

Implementing I/O Bandwidth Sharing Scheme between Multiple Linux Containers based on Dm-zoned for Zoned Namespace SSDs

Seokjun Lee¹, Sungyong Ahn²

¹Master, School of Computer Science and Engineering, Pusan National University, Korea

²Associate Professor, School of Computer Science and Engineering, Pusan National University, Korea

¹sky_lark0401@pusan.ac.kr, ²sungyong.ahn@pusan.ac.kr

Abstract

In the cloud service, system resource such as CPU, memory, I/O bandwidth are shared among multiple users. Particularly, in Linux containers environment, I/O bandwidth is distributed in proportion to the weight of each container through the BFQ I/O scheduler. However, since the I/O scheduler can only be applied to conventional block storage devices, it cannot be applied to Zoned Namespace(ZNS) SSD, a new storage interface that has been recently studied. To overcome this limitation, in this paper, we implemented a weighted proportional I/O bandwidth sharing scheme for ZNS SSDs in dm-zoned, which emulates conventional block storage using ZNS SSDs. Each user receives a different amount of budget, which is required to process the user's I/O requests based on the user's weight. If the budget is exhausted I/O requests cannot be processed and requests are queued until the budget replenished. Each budget refill period, the budget is replenished based on the user's weight. In the experiment, as a result, we can confirm that the I/O bandwidth can be distributed on their weight as we expected.

Keywords: Zoned Namespace SSD, dm-zoned, Proportional I/O Distribution, Cgroup

1. Introduction

Recently, the cloud service has been widely used to share system resources among multiple users. Specifically, because Linux container isolates a computing environment at the operating system level, it has the advantage of not requiring separate operating systems and low performance overhead. In Linux, a Cgroup subsystem [1] is used to share system resources among containers. In particular, I/O bandwidth of storages can be shared using *blkio* subsystem of Cgroup[X]. Note that the *blkio* subsystem uses a BFQ scheduler[2] to sharing different amounts of bandwidth among multiple Linux containers based on their own I/O weight. However, this scheme cannot be applied to Zoned Namespace(ZNS) SSDs because it is applicable for conventional block interface storages.

ZNS[3] is an emerging storage interface, which divides space of a storage device into certain size unit named

Manuscript Received: october. 23, 2023 / Revised: october. 28, 2023 / Accepted: November. 4, 2023

Corresponding Author: sungyong.ahn@pusan.ac.kr

Tel: +82-51-510-2422

Associate Professor, school of Computer Science and Engineering, Pusan National University, Korea

Zone. ZNS SSDs expose Zone information to the host through zone-based interface, so that the host can control data placement. However, as we mentioned above, the BFQ scheduler only can be used in block I/O layer which ZNS SSD cannot use. Therefore, in previous Linux container system, the proportional I/O bandwidth distribution is not supported for ZNS SSDs.

In this paper, we implement a weight-based proportional I/O bandwidth distribution scheme for multiple Linux containers sharing ZNS SSD. The proposed scheme has been implemented in the Linux Kernel device mapper for zone-based storage called dm-zoned[4]. Dm-zoned emulates a virtual block storage device by using a conventional block storage and a zone-based storage. By using this emulated block device, we can use legacy filesystem on a ZNS SSD. But an emulated block device cannot properly use kernel I/O schedulers. To check whether the I/O scheduler can work on emulated block device, we tried to apply BFQ scheduler to conventional block device composing an emulated block device. However, in this case, I/O bandwidth was not distributed proportionally based on the weight of each container and I/O performance was significantly decreased.

Therefore, we implemented new I/O bandwidth sharing scheme inside dm-zoned. The proposed scheme allocates a resource called budget to each container in proportion to their I/O weight. For this purpose, we modified dm-zoned to identify which container issued each I/O request. Unlike the conventional dm-zoned having single I/O queue, proposed scheme has separate I/O request queue for queuing I/O requests issued by each container. Each container must consume I/O budgets to deliver I/O requests to a storage device. If the budget is exhausted, the I/O request will be delayed until the next budget refill period. As we mentioned above, because I/O budgets are allocated in proportion to I/O weight I/O bandwidth of ZNS SSD can be shared proportionally among multiple containers. According to experiment results, the proposed scheme can distribute I/O bandwidth of ZNS SSD to multiple containers in proportion to I/O weight.

The remainder of this paper is as follows. In a Section 2, we introduce the background information and motivation of this paper. And Section 3 presents an introduction and explanation of our proposed scheme. Section 4 shows experiments using the our proposed scheme and the results, and ends with conclusions in Section 5.

2. Zoned Namespace SSD

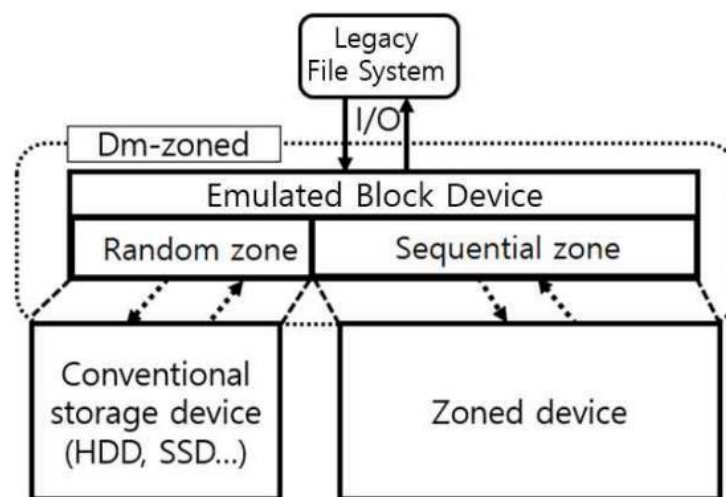


Figure 1. ZNS virtualization through dm-zoned device mapper

ZNS SSD that unlike conventional block device SSD divide NAND flash memory space into certain size called zone and aware to host. Host can select a zone which logical block will be written with zone-based interface. So we expect the host can write I/O stream to different zones to each other and this can remove performance interference and isolate I/O request from multiple user's environments to each other.

However, ZNS SSD is different from conventional block device. ZNS SSD has sequential write constraint, cannot use a legacy file system. To use the ZNS SSD, user needs to use zone-specific file system. In other way, we can get help from dm-zoned device mapper can use legacy file system and without any extra modifications. Dm-zoned combines conventional storage device and a zoned device to create emulated block device. As shown in Figure 1, dm-zoned emulate a virtual block device and expose it to the host can use. In an environment using dm-zoned, legacy file system can use to emulated block device no need to use a zone-specific file system. Said above, ZNS SSD has sequential write constraint, so process the random write request with a conventional block device, which is included in emulated block device.

But if we use the dm-zoned to use ZNS SSD, cannot use I/O scheduler for the emulated block device. Emulated block devices send the I/O request to conventional block devices or zoned block devices that make up the emulated block device. Considering this operation scheme of dm-zoned, we can think the way that set the scheduler to conventional block device that construction of emulated block devices. But, according to our experiments, scheduler does not work that we expect. BFQ scheduler, which we can use in Linux Kernel environment, can divide I/O bandwidth proportionally by pre-set user's weight. But we mentioned above, I/O scheduler does not work when using dm-zoned environments. As shown in Section 4.1 result, BFQ scheduler does not work and even it led significant performance decrease. Therefore, in this paper, we propose the weighted proportional I/O bandwidth sharing scheme for dm-zoned environments.

3. Weighted Proportional I/O Bandwidth Sharing Scheme for ZNS SSD

As we know that confirmed above, we implement weighted proportional I/O bandwidth scheme for ZNS SSD in a dm-zoned environment, in this paper. We analyze and modify the original operating scheme of dm-zoned and make scheme that we propose can work. To implement the weighted proportional I/O bandwidth sharing scheme, we designed a new operation base and weighted proportional I/O budget refill policy[5].

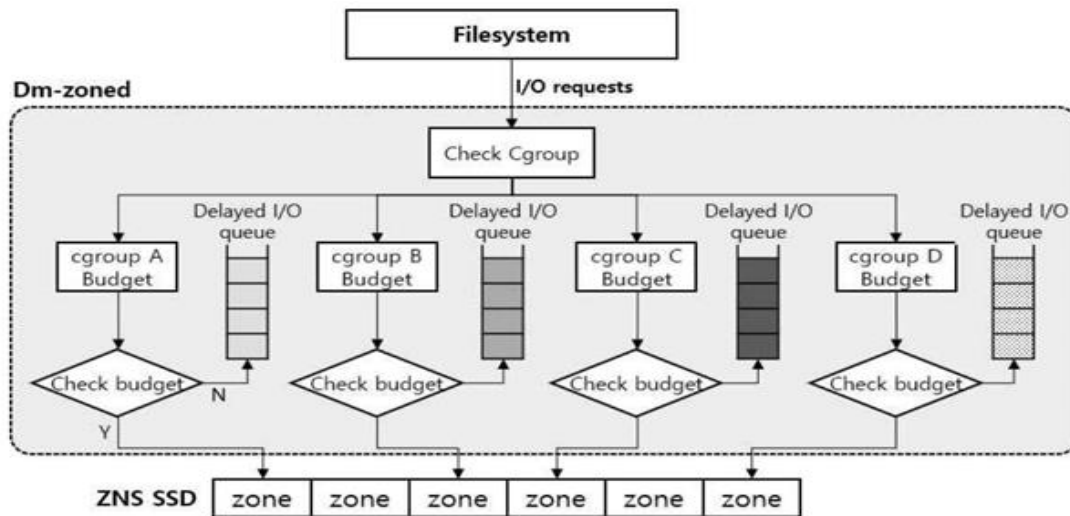


Figure 2. Modified Cgroup Based Dm-zoned Operating Base

3.1 Modify Dm-zoned I/O Process Operating Base

Dm-zoned has been implemented to process a requested write and read from the host based on the chunk that the request belongs to. Collect all requests that belong to the same chunk area and process them at once. In this case for multiple users I/O requests have mixed in the same chunk area. In this paper, in order to individually control the user's I/O bandwidth individually, modified the operating base of dm-zoned to collect I/O requests that belong to same Cgroup. Figure 2 shows the modified scheme for collection and processing I/O requests. The proposed schemes first check the request's Cgroup information of a request, then check whether the Cgroup structure is exists or not. If it does not exist, create a new structure for that Cgroup and collect the request. Each Cgroup structure has budget to use to process an I/O request that user themselves. If some Cgroup used up all budget, that Cgroup's request will be delayed until budget is replenished. The delayed request will be queued in the delayed queue. Each Cgroup's budget will be refill preset refill period. Amount of refill budget will be followed based on their weight proportionally. Delayed requests that are queued in the delayed queue will be processed when the budget is replenished.

3.2 Weight-Based Budget Set and Refill Policy

After implementing a Cgroup-based I/O request collect and process scheme, we need to make budget refill policy to share an I/O bandwidth with weight-based proportional among multiple users. To share bandwidth with weight-based proportional, the host continuously checks the user who has the highest weight (T_{cg}). At each budget refill time, check a budget that T_{cg} has used since the last budget refill time (T_{use}). Give enough budget to the T_{cg} that will never cause an I/O request delay until the next budget refill time. For the other Cgroup that is not T_{cg} need calculate a budget that needs to be replenished. Check the T_{use} and calculate the amount of budget to be paid in proportion to the weight of T_{cg} and this Cgroup. For example, if the weight value of T_{cg} is 1000 and the weight value of Cgroup to be replenished at this budget refill time is 100, this Cgroup will be refilled as an amount of 1/10 of T_{use} . Through this refill policy, we can distribute I/O bandwidth proportionally based on the weight of each Cgroup.

3.3 Checking Cgroup State Idle

We need to check whether a Cgroup is idle or not. If the current T_{cg} is idle, we need to change the T_{cg} before the next budget refill time. If the T_{cg} is not change, despite of the current T_{cg} is idle, T_{use} goes to 0 and this will affect to budget refill policy. Cgroup budget that does not have T_{cg} will be calculated and refilled based on T_{use} . If T_{use} is 0, every other Cgroup will get no budget until the current T_{cg} is working again. Accordingly, schemes that we propose was made to check continuously each Cgroup is idle or not. This scheme continuously updates each Cgroup's last I/O request time, checking this time at each budget refill time. For each Cgroup, if there is no request after 90% in a budget refill cycle since the last budget refill time, judge that Cgroup is idle and exclude it from budget calculation and refill.

In some cases, budget needs to be replenished even through a Cgroup is idle. Cgroup may have an I/O request in the delayed queue, in which case budget needs to be replenished to process the delayed I/O request. So, before excluding a budget refill target, we check the delayed queue to see if any I/O request is queued there.

4. Experiments

Table 1. Experimental Environment

Host	CPU	Intel(R) Xeon(R) Gold 218 CPU@ 2.30GHz 64 Core
	Memory	128GB
FEMU	Core	32 Core
	Memory	Linux Kernel 5.10.136
	System	16GB Block Device SSD
	Storage	32GB Zoned Namespace SSD

Proposed scheme is implemented in Linux Kernel version 5.10.136, the experimental environment can be found in Table 1. We cannot get physical ZNS SSD, so we use FEMU[6] that ramdisk based SSD virtualization virtual machine for this study. We virtualize a conventional block device and ZNS SSD for this experiment. Assuming an experiment with exist 4 users, set each weight 100, 250, 500, 1000 and test about I/O bandwidth. We run the FIO I/O workload simulator[7] experiments (4.1), UMASS[8] real workload experiments(4.4), check a fairness index[9] about our scheme (4.2) and check that Cgroup idle state checking works, which we explained in Section 3.3 (4.3).

4.1 FIO I/O Workload Simulator Experiments

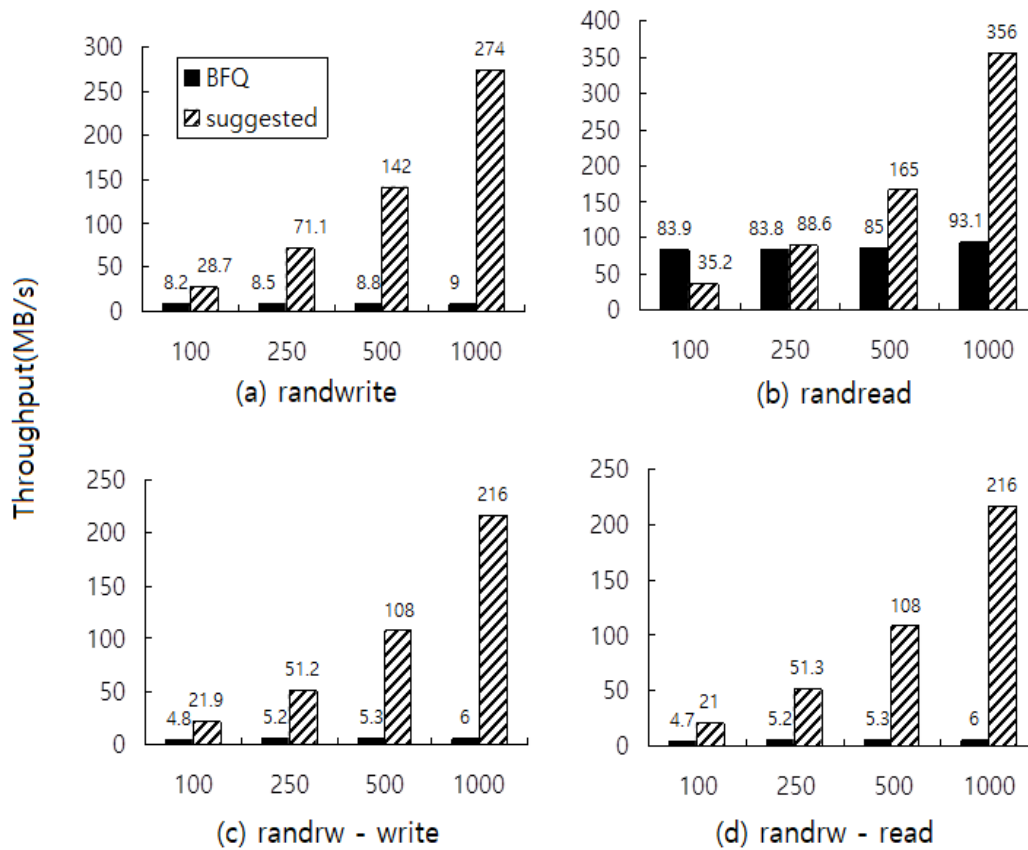


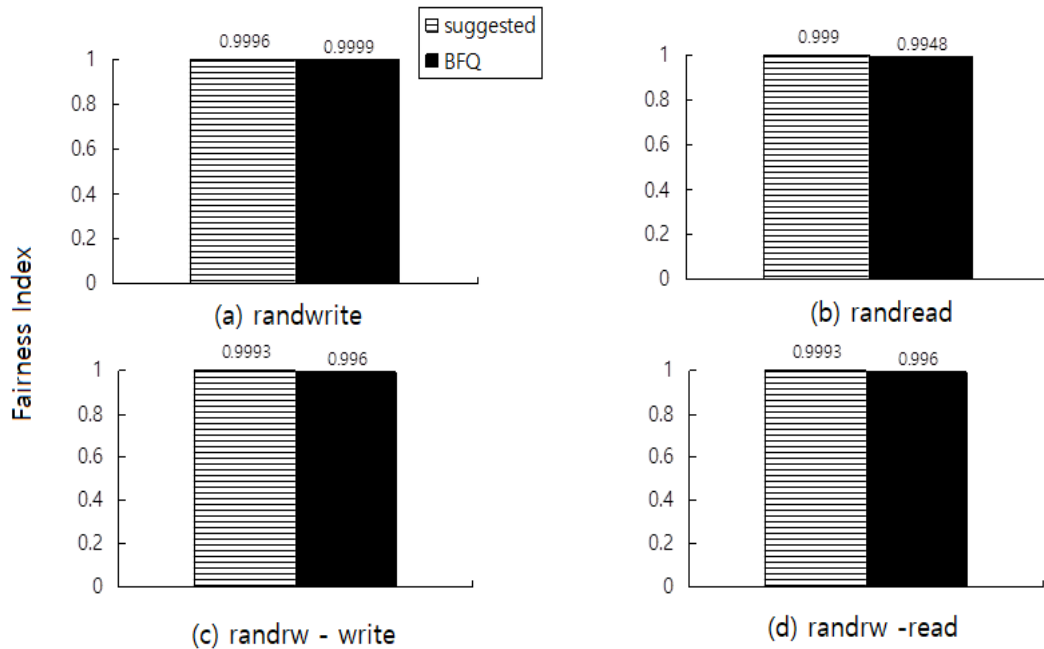
Figure 3. FIO I/O Simulator Experiments

Table 2. FIO Experiment Environment

ioengine	libaio
size	2G
direct	enable
numjobs	1
runtime	30s
rw	randwrite, randread, randrw(50:50)
bs	4KB
iodepth	32

We conducted FIO I/O simulator experiments. The FIO experiments environment can be found in table 2. Comparison target BFQ is that we have set a BFQ scheduler for conventional block device that construction of emulated block devices as we discussed at Section 2. We conduct the same experiments on both. As you can see in a Figure 3, the proposed scheme looks works well as we expected. But the result of BFQ, bandwidth is not distribute proportionately. There even seems to be a performance problem. We expect that, scheduler does not work to ZNS SSD because of ZNS SSD's sequential write constraint. BFQ scheduler can be designed to consider conventional block device has not considered ZNS device. And we can see the result in Figure 3 b likewise, weight-based bandwidth distribution does not work but shows more higher throughput compared to other results. It is ZNS SSD have sequential write constraint, but this constraint does not affect to read, so the following results occurred we think. Through this result, we can confirm that we propose scheme worked.

4.2 Fairness Index of Weighted proportional I/O Bandwidth Sharing Scheme

**Figure 4. Fairness Index**

We confirm the fairness about our scheme. The comparison target BFQ of this experiment is result about using BFQ scheduler to the conventional block device in general environment that does not use dm-zoned different from Section 4.1. Fairness Index which to confirm the fairness is more fair that value is closer to 1. Proposed scheme has high fairness even when compared to BFQ scheduler. Through this experiment we confirm, our weighted proportional I/O bandwidth sharing scheme distributes bandwidth fairly to multiple users.

4.3 Cgroup Idle State Checking and I/O Bandwidth Distribution Experiments

We perform the experiment about function that we explained in Section 3.3 works. It is the same environment with Section 4.1 FIO experiment, but each Cgroup has different runtime in this experiment. The runtime of each Cgroup were set as follows: 1. Cgroup which has 1000 as weight has 30 seconds runtime 2. Cgroup which has 500 as weight has 40 seconds runtime 3. Cgroup which has 250 as weight has 50 seconds runtime 4. Cgroup which has 100 as weight has 60 seconds runtime. We expect about this experiment is when Cgroup which have highest weight fell in idle state that no I/O request issued, it changes highest weight Cgroup to the user with next highest weight (of course not it must be in idle state). If it works, the next highest weight Cgroup's bandwidth must be increased, and the other Cgroup's bandwidth will also change proportionally by changed highest weight. As you can see in Figure 5, every 30, 40 and 50 seconds, the highest weight Cgroup has fallen to idle. Accordingly, the highest weight Cgroup is changed and the bandwidth of this Cgroup is increased.

Figure 5 also shows that the bandwidth is not proportionally distributed at the beginning of the experiment, it's because at the first state of the experiment, there is no standard value to calculate a budget. However, as you can see, after the first time that makes standard value to calculate a budget, the bandwidth is gradually adjusted in proportion to the weight. And we can confirm that when some Cgroup became idle, the bandwidth also changed accordingly.

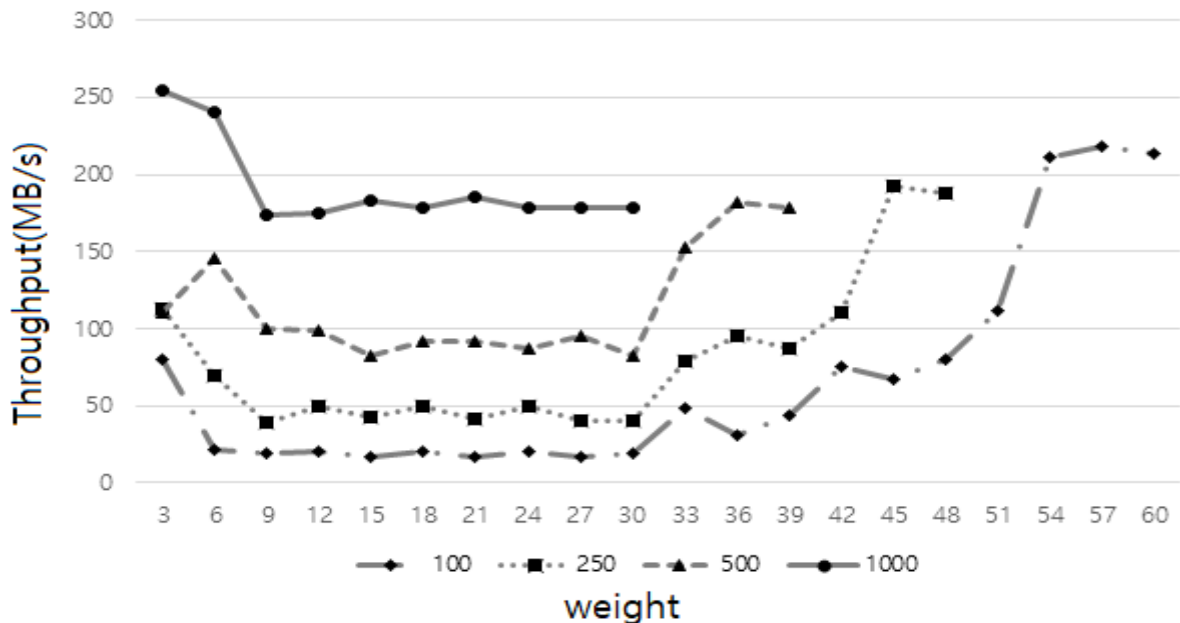


Figure 5. Bandwidth Change According to Cgroup Idle State

4.4 UMASS Real Workload Experiments

Table 3. UMASS Real Workload Experiments Environment

iodepth	32	
threads	8	
runtime	60s	
Read/Write ratio	Financial	Read 18%, Write 82%
	Websearch	Read 99%, Write 1%

We also conducted experiments for real workload. Trace Replay[10], which is replay tool for real workload, experiment result is shown in a Figure 6. Experiments environment of this is shown in Table 3. We use Financial and Websearch workload collected from UMASS. Financial is write-intensive workload and Websearch is read-intensive workload. As a result, we can see, all each workload is proportionally distributed based on weight. But the Financial workload in case, bandwidth distribution is not accurate compare with Websearch workload. It has less accurate compare with FIO experiments but weighted proportional bandwidth distribution also worked in real workload experiment.

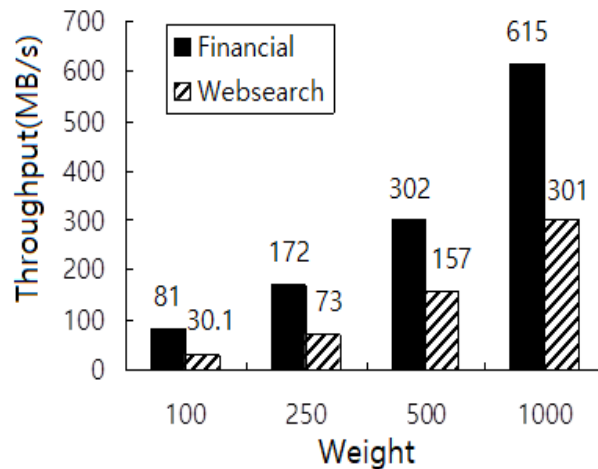


Figure 6. UMASS Real Workload Experiment

5. Conclusion

In this paper, we confirmed that the BFQ I/O scheduler cannot work on environment using dm-zoned device mapper. According to our experiment, the sequential write constraint of ZNS SSD is the cause of BFQ scheduler did not work well for the ZNS SSDs. So, we implemented weighted proportional I/O bandwidth sharing scheme for using dm-zoned environment. Set a weight value for each Cgroup user and distribute different bandwidth each Cgroup user according to their weight. For this scheme, we modified I/O process operating base. To control the I/O bandwidth for each user, we collect I/O requests that belong to the same Cgroup. Each Cgroup has a delayed queue that queues I/O requests when budget is exhausted. And we replenished the budget periodically based on each user's weight. In the result of several experiments, we can see that our proposed scheme can distribute the bandwidth proportionally based on weight. In the future study, we can expect the individual zone allocation for each Cgroup. With an individual zone allocation and study of this paper, we expect a study on parallelism of I/O processing and performance isolation.

Acknowledgement

This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the Convergence security core talent training business(Pusan National University) support program(IITP-2023-2022-0-01201) supervised by the IITP(Institute for Information & Communications Technology Planning & Evaluation).

This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2023-2020-0-01797) supervised by the IITP(Institute for Information & Communications Technology Planning & Evaluation)

References

- [1] Cgroup, [Online] Available: <https://www.kernel.org/doc/Documentation/group-v1/cgroups.txt> ,2018.
- [2] BFQ Scheduler, [Online] Available: <https://www.kernel.org/doc/Documentation/block/bfq-iosched.txt>
- [3] Zoned Namespaces (ZNS) SSDs, [Online] Available: <https://zonedstorage.io/introduction/zns/>.
- [4] dm-zoned-tools, [Online] Available: <https://github.com/westerndigitalcorporation/dm-zoned-tools>.
- [5] S. Ahn, K. La., and J. Kim, "Improving I/O Resource Sharing of Linux Cgroup for NVMe SSDs on Multi-core Systems," 8th USENIX Workshop on Hot Topics in Storage and File Systems(HotStorage 16). pp. 111-115, 2016.
DOI: <https://dl.acm.org/doi/10.5555/3026852.3026875>
- [6] H. Li, et. al., "The case of FEMU: cheap, accurate, scalable and extensible flash emulator," 16th USENIX Conference on File and Storage Technologies (FAST 18), pp. 83-90, 2018.
DOI: <https://dl.acm.org/doi/10.5555/3189759.3189767>
- [7] J. Axboe, "Flexible I/O tester," [Online] Available: <https://github.com/axboe/fio>
- [8] UMASS trace, [Online] Available: <https://traces.cs.umass.edu/>
- [9] R. K. Jain, et al. "A quantitative measure of fairness and discrimination," Eastern Research Laboratory, Digital Equipment Corporation, 1984.
DOI: <https://doi.org/10.48550/arXiv.cs/9809099>
- [10] trace-replay, [Online] Available: <https://github.com/yongseokoh/trace-replay>