Host-Level Workload-Aware Budget Compensation I/O Scheduling for Open-Channel SSDs

Sooyun Lee, Kyuhwa Han and Dongkun Shin
Department of Electrical and Computer Engineering, Sungkyunkwan University, Suwon, Korea
{sooyun802, hgh6877, dongkun}@skku.edu

***Abstract*—In datacenters and cloud computing, Quality of Service (QoS) is an essential concept as access to shared resources, including solid state drives (SSDs), must be ensured. The previously proposed workload-aware budget compensation (WA-BC) scheduling algorithm is a device I/O scheduler for guaranteeing performance isolation among multiple virtual machines sharing an SSD. This paper aims to resolve the following three shortcomings of WA-BC: (1) it is applicable to only SR-IOV supporting SSDs, (2) it is unfit for various types of workloads, and (3) it manages flash memory blocks separately in an inappropriate manner. We propose the host-level WA-BC (hWA-BC) scheduler, which aims to achieve performance isolation between multiple processes sharing an open-channel SSD.**

***Index Terms*—quality of service, solid state drives, open-channel SSDs, I/O scheduler, performance isolation**

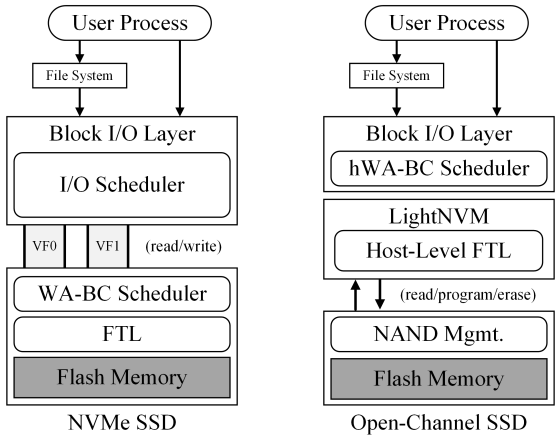
I. Introduction

With the increase in performance per cost of solid state drives (SSDs), SSDs have been adopted in various areas from smartphones to datacenters. Such transition stands out especially in environments requiring high I/O throughput such as datacenters and cloud computing where multiple processes or virtual machines share storage resources. Sharing of resources induces I/O interference between different processes; accordingly, many studies have been conducted to solve this problem via guaranteeing quality of service (QoS) [1].

The workload-aware budget compensation (WA-BC) scheduler [2] is a budget based device-level scheduler proposed to guarantee performance isolation among applications sharing an SSD. It focuses on garbage collection (GC) as the source of interference and evaluates each application's contribution to GC. However, WA-BC holds three drawbacks. First, WA-BC is exclusively for single root I/O virtualization (SR-IOV) [3] supporting SSDs. Secondly, GC contribution assessment relies solely on the write amplification factor (WAF). Third, maintaining of WAF values for each application requires separate management of flash memory blocks as such as multi-streamed SSDs [4]. Such drawbacks limit the applicability of WA-BC to various types of workloads as well as environments.

We propose the *host-level workload-aware budget compensation* (hWA-BC) scheduler for Open-Channel SSDs (OC-SSDs) [5], [6]. hWA-BC monitors both (i) the ratio of valid user data to the amount of consumed storage and (ii) the ratio of read to write requests of each process for GC contribution assessment. It then redefines the I/O request costs based on each process's assessment to either penalize or compensate the processes; thereby achieving performance isolation.

978-1-7281-3854-1/19/$31.00 ©2019 IEEE



(a) WA-BC (b) hWA-BC

Fig. 1. Overall architecture of WA-BC and hWA-BC.

II. The hWA-BC Scheduler

The hWA-BC scheduler is a host-level scheduler which aims to achieve fairness amongst multiple processes sharing an OC-SSD. Fig. 1(b) shows the overall architecture of hWA-BC. The host-level flash translation layer (FTL) is responsible for monitoring the status of each process.

hWA-BC allocates a predefined budget to all processes issuing I/O requests to the shared OC-SSD. For every serviced I/O request, the cost of each request is deducted from the given budget. Processes whose budget no longer remains are excluded from scheduling. The attainment of request cost by hWA-BC utilizes the regression-based cost modeling technique introduced by the BCQ [1] scheduler.

The hWA-BC scheduler follows two principles for penalizing/compensating the budgets: (i) the owner process of the valid pages of victim blocks selected during GC should be penalized, and (ii) read-intensive processes should be compensated compared to write-intensive processes. Two factors, the *read-write request ratio* and the *valid-invalid flash page ratio*, help hWA-BC evaluate a process's contribution to GC. The following equation represents how write costs are modified:

$$cost'_w = cost_w \times [\ 1 + \alpha \times \frac{valid}{invalid} + (1-\alpha) \times \frac{write}{total}\ ]$$

where $cost'_w$ is the modified write cost, $valid$ and $invalid$ represent the number of valid and invalid flash pages, and

$write$ and $total$ represent the number of write requests and the sum of both read and write requests. The degree of how much each factor penalize the write cost is controlled by α, a value between 0 and 1.

III. Evaluation

Our experimental setup consists of a server equipped with an Intel Xeon E5-2630 v3 (2.40 GHz, 16 cores) and 32 GB DRAM. A qemu-based emulator, qemu-nvme [7], was used to emulate the OC-SSD. The guest machine was set up with 16 GB memory, 8 CPU cores, and a 20 GB (emulated) OC-SSD. The host-level FTL used in our implementation was *pblk* [8].

To evaluate the applicability of hWA-BC to various types of workloads, a read-intensive workload which holds large amounts of cold data was run together with a random write workload. For this experiment, we used the fio benchmark [9] with the following workload configurations: 4 KB block size, queue depth 16, direct I/O, and libaio ioengine. Both processes were run for 600 seconds. The OC-SSD device was filled to its 85% utilization beforehand the experiment to invoke GC.

Fig. 2 shows the bandwidth of each workload with the original I/O scheduler and the hWA-BC scheduler. The two schedulers show a complete inversion in performance; however, hWA-BC achieves fairness by providing higher performance to the read workload. Although the read workload holds lots of cold data, its number of invalidated pages will be minimal. On the other hand, since the write workload performs random I/O, it will likely have contributed more to the occurrence of garbage collection. Therefore, regarding fairness, the write workload should be penalized, and the read workload should be compensated as it experienced interference by the garbage collection caused by the write workload. From Fig. 2(b), we can see that hWA-BC succeeded in compensating/penalizing each workload according to its contribution to interference. Moreover, the overall throughput of the original and hWA-BC scheduler was 201 MB/s and 193 MB/s, respectively. From this, we can assume that the overhead of hWA-BC (e.g., cost regression analysis, status management) is negligible.

The fluctuations in the bandwidth of the write workload in Fig. 2(a) is due to the write buffering of pblk. As the write buffer is filled up faster with write requests in batches, processes will have to wait longer until there is enough space within the buffer. This explains the sharp drops in bandwidth.

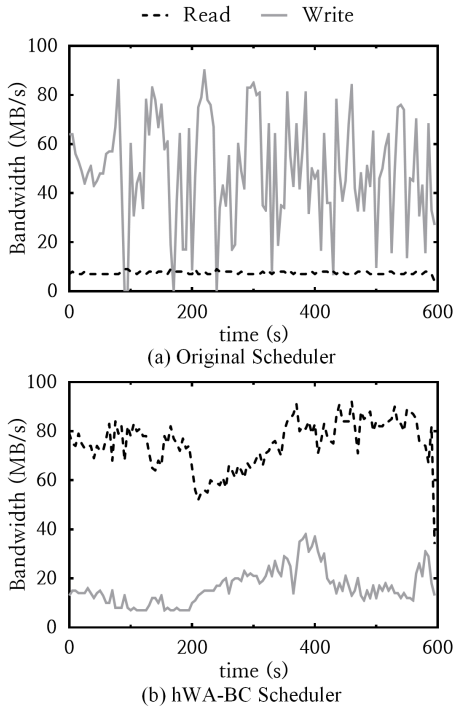


Fig. 2. The throughput (MB/s) of sequential read and random write workloads during garbage collection.

IV. Conclusion

We propose a novel I/O scheduler, hWA-BC, which solves the three problems of the WA-BC scheduler mentioned above. hWA-BC consolidates with the block I/O scheduler; thereby no redundant I/O scheduling occurs. In addition, we propose an algorithm that calculates read and write costs considering the read-write patterns. Finally, we propose a method to determine how much each process contributed to garbage collection even when data from multiple processes share a single flash block. We developed a prototype of hWA-BC and measured its performance using nvme-qemu. We will develop an hWA-BC prototype based on the openSSD Cosmos platform [10] in the future and observe the effects of hWA-BC on various workloads such as virtual machine environments.

Acknowledgment

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT). (No. 2016R1A2B2008672)

References

[1] Q. Zhang, D. Feng, F. Wang, and Y. Xie, "An efficient, QoS-aware I/O scheduler for solid state drive," in *proceedings of the IEEE 10th International Conference on High Performance Computing and Communications & International Conference on Embedded and Ubiquitous Computing*, 2013, pp. 1408–1415.

[2] B. Jun and D. Shin, "Workload-aware budget compensation scheduling for NVMe solid state drives," in *proceedings of the IEEE Non-Volatile Memory System and Applications Symposium*, 2015, pp. 1–6.

[3] "Single root I/O virtualization," 2015. [Online]. Available: http://pcisig.com/single-root-io-virtualization-specification-10

[4] J.-U. Kang, J. Hyun, H. Maeng, and S. Cho, "The multi-streamed solid-state drive," in *proceedings of the 6th USENIX Workshop on Hot Topics in Storage and File Systems*, 2014.

[5] M. Bjørling *et al.*, "Open-channel solid state drives," *Vault, Mar*, vol. 12, p. 22, 2015.

[6] W. K. Josephson, L. A. Bongo, K. Li, and D. Flynn, "DFS: A file system for virtualized flash storage," *ACM Transactions on Storage*, vol. 6, no. 3, p. 14, 2010.

[7] "qemu-nvme," 2018. [Online]. Available: https://github.com/OpenChannelSSD/qemu-nvme

[8] M. Bjørling, J. González, and P. Bonnet, "LightNVM: The linux open-channel ssd subsystem," in *proceedings of the 15th USENIX Conference on File and Storage Technologies*, 2017, pp. 359–374.

[9] "Fio benchmark," 2014. [Online]. Available: http://freecode.com/projects/fio

[10] Y. H. Song, S. Jung, S.-W. Lee, and J.-S. Kim, "Cosmos openSSD: A PCIe-based open source SSD platform," *proceedings of the Flash Memory Summit*, 2014.