Towards SLO Complying SSDs Through OPS Isolation

Jaeho Kim and Donghee Lee, *University of Seoul;* Sam H. Noh, *Hongik University*

https://www.usenix.org/conference/fast15/technical-sessions/presentation/kim_jaeho

This paper is included in the Proceedings of the
13th USENIX Conference on
File and Storage Technologies (FAST '15).

February 16–19, 2015 • Santa Clara, CA, USA

ISBN 978-1-931971-201

Towards SLO Complying SSDs Through OPS Isolation

Jaeho Kim[1], Donghee Lee[1], and Sam H. Noh[2]

[1]University of Seoul, {kjhnet10, dhl_express}@uos.ac.kr
[2]Hongik University, http://next.hongik.ac.kr

Abstract

Virtualization systems should be responsible for satisfying the service level objectives (SLOs) for each VM. Performance SLOs, in particular, are generally achieved by isolating the underlying hardware resources among the VMs. In this paper, we show through empirical evaluation that performance SLOs cannot be satisfied with current commercial SSDs. We show that garbage collection is the source of this problem and that this cannot be easily controlled because of the interaction between VMs. To control the effect of garbage collection on VMs, we propose a scheme called OPS isolation. OPS isolation allocates flash memory blocks so that blocks of one VM do not interfere with blocks of other VMs during garbage collection. Experimental results show that performance SLO can be achieved through OPS isolation.

1 Introduction

The use of flash memory based Solid State Drives (SSDs) is now commonplace and is being extended to server virtualization [1, 2]. Virtualization systems should be responsible for satisfying the service level objective (SLO) for each VM. Performance service level objectives (SLOs), in particular, are generally achieved by isolating the underlying hardware resources among the VMs. Consequently, many studies for allocating the resources for each VM have been conducted and existing products such as VMware ESX server hypervisor that provide isolated CPU and memory are available [3, 4].

Recent studies making use of SSDs as a shared cache resource among virtual machines (VM) in virtualization systems have been conducted [1, 2]. In this work, we revisit this issue, first, by quantitatively examining IO performance and interference among the VMs within the SSD. We show that depending on the status of the SSD, experimental results can vary significantly, and this difference comes from the interference among the VMs. We then propose *OPS (Over-Provisioning Space) Isolation* at the FTL (Flash Translation Layer) layer such that the OPS of each VM is isolated from being affected by other VMs. We show that performance SLOs of VMs can be satisfied through OPS isolation.

Table 1: Characteristics of IO workloads

Workload	Request Total	Write Ratio	Average Write Size
Financial	7.1GB	0.76	14KB
MSN	14.6GB	0.96	27KB
Exchange	9.8GB	0.67	17KB

The rest of the paper is organized as follows. In the next section, we present the motivation of this work and work related to this study. In Section 3, we look into the internals of SSDs to understand the effects of garbage collection on concurrently executing VMs. In Section 4, we present OPS isolation, the main contribution of this work along with performance evaluations. Finally, in Section 5, we give a summary and conclude.

2 Motivation and Related Work

As motivation, we conduct a set of experiments and observe the performance results that are returned. We show that the performance reported by SSDs vary widely even when executing the same workloads and that the performance of SSDs are strongly affected by their state, which is difficult to control. The results serve as motivation to develop SSDs that are performance predictable.

All experiments in this section are conducted using a commercial SSD that is purchased off-the-shelf. The product uses MLC-based flash memory with a capacity of 128GB. The experiments conducted start from either a clean state or an aged state. Aging is conducted by issuing random writes (including overwrites) of sizes ranging from 4KB through 32KB for a total write that exceeds the SSD capacity. As the SSD becomes full, the SSD becomes busy performing garbage collection. We consider an SSD at this state to be an aged SSD.

Table 2: KVM environment

Description	Host	VM-1∼4
CPU core	8	1
Memory size	32GB	1GB
OS	Ubuntu-14.x with KVM	Ubuntu-14.x
Storage	Dedicated storage	Each 30GB SSD

Three workloads, specifically, Financial, MSN, and Exchange, are used in the experiments. The details regarding the characteristics of the workloads are shown in Table 1. The original workloads used here are traces provided by UMass Trace Repository and MSR [5, 6]. As the experiments in this section use real SSDs, we require real IO requests. Hence, we make use of a replayer tool that takes requests from the trace and turns them into real requests to the device [7]. For each request, a single threaded replayer waits for it to complete, upon which various statistics are gathered.

2.1 Effect of SSD aging

We conduct a set of experiments to show how VMs are affected by various conditions of the storage system. First, we show how proportionality varies as the state of the SSD varies. Our goal is to proportionally distribute IO usage of a shared SSD among VMs. To do so, we first create kernel-based virtual machine (KVM) VMs with the same workloads. The VM settings and the rest of the environment for the experiments are summarized in Table 2. We use the Cgroup [8] Linux feature that limits, accounts, and isolates hardware resource usage of process groups to assign different weights to the VMs (allocating higher throughput for higher weights). We then measure IO performance of each VM.

The results in Figure 1 show that for all the workloads, on the HDD, proportionality is close to the IO weight except for VM-10. However, for the SSDs, proportionality deviates. Note that deviation is worse for the aged SSD than the clean SSD. One can conjecture that this is due to garbage collection (GC), which is largely considered to be the bandit of all wrong in SSDs. Indeed, we show later that during GC, VMs are actually moving other VM's data around, which is unnecessary data movement from the GC triggering VM's point of view, resulting in inaccurate performance control. Another observation from Figure 1 is that the effect of GC on the various VMs is not uniform. That is, we understand that GC is affecting performance in a negative way, but how each VM is affected is not clear.

2.2 Effect of concurrent execution

We perform another set of experiments, this time with a mix of workloads. Four sets of results are presented in Figure 2, and we discuss how they were obtained and

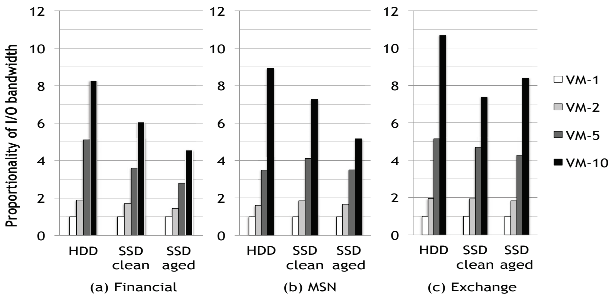


Figure 1: IO bandwidth with Cgroup relative to VM-1 for various workloads. (Notation: VM-x, where x is weight value.)

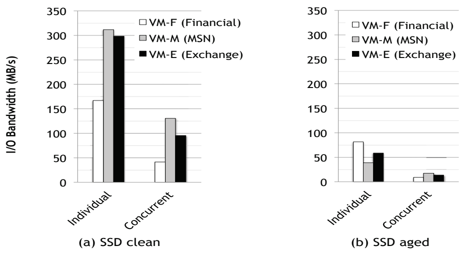


Figure 2: IO bandwidth of individual and concurrent execution of VMs.

what they imply. Figure 2(a) shows the observed bandwidth with a clean SSD. The individual results are obtained by executing each workload starting from a clean SSD for each workload. The concurrent results are obtained by executing the three workloads concurrently on a clean SSD. The three workloads are exactly the same for both individual and concurrent executions, so the total footprint is also the same.

We observe from the results, however, that concurrent execution performs markedly worse than executing each VM individually. With concurrent execution, each VM performance is roughly a third of each individual execution with some deviation among the individual VMs. We also observe that bandwidth is not being consumed in full with the total bandwidth consumed by the three concurrent workloads being roughly 270MB/s.

The results for Figure 2(b) were obtained in a manner similar to that of the clean SSD, only that the SSD goes through an aging process that was described earlier. Three points are noteworthy regarding these results. First, the overall performance drop is significant. Again, the culprit will be GC. Second, we also see that with concurrent execution the performance drop is significant compared to individual execution as was observed with the clean SSD, but the drop is more significant. Finally, and more importantly, the effect of aging on the individual VMs varies considerably depending on the VM. That is, the effect of aging is not uniform. For example, for the individual execution, while the bandwidth of VM-F

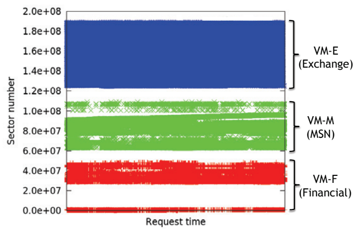


Figure 3: Sector access pattern of the Financial, MSN, and Fileserver workloads.

Table 3: Parameters of SSD simulator

Parameter	Description
Page size	4KB
Block size	512KB
Page read	60us
Page write	800us
Block erase	1.5ms
Page Xfer latency	102us

is reduced by only half, for VM-M, observed bandwidth is reduced to only 15% of the clean SSD case.

Again, we reach the same conclusion as in the previous subsection. That is, we point our fingers at GC for the reduction in performance. However, the effect of GC on individual VMs is not at all uniform. We know GC have negative effects, but how the VMs are being affected is not clear.

2.3 Related Work

In virtualization systems, service level objectives (SLOs) for VMs is achieved through transparent allocation of resources for each VM. Products such as VMware ESX server hypervisor provides isolated CPU and memory to satisfy SLOs [3, 4]. Numerous studies have been conducted to satisfy SLOs for VMs [1, 2, 9, 10, 11, 12, 13, 14]. In particular, DeepDive identifies and manages performance interference between VMs sharing hardware resources [11]. It is regarded as the first end-to-end system that handles interference of major resources such as CPU, memory, and IO.

Studies to provide IO SLOs among VMs have also been conducted [9, 10, 14]. mClock provides proportional-share fairness among the VMs through IO scheduling of the hypervisor [10]. Research on allocation of a shared SSD cache for VMs have also been conducted [1, 2]. S-CAVE effectively manages a shared SSD cache by using runtime information among VMs [1]. vCacheShare addresses the allocation decision for server flash cache (SFC) based on IO access characteristics of running VMs [2]. The goal of their work is in maximizing the utilization of the SSD cache and achieving performance isolation. Our work shows that controlling the SSD from outside the SSD is difficult as one cannot

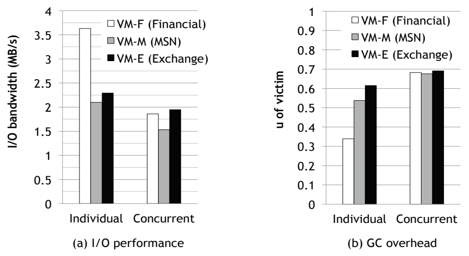


Figure 4: (a) IO performance and (b) GC overhead with SSD simulator

control the internal workings of GC.

A recent study called the Multi-streamed SSD proposes a technique similar to what we propose [15]. Here, Kang et al. propose to make changes to the block device interface to manage blocks based on what they call streams, that is, blocks with similar expected lifetime. This work is different from ours in that their focus is on maximizing the overall performance of SSDs through workload independent block characterization, while we concentrate on controlling each VM within an SSD for SLO compliance.

3 Understanding the Effect of GC

In this section, we first discuss the effect of GC on individual workloads when workloads are run concurrently. This is done using a simulation environment. Then, we present experimental results that imply that commercial SSDs have similar effects.

3.1 GC effect on concurrent workloads

To analyze GC overhead with concurrent workloads, we conduct experiments with SSD extension for DiskSim as the internal workings can be monitored. DiskSim employs a page-mapped FTL used in most SSD products. As for GC, it uses a greedy policy to select the victim block when the number of free blocks drops below a certain threshold. Other parameters of the simulator are presented in Table 3. Also, we use the same workloads as the previous section, and to take into account the VM nature of the previous experiments, the traces that we use for these experiments are those captured as the experiments are performed in Section 2. Fig. 3 shows the sector access patterns for the workloads. The figure shows distinct bands of space being accessed by each workload.

Figure 4(a) shows the IO performance for cases where the workloads are executed individually and concurrently, similarly to those described in Section 2. Here, we age the DiskSim simulator in similar fashion as the commercial SSD before the performance is measured. For the individually run case, the trend in performance is similar to those obtained for the real SSD. For the concurrent case, the trend is quite different, but this is

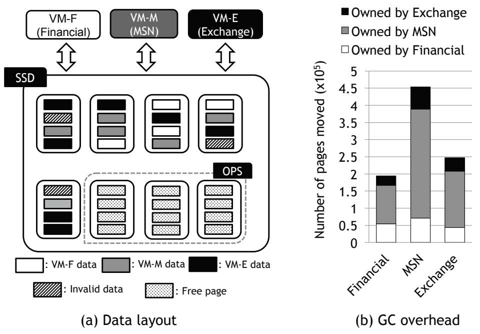


Figure 5: (a) Data layout of concurrent workloads in conventional SSD and (b) number of pages moved for each workload during GC.

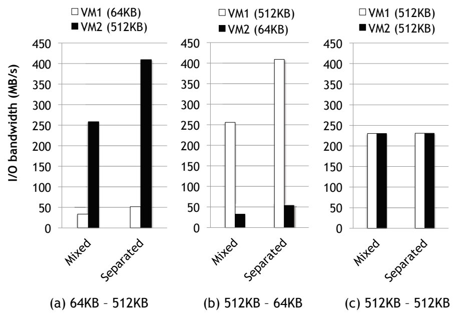


Figure 6: IO bandwidth of VMs generating synthetic requests on a commercial SSD after the 'Mixed' and 'Separated' initialization steps. With 'Mixed', initialization is done with VM1 and VM2 workloads executed concurrently, while for 'Separated', initialization is done by first executing VM1, followed by VM2 execution.

expected as the FTL employed will be different and the three workloads are simultaneously affecting the FTL in various ways. Note, however, that though the exact performance trend may be different, the performance drop of the individual workload varies as was observed for the commercial SSD.

Let us now turn to the reason behind this observation. For this, we observe the internal status and movements of the pages within the device. This is done by tagging each page (in the OOB (Out-Of-Band) area) with the ID of the particular VM that instigated the request and monitoring the tags as the experiments are conducted.

Figure 4(b) shows the average number of valid pages (denoted u, for utilization) of the victim blocks selected for GC. This value is lower when each workload is executed individually than when the workloads are executed concurrently. In particular, the difference in u is proportional to the difference in performance observed in Figure 4(b), that is, largest for Financial and smallest for Exchange. This is to say that while each workload is being negatively influenced by each other as they execute concurrently, Financial is being influenced the most.

The reason behind this negative influence can be explained through Figure 5. Figure 5(a) shows a data layout of a typical SSD when requests from multiple workloads arrive concurrently. The FTL takes each page and randomly places them among the available blocks. Consequently, blocks contain pages from various workloads. Hence, upon an erase while servicing a particular workload, live pages from other workloads in the victim block will be moved to a new block during the GC process.

Figure 5(b) shows the number of pages that are moved during GC for each workload. The pages are distinguished by the owner of the page when they are moved. For example, of the 190K number of pages moved while executing the Financial workload, only 30% of them are those of its own. This says that though GC is a necessity, much of the work involved in the GC process are actually unnecessary work induced by other workloads. Then the solution to this problem is to find a means to isolate the GC process so that GC for each workload does not interfere with other workloads.

3.2 Observation in commercial SSDs

In the previous subsection, we showed results that alluded to the interfering phenomenon using the DiskSim simulation environment. Though simulations are the basis of many important studies and innovations, one still has to wonder if what we observed in the previous subsection actually occurs in real SSDs. To verify this, we perform the following set of experiments.

Taking the commercial SSD that we used previously, we create two VMs, VM1 and VM2 that generates writes and over-writes of 64KB and 512KB sized random requests, which represent small and large requests, respectively. The choice of the two sizes is to vary the mix of data within blocks as will be described below. Hence, the results that we show do not vary for different size choices so long as the two sizes differ by some significant value.

With the two VMs, we perform two different experiments. In the first, starting from a clean SSD, the two VMs are run simultaneously as an initialization step for some amount of time. As a result, the SSD will be populated with data from VM1 and VM2 resulting in data from the two VMs being intermixed. Then, the VMs are run again, but this time the performance is measured and is reported as 'Mixed' in Figure 6(a). In the second set of experiments, we also go through an initialization process but this time the VMs are run one at a time filling in the same amount of data as before. This time, because of the request size and as the VMs are run in sequence, the FTL will (generally) not place data from the two VMs within the same block. Then, the VMs are run and the performance measured. The results for these are reported as 'Separated' in Figure 6(a). Note that the 'Separated' scenario performs substantially better than 'Mixed'.

The results shown in Figure 6(b) are results from the

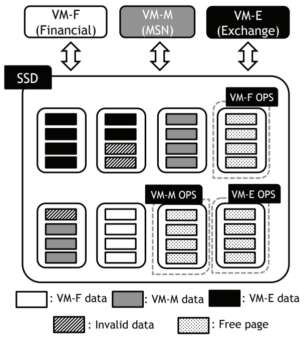
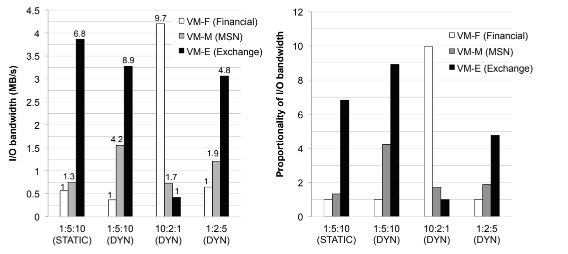


Figure 7: Sample data layout with OPS isolation



(a) I/O bandwidth (b) I/O proportionality

Figure 8: Evaluation results

same experiments with only the order of initialization (512KB first, then 64KB random writes) for the 'Separate' result being different. Figure 6(c) shows the results for the same sequence of experiments, but with same sized (512KB) random writes. These results are shown to contrast them to those of Figures 6(a) and (b).

The reason the 'Separated' scenario performs substantially better than the 'Mixed' scenario is likely because in the 'Separated' scenario the pages from one VM do not negatively influence the other VM. Though we cannot be definite regarding the workings of the SSD due to their propriety nature, these results are in line with the findings of the DiskSim evaluation.

4 SLO Complying SSD and Its Performance Evaluation

In this section, we discuss how the negative effect of garbage collection can be mitigated so that SLO requests may be satisfied. We start by reviewing previous work that formulates IO performance of flash memory based SSDs. We propose OPS isolation as a means to control the performance of individual VMs. Experimental results showing that performance proportionality of VMs can be obtained through OPS isolation are presented.

4.1 Calculating IOPS of SSD

To guarantee IO performance SLO among VMs sharing an SSD, we need to understand the relation between IO performance and the GC overhead. Performance characterizations of NAND flash memory SSDs have been studied extensively, and from these it is well understood that write IO performance can be represented as shown in Equation 1 [16, 17].

$$IOPS_{SSD_W} = \frac{1}{t_{GC} + t_{PRG} + t_{Xfer}} \tag{1}$$

where t_{PRG} and t_{Xfer} are constant values determined by the flash chip manufacturers representing the time to program a page and the time to transfer a page, respectively, and t_{GC}, which is the time to GC defined as $t_{GC} = WAF(u) \cdot t_{PRG}$. $WAF(u)$, which stands for Write Amplication Factor and which is a function of u, the utilization of the flash memory blocks, refers to the additional page writes caused by GC to service the write requests [16]. Studies have shown that $WAF(u)$ can be represented as shown in Equation 2 where N_p is the number of pages per block. Note that u can be measured from the SSD or can be estimated from the ratio of the user data and initial OPS size [18, 19]. Also note, however, u is a value that represents the entire SSD.

$$WAF(u) = \frac{u \cdot N_P}{(1-u) \cdot N_P} = \frac{u}{(1-u)} \tag{2}$$

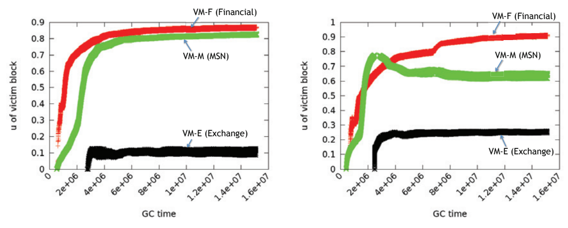
From Equations 1 and 2, we know that write performance of SSDs is determined by the GC overhead, which is determined by u, which in turn, is determined by the OPS [20]. Therefore, to control IO performance, managing OPS properly is imperative.

Typically, OPS is globally managed in SSDs. Hence, VM based IO performance guarantees are difficult, if not impossible, to handle. We propose to isolate OPS handling so that OPS is managed per VM. This allows more manageable control over IO performance for each VM.

4.2 OPS isolation

To satisfy SLO requests from VMs, we propose to dedicate flash memory blocks, including OPS, to each VM separately when allocating pages to VMs so that interference can be prevented during GC. Figure 7 shows an example of how blocks would be allocated among the three VMs concurrently requesting flash space. Contrasting this figure with that of Figure 5(a) shows how the two differ. Observe in Figure 7 all blocks consists of free pages or pages from only one VM. As OPS is also dedicated to a single VM, write requests from each VM will be placed only within the same block preventing pages from different VMs from being mixed.

To satisfy SLO requests, IO performance must also be guaranteed. As discussed in Section 4.1, performance is eventually influenced by the OPS allocated to each VM. Algorithm 1 presents the algorithm that we use to partition the OPS among the competing concurrent VMs. For this study, we simply take the proportional division of the total possible IOPS as satifying the SLO request.



(a) u with Static (b) u with Dynamic

Figure 9: Average u of victim block along GC time

Initially, the flash memory blocks, including the OPS, are partitioned among the competing VMs based on the weight requested. Using the IOPS specified for the SSD, we calculate the estimated IOPS of each VM based on the requested weight. Then, a separate u for each VM can be calculated using Equation 1. After this initialization phase (lines 3 through 11), the OPS size is dynamically and periodically adjusted to maintain the IOPS that was initially designated (lines 12 through 20). For example, take 3 VMs, A, B, and C that are given target proportional IO performance weights of 1, 3, and 6 with higher weights being allotted higher bandwidth. The initial OPS size for each VM is set by the ratio of the weights (lines 3 and 4), that is, 10%, 30%, and 60% of the total OPS is allotted to VMs A, B, C, respectively. If we assume that the specified IOPS for the SSD is 1000, the target IOPS is also set by the weight designated for each VM (line 9). Finally, the target utilization is set for each VM using Equation 1 (line 11). Then, utilization is monitored so that the OPS size can be adjusted if the utilization drifts from the target utilization. This adjusting is done before every GC with the OPS size increased or decreased by one block when necessary.

To implement features such as this in an SSD, the storage interface must change. Recent studies such as the Multi-streamed SSD [15] have proposed changes to the interface for enhanced performance benefits. Similarly, our method requires minimal information, such as a tag identifying the workload, which is already provided with eMMC flash [21], and the SLO requirement such as weight, to be transferred to the SSD.

4.3 Performance Evaluation

To evaluate the SLO complying SSD that we propose, we implement Algorithm 1 in the DiskSim SSD extension [22] that we used in Section 3. For the workloads, we again use the same traces that were previously used.

Figure 8(a) shows the results for the various workloads as the weight of each VM is given differently. In the figure, the x-axis shows groups of VMs that are executed concurrently with the weights allotted to the VMs. For the static case, only lines 3 and 4 of Algorithm 1 are executed and the OPS size does not change throughout the execution. For the rest of the results, the OPS size is

Algorithm 1 OPS Allocation

```
 1: //N: Number of VMs running concurrently
 2: //W(VM$_i$) refers to weight given to VM$_i$
 3: **for** each VM$_i$ **do** //Initialize OPS size for VM$_i$
 4:     OPS(VM$_i$) $\leftarrow$ OPS$_{total}$ $\times$ Ratio of W(VM$_i$)
 5: //Use SSD IOPS value
 6: IOPS$_{total}$ $\leftarrow$ SSD IOPS specification
 7: **for** each VM$_i$ **do**
 8:     //Divide total IOPS according to VM$_i$ weight
 9:     IOPS(VM$_i$) $\leftarrow$ IOPS$_{total}$ $\times$ Ratio of W(VM$_i$)
10:     //Find $u$ for each VM$_i$ using Equation 1
11:     $u$(IOPS(VM$_i$)) $\leftarrow$ Equation 1
12: **Begin Do** periodically adjust OPS:
13: **for** each VM$_i$ **do**
14:     //Otherwise if current utilization is higher
15:     **if** $u$(Cur(VM$_i$)) > $u$(IOPS(VM$_i$)) **then**
16:         Increase OPS(VM$_i$)
17:         //Find VM$_i$ with max current utilization
18:         VM$_i$ $\leftarrow$ Max(VM($u$(Cur(VM$_i$))))
19:         Decrease OPS(VM$_i$)
20: **End Do**
```

dynamically adjusted according to Algorithm 1. The y-axis represents the absolute bandwidth achieved and the numbers on top of each bar represents the performance ratio relative to the bar with the smallest weight. For easy comparison, Figure 8(b) shows the same results in Figure 1 format.

The results show that using OPS isolation and dynamically adjusting the OPS size based on u results in quite accurate proportionality of IO bandwidth. However, static OPS isolation is not effective as there is no leeway to adjust the OPS size according to the workload characteristics.

Figure 9(a) shows how u changes when OPS is set to a static value determined by the proportional weight of the VMs. In contrast, Figure 9(b) shows u changing when the whole of Algorithm 1 is employed, dynamically adjusting the OPS size as need be.

5 Conclusion

In this paper, we showed that performance SLOs cannot be satisfied with current commercial SSDs because of garbage collection interference among competing virtual machines (VM). To resolve this problem, we proposed OPS isolation, a scheme that allocates flash memory blocks in such a way that blocks are not shared among VMs, but are wholly dedicated to each individual VM. Our experimental results showed that OPS isolation is an effective way for SSDs to provide performance SLOs to competing VMs.

Acknowledgement

We would like to thank our shepherd John Strunk and the anonymous referees for their constructive comments that helped to improve the presentation of the paper. We also thank Yongseok Oh for providing us with the trace replayer and Sooyeong Bae for helping with some of the experiments. This research was supported in part by Seoul Creative Human Development Program funded by Seoul Metropolitan Government(No. HM120006), by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MEST) (No. 2012R1A2A2A01045733), and by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology(2010-0025282).

References

[1] Tian Luo, Siyuan Ma, Rubao Lee, Xiaodong Zhang, Deng Liu, and Li Zhou. S-CAVE: Effective SSD Caching to Improve Virtual Machine Storage Performance. In *Proc. of International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 103–112, 2013.

[2] Fei Meng, Li Zhou, Xiaosong Ma, Sandeep Uttamchandani, and Deng Liu. vCacheShare: Automated Server Flash Cache Space Management in a Virtualization Environment. In *Proc. of USENIX Conference on USENIX Annual Technical Conference (ATC)*, pages 133–144, 2014.

[3] VMware Inc. Distributed Resource Scheduler. `http://www.vmware.com/files/pdf/VMware-Distributed-Resource-Scheduler-DRS-DS-EN.pdf`.

[4] Carl A. Waldspurger. Memory Resource Management in VMware ESX Server. *SIGOPS Oper. Syst. Rev.*, 36(SI):181–194, 2002.

[5] UMASS TRACE REPOSITORY. OLTP Application I/O. http://traces.cs.umass.edu, 2002.

[6] Dushyanth Narayanan, Eno Thereska, Austin Donnelly, Sameh Elnikety, and Antony Rowstron. Migrating Server Storage to SSDs: Analysis of Tradeoffs. In *Proc. of ACM European Conference on Computer Systems (EuroSys)*, pages 145–158, 2009.

[7] Yongseok Oh. Trace-replay. `https://bitbucket.org/yongseokoh/trace-replay`.

[8] Paul Menage. CGROUPS. `https://www.kernel.org/doc/Documentation/cgroups/cgroups.txt`.

[9] Ajay Gulati, Irfan Ahmad, and Carl A. Waldspurger. PARDA: Proportional Allocation of Resources for Distributed Storage Access. In *Proc. of USENIX Conference on File and Storage Technologies (FAST)*, pages 85–98, 2009.

[10] Ajay Gulati, Arif Merchant, and Peter J. Varman. mClock: Handling Throughput Variability for Hypervisor IO Scheduling. In *Proc. of USENIX Conference on Operating Systems Design and Implementation (OSDI)*, pages 1–7, 2010.

[11] Dejan Novaković, Nedeljko Vasić, Stanko Novaković, Dejan Kostić, and Ricardo Bianchini. DeepDive: Transparently Identifying and Managing Performance Interference in Virtualized Environments. In *Proc. of USENIX Conference on Annual Technical Conference (ATC)*, pages 219–230, 2013.

[12] R. Prabhakar, S. Srikantaiah, C. Patrick, and M. Kandemir. Dynamic Storage Cache Allocation in Multi-Server Architectures. In *Proc. of Conference on High Performance Computing Networking, Storage and Analysis (SC)*, pages 8:1–8:12, 2009.

[13] David Shue, Michael J. Freedman, and Anees Shaikh. Performance Isolation and Fairness for Multi-tenant Cloud Storage. In *Proc. of USENIX Conference on Operating Systems Design and Implementation (OSDI)*, pages 349–362, 2012.

[14] Eno Thereska, Hitesh Ballani, Greg O'Shea, Thomas Karagiannis, Antony Rowstron, Tom Talpey, Richard Black, and Timothy Zhu. IOFlow: A Software-defined Storage Architecture. In *Proc. of ACM Symposium on Operating Systems Principles (SOSP)*, pages 182–196, 2013.

[15] Jeong-Uk Kang, Jeeseok Hyun, Hyunjoo Maeng, and Sangyeun Cho. The Multi-streamed Solid-State Drive. In *USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage)*, 2014.

[16] Xiao-Yu Hu, Evangelos Eleftheriou, Robert Haas, Ilias Iliadis, and Roman Pletka. Write Amplification Analysis in Flash-based Solid State Drives. In *Proc. ACM International Ststems and Storage Conference (SYSTOR)*, pages 10:1–10:9, 2009.

[17] Yongseok Oh, Jongmoo Choi, Donghee Lee, and Sam H. Noh. Caching Less for Better Performance: Balancing Cache Size and Update Cost of Flash Memory Cache in Hybrid Storage Systems. In *Proc. of USENIX Conference on File and Storage Technologies (FAST)*, pages 313–326, 2012.

[18] Hunki Kwon, Eunsam Kim, Jongmoo Choi, Donghee Lee, and Sam H. Noh. Janus-FTL: Finding the Optimal Point on the Spectrum between Page and Block Mapping Schemes. In *Proc. International Conference on Embedded Software (EMSOFT)*, pages 169–178, 2010.

[19] Wenguang Wang, Yanping Zhao, and Rick Bunt. HyLog: A High Performance Approach to Managing Disk Layout. In *Proc. of USENIX Conference on File and Storage Technologies (FAST)*, pages 145–158, 2004.

[20] Radu Stoica and Anastasia Ailamaki. Improving Flash Write Performance by Using Update Frequency. *Proc. VLDB Endowment*, 6(9):733–744, 2013.

[21] JEDEC. Data Tag Mechanism of eMMC, JEDEC Standard Specification No. JESD84-B45. `http://www.jedec.org/sites/default/files/docs/jesd84-B45.pdf`.

[22] Vijayan Prabhakaran and Ted Wobber. SSD Extension for DiskSim Simulation Environment. `http://research.microsoft.com/en-us/downloads/b41019e2-1d2b-44d8-b512-ba35ab814cd4`, 2009.