



저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:



저작자표시. 귀하는 원 저작자를 표시하여야 합니다.



비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리와 책임은 위의 내용에 의하여 영향을 받지 않습니다.

이것은 [이용허락규약\(Legal Code\)](#)을 이해하기 쉽게 요약한 것입니다.

[Disclaimer](#)



M.S. THESIS

Optimization of Distributed Storage on Commodity
SSD using NVDIMM

NVDIMM을 상용 SSD와 사용할 때의 분산 스토리지 최적화

AUGUST 2017

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Taeksang Kim

M.S. THESIS

Optimization of Distributed Storage on Commodity
SSD using NVDIMM

NVDIMM을 상용 SSD와 사용할 때의 분산 스토리지 최적화

AUGUST 2017

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Taeksang Kim

Optimization of Distributed Storage on Commodity
SSD using NVDIMM

NVDIMM을 상용 SSD와 사용할 때의 분산 스토리지
최적화

지도교수 염현영
이 논문을 공학석사학위논문으로 제출함
2017년 06월

서울대학교 대학원
컴퓨터 공학부
김택상

김택상의 석사학위논문을 인준함
2017년 07월

위 원 장 _____ 장병탁 (인)
부위원장 _____ 염현영 (인)
위 원 _____ 염현상 (인)

Abstract

Optimization of Distributed Storage on Commodity SSD using NVDIMM

Taeksang Kim

School of Computer Science Engineering
Collage of Engineering
The Graduate School
Seoul National University

As cloud computing has been dominant, people become interested in distributed storage system used in data center. One of the most attractive things is ceph distributed storage. That's because it has various advantages like high scalability, fault tolerance and rebalancing data objects automatically. Client can communicate the node which has the target data object directly, without extra communication with metadata server. Ceph distributed storage is said to be petabyte scalable for these reasons. In addition, the replication functionality of ceph ensures high availability, fault tolerance and no single point of failure. That's why ceph has been selected as our target system.

Even if ceph has various advantages like scalability, it has not been designed for the usage of NVDIMM with commodity SSD. So, the research focuses on how the ceph distributed storage has to be optimized for NVDIMM with commodity SSD. First, how the internal implementation of ceph has to be optimized for using NVDIMM as write buffer will be explained. This optimization has been implemented by making worker threads issue I/O request to NVDIMM and reply to client. This optimization improves the performance of write request. Second, how the mixed workload, which contains read and write request, should

be dealt with will be explained. The above-mentioned optimization for write request can't improve the performance of mixed workload. The reason is that read requests can race with flush command. To remove the race, separation of read request and flush command should be used.

Keywords: RTC, BlueStore, Ceph, Distributed storage, SSD, NVDIMM

Student Number: 2015-21238

Contents

Abstract	i
Contents	iii
List of Figures	v
List of Tables	vi
Chapter 1 Overview	1
Chapter 2 Background	4
2.1 Ceph Distributed Storage	4
2.2 The Architecture of Ceph BlueStore	5
2.3 Optimization for Ceph Distributed Storage	6
2.4 NVM for Write Buffer	6
2.5 Optimization of Read Request	7
Chapter 3 Design and Implementation	8
3.1 Analysis of Ceph BlueStore	8
3.1.1 Critical Path of Write without WAL	9
3.1.2 Critical Path of Write with WAL	10
3.2 Optimization of Critical Path of Write Request	10
3.3 Separation of Read Request and Flush Command	13

Chapter 4 Evaluation	14
4.1 The optimization of write request for NVDIMM	15
4.2 Separation of Read and Flush Command	16
Chapter 5 Conclusion	19
Chapter 6 Discussion	20
Bibliography	21
요약	23

List of Figures

Figure 3.1	The flow of write request in BlueStore	9
Figure 3.2	The sequence diagram of write request flow in BlueStore	11
Figure 3.3	The sequence diagram of proposed write request flow . .	12
Figure 3.4	The problems that reader races with background flush .	13
Figure 4.1	The throughput of basic BlueStore and rtc according to threads (1 OSD)	15
Figure 4.2	The throughput of basic BlueStore and rtc according to threads (2 OSDs, 2 replicas)	16
Figure 4.3	The maximum throughput, according to workload and combination	17
Figure 4.4	The normalized throughput, according to workload and combination	18

List of Tables

Chapter 1

Overview

In Distributed Storage, the performance of write request is lower than that of read request. That's because write request generates various operations for availability (replication) and consistency (journaling). Such a heavy write request can be accelerated with NVM(Non-Volatile Memory) like NVDIMM. NVDIMM can be used as burst buffer or Logging device to accelerate write request. If metadata and data journal is written to NVDIMM, of which properties are low latency and non-volatility, the latency of write request can be improved.

Some storage or file system like [1] has used already NVDIMM as write buffer. Our target system, ceph BlueStore, also has separate paths for metadata and data. Even if metadata managed by RocksDB is stored in NVDIMM, there is no performance increase in ceph BlueStore. That's because the BlueStore has not been developed for making use of NVDIMM. To improve the performance of BlueStore using NVDIMM, it needed to be modified for that acceleration with NVDIMM.

The performance of write request has been improved up to 5.78x(FIO 4 threads). The increased throughput is only for write only workload. The per-

formance doesn't change from the performance of original BlueStore for mixed workload with read and write request. Further optimization is required for various workload, mixed workload, not only for write-only workload.

There is a possibility that read request can be slower than write request, if NVDIMM is used as logging device and commodity SSD without super capacitor is used as data storage. That's because NVDIMM is to accelerate just write request, not read request. If NVDIMM is used as write buffer, the critical path of write request has no SSD I/O which leads to high latency. On the other hand, almost all read requests have to be issued to SSD directly, even if memory cache, which is much smaller than data storage (SSD), handles some read request without accessing SSD.

The fact that NVDIMM is only for write request isn't enough for explaining the fact that there is no difference between the performance of original version and that of optimized version for write request, in case of mixed workload. When NVDIMM is used as write buffer, data write and flush command is issued to SSD in background manner. If a read request is issued at that time, the request can race with flush command issued by background thread. That race causes several problems such as increasing read latency and making the I/O handling thread pool full. That's why the write optimization for NVIDMM isn't helpful for mixed workload with read and write request.

To overcome this problem, another optimization has been applied, which is the separation of read request and flush command to SSD. The basic idea is like existing solution for [7]. There are some differences from [7]. This optimization doesn't separate read and write request. Flush command is only the thing to be separated, not write request. A separate device is dedicated for flush command. The other device is for direct I/O request without flush command. This simple approach increases the performance of mixed workload up to 3.47x. When read/flush separation is combined with write optimization for NVDIMM, the performance increased dramatically, regardless of workload.

What contents the paper consists of will be explained, roughly. First, this paper explains the background of the topic. It contains two parts. One thing is restricted to ceph like the overview of ceph itself and existing optimization applied to ceph. The other is about the proposed optimization for ceph in this paper, which includes the write optimization with NVDIMM and read/flush separation for commodity SSD.

Design and Implementation chapter follows the background chapter. The detail of write optimization for NVDIMM and read/flush in the chapter will be described. It will be explained how these optimizations is applied to ceph. The first step for write optimization is to analyze the control flow of ceph BlueStore. Problems had been detected, after control flow of ceph BlueStore is analyzed. The problem has been resolved by decoupling of foreground job from background jobs. In the secion, the design for separation of read request and flush command will be suggested.

In evaluation chapter, fio micro benchmark is used to evaluate suggested optimization. At first, the performance gain of the optimization of write request for NVDIMM will be shown. The throughput varies, as the number of threads increases. Second, the experiment result of the separation of read request and flush command.

Conclusion chapter summarizes the proposed optimizations of ceph BlueStore. Main contribution will be described in abstract level. Lastly, Discussion chapter refers to what optimization is adequate for SSD with super capacitor, which has no flush command overhead. The research direction will be proposed for very fast storage like SSD with super capacitor in that chapter.

Chapter 2

Background

This chapter explains the background about this paper, briefly. There is brief explanation about ceph distributed storage which is petabyte scalable [9]. As our target system, ceph distributed storage needs to be introduced. Next, this chapter shows what optimization has been applied to ceph distributed storage. The existing optimization is just about the architecture of program like lock and throttling. Unlike it, the proposed scheme is an optimization for certain hardware, NVDIMM and commodity SSD. Next two backgrounds are some techniques related this proposal. One of the techniques is related to the usage of NVDIMM for accelerating write request. The other existing technique that multiple devices or nodes are used for accelerating read request will be shown.

2.1 Ceph Distributed Storage

Ceph is one of the popular distributed storage, because of various advantages like extreme scalability, automatic rebalance of data, fault tolerance, etc. [9] In ceph, client communicates with OSD(Object Storage Daemon), which has ability to do computation and store data. The basic concept is OSD(Object

Storage Daemon) responsible for communication and storage node. Unlike other distributed storages, ceph has no metadata server, which leads to removing the performance bottleneck from single metadata server. When client wants to send I/O request, it has only to communicate OSDs dealing with the PGs (Placement Group) to which data are assigned. There is no central metadata server in this process. The only thing needed is cluster map, which is the information about PG management over all OSDs, periodically taken from ceph monitor daemon.

Crush algorithm is one of the most important notions in ceph. Crush takes cluster map, rule and PG. Then, it assigns PG to actual devices, through the algorithm. The placement of data object is well-balanced over all OSDs, which makes ceph scale out. With this algorithm, client can communicate with OSDs directly, without communicating central metadata server. [8] Ceph distributed storage has been selected as our target system, because of the properties like scalability and no single point of failure.

2.2 The Architecture of Ceph BlueStore

BlueStore is a promising storage module in ceph. That's because BlueStore overcomes the limitation of existing storage module, FileStore. FileStore has a big problem that it uses local file system. The main problem is that local file system may do journaling of the journaling of ceph. This problem is referred to as journal of journal problem. The other problem is that local file system is heavy, because it has been developed for general purpose. To resolve these problems, ceph BlueStore uses BlueFS that is lightweight file system in user space. This light file system uses three raw devices to store data object, which resolves two problems by removing journaling of kernel and overhead of heavy local file system. The three raw devices that BlueStore uses are for journaling, metadata and data object, respectively. RocksDB manages metadata with two devices of three devices mentioned above. They are wal(write ahead logging) device and db device, which are journaling device and metadata device, respectively. This

division of raw devices makes it easy to apply our proposed optimization to BlueStore.

2.3 Optimization for Ceph Distributed Storage

Existing optimization for ceph will be explained briefly. As described above, data object has to be assigned to PG in ceph distributed system. Then, PG is remapped to actual OSDs by crush algorithm. Through this process, client can communicate with OSDs, without a central metadata server. [6] improves the performance of single OSD. When OSD receives a request from client, it parses the request and queues it to worker thread pool. There is a problem that worker which waits for PG lock cannot be used by the request of other PG. That problem is called as head of line blocking. [6] resolves this head of line blocking in worker thread pool by adopting pending queue. If a situation that a worker has to wait for PG lock occurs, the worker puts the job into pending queue and handle other jobs from its own queue. There are other several optimizations about thread and throttling configuration. The approach of proposed optimization is same as that of [6], in that the improvement of the performance of single OSD leads to the overall performance of ceph cluster. Unlike [6], the proposed optimization is specialized for (NVDIMM + commodity SSD)

2.4 NVM for Write Buffer

The effective usage of NVM like NVDIMM has been studied. NVM can be used as write buffer. In [1], NVM is used as write buffer for improving application performance. The scheme focuses on finding the dependency of critical I/O, to determine what write is stored into NVM. But our proposed optimizations don't have to consider what kinds of write are stored into NVM. The kinds of write, which are metadata and journal, to be stored into NVM is already

determined easily. That's because journal, metadata, data are already stored into their own raw device. In addition, the scheme used in [1] is implemented on kernel layer, using hint given by user application. Our proposed scheme doesn't modify kernel layer.

File system uses NVM for performance in [3]. It uses undo log for metadata to reduce log tracking cost. Redo log is used for data, to reduce the copy cost for data, compared to undo log [3]. Unlike it, our proposed scheme uses only redo log for data and metadata, because BlueStore has to keep committed request in main memory, before the request is applied to SSD. There is no additional tracking overhead with this policy. The scheme in [3] makes use of byte-addressability of NVM. But our proposed scheme uses NVDIMM as Block Device. There may be some overhead of using NVDIMM as block device. But it has better portability to existing system using block device

2.5 Optimization of Read Request

For the stable performance of read request on SSD, erasure coding is used in [2]. Through erasure coding, SSDs are divided into reader and writer SSDs. The role of reader and writer changes, as sliding window indicating reader. When read request comes in, read request is reconstructed by the SSDs of sliding window. RAID1+0 is used for same purpose in [7]. In this case, there is no overhead caused by running erasure coding algorithm. But shortcoming of [7] is 100% space overhead. The technique similar with [2] can be applied to internal architecture of SSD [4] and cluster of memory cache [5]. All of them improve the tail latency of SSD or cluster of memory cache. As motivated by them, one of our proposed schemes, separation of read and flush command, has been developed. Unlike existing techniques separating read and write request, our proposed scheme separates read request and flush command.

Chapter 3

Design and Implementation

In this chapter, our design of proposed techniques will be explained. One of the techniques is the optimization of critical path of write request. The other is optimization of read request with multiple SSDs. Before detail of the proposed schemes, this chapter shows the analysis of ceph BlueStore. Then, other two techniques follow the analysis.

3.1 Analysis of Ceph BlueStore

In this part, the I/O flow of ceph using Bluestore will be analyzed. When client tries to write data object to the ceph cluster, it first calculates the PG to which the data object belongs. After PG is calculated, crush algorithm finds the appropriate OSD for this PG with cluster map which was taken from ceph monitor periodically. Finally, client can send I/O request with the OSD, directly. This process is dealt with in client side.

In server side, the OSD receives I/O request from the client. Messenger in OSD, which handles network communication, receives the request. It parses the request and queue the request into OSD::ShardedOpWQ. What shard the

request is forwarded to can be determined by a numerical expression, which is the id of PG modulo the number of shard. OSD::ShardedOpWQ handles the request in store module, which is BlueStore in this paper. After the store module processes the request, the request is queued to the queue of messenger. After messenger takes the request, messenger sends the message that commit or applied operation has been finished to client or primary replica.

All optimizations in this paper is implemented in store module, not Messenger. From now on, this section focuses on our target store module, BlueStore. BlueStore is divided into two parts, which are path of WAL(Write Ahead Logging) and path without WAL(Write Ahead Logging). Next Two subsections explain the BlueStore's path.

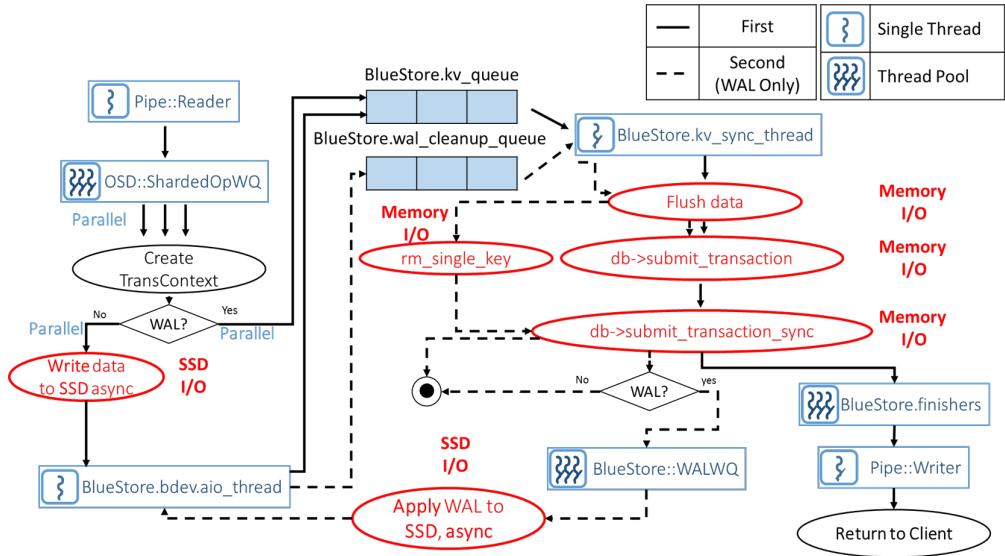


Figure 3.1 The flow of write request in BlueStore

3.1.1 Critical Path of Write without WAL

Figure 3.1 indicates the flow of write request in BlueStore. When WAL isn't used, OSD::ShardedOpWQ issues I/O request to SSD, asynchronously. After the I/O is handled in SSD, BlueStore.bdev.aio_thread takes finished I/O re-

quest and queue it into BlueStore.kv_queue. BlueStore.kv_sync_thread gets all requests in BlueStore.kv_queue at once and issues flush command to SSD. After flush command, BlueStore.kv_sync_thread stores metadata into WAL Device(NVDIMM). The requests processed by BlueStore.kv_sync_thread will be queued into BlueStore.finishers, which is a thread pool replying to client or primary replica. BlueStore.finishers does post processing about the requests and send them to Messenger.

3.1.2 Critical Path of Write with WAL

The path with WAL is similar with above process. Figure 3.1 also indicates the critical path of WAL. When a write request needs WAL, the multiple threads in OSD::ShardedOpWQ thread pool queue write request into BlueStore.kv_queue, concurrently. After that, BlueStore.kv_sync_thread takes all requests from BlueStore.kv_queue and issues flush command to SSD. The thread writes metadata and data journal into WAL device(NVDIMM) about the taken requests and queue the requests to BlueStore.finishers. The BlueStore.finishers replies to client or primary replica about the requests.

After reply is sent to appropriate target which is client or primary replica, BlueStore::WALWQ thread pool issues write I/O about the request to SSD, asynchronously. BlueStore.bdev.aio_thread takes the finished write I/O requests and queues them into BlueStore.wal_cleanup_queue. BlueStore.kv_sync_thread takes them from the queue and removes data journal from WAL device(NVDIMM).

3.2 Optimization of Critical Path of Write Request

From now, this paper considers only the path of WAL, not without WAL, because this path is useful for optimization. How to make every I/O follow the path of WAL will be explained in evaluation chapter.

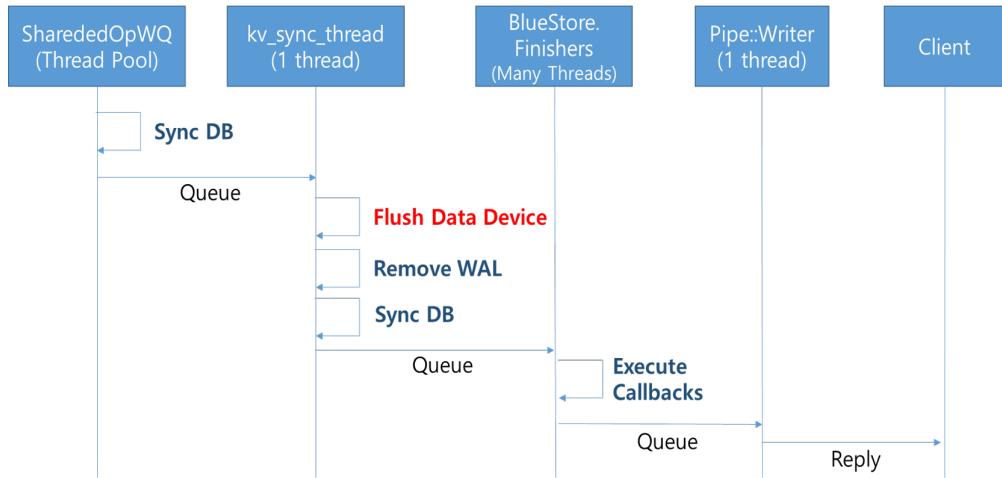


Figure 3.2 The sequence diagram of write request flow in BlueStore

Figure 3.2 is the sequence diagram indicating the critical path of BlueStore. Data journal and metadata of the requests in BlueStore.kv_queue is stored into RocksDB by BlueStore.kv_sync_thread. The thread issues flush command into Data Device (SSD). The thread removes data journal about requests previously committed, which is in BlueStore.wal_cleanup_queue. It executes sync operation of RocksDB. After that, BlueStore.finishers calls callback context doing post processing about the requests.

In this process, there is a problem that reply to client is delayed, because of flush command, which is high overhead in commodity SSD, to SSD. The role of the flush command called by BlueStore.kv_sync_thread is to ensure the durability of data issued to SSD, before removing data journal about the requests in BlueStore.wal_cleanup_queue. The wait time for flush command can be omitted in case of the requests in BlueStore.kv_queue

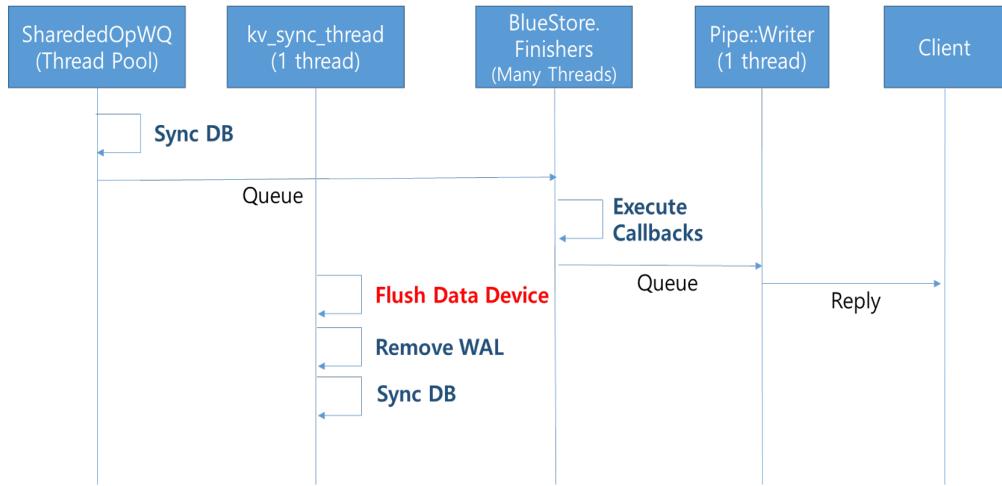


Figure 3.3 The sequence diagram of proposed write request flow

Figure 3.3 is the proposed scheme for the optimization of the path of write request in BlueStore. If ShardedOpWQ thread pool stores WAL(data journal) and metadata into WAL device(NVDIMM), the thread pool can queue the corresponding request into the queue of BlueStore.finishers, to reply to client. By making BlueStore.kv_sync_thread background thread, the flush command to be issued to data device(SSD) in critical path disappears.

3.3 Separation of Read Request and Flush Command

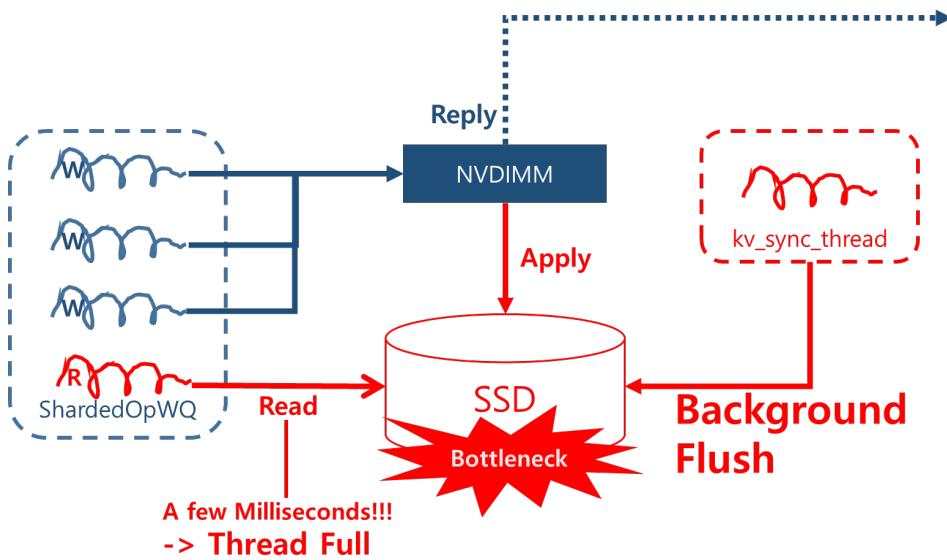


Figure 3.4 The problems that reader races with background flush

Even if write request optimization for NVDIMM is applied to BlueStore, the performance of mixed workload which contains read and write requests is rarely improved. That's because foreground ShardedOpWQ thread pool races with the flush command issued by BlueStore.`kv_sync_thread` which is background thread. This problem is described in figure 3.4. To overcome this race, separation of read request and flush command has been designed. It is similar with what was mentioned in background chapter. It needs two devices (SSDs) for Direct I/O and flush command, respectively. Read request cannot be served by a device dedicated for flush command. Read request is served by the other device. By doing so, read request can avoid waiting for flush command which is significant overhead in commodity SSD without super capacitor.

Chapter 4

Evaluation

This chapter evaluates proposed scheme in this paper. The proposed scheme consists of the optimization of write request for NVDIMM and separation of read request and flush command. The benchmark used is fio micro benchmark. The node consists of I7-4790 CPU(8 threads with hyper threading), 32GB main memory, 256G Samsung 850 pro SSD and 8G NVDIMM emulated by ramdisk. The setting is that an OSD is assigned to one node and a client is assigned to one node. The configuration of ceph is that simple messenger and BlueStore are used. Other configurations are default, except for removing throttle. All experiments use single RBD client. To make every I/O request of BlueStore follow the path using WAL, the 100G data is written before we start an experiment.

4.1 The optimization of write request for NVDIMM

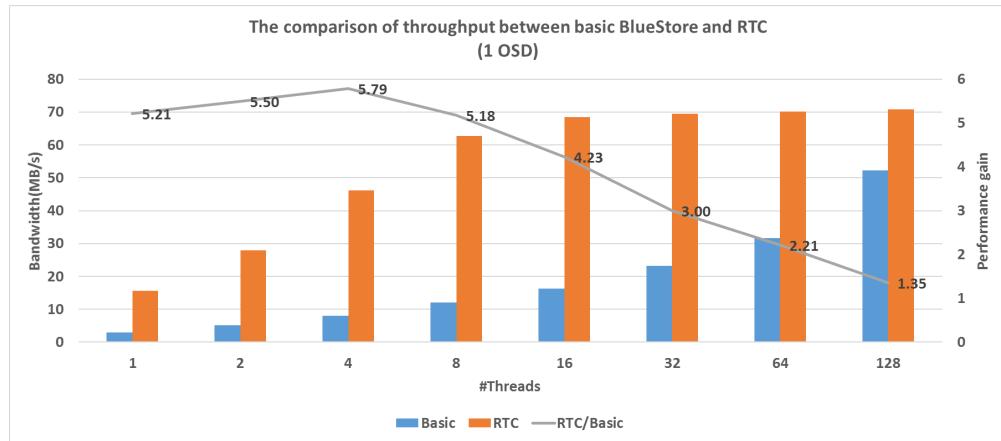


Figure 4.1 The throughput of basic BlueStore and rtc according to threads (1 OSD)

This optimization will be called as RTC(Run To Completion) in this chapter, because ShardedOpWQ handles I/O request on NVDIMM, without queuing to BlueStore.kv_sync_thread. It also removes flush command to be issued to SSD from critical path. Figure 4.1 and figure 4.2 shows how much the performance of write request of RTC is improved, as the number of threads in Fio increases. In figure 4.1, the cluster consists of single OSD. In this setting, the maximum throughput of RTC and basic BlueStore are about 70MB/s and 52MB/s respectively. The maximum performance of RTC is 1.35x better than that of basic BlueStore. In 4 threads option, the performance is improved from 8MB/s to 46MB/s, which is 5.79x. The improvement is also effective in case of the cluster that consists of 2 OSDs and 2 Replica.

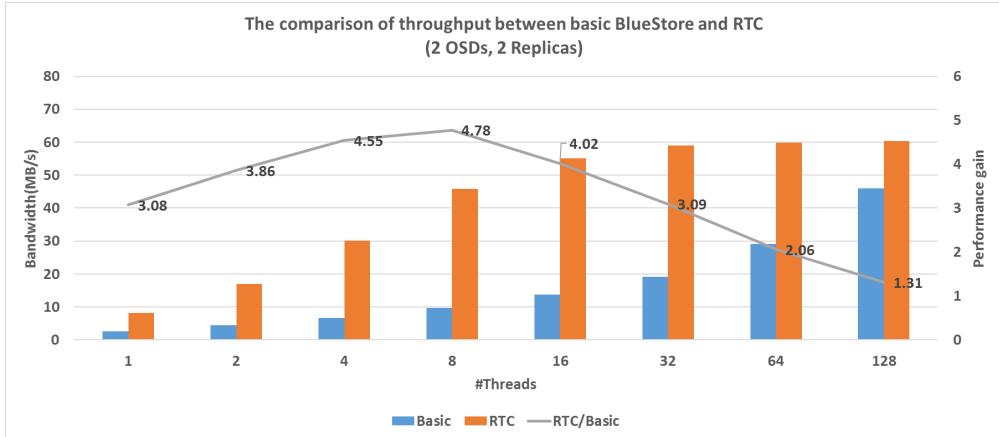


Figure 4.2 The throughput of basic BlueStore and rtc according to threads (2 OSDs, 2 replicas)

There is also significant improvement on that setting, as seen in figure 4.2. The maximum improvement is at 8 threads, which is 4.78x from 9.6MB/s to 45.8MB/s. The improvement of maximum performance is 1.31x from 46MB/s to 60.2MB/s.

4.2 Separation of Read and Flush Command

The ceph cluster setting is that a cluster consisting of single OSD serves a single client. In this experiment, maximum throughput is provided as the ratio of read and write changes.

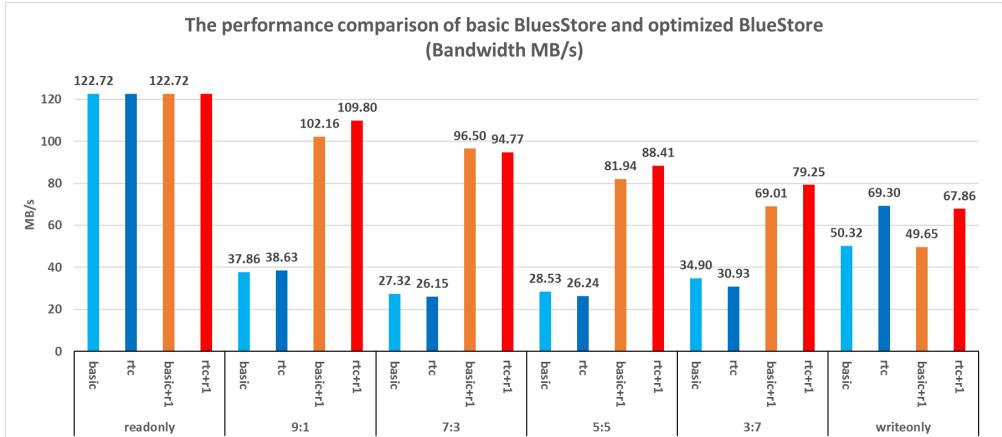


Figure 4.3 The maximum throughput, according to workload and combination

Figure 4.3 indicates the performance of combination of proposed optimization, as the ratio of read and write changes. The lowest x axis of the figure indicates workload. The x axis above it indicates the combination of proposed optimization. Y axis of it indicates throughput. From this figure, RTC(Optimization of write request for NVDIMM) is effective on only write-only workload. There is no performance gain in other workloads. The second optimization, separation of read and flush command, is effective on all workloads except for write-only workload. The combination of two optimizations, which is rightmost bar, is effective on all workloads.

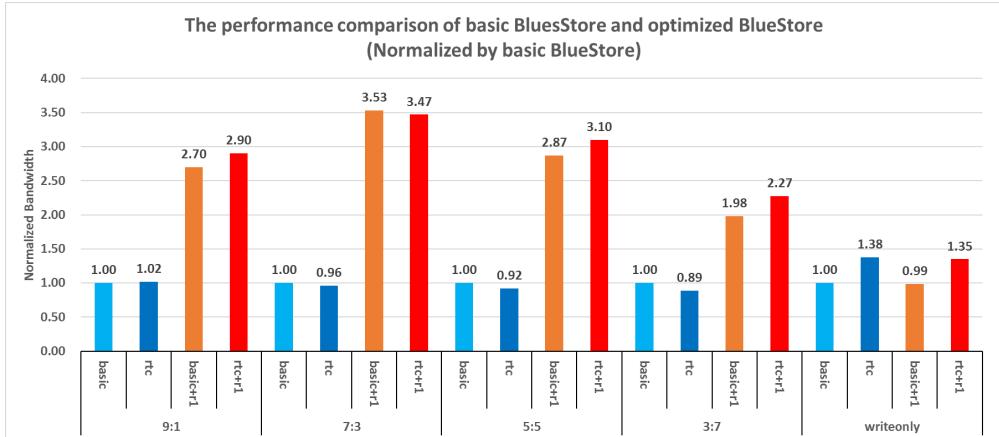


Figure 4.4 The normalized throughput, according to workload and combination

Figure 4.4 normalizes the performance of various setting by the basic BlueStore's performance. The separation of read and flush command is very effective on mixed workload. In addition, the performance of combination of optimization for NVDIMM and read/flush separation is better than the case adopting only read/flush separation, except 7:3 workload. In that case, the overhead of lock contention of RTC is heavier than that of flush command.

Chapter 5

Conclusion

Even if ceph distributed storage is popular, its architecture is inappropriate for the usage of NVDIMM as buffer. To overcome this problem, BlueStore, which is promising store module in ceph, has been optimized by adopting Run To Completion model that workers in critical path doesn't sleep for heavy job like SSD I/O. Unlike [6], it considers hardware used in BlueStore. The performance of write request improves up to 5.79x which is experimented in the environment of 4 threads in single OSD. In addition, the proposed scheme optimizes not only write-only workload, but also mixed workload which has read and write I/O. Because commodity SSD without super capacitor has significant overhead of flush command, the performance of read request in mixed workload can be improved by separating flush command and read request. This optimization leads to performance improvement up to 3.47x. When commodity SSD without super capacitor is used, it's important to handle flush command well.

Chapter 6

Discussion

There is another way to accelerate write request by asynchronous checkpoint. But it requires more NVDIMM space. Future work of this paper is to reduce the overhead of RAID1 solution. Current RAID1 solution has 100% overhead of space. This overhead can be reduced by adopting RAID5 or erasure coding algorithm.

The proposed solutions in this paper are limited to the usage of commodity SSD without super capacitor. There are many other SSDs with super capacitor which has no overhead of flush command, because of their super capacitor. There is another technique for this kind of SSDs. In this case, the locking overhead of key value module is dominant factor of bottleneck. To overcome this problem, the key value db for metadata has to be sharded.

Bibliography

- [1] S. W. Kim, H. J. Kim, S. H. Kim, J. W. Lee and J. K. Jeong, “Request-Oriented Durable Write Caching for Application Performance” in *Proceedings of the 2015 USENIX conference on USENIX Annual Technical Conference*, Santa Clara, USA, July 2015. pp. 193-206.
- [2] D. Skourtis, D. Achlioptas, N. Watkins, C. Maltzahn, and S. Brandt, “Flash on Rails: Consistent Flash Performance through Redundancy” in *Proceedings of the 2014 USENIX conference on USENIX Annual Technical Conference*, Philadelphia, USA, June 2014. pp. 463-474.
- [3] J. Ou and J. Shu, “Fast and Failure-Consistent Updates of Application Data in Non-Volatile Main Memory File System” in *IEEE 32nd International Conference on Massive Storage Systems and Technology*, Santa Clara, USA, May 2016.
- [4] S. Yan, H. Li, M. Hao, M. H. Tong, S. Sundararaman†, A. A. Chien, and H. S. Gunawi, “Tiny-Tail Flash: Near-Perfect Elimination of Garbage Collection Tail Latencies in NAND SSDs” in *Proceedings of the 2017 USENIX conference on File and Storage Technologies*, Santa Clara, USA, March 2017.
- [5] K. V. Rashmi, M. Chowdhury, J. Kosaiyan, I. Stoica, and K. Ramchandran, “EC-Cache: Load-Balanced, Low-Latency Cluster Caching with On-

line Erasure Coding” in *Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation*, Savannah, USA, November, 2016. pp. 401-417.

- [6] M. W. Oh, J. G. Eom, J. Y. Yoon, J. Y. Yun, S. M. Kim and H. Y. Yeom, “Performance Optimization for All Flash Scale-Out Storage” in *2016 IEEE International Conference on Cluster Computing*, Taipei, Taiwan, September, 2016, pp. 316-325.
- [7] W. Shin, M. C. Kim, K. D. Kim, and H. Y. Yeom, “Providing QoS through Host Controlled Flash SSD Garbage Collection and Multiple SSDs” in *International Conference on Big Data and Smart Computing (BigComp)*, Jeju, Korea, February, 2015. pp. 111-117.
- [8] S. A. Weil, S. A. Brandt, E. L. Miller, and C. Maltzahn, “CRUSH: Controlled, Scalable, Decentralized Placement of Replicated Data” in *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing (SC ’06)*, Tampa, USA, November, 2006.
- [9] S. A. Weil, A. W. Leung, S. A. Brandt, and C. Maltzahn, “RADOS: A Scalable, Reliable Storage Service for Petabyte-scale Storage Clusters” in *Proceedings of the 2nd international workshop on Petascale data storage*, New York, USA, November, 2007. pp. 35-44.

요약

클라우드 환경이 강조되면서, 최근 분산 스토리지에 대한 관심이 늘어나고 있다. 그 중에서, ceph 분산 스토리지는 가장 주목받고 있는 분산 스토리지 중 하나이다. 그 이유로는 고 확장성과, 장애 허용도, 자동적인 데이터 분배에 있다. ceph 클라이언트가 I/O요청을 클러스터에 보낼 때, 별도의 메타데이터서버와의 통신없이, 데이터를 가지고 있는 노드와 직접 통신이 가능하다. 레플리케이션(Replication) 기능을 통해, 고가용성, 장애허용도 측면에서도 높은 점수를 받고 있다. 이러한 이유로 본 연구에서는 ceph 분산 스토리지를 타겟 시스템으로 선정하였다.

이러한 ceph 분산스토리지는 위에 언급된 것처럼, 고 확장성은 가지고 있지만, 새로운 하드웨어를 추가했을 때의 노드 자체의 성능은 개선이 되지 않는다. 본 논문은 NVDIMM과 같은 고성능의 하드웨어를 상용 SSD와 함께 사용하였을 때, 어떻게 Ceph 분산스토리지를 최적화해야하는 지에 대하여 다루고있다. 첫번째, NVDIMM을 쓰기 베퍼로 사용하기 위해, ceph 분산스토리지의 내부적인 구현이 어떻게 바뀌어야 하는지 설명한다. Worker 쓰레드가 컨텍스트 스위칭을 하지 않고, NVDIMM에 직접 I/O요청을 한 후에, 클라이언트에게 응답을 보내는 방법으로 성능을 개선시킬 수 있다. 두번째, 읽기와 쓰기 요청이 섞일 때, 어떻게 해당 요청들에 대한 성능을 향상시킬지 설명한다. 앞서 설명된 최적화로는 읽기와 쓰기 요청이 섞일 때의 성능을 개선시킬 수 없다. 읽기 요청이 비동기적으로 수행되는 flush 명령과 경쟁을 하기 때문이다. 읽기 요청과 flush 명령어를 분리시켜 경쟁을 제거함으로써, 읽기와 쓰기요청이 섞일 때에도 성능을 개선시킬 수 있다.

주요어: RTC, BlueStore, Ceph, Distributed storage, SSD, NVDIMM

학번: 2015-21238