, (6),(7)). We believe that the reason for these operations taking too

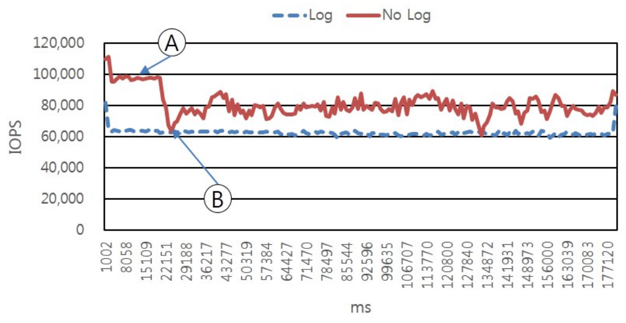


Figure 4: Performance comparison (Log vs No log)

long time is PG lock overhead because all the operations above need to acquire PG locks as shown in the Figure 2.b. From this analysis, we have observed that the delay caused by the PG lock can raise up to 9 ms when the total latency of a write request is about 17 ms.

2.4. Logging and filestore overhead

We found following two phenomena about logging during performance test. (This is not file system logging in Ceph, just for log that prints in /var/log.) First, random write performance is improved when turning off logging option. Second, sequential write performance is not changed even through logging is turned off. These results mean that current logging system of Ceph causes performance problems when low latency is needed.

Figure 4 shows the result when PG lock minimization and system tuning are applied. In the case of No log, high performance is sustained a few seconds at point A. Then, performance fluctuation begins from point B. We found that filestore queue as mentioned in Figure 2 (marked as contention) is growing as time passes. Therefore, fluctuation begins from point B. This is because the filestore can not process the I/O requests as fast as they arrive, which results in degraded OSD cluster performance. The contention in filestore prevents receiving any more input from clients since Ceph controls the number of operations doing journaling and writing. Therefore, requests are backed up until filestore is available (Figure 3.3) with PG lock.

3. Design and Implementation

3.1. Minimizing coarse-grained lock

As described in Section 2.3, current Ceph has huge overhead since PG based coarse-grained locking is used. Figure 5 and Figure 6 show the design of our optimization for PG lock. We focus on three parts discovered in Section 2 where the PG lock is held while waiting; processing requests from PG queue, processing completion for writing journal or filestore, and handling ack.

First of all, we additionally implement a pending queue in order to optimize process without unnecessary waiting.

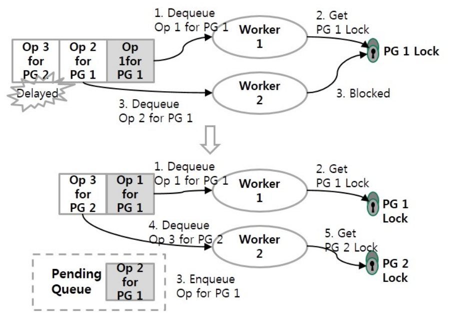


Figure 5: PG Lock based pending queue

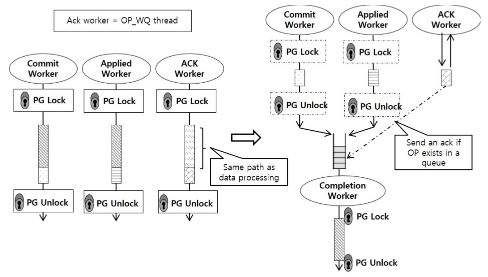


Figure 6: Reducing critical path and dedicated processing

As shown in Figure 5, when an I/O request is delivered from a client, it is enqueued to the appropriate PG queue by the worker (OP_WQ thread) after obtaining the corresponding PG lock. If another request belonging to the same PG arrives before the first request is finished, it has to be blocked since the necessary PG lock is already held by previous request, which in turn blocks the whole process until the PG lock becomes free. To alleviate this blocking, we introduce a pending queue for each PG queue so that the subsequent requests will not be blocked if they do not belong to the same PG. By introducing this pending queue, worker 2 can process OP 3 as described in Figure 5 because OP 2 is inserted in pending queue without blocking OP 3. The pending queue not only removes unnecessary blocking when multiple requests arrive at the same time but also protects data ordering per PG because data requests are enqueued to the pending queue sequentially. Therefore, the order between read and write or between write and write for the same PG is strictly enforced. Second, we reduced the size of the critical section governed by PG lock in case of processing completion and enhanced parallelism by adding batching based dedicated worker and OP lock. As shown in Figure 6, commit worker (responsible for journal completion), applied worker (data completion) known as finisher in Section 2, and ack worker all request PG lock. Each of these workers is modified to handle minimal operations while holding OP lock such as reference counting and the rest of the works which need PG lock are deferred. A dedicated completion worker is introduced to process these deferred operations in a batching manner. Third, we speed up the ack processing. From the latency analysis (in section 2), we found that the latency related to ack processing such as sending, receiving, logging and counting is high. This is because ack processing and data processing share the same path and compete with one another (OP_WQ thread in Figure 3), even though the only purpose of ack message is to notify completion of data request. The fact that the ack messages are enqueued to the PG queue significantly increases the lock contention. We modified ack processing, so that the ack messages are processed right away without enqueueing them to the PG queue.

Our design does not revise the entire lock scheme. Although highly tempted, we did not revise the entire PG lock scheme since it is the basis of the recovery system. PG lock protects data ordering and writing pg log sequentially. For instance, PG log is used to recover PG metadata which represents system status (via synchronizing PG status between Primary and Replicated OSD). Therefore, it should be written sequentially in order to do rollback to the previous state. PG lock also helps data ordering for strong consistency. Strong consistency, which is a core design principle in Ceph guarantees to read the most recently written data, therefore strict ordering is needed. If we totally revise the traditional lock scheme in Ceph, we can not guarantee system reliability including many critical operations which depend on this lock scheme as mentioned above. Therefore, we look for a solution which maintains data ordering per PG without changing basic lock scheme while reducing the overhead.

In our design, we bring out performance improvement via the following optimizations. 1) The messenger thread can process network packets faster from client because I/O requests can be queued without delay 2) Threads are utilized fully because the delay which caused by PG lock is minimized. 3) Faster response is possible because ack messages are processed without delay. 4) Parallelism is also enhanced because a dedicated thread handles delayed works. Multiple completion per PG can be processed at once.

Our design enhances latency while protecting data ordering per PG, resulting in no inconsistency. One weak point is that client can receive unordered write acks from OSD. This is because completion worker scheme (batching, fast return ack) may return unordered acks even though OSD writes data sequentially. Therefore, we added logic that sends client sequential acks if a client wants to receive ordered acks as requested. Completion worker can sort these unordered acks before sending them to clients.

3.2. Throttling and system tuning

In this section, optimizations for throttling and system tuning is explained. By system tuning, we mean not the code change but changing control options for TCP/IP stack or library configurations.

Minimizing coarse-grained lock explained in the previous section can decrease latency, but this was not sufficient. Performance fluctuation is still observed when I/O tests continue even decreased latency. It is suspected that fluctuations is caused by the throttle logic in Ceph because fluctuations are being repeated. However, this phenomenon is not fixed by changing one parameter. Performance degradation disappears only when combination of parameters for throttle are fixed together. The first parameter is filestore_queue_max_ops, which controls the maximum number of operations in journal and filestore, and the second parameter is osd_client_message_cap which controls the maximum number of messages that OSD can handle. Theses parameters are set based on HDD capacity. Thus performance degradation can occur even if Ceph is applied our optimization that can process request faster. Most of the distributed filesystems have throttle logic in order to support balanced performance or QoS, and export a tunable parameter to change the throttle policy easily. However, finding a suitable parameter is difficult without the knowledge of the internal I/O structure [14]. In particular, understanding the entire logic for throttle is needed because each part of Ceph (such as filestore, messenger and journal) have inherent throttle policy. Throttle parameter is determined as 30K IOPS, because A single block device (consist of 3 SSDs) can perform 30K IOPS in sustained state. Throttle parameter for journal has no impact because writing journal (NVRAM) is very fast.

In system tuning, the first optimization is changing memory allocator. As a result of CPU profiling using perf tool, we could tell most of CPUs used for Ceph was consumed by memory allocator. Current Ceph uses tcmalloc library without using memory pool internally. This is not a problem in large sequential workload. However, small random workloads need more responsibility and parallelism for memory handling than large sequential workload so memory allocator causes overhead in I/O path due to frequent allocation and release [22]. We replaced tcmalloc with Jemalloc since it outperforms other methods when dealing with small random workload. (among tcmalloc, jemalloc and standard malloc) The second optimization is to disable tcp nagle algorithm. In the performance as shown in Section 2.1, latency is high in the case when fewer requests are handled. This problem is caused by TCP nagle algorithm in Linux TCP/IP. Thus we simply disable this algorithm.(KRBD enable TCP nagle in Centos 7.0) TCP nagle algorithm is known to show good performance with large sequential workload but has worse performance with small random workload [21].

3.3. Non-blocking logging

As we mentioned in section 2.3, the performance of Ceph depends on whether logging (for debugging) is turned on or off, which means that logging system had considerable overhead. Performance is improved if logging is simply turned off. However, production systems need logging because it gives valuable information in case of system failure or checking system behavior.

Ceph log system has two different modes for logging. First, in-memory logging when the system writes logs in the memory unless there are some abnormal events. Second, filestore logging when the system writes logs to the storage

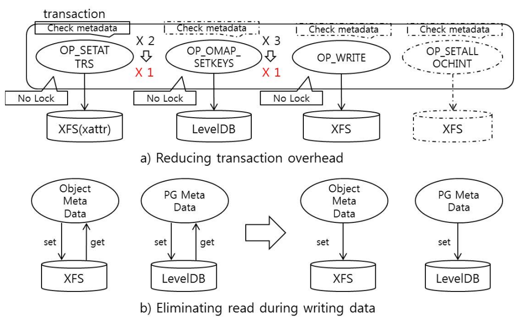


Figure 7: Light-weighted transaction

directly. In the case of in-memory mode, the system maintains a certain amount of log entries. Therefore, the first log entry is overwritten when the number of log entries reaches the limit. These logs contain the information regarding the progress of OSD operations. Although this is not critical, Ceph still waits for the logging to be completed before proceeding resulting in non-negligible delay even for memory writing. We made all the in-memory logging non-blocking without compomising the integrity of the Ceph system.

When an I/O request is processed in the internal OSD, log entries are created in each step of I/O path and are transmitted to the logging thread. However, logging is based on HDD style processing that has only a single write thread. Therefore, when small I/O is requested, the logging sometimes takes longer than the actual I/O itself.

We have changed all the logging from synchronous to asynchronous so that it will not be on the critical path anymore. The I/O request no longer waits for the logging. In addition, we made the single thread structure, which is responsible for logging, to be multi threaded so that parallel processing is possible to exploit the SSD characteristics. We also tacked the overheads related with string operations The way current logging is handled is to make a new log entry each time it is needed, where the log entry was constructed as string of words using standard library which results in quite an overhead. Because it causes memory allocation overhead as mentioned in Section 3.2. we introduced a log cache where the log entry strings can be stored and retrieved without making them over and over again if the same log is stored multiple times reducing the number of string operations as well as the new entry assignments.

Non-blocking logging used limited size of memory because OSD can process a certain amount of operation by throttling logic as mentioned in Section 3.2. A disadvantage of non-blocking logging is that the logs can be lost in the case of sudden system failure (power off). However this can also happen in existing logging system. Also, it can be easily rectified if NVRAM is used.

3.4. Light-weight transaction

As mentioned in Section 2.4, filestore in current Ceph is unable to handle high speed I/O and causes performance drop while handling write operation. Even though OSD sends ACK right after completion of writing journal and the performance of writing journal directly affects to the entire write performance, as explained earlier, eventually writing performance of filestore determines performance of write operation if writing speed in filestore is not sufficient because journal size is not infinite. For this reason, we make light-weighted transaction through optimization techniques as following. First, number of operation used in a transaction is reduced. A write operation is actually written as a transaction along with other metadata when Ceph writes the requested data. Figure 7 shows how requested data is stored in file system (OP_WRITE), PG log and OMAP data are stored (OP_OMAP_SETKEYS) in Key-value DB (Level-DB or Rocks DB), and object metadata is stored (OP_SETATTRS) as xattr in file system. The redundancy is removed and operations in this transaction is minimized. We also reduced system calls and operations that checks metadata to one time per each transaction. Second, the situation where read and write operations are requested simultaneously is prevented by using write through cache. As mentioned earlier in [15], there is considerable performance degradation when read and write requests are submitted at the same time to the SSD. To prevent this performance degradation, we used caching of objects and PG metadata to avoid reading metadata when write requests are being handled (Figure 7.b).

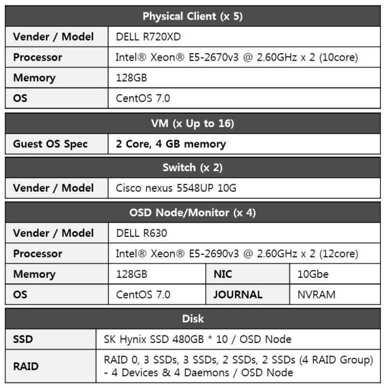
We have considered the following points in implementating the light-weight transaction. First, we reduced the use of Key-value DB and system calls. As described earlier, Ceph uses Key-value DB to write metadata. The amount of written data is small (around 12 $\sim$ 729 bytes), but it makes many operations in small sized I/O pattern. In this case, when write request is continued, latency of each requested operation becomes unstable because key-value DB performs compaction or construction of immutable table [16] [17]. Also, considerable amount of data is written additionally due to the write amplification inherent in LSM based Key-value DB. According to our observation, when a client writes a total of 2GB using 4MB block size, 30MB of additional data is written. However, if the block size is 4KB instead, 2GB of additional data is written. Performance degradation caused by this write amplification problem become much severe when SSD is in sustained state than in clean state. Because SSD in sustained state needs erasing and handling mapping table before writing. The solution is to minimize operations in a batching manner when transaction is written to Key-value DB. Because less operation can not only enhance responsiveness of Key-value DB but also results in less merge operations.

Also, we have minimized the number of system calls. Since various modules (Network, Key-value DB, Cache, File Store and Journal) use system calls, the total number of system calls are significant. Furthermore, various types of system calls such as (open, write, stat) are repeated to the same file especially in filestore. We believe this is to facilitate the readability and reusability of modules in Ceph code base. However, in terms of system performance, it is purely not necessary. We have removed all the redundant systems calls. Next, we have removed the system call set-alloc-hint (fallocate) from the path handling random workloads. Although it could be beneficial for sequential workloads, it is of no use for random access. When using HDDs, it was of no consequence since the random access is so slow for HDD. Second, we have removed reading meta-data from storage during write operation. Our performance test showed that considerable amount of read operations are always induced while handling write operation due to metadata (around 15MB/s per disk). General file system writes metadata along with writing data. However, reading metadata from storage is required if metadata is not cached. In this case, writing metadata needs Read-Modify-Write. In distributed file systems like Ceph, a greater amount of metadata than local filesystem is read because metadata of the distributed file system (such as PG Log, Rollback, Snapshot information etc.) that is loaded on local system is needed, besides metadata of local system itself. Thus, we avoid reading metadata from storage by maximizing the use of cache (write through) because most of the metadata exist in memory. Write through cache has an advantage that can avoid inconsistent state because data is written directly to storage. Even though SSD provides high IOPS and relatively better performance in mixed pattern, latency increases when load of reading metadata is piled up along with overhead of Key-value DB as described earlier. Therefore, minimizing reading operation is key to improving the performance of write operation.

Metadata cache has a disadvantage due to limited size. As object data is growing, caching has less effect because system can not manage all of the metadata. However most of distributed file system adapt 4 MB as default block size. In the case of 4MB block size, 10TB capacity needs 2.5GB if each object needs 1KB metadata (most of the object metadata are under 270 bytes in reality). Typical storage system uses less memory, because only a certain amount of memory is used for buffer. Thus, we think memory usage of metadata cache is reasonable. Actually, each OSD node of Ceph uses only less than 10GB memory (total memory usage, except for buffer cache) if the storage capacity of the Ceph cluster is completely utilized (each node has 5TB storage capacity).

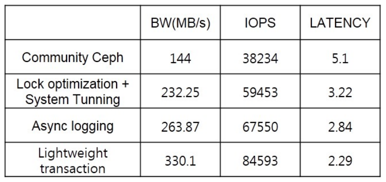
4. Evaluation

We have evaluated the throughput and latency of Solid-Fire, which is a commercial all flash scale-out system, Ceph 0.94.4 which is community version of Ceph and optimized Ceph (AFCeph) based on 0.94.1 community version and compared these three systems. We call optimized Ceph AFCeph. Our evaluation is based on block storage, but the performance of object and file storage also would be similar because OSD in Ceph, which is the main optimized part, is common for all three storage system.



Physical Client (x 5)			
Vender / Model	DELL R720XD		
Processor	Intel® Xeon® E5-2670v3 @ 2.60GHz x 2 (10core)		
Memory	128GB		
OS	CentOS 7.0		
VM (x Up to 16)			
Guest OS Spec	**2 Core, 4 GB memory**		
Switch (x 2)			
Vender / Model	Cisco nexus 5548UP 10G		
OSD Node/Monitor (x 4)			
Vender / Model	DELL R630		
Processor	Intel® Xeon® E5-2690v3 @ 2.60GHz x 2 (12core)		
Memory	128GB	NIC	10Gbe
OS	CentOS 7.0	JOURNAL	NVRAM
Disk			
SSD	SK Hynix SSD 480GB * 10 / OSD Node		
RAID	RAID 0, 3 SSDs, 3 SSDs, 2 SSDs, 2 SSDs (4 RAID Group) - 4 Devices & 4 Daemons / OSD Node		

Figure 8: Setup of the experiment



	BW(MB/s)	IOPS	LATENCY
Community Ceph	144	38234	5.1
Lock optimization + System Tunning	232.25	59453	3.22
Async logging	263.87	67550	2.84
Lightweight transaction	330.1	84593	2.29

Figure 9: Performance improvement with clean state SSDs (fio, direct, 4K random write)

4.1. Experimental setup

Figure 8 describes our experimental environment. The number of client node was five and each client node had up to 16 VMs and total of 80 VM were created and tested. We used PMC 8GB NVRAM product as journaling disk. Each OSD node used one NVRAM and the number of OSD daemons per node was 4 so each OSD daemon used 2GB for the journal usage. Also, each OSD node is with 10 SSDs and OSD 1∼4 uses 3,3,2,2 SSDs respectively and SSDs are tied up as RAID 0. If more than 4 OSD are used, we do not achieve performance gain because OSDs used significant CPU. We used FIO tool on each client for performance evaluation and set replication factor as 2. Clean state means that Ceph and SSD are not fully filled by the data and sustained state means that the SSD is already saturated by sufficient data.

4.2. Performance improvement

Figure 9 shows performance improvement for each optimization scheme as described in the previous sections. We used SSD of clean state to test and created block device (100GB per client) through KRBD. Note that this is clean performance (SSD, local filesystem and Ceph are initial state). Thus Community Ceph has better performance because the small image size causes less reading metadata. We can see the evaluation result of non-blocking logging and light-weight transaction including the optimization result of previous step. Our evaluation results show that the performance improvement is more than two times.

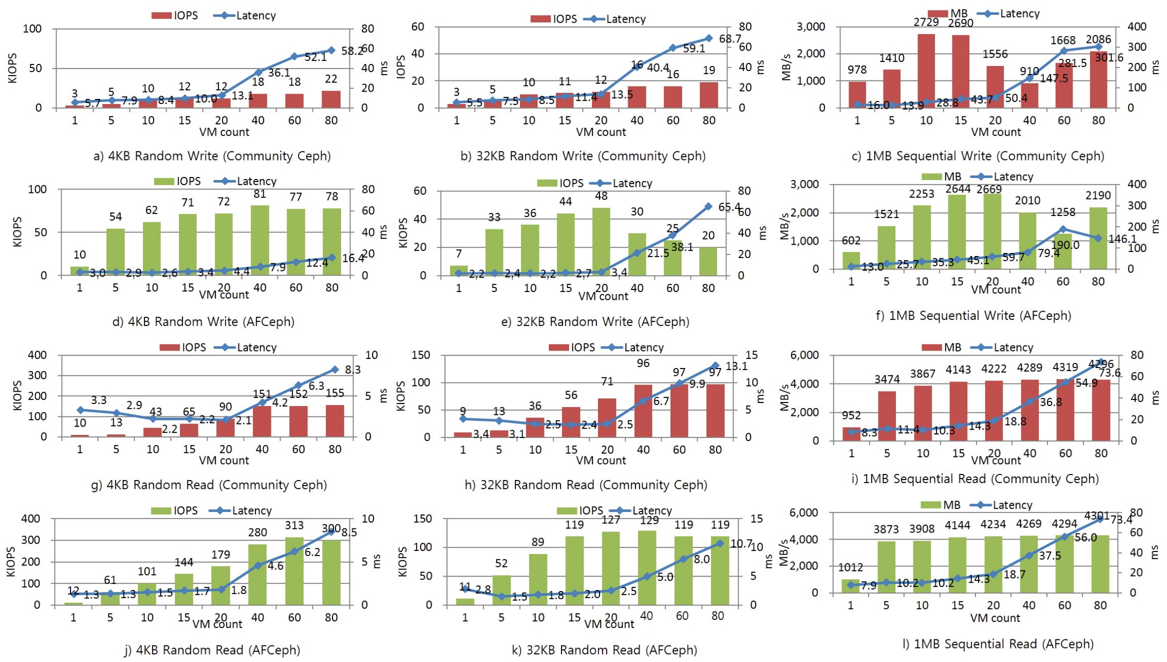


Figure 10: Vitual Machine performance comparision (Community Ceph: (a, b, c, g, h, j), AFCeph: (d, e, f, j, k, l))

4.3. Virtual machine performance

We increased the number of VMs on each client to test how much performance the Ceph cluster could achieve as the load was increased. In VM test, we created VMs on client nodes through KVM and made one block device (RBD) per VM. X axis is the number of total VMs. For example, 10 means 2 VMs per client node, 15 means 3 VMs per client node. Each VM had 1 image and we tested up to 80% of a whole disk capacity after we reached sustained state. The VM test result is shown in Figure 10. We summarize the results and select the best results from FIO test which is executed using increasing number of threads and iodepths.

4KB random write results show that the Community Ceph has 22 KIOPS as its maximum performance with latency of 58.2 ms in case of 80 VMs. Also, in the case of 40 VM cases or higher, the latency is increased rapidly because reading operation on metadata affected latency while the load is getting heavier. On the other hand, the AFCeph shows the maximum performance of 81 KIOPS and its latency was 7.9 ms. This figures show the AFCeph performed four times better with around 75% smaller in latency when it is compared to that of Community Ceph. Also, the latency is better on all points from 10 VM to 80 VM test. This means that latency reduction scheme in lock optimization and non-blocking logging had actual effects on reducing latency in the case of small sized I/O and avoiding read operation during write request in light-weighted transaction which prevents an explosive increase in latency. 32KB random write results show the AFCeph shows almost 4 times better performance with the smaller latency which is less than 5 ms compared to Community Ceph. The performance is declined for the AFCeph at the 40 VM case or higher because the journal is full. Therefore, flush operation is needed and fragmentation by bypassing system calls like set-alloc-hint for small sized block causing random workload. An NVRAM used as journal disk is faster than SSDs being used as filestore. If journal is full with its data, the system gets blocked until some of data in journal is flushed to filestore. As a result, performance fluctuation is observed. In Community Ceph, its slow performance does not generate journal data to fill up the NVRAM. In the sequential write case, Community Ceph and AFCeph show similar performance (except for VM 20, 40, 60 cases). In these results, the performance fluctuation is also observed and this is because NVRAM is full with journal data as explained above.

4KB random read performance results show the AFCeph shows higher IOPS with less latency than Community Ceph in the fewer numbers of VMs, i.e. under lighter load test in the case of less than VM 40. Under the heavy load test, the AFCeph shows similar latency with Community Ceph but 2 times higher IOPS than Community Ceph. Enqueuing optimization with pending queue makes IOPS is increased because the read requests of other PG can be processed without delay. Also, non-blocking logging and memory allocator changing causes read latency reduction. 32KB random read performance test shows the similar result to 4KB result and AFCeph is superior to Community Ceph. In the sequential read case, Community Ceph and AFCeph show similar performance.

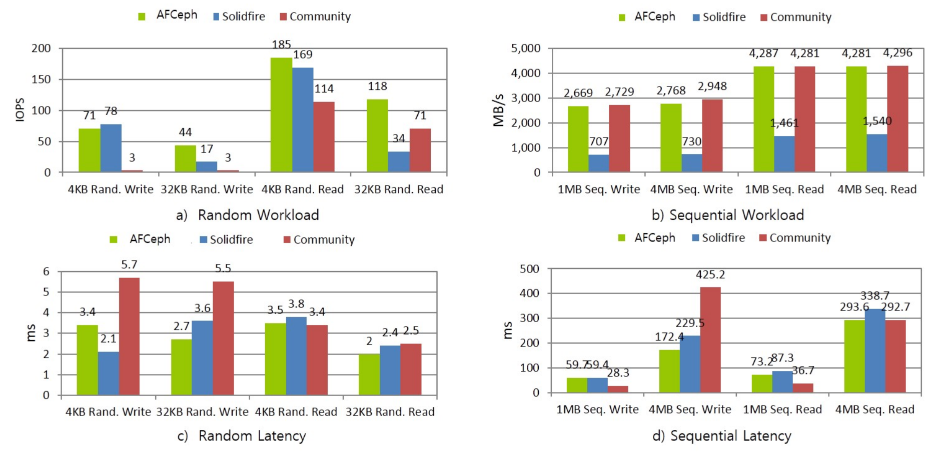


Figure 11: Virtual Machine max performance comparison: SolidFire vs AFCeph vs Community Ceph

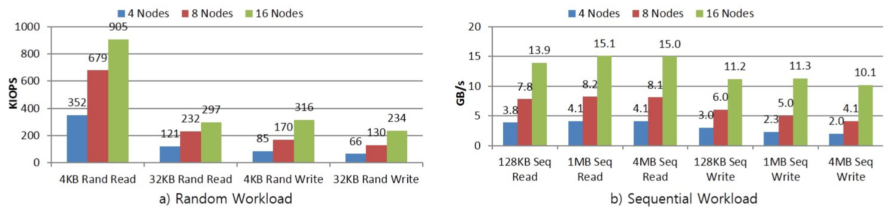


Figure 12: AFCeph scale-out test

4.4. SolidFire vs AFCeph

We wanted to see whether AFCeph has a reasonable performance compared to that of SolidFire which is one of the most competitive solution for All-Flash scale-out storage. SolidFire used the same configurations with Ceph for the fair comparison. It was consisted of 4 nodes and each node had 10 SSDs and a NVRAM. Network configuration is 10Gb x 2 bonding which is one more than in Ceph but public and private network are not separated. Deduplication is enabled (mandatory options). Fully random data is used for test in order to see real I/O performance. Therefore, the result of SolidFire includes overhead of deduplication processing. Figure 11 shows the best VM-based performance results considering IOPS and latency.

In a result of 4KB random write (Figure 11.a, c), the Community Ceph shows 3 KIOPS because we extract the values from minimal latency (5.7 ms) to compare to SolidFire and the AFCeph with similar latency. 3 KIOPS is almost the same as HDD-based Ceph. The AFCeph has 71 KIOPS with 3.4 latency. This result is less than SolidFire in IOPS (78K) and 1 ms higher in latency. But AFCeph has 20 times higher IOPS and 2 ms lower latency than Community Ceph. AFCeph has higher performance than both Community Ceph and SolidFire in the 32K random write case. SolidFire is optimized for using 4KB fixed chunk size for deduplication so its performance is decreased after non-4KB workload.

In the case of random read, the AFCeph shows an admirable result compared to others. Specially SolidFire performance is degraded significantly in 32KB case. In sequential workload (Figure 11.b, d), both performance of community and AFCeph is 3 or 4 times faster than SolidFire. As we mentioned before, in SolidFire, client's sequential workload would be random workload in the storage cluster because SolidFire divides all inputs to 4KB unit for deduplication.

4.5. Scale-out performance test

In this experiment, we examined the scalability of the system with increasing the number of OSD nodes and clients. (Figure 12) The same hardware and software was used for this experiment and Figure 11 shows the result.(One exception is that SSDs are clean state) For both the sequential and random workloads, (Figure 12.a) the performance is increased with the increasing number of OSD nodes regardless of the block size and RW type.

In the case of random read with 16 nodes, however, the performance improvement is not as big as we expected. (Figure 12.a) We think this issue is due to the high CPU consumption of Ceph's Simple Messenger which handles all of network processing in Ceph because messenger's structure is not scalable and have receiver and sender threads for each connection. Except this case, all workload performances are directly proportional to the number of OSD nodes.

5. Related Works

SolidFire [10] has advanced features such as deduplication and compression for SSD endurance and data efficiency, and per-volume QoS. Its architecture is designed for deduplication because it calculates hash of 4KB unit chunk and saves it in NVRAM. However, unlike Ceph, it needs metadata server for data communication and supports only block storage service such as iSCSI. Its deduplication architecture using 4KB fixed unit chunk leads to fragmentation and low sequential workload performance.

To reduce frequent context switching and lock contention, minimizing coarse-grained lock [25] optimization uses similar scheme with interrupt coalescing [26] by making dedicated thread for completion processing. However, by newly adopting pending queue structure during I/O issuance, thread utilization is raised and unnecessary ack processing during lock processing is removed.

There is an approach that tries to optimize random performance for Ceph [27]. They also tuned memory allocator, tcp nagle. However they do not address Throttle, PG lock, logging, processing transaction.

Classifying critical write and handling critical write are important for performance [28]. They send critical writes as well as non-critical writes with dependency to critical write to NVM in order to minimize latency. Our non-blocking logging uses similar approach.

IndexFS [29] uses similar approach as light-weight transaction. For instance, it uses batched insertion to minimize level DB overhead and uses metadata cache (write back) on client. However, our optimization is based on server side and uses batched insertion in the case of inner operations in transaction. Also, light-weight transaction uses metadata cache (write through) in order to avoid read-modify-write. On the other hand, IndexFS uses write back cache for metadata in order to make bulk insertion.

6. Conclusion

This paper analyzed the problems that occur when scale-out storage system uses SSD and suggest various optimization techniques such as lock optimization, non-blocking logging, and light-weight transaction to solve the performance problems. As a result, we obtain comparable random workload performance with the commercial solution, and a far higher sequential workload performance than with the commercial solution.

Our works does not influence Ceph negatively because it preserves the basic semantics of Ceph. Additionally, we verfified the stability using the Ceph QA suite which is called Teuthology (we passed RBD test). We submitted lock optimization patch to Ceph community. Other optimizations will be submitted after codes are reworked for latest Ceph version.

References

[1] Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung: The Google file system. SOSP 2003: 29-43

[2] OPENSTACK USER SURVEY, 2015. https://www.openstack.org /assets/survey/Public-User-Survey-Report.pdf

[3] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, Carlos Maltzahn: Ceph: A Scalable, High-Performance Distributed File System. OSDI 2006: 307-320

[4] OpenStack Swift. http://docs.openstack.org/developer/swift/

[5] GlusterFS. http://www.gluster.org/

[6] Brad Calder et al., : Windows Azure Storage: a highly available cloud storage service with strong consistency. SOSP 2011: 143-157

[7] Tyler Harter et al.,: Analysis of HDFS under HBase: a facebook messages case study. FAST 2014: 199-212

[8] Changwoo Min et al.,: SFS: random write considered harmful in solid state drives. FAST 2012: 12

[9] Raja Appuswamy et al., : Scale-up vs scale-out for Hadoop: time to rethink? SoCC 2013: 20:1-20:13

[10] SolidFire. http://www.solidfire.com/

[11] Young Jin Yu et al.,: Optimizing the Block I/O Subsystem for Fast Storage Devices. ACM Trans. Comput. Syst. 32(2): 6 (2014)

[12] Eric Seppanen, Matthew T. O'Keefe, David J. Lilja: High performance solid state storage under Linux. MSST 2010: 1-12

[13] Weil, S.A.: Ceph: reliable, scalable, and high-performance distributed storage. PhD thesis, Santa Cruz, CA, USA (2007)

[14] Eno Thereska et al.,: IOFlow: a software-defined storage architecture. SOSP 2013: 182-196

[15] Stan Park, Kai Shen: FIOS: a fair, efficient flash I/O scheduler. FAST 2012: 13

[16] Xingbo Wu, Yuehai Xu, Zili Shao, Song Jiang: LSM-trie: An LSM-tree-based Ultra-Large Key-Value Store for Small Data Items. USENIX Annual Technical Conference 2015: 71-82

[17] Russell Sears, Raghu Ramakrishnan: bLSM: a general purpose log structured merge tree. SIGMOD Conference 2012: 217-228

[18] Hyeontaek Lim et al.,: SILT: a memory-efficient, high-performance key-value store. SOSP 2011: 1-13

[19] SSangjin Han, Scott Marshall, Byung-Gon Chun, Sylvia Ratnasamy: MegaPipe: A New Programming Interface for Scalable Network I/O. OSDI 2012: 135-148

[20] Eunyoung Jeong et al., : mTCP: a Highly Scalable User-level TCP Stack for Multicore Systems. NSDI 2014: 489-502

[21] Jeffrey C. Mogul, Greg Minshall: Rethinking the TCP Nagle algorithm. Computer Communication Review 31(1): 6-20 (2001)

[22] John K et al.,: The RAMCloud Storage System. ACM Trans. Comput. Syst. 33(3): 7 (2015)

[23] Douglas B. Terry, Vijayan Prabhakaran, Ramakrishna Kotla, Mahesh Balakrishnan, Marcos K. Aguilera, Hussam Abu-Libdeh: Consistency-based service level agreements for cloud storage. SOSP 2013: 309-324

[24] Youyou Lu, Jiwu Shu, Long Sun, Onur Mutlu: Loose-Ordering Consistency for persistent memory. ICCD 2014: 216-223

[25] M. Herlihy, N. Shavit, The Art of Multiprocessor Programming (Morgan Kaufmann, March 2008)

[26] Xiaolin Chang, Jogesh K. Muppala, Zhen Han, Jiqiang Liu: Analysis of Interrupt Coalescing Schemes for Receive-Livelock Problem in Gigabit Ethernet Network Hosts. ICC 2008: 1835-1839

[27] Optimizing Ceph for All-Flash Architectures, Sandisk. https://events.linuxfoundation.org/sites/events/files/slides/ optimizing_ceph_flash.pdf

[28] Sangwook Kim, Hwanju Kim, Sang-Hoon Kim, Joonwon Lee, Jinkyu Jeong: Request-Oriented Durable Write Caching for Application Performance. USENIX Annual Technical Conference 2015: 193-206

[29] Kai Ren, Qing Zheng, Swapnil Patil, Garth A. Gibson: IndexFS: Scaling File System Metadata Performance with Stateless Caching and Bulk Insertion. SC 2014: 237-248