Workload-Aware Budget Compensation Scheduling for NVMe Solid State Drives

Byunghei Jun
Memory Division
Samsung Electronics Co.
Hwasung, Korea
Email: byunghei.jun@samsung.com

Dongkun Shin
College of Information and Communication Engineering
Sungkyunkwan University
Suwon, Korea
Email: dongkun@skku.edu

***Abstract*—Recently, solid state drives (SSDs) are replacing hard disk drives (HDDs) in datacenter storage systems in order to reduce power consumption and improve I/O performance. Additionally, in order to mitigate the performance bottleneck at I/O interface between host and SSD, the PCIe-leveraging NVMe SSD is emerging for datacenter SSDs. The NVMe interface supports the I/O virtualization mechanism called single root I/O virtualization (SR-IOV), which is a device self-virtualization technique for supporting direct paths from virtual machines (VMs) to I/O devices. Multiple virtual machines can share an SR-IOV-supporting physical device without intervention of virtual machine monitor. SR-IOV-supporting SSD should provide a device-level scheduler which can schedule the requests from multiple VMs considering performance isolation and fairness. In this paper, we propose a workload-aware budget compensation scheduling algorithm for the device-level request scheduler. To guarantee the performance isolation, the device-level scheduler estimates the contribution on the garbage collection (GC) cost of each virtual machine in the SSD device. Based on the estimated GC contributions, the budget of each VM is compensated for performance isolation. We experimented the effects of the proposed technique with an SSD simulator. The experiments showed that the scheduler can guarantee the performance isolation when multiple VMs share an NVMe SSD with different workloads.**

I. Introduction

Virtualization is the key component of cloud computing. Virtualization can allow the limited hardware resources, such as CPU, Memory, and I/O devices, to be shared among multiple guest machines through virtual machine monitor (VMM) or hypervisor. For I/O virtualization, a software-based sharing technique can be used, where the hypervisor performs the multiplexing and de-multiplexing of I/O requests from many virtual machines (VMs). However, due to the additional software stack, there is a performance overhead for the virtualized hardware compared with native hardware [1]. In order to reduce the virtualization performance gap, a direct assignment approach such as IOMMU [2] and VT-d [3] can be used, which allows direct access from a guest VM. The guest VM can thus perform I/O operations without VMM intervention. However, the direct assignment has limited scalability since a physical device can only be assigned to one VM. To resolve the problem of direct assignment, the self-virtualization techniques are proposed [4], [5], where a physical device provides host with multiple virtual devices, each of which can be assigned to each VM. Therefore, it is possible to assign a physical device to multiple VMs without any software involvement.

Recently, as a self-virtualization interface, PCI-SIG [6] announced the single root I/O virtualization (SR-IOV) interface [7] based on PCI Express (PCIe) interface. SR-IOV can be used at PCIe-based devices such as network interface card (NIC) [8] or solid state disk (SSD) [9]. In particular, the recently proposed NVMe SSD standard interface, which leverages the PCIe interface, includes the SR-IOV feature. Although there is no software involvement when the SR-IOV-supporting device is used, the device should handle the scheduling issues such as performance isolation and fair resource distribution among multiple VMs instead of hypervisor.

In this paper, we propose a device-level scheduling technique for SR-IOV-supporting NVMe SSD. The scheduler can provide the fair resource allocation considering the different I/O workloads of VMs without mitigating the utilization of internally parallel flash memory chips of SSD. The time-slice-based scheduling [10] or the queue-based scheduling [11] has been used in the previous host-level schedulers. Our device-level scheduler adopts the queue-based scheduling technique since NVMe SSD supports multiple command queues in order to utilize the internal parallelism of flash array in SSD. The queue-based scheduling provides a separated queue for each VM, and assigns a predefined time budget to each queue. While SSD processes the requests from VMs, the consumed time is deducted from the budget of the corresponding VM.

There are several issues on the time budget management. The first issue is how to calculate the cost of a request at the parallel architecture of SSD. Since multiple I/O requests can be processed at the multiple flash chips simultaneously and one request can be handled by multiple flash chips, the latency of a request cannot be used as a metric of resource usage.

The second issue is how to guarantee the performance isolation among VMs. During the request handling, the internal firmware of SSD, called flash translation layer (FTL), should perform several internal operations such as garbage collection (GC), wear-leveling, mapping table handling, etc. Depending on the storage access pattern, the internal overhead will be different. For example, whereas read requests and sequential write requests invoke little GC overheads, random and small write requests will invoke large GC overheads. One problem is that the effects of workload will not appear immediately. When a VM generates many random and small write requests, other following VMs may suffer from the garbage collections

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2013R1A1A2A10013598).

978-1-4673-6688-5/15/$31.00 ©2015 IEEE

caused by the previous requests. Therefore, it is very difficult to quantify the contribution of each VM on performance delay. In order to avoid the performance interference between VMs, the scheduler should have a budget compensation mechanism considering the different I/O workloads of multiple VMs.

The proposed device-level scheduler for NVMe SSD is a workload-aware budget compensation (WA-BC) scheduler, which uses a regression-based request cost modeling technique and a budget compensation technique. The request cost modeling technique can estimate the costs of multiple VMs considering the read and write portions of each VM's workload. The budget compensation technique can calculate the performance delay contribution of each VM and can adjust the time budgets of concurrently running VMs.

The rest of this paper is organized as follows. Section II discusses related works. Section III presents the overall architecture of the proposed scheduler, and Section IV describes the budget compensation technique. Section V demonstrates the experimental results. Section VI concludes the paper with a summary and future works.

II. Related Works

There have been several researches on I/O virtualization, which are mostly focused on the optimization of hypervisor [12], [13]. In particular, in order to provide the fairness, CFQ-CRR [14] compensates the overtime in the next scheduling period if a VM uses resources during more than the assigned time in the current scheduling period. mClock [10] tried to minimize the maximum latency by giving the highest priority to the request which is most delayed. These previous researches assume HDD as a storage, which has the same read and write costs. However, SSD has different read and write costs, and the write requests have different costs depending on the storage access pattern.

FIOS [15] and FlashFQ [11] are the host-level schedulers targeting for SSD. They consider the different read and write costs of SSD, and thus read requests are scheduled with a higher priority. In addition, they tried to schedule as many requests as possible concurrently to maximize the utilization of parallel flash chips in SSD. However, they use an offline calibration to estimate read and write costs. Therefore, the run-time cost fluctuations by garbage collections and the performance interferences among different VMs were not considered.

BCQ [16] is the host-level scheduler for fairness among VMs. Considering the runtime varying cost, it recalculates the read and write costs periodically at runtime with a simple regression model. Additionally, it isolates the performances of read-intensive VMs and write-intensive VMs by applying different read and write costs of SSD. However, BCQ does not provide the performance isolation among VMs with different write workloads.

Song and Yang [9] proposed a hardware partitioning technique for SR-IOV-supporting SSD. The technique can guarantee the performance isolation perfectly by dedicating separated channels or flash chips to each VM. However, the parallel architecture of SSD can be under-utilized, especially when the workloads of VMs are not balanced. Compared with the hardware partitioning technique, our approach can fully utilize the parallel architecture of SSD while guaranteeing the performance isolation. The requests of each VM can be handled at any channel or flash chip of SSD.

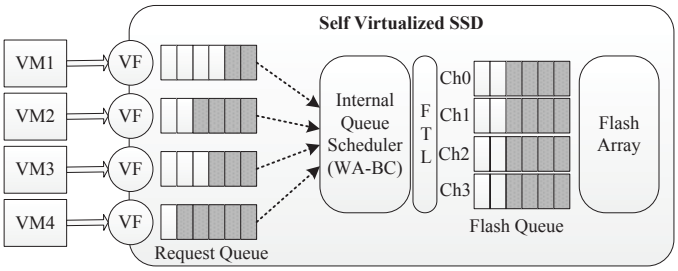


Fig. 1. The internal architecture of self-virtualized SSD.

III. The Architecture of WA-BC Scheduler

The SR-IOV-supporting NVMe SSD provides host with a physical device (physical function, PF) for management and several virtual devices (virtual functions, VFs) each of which can be dedicated to each VM. A VF has its own interrupt and memory area, and the VM can communicate with its VF directly. The SR-IOV SSD has a request queue per each VF as shown in Fig. 1. Each VM sends its requests through the dedicated VF, and the requests are inserted into the corresponding queue. The SR-IOV SSD should have an internal scheduler which is responsible for the fair request scheduling. We propose the workload-ware budget compensation (WA-BC) scheduler for the internal scheduler. Since one request can be striped into multiple flash chips and the basic read or write unit of flash memory is page (e.g., 4 KB), the scheduling is also performed in the unit of flash page. Therefore, a host request is divided into several page-sized requests before it is inserted into the request queue. The WA-BC scheduler selects one queue among multiple queues for next scheduling in the round-robin manner, and sends the first one request in the selected queue to a flash queue. There is one flash queue per one channel, and the target flash queue is determined by the chip allocation algorithm of FTL. Since the requests in different flash queues can be handled concurrently, the requests of different VMs can be processed simultaneously.

The WA-BC scheduler allocates a predefined time budget to each request queue initially, and reduces the time budget whenever a page-sized request is processed. If there is no remaining time budget of a request queue, the request queue is excluded from the scheduling until its budget is replenished. If all the non-empty request queues have no remaining budgets, the scheduler initializes the budgets of all the request queues. The proposed WA-BC scheduler uses a work-conserving scheduling policy in order to maximize the utilization of SSD. Therefore, if there is no waiting request in a request queue, the scheduler skips the queue even though it has a remaining time budget.

In order to manage the remaining time budget, the scheduler should know the time cost consumed by each request. The time cost is different with the request latency. Since multiple requests can be processed concurrently, it is not trivial to estimate the time cost of each request. Moreover, SSD can invoke additional internal operations to manage flash memory such as garbage collection, wear-leveling, and mapping table management. Therefore, the latency of a request will vary depending on the internal status of SSD.

In order to resolve this problem, the WA-BC scheduler uses the regression-based cost modeling technique, which is

introduced by the BCQ [16] scheduler. The on-line profiler measures the execution time for handling a predefined number of requests. When we denote the execution time of i-th profiling interval as T_i, it can be represented as shown in (1).

$$T_i = C_R \times N_R(i) + C_W \times N_W(i) \quad (1)$$

$N_R(i)$ and $N_W(i)$ are the numbers of read requests and write requests in the i-th profiling interval, respectively. (Note that each request is page-sized.) C_R and C_W are the costs of read and write requests to be estimated, respectively. If K number of profiling intervals are elapsed, the WA-BC scheduler can have K number of regression equations and it can get the values of C_R and C_W by performing a linear regression. Since write requests require longer latencies than read requests do, the read and write costs are separated in order to isolate the performances of read-intensive VMs and write-intensive VMs. Since the cost values of C_R and C_W are recalculated at every K profiling intervals, the varying internal SSD overheads can be reflected. The number of profiling intervals can be selected considering the regression error and the sensitivity on workload changes.

The regression analysis cost should be negligible. We evaluated the time and space overheads of regression analysis by compiling the program code for ARM Cortex-R4 processor. The evaluation showed that 320 bytes of code and 1,000 CPU cycles (3 μs on 400 MHz clock) are required additionally for the regression analysis when K is 20.

IV. Budget Compensation

If the workload of one VM generates small and random write requests frequently, it will invoke frequent and high-cost garbage collections. The garbage collection is invoked when there are little available flash blocks, and it generally reclaims old blocks since the old blocks have many invalid pages. Therefore, the VMs which cause the garbage collections and the VMs which suffer from the garbage collections can be different. That is, the internal SSD operations will delay the request handling of other VMs as well as the GC-caused VM. This is definitely unfair.

To resolve the problem, we need a performance isolation scheme for fairness. For the fair scheduling, the contribution of each VM on the internal overhead should be quantified, and each VM should be compensated or penalized based on the contributions. If a VM has a smaller contribution on the garbage collection cost, the VM should be compensated with more time budget. For example, Fig. 2 shows the write request handling costs of two VMs, $\mathrm{VM_A}$ and $\mathrm{VM_B}$. Assume that $\mathrm{VM_A}$ generates random write requests and $\mathrm{VM_B}$ generates sequential write requests. C_{W-A} and C_{W-B} are the write costs of $\mathrm{VM_A}$ and $\mathrm{VM_B}$ when there is no garbage collection. If garbage collections are invoked during the handling of the VMs and the garbage collection cost are evenly distributed to the VMs, each cost is increased by $\mathrm{C_{GC}}$ as shown in Fig. 2(b). However, since $\mathrm{VM_A}$ and $\mathrm{VM_B}$ have different workloads, the compensation algorithm should estimate the contribution of each VM. With the contributions, the scheduler can recalculate the compensated costs as shown in Fig. 2(c). Based on the compensated cost, the time budgets of two VMs are reduced by the WA-BC scheduler. Among several internal SSD operations, we focus on only the garbage collection overhead in this paper since the GC overhead occupies the largest portion and other overheads also can be easily handled with the same approach.

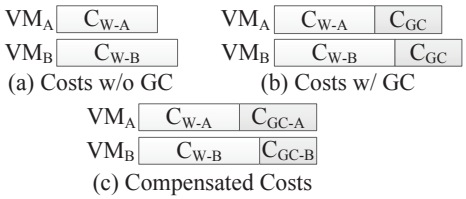


Fig. 2. Cost compensation.

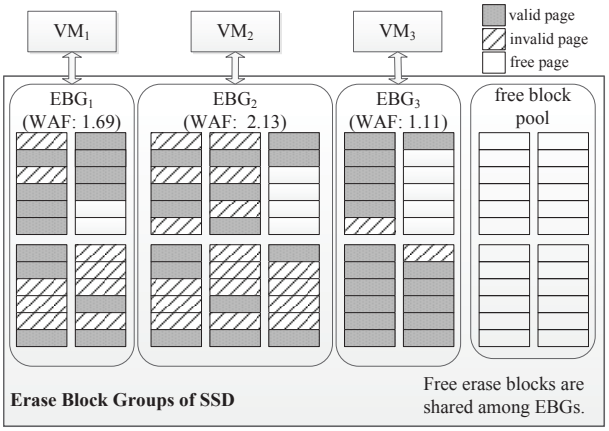


Fig. 3. The erase block separation in the multi-streamed SSD.

For the budget compensation, the scheduler should know how much each VM contributes to the SSD-internal overhead and how much each VM suffers from the overhead. However, it is difficult to estimate the contribution of each VM on the garbage collection cost, since the requests of multiple VMs are concurrently processed in SSD and they can share flash memory blocks. In order to estimate the workload characteristic of each VM, we need to manage separated flash memory blocks for each VM as proposed in the multi-stream SSD architecture [17].

Fig. 3 shows the architecture of multi-streamed SSD. SSD has several flash chips, and a flash chip is composed of several erase blocks. In the multi-streamed SSD, an erase block can be used for only one application or VM. Therefore, the data from different VMs cannot be mixed within an erase block. Each VM has its own erase block group (EBG). When a VM needs more erase blocks, they can be allocated from the free block pool. When the garbage collection generates free blocks, the free blocks are inserted into the free block pool. Therefore, each physical block can be used for different VMs alternatively whenever it is allocated.

Since each VM has its dedicated EBG, the WA-BC scheduler can monitor the workload pattern of the target VM by observing the corresponding EBG. In particular, our scheme measures the write amplification factor (WAF) of each VM in order to estimate the GC contribution of the VM. The WAF stands for the ratio between the amount of consumed storage space and the amount of valid user data as shown in (2). The WAF of a VM is measured with the allocated EBG of the VM. α_i is the WAF value of VM_i. u_i and v_i are the amount of used pages and valid pages in the EBG_i, respectively. The used pages include both valid pages and invalid pages. α in (3) is the WAF value of total workload.

$$\alpha_i = u_i / v_i \quad (2)$$

$$\alpha = \sum u_i / \sum v_i \tag{3}$$

If a VM has a higher WAF value, the workload of the VM consumes more storage space to store data. Therefore, a high WAF of VM invokes the garbage collections more frequently, and the VM has a higher contribution on the garbage collection cost.

For example, in Fig. 3, α_1 (the WAF of EBG_1) is 1.69 since the number of used pages (u_1) is 22 and the number of valid pages (v_1) is 13. α_2 and α_3 are 2.13 and 1.11, respectively. From the WAF values, we can know that VM_2 is most contributive to the garbage collections. The WA-BC scheduler calculates the relative write cost of each VM with the VM's WAF.

In order to estimate each VM's contribution, we can exploit C'_W, which is a normalized cost of a write request assuming that there is no garbage collection. If we assume that the garbage collection cost is proportional to the WAF of total workload, we can define C'_W with C_W and α, as shown in (4). C_W and α are calculated with (1) and (3), respectively. Since each VM uses its dedicated EBG under the multi-streamed architecture, C'_W can be also defined with C_{W_i} and α_i, as shown in (5). Then, we can calculate the value of C_{W_i} with (6). C_{W_i} is the compensated cost considering the WAF of VM_i.

$$C'_W = C_W / \alpha \tag{4}$$

$$C'_W = C_{W_i} / \alpha_i \tag{5}$$

$$C_{W_i} = C_W \times \frac{\alpha_i}{\alpha} \tag{6}$$

For example, assume that there are two VMs, VM_1 and VM_2. If C_W, α, α_1, and α_2 are 16, 1.6, 3.2, and 1.2, respectively, C_{W_1} and C_{W_2} are 32 and 12, respectively. Whereas VM_1 has a higher write cost due to its higher WAF, VM_2 has a lower write cost due to its lower WAF. The calculated write cost of each VM is deducted from the VM's time budget whenever a write request of the VM is processed by SSD.

V. Experiments

A. Experimental Environments

We used the SSD simulator proposed in [18], which is an event-driven and trace-based simulator, in order to evaluate the effects of the proposed algorithm. Several scheduling algorithms such as BCQ [16] and WA-BC are implemented on the SSD simulator. Originally, the BCQ algorithm is proposed for a host-level scheduler. For comparison, we implemented the BCQ scheduler as an SSD-internal scheduler. In addition, the multiple queues of SR-IOV interface and the multi-stream technique are also implemented. The queue depth is 65,535 in order to follow the NVMe specification. The FTL of SSD uses the page-level address mapping algorithm, and the address mapping entries are fully loaded in the internal DRAM of SSD during runtime. A write request can be allocated with any flash chip dynamically considering the waiting requests of flash chips. The over-provisioning area is 3%.

The input I/O traces are collected from a virtualization system, which uses Debian Xen 4.1 [19] as a hypervisor, and the kernel version of the hypervisor is 3.15.8. We modified the hypervisor kernel by inserting *ftrace* calls in order to make the I/O request traces of VMs. Four synthetic workloads generated by *fio* utility and four real user workloads are selected as target

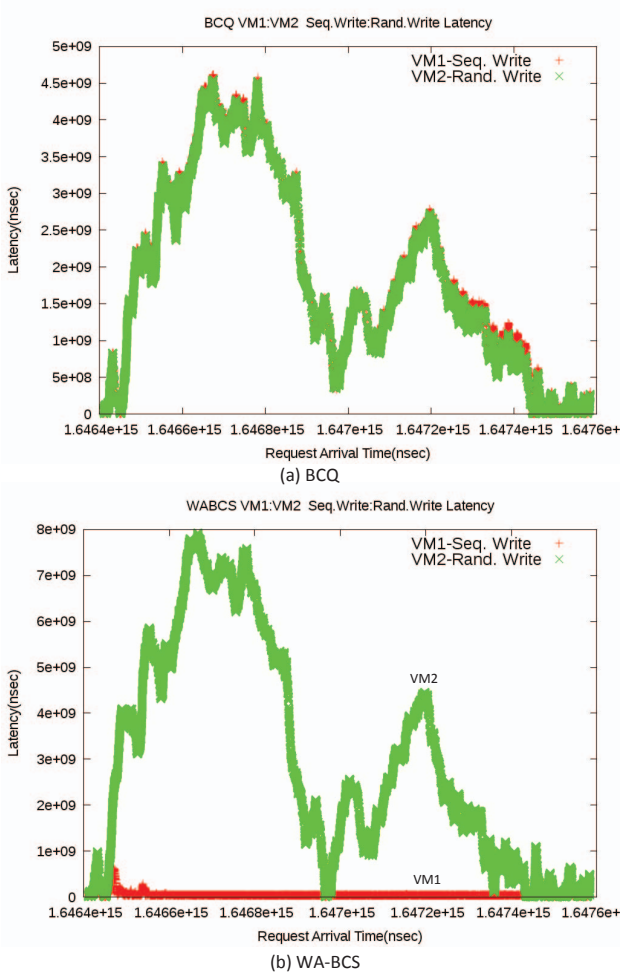
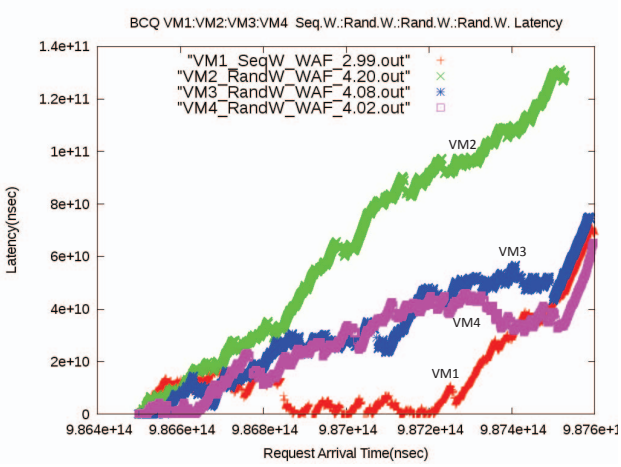


Fig. 4. Case 1: VM1 has the sequential write workload, and VM2 has the random write workload.

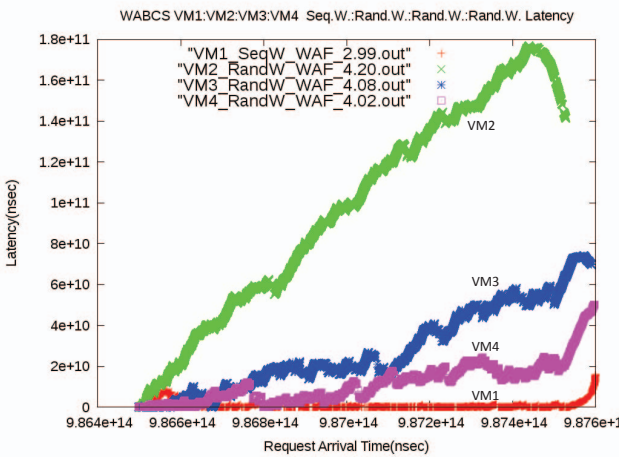
workloads. Each synthetic workload generates only one type of requests among sequential read, sequential write, random read, and rand write. The `O_DIRECT` option is used to disable cache, and the storage space is 4 GB. Initially, the storage space except the over-provisioning area is filled sequentially. Then, 1 GB of data are read or overwritten with the specified access patterns. Each request size is 4 KB. For the real user workloads, three `filebench` workloads, i.e., `webserver`, `varmail`, and `file server`, and a linux kernel code copy scenario are used.

B. Case 1: Two VMs with different write patterns

VM1 has the sequential write workload, and thus its WAF is close to 1. VM2 has the random write workload. Fig. 4 shows the latencies of each request under different scheduling algorithms. The latency varies during the experiments depending on the burstness or idleness of the workloads. Due to the performance gap between real SSD and simulator, there are significantly long latencies. Since BCQ does not discriminate sequential write requests and random write requests, VM1 and VM2 show similar latency patterns under the BCQ scheduler as shown in Fig. 4(a). However, under the WA-BC scheduler, the performance of VM1 is completely isolated from the workload of VM2, and thus the latencies of sequential write requests are very short.



(a) BCQ



(b) WA-BCS

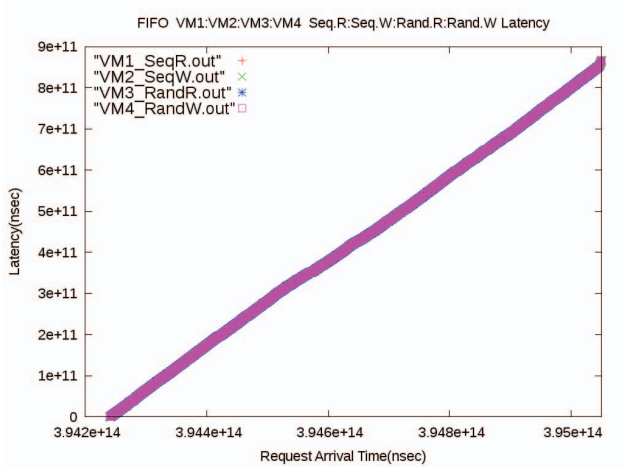
Fig. 5. Case 2: Four VMs have different write workloads.

C. Case 2: Four VMs with different write patterns and different WAFs

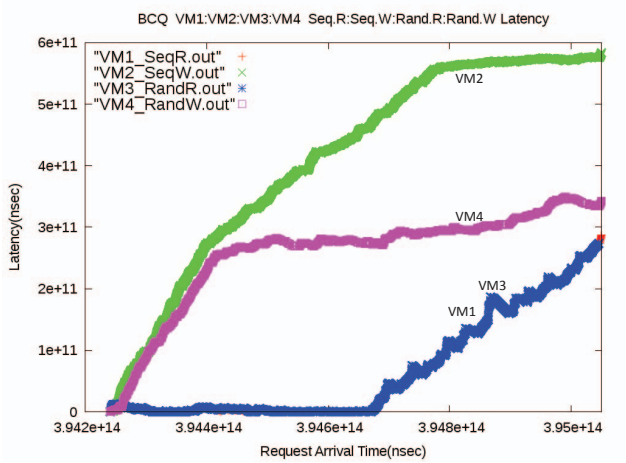
In order to see the effect of budget compensation based on WAF values, different write patterns and different WAFs are applied to four VMs. Whereas VM1 has the sequential write pattern, other VMs have random write patterns. VM1, VM2, VM3, and VM4 have 2.99, 4.2, 4.08, and 4.02 of WAFs, respectively. The different WAFs are generated by changing the overwrite frequency. Under BCQ, VM3 and VM4 show similar latencies even though their workloads have different WAFs as shown in Fig. 5(a). Moreover, the latency of VM1 is also affected by other VMs. However, the WA-BC scheduler provides different performances depending on the different WAFs as shown in Fig. 5(b).

D. Case 3: Four VMs with different workloads

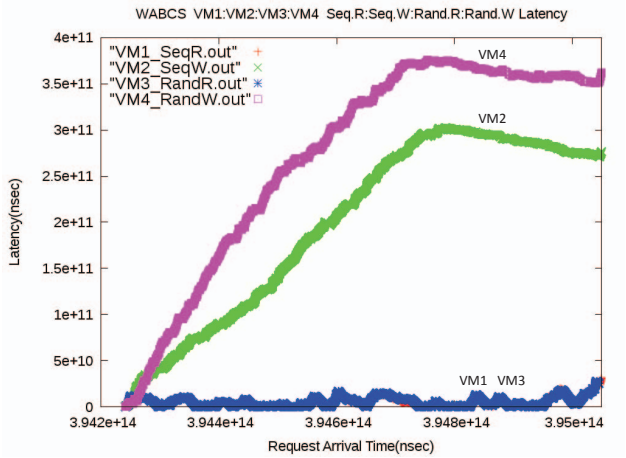
Fig. 6 shows the results of four VMs with different workloads, i.e., sequential read (VM1), sequential write (VM2), random read (VM3), and random write (VM4). The FIFO, BCQ, and WA-BC scheduling algorithms are compared. The FIFO scheduler shows the same latencies for all the VMs since no performance isolation and compensation techniques are applied. Under the BCQ scheduler, VM2 has longer latencies than VM4, even though VM2 has the sequential workload. In addition, the read-intensive VM1 and VM3 are affected by the write-intensive VMs. However, under the WA-BC scheduler,



(a) FIFO



(b) BCQ



(c) WA-BCS

Fig. 6. Case 3: Four different workloads (sequential read, sequential write, random read, and random write) on four VMs.

the performances of read-intensive VMs are completely isolated from the workloads of write-intensive VMs, and VM4 has longer latencies than VM2 since VM4 has the random write pattern.

E. Case 4: Real workloads

Fig. 7 shows the results of real workloads. Table I describes the characteristics of four real workloads. The WA-BC scheduler shows differentiated performances based on the different

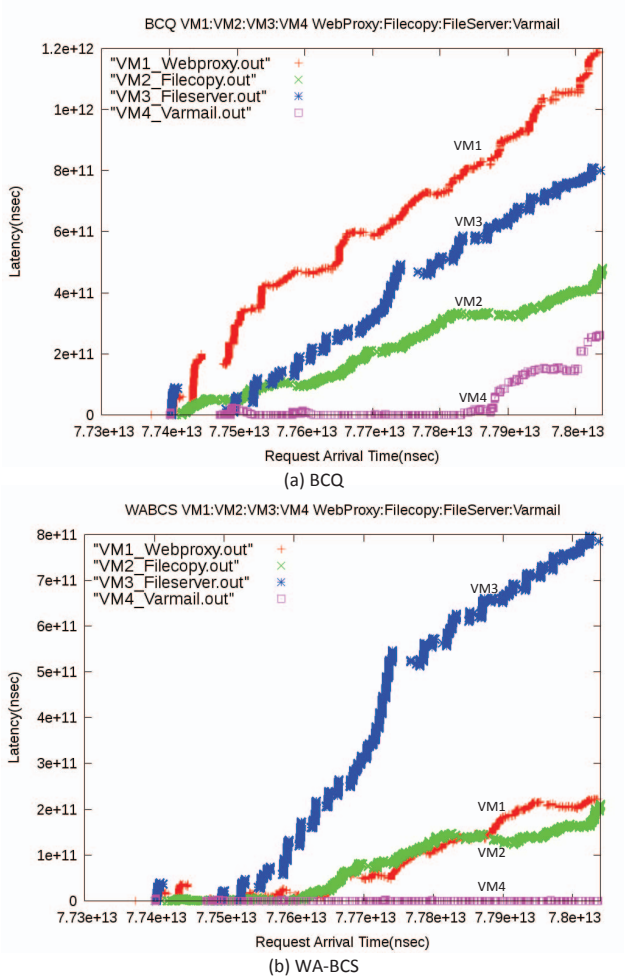


Fig. 7. Case 4: Latency of four VMs with real workloads.

TABLE I. CHARACTERISTICS OF REAL WORKLOADS.

	VM1 WebProxy	VM2 FileCopy	VM3 FileServer	VM4 Varmail
Avg. request size(KB)	16	127	21	15
Write Ratio(%)	35.61	99.95	100.00	99.96
WAF	4.66	5.15	14.37	2.53

WAFs of workloads. However, under the BCQ scheduler, even though VM1 has a lower WAF and a lower write ratio than those of VM2 and VM3, VM1 has the worst performance. Although BCQ uses different costs for write and read requests, the read-intensive VM1 shows long latencies since its write requests are significantly delayed.

VI. CONCLUSIONS

The SR-IOV SSD is a promising device for storage virtualization since it can reduce the hypervisor overhead. However, the SR-IOV-supporting SSD should provide a device-level scheduler for performance isolation and fairness among multiple virtual machines. Since several SSD-internal operations such as garbage collection can affect the performances of VMs arbitrary, we need to estimate the contribution of each VM exactly and compensate each VM based on the contribution in order to provide the performance isolation. In this paper, we proposed the workload-aware budget compensation (WA-BC) scheduler for SR-IOV-supporting NVMe SSD. The WA-BC scheduler quantifies the contribution of each VM on garbage collection overhead by measuring the write amplification factor (WAF) of the erase block group dedicated to the VM in the multi-streamed SSD. Based on the estimated contribution, the WA-BC scheduler gives a compensated time budget to each VM. We verified the effects of the proposed scheme with an SSD simulator and several VM workloads. In all the experiments, the read-intensive and low-WAF VMs show higher performances than the write-intensive and high-WAF VMs do. With the budget compensation scheduler, it will be possible to implement the cloud service level agreement (SLA). As future works, we will evaluate our budget compensation scheduler in a real SR-IOV device. In addition, we will handle other SSD-internal operations such as wear leveling and map loading operation.

REFERENCES

[1] Xen and XenServer Storage Performance. [Online]. Available: http://events.linuxfoundation.org/sites/events/files/slides/20131025%20-%20Storage%20Performance%20PDF.pdf

[2] AMD I/O Virtualization Technology (IOMMU) Specification. [Online]. Available: https://support.amd.com/TechDocs/48882_IOMMU.pdf

[3] Intel Virtualization Technology for Directed I/O. [Online]. Available: http://www.intel.com/content/www/us/en/embedded/technology/virtualization/vt-directed-io-spec.html

[4] H. Raj and K. Schwan, "High Performance and Scalable I/O Virtualization via Self-virtualized Devices," in *Proc. of the 16th International Symp. on High Performance Distributed Computing*, 2007, pp. 179–188.

[5] J. Liu *et al.*, "High Performance VMM-bypass I/O in Virtual Machines," in *Proc. of the 2006 USENIX Conf. on Annual Technical Conference*.

[6] PCI-SIG. [Online]. Available: https://www.pcisig.com

[7] Single Root I/O Virtualization. [Online]. Available: https://www.pcisig.com/specifications/iov/single_root/

[8] M. Musleh *et al.*, "Bridging the Virtualization Performance Gap for HPC Using SR-IOV for InfiniBand," in *Proc. of the 2014 IEEE International Conf. on Cloud Computing*, 2014, pp. 627–635.

[9] X. Song *et al.*, "Architecting flash-based Solid-State Drive for high-performance I/O virtualization," *Computer Architecture Letters*, vol. 13, no. 2, 2013.

[10] A. Gulati *et al.*, "mClock: Handling Throughput Variability for Hypervisor IO Scheduling," in *Proc. of the 9th USENIX Conf. on Operating Systems Design and Implementation*, 2010.

[11] K. Shen and S. Park, "FlashFQ: A Fair Queueing I/O Scheduler for Flash-based SSDs," in *Proc. of the 2013 USENIX Conf. on Annual Technical Conference*, 2013, pp. 67–78.

[12] D. Ongaro *et al.*, "Scheduling I/O in Virtual Machine Monitors," in *Proc. of the 14th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, 2008.

[13] M. Kesavan *et al.*, "On Disk I/O Scheduling in Virtual Machines," in *Proc. of the 2nd Conf. on I/O Virtualization*, 2010.

[14] S. R. Seelam and P. J. Teller, "Virtual I/O Scheduler: A Scheduler of Schedulers for Performance Virtualization," in *Proc. of the 3rd International Conf. on Virtual Execution Environments*, 2007.

[15] S. Park and K. Shen, "FIOS: A Fair, Efficient Flash I/O Scheduler," in *Proc. of the 10th USENIX Conf. on File and Storage Technologies*, 2012.

[16] Q. Zhang *et al.*, "An Efficient, QoS-Aware I/O Scheduler for Solid State Drive," in *Proc. of 2013 IEEE International Conf. on Embedded and Ubiquitous Computing*, 2013, pp. 1408–1415.

[17] J.-U. Kang *et al.*, "The multi-streamed solid-state drive," in *Proc. of the 6th USENIX Conf. on Hot Topics in Storage and File Systems*, 2014.

[18] Y. Hu *et al.*, "Performance Impact and Interplay of SSD Parallelism Through Advanced Commands, Allocation Strategy and Data Granularity," in *Proc. of the International Conf. on Supercomputing*, 2011, pp. 96–107.

[19] Xen Hypervisor on AMD64. [Online]. Available: https://packages.debian.org/wheezy/xen-hypervisor-4.1-amd64