Deterministic I/O and Resource Isolation for OS-level Virtualization In Server Computing

Miryeong Kwon[1], *Donghyun Gouk*[1], *Changrim Lee*[1], *Byounggeun Kim*[2], *Jooyoung Hwang*[2], *Myoungsoo Jung*[1]
Computer Architecture and Memory Systems Laboratory,
Korea Advanced Institute of Science and Technology (KAIST)[1], *Samsung*[2]

1 INTRODUCTION AND MOTIVATION

Docker container technology is applied to diverse computing domains such as datacenter, serverless, cloud, and high-performance computing thanks to its predictable development and efficient resource isolation [1,2]. Docker can isolate the execution of specific applications from running other applications as Sandbox [3]. Specifically, users can explicitly set a resource limit for each service by using *control groups* (cgroups) and namespace. Therefore, different services can be free from conflicting dependencies and resource contention. Since this consistent environment supports completely isolated tenants, it is used to improve computing utilization and portability [4,5]. For example, LEMP (Linux, Nginx, MySQL, & PHP) in the Google cloud platform can create multiple tenants per node, each running a separate web server (Nginx), fast CGI (PHP-FPM), and database (MySQL) [6].

While Docker is one of the best options to increase the computing utilization by sharing and/or isolating its resources, it does not yet well manage underlying storage [7]. Docker's virtualized environment enables multiple container executions atop the host-side kernel directly, which is different from hardware stack virtualization of the conventional virtual machines (VMs) [8]. Because of this OS-level virtualization, launching a container is much faster and lighter than executing a VM [9]. However, the multiple containers and host kernel often share the persistent states on a solid-state drive (SSD), which can unfortunately interfere with their executions and I/O services of any peers at a device-level (Figure 1). As modern SSDs in practice exploit internal parallelism to enhance performance [10,11], such interference introduces many storage resource conflicts, thereby degrading the performance. To manage the storage resource, ones may utilize proportional bandwidth of the blkio cgroup adopted by Docker [12] and assign different proportions to each of multiple containers, individually. However, this approach cannot address the interference issue for multi-container execution due to two root causes. First, even though the blkio cgroup throttles the target I/O queue with different proportions [13], the flash-level resource conflicts cannot be resolved as the host is unaware

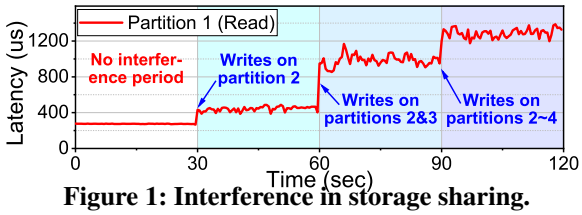


Figure 1: Interference in storage sharing.

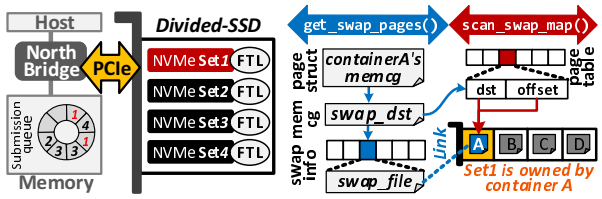
of the physical layout of the underlying SSD [14]. Second, metadata I/O services, including page frame reclaiming, are not taken by I/O throttling of the blkio cgroup.

2 DC-STORE FRAMEWORK

In this paper, we introduce *DC-Store*, a storage framework that supports deterministic I/O performance and resource isolation for the multi-container execution environment. DC-Store consists of two major components, called *Divided SSD* (Figure 2a) and *I/O Tacker* (Figure 2b). Generally speaking, Divided SSD addresses I/O interference at the hardware-level, thereby offering deterministic performance per volume. At the same time, I/O Tacker isolates I/O requests of noisy neighbors from the execution of other peers by considering each container's volume ownership at the software-level.

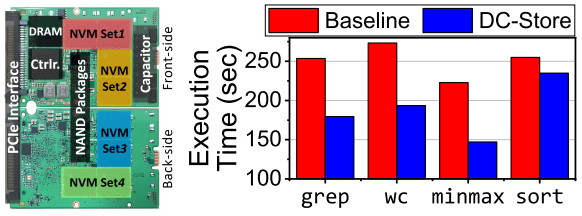
Hardware-level design. We design and implement Divided SSD, which separates physical flash channels and internal resources to support multiple NVM sets, which were very recently included by NVMe workgroup (NVMe 1.4). In specific, we statically partition the computing and hardware resources per NVM set such that different I/O requests heading across different NVM sets can have no or minor flash-level resource conflicts. The proposed Divided SSD can provide an independent environment, which allows multiple containers to operate without the device-level I/O interference. It also enables us to use a large size of shared storage with better utilization per node, which is in typical required by modern data centers [15].

Nevertheless, when multiple containers mount each of different NVM sets (provided by our Divided SSD), we still observe that the user experience of some containerized applications degrades from an execution time angle. Even though



(a) Overview of Divided SSD. (b) I/O Taker.

Figure 2: Overview of DC-Store framework.



(a) HW prototype. (b) Execution time comparison.

Figure 3: Prototype and performance of DC-Store.

Linux cgroups and namespaces can limit/split the CPU, memory, and blkio subsystems for different containers, if there is a noisy container that unintentionally issues page-out I/O requests imposed by a low on virtual memory, the performance of other peers is severely interfered and degraded. Since Docker is not involved in metadata I/O management (and Linux treats containers as just like other processes), it cannot fully support the consistent environment for completely isolated tenants.

Software-level design. To address this shortcoming, the proposed I/O Tacker modifies Linux kernel in order to enable per-container page frame reclaiming by considering the container ownership and exposed NVM set organization. The current version of Linux applies the same page-in/out procedures to all processes and containers, which can introduce container-level performance interference. In contrast, our I/O Tacker assigns a per-container swap area and informs its the swap location to the kernel by modifying the kernel memory controller. Therefore, the noisy neighbor performs page in/out only from/to its own NVM sets. To this end, we also revise the Docker stack for users to pass through the swap-pinning information from the Docker client to the container engine. The I/O Tacker's swap pinning mechanism being aware of the container ownership can isolate noisy neighbors from other containers, thereby addressing the performance degradation of all containers running on the shared storage.

3 EVALUATIONS AND CONCLUSION

Our results show that even though the average latency of our Divided SSD prototype (Figure 3a) is worse than a baseline NVMe SSD for a single volume accesses (with a specific pattern), it provides better and deterministic I/O performance under concurrent storage accesses. As shown in Figure 3b, when we co-run containerized data-intensive applications with memory-hungry containers (e.g., LEMP), the proposed DC-Store improves user-level experiences of the applications by 31%, on average, without degrading the performance of such memory hungry noisy neighbors.

In this paper, we proposed DC-Store to offer deterministic I/O performance for a multi-container execution environment. We prototyped both hardware (Divided SSD) and software (I/O Tacker) modules of our DC-Store, and evaluate them in a real system.

ORIGINAL PUBLICATION

M. Kwon, D. Gouk, C. Lee, B. Kim, J. Hwang, and M. Jung. 2020. DC-Store: Eliminating Noisy Neighbor Containers using Deterministic I/O Performance and Resource Isolation. USENIX FAST, 183-191. https://www.usenix.org/system/files/fast20-kwon.pdf

ACKNOWLEDGEMENTS

This research is mainly supported by NRF 2021R1A2C4001773, KAIST Start-up package (G01190015), and Samsung grant (G01190271/G01200447).

References

[1] A. Eivy, "Be wary of the economics of "serverless" cloud computing," *IEEE Cloud Computing*, vol. 4, no. 2, pp. 6–12, 2017.

[2] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, 2014.

[3] J. Shukla, "Application sandbox to detect, remove, and prevent malware," Jan. 17 2008. US Patent App. 11/769,297.

[4] A. J. Younge, K. Pedretti, R. E. Grant, and R. Brightwell, "A tale of two systems: Using containers to deploy hpc applications on supercomputers and clouds," in *2017 IEEE International Conference on Cloud Computing Technology and Science*, pp. 74–81, IEEE, 2017.

[5] B. P. Rimal and M. Maier, "Workflow scheduling in multi-tenant cloud computing environments," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 28, no. 1, pp. 290–304, 2016.

[6] S. Sakr, "Cloud-hosted databases: technologies, challenges and opportunities," *Cluster Computing*, vol. 17, no. 2, pp. 487–502, 2014.

[7] P. Sharma, L. Chaufournier, P. Shenoy, and Y. Tay, "Containers and virtual machines at scale: A comparative study," in *Proceedings of the 17th International Middleware Conference*, p. 1, ACM, 2016.

[8] "VirtualBox." https://www.virtualbox.org/.

[9] R. Morabito, J. Kjällman, and M. Komu, "Hypervisors vs. lightweight virtualization: a performance comparison," in *2015 IEEE International Conference on Cloud Engineering (IC2E)*, pp. 386–393, IEEE, 2015.

[10] M. Jung and M. T. Kandemir, "Sprinkler: Maximizing resource utilization in many-chip solid state disks," in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, pp. 524–535, IEEE, 2014.

[11] M. Jung, W. Choi, S. Srikantaiah, J. Yoo, and M. T. Kandemir, "Hios: A host interface i/o scheduler for solid state disks," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3, pp. 289–300, 2014.

[12] "cgroup-v1 blkio." https://www.kernel.org/doc/Documentation/cgroup-v1/blkio-controller.txt.

[13] S. Ahn, K. La, and J. Kim, "Improving i/o resource sharing of linux cgroup for nvme ssds on multi-core systems," in *8th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage)*, 2016.

[14] M. Jung, "Exploring parallel data access methods in emerging non-volatile memory systems," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 28, no. 3, pp. 746–759, 2016.

[15] C. Petersen and A. Huffman, "Solving latency challenges with nvm express ssds at scale," *Flash Memory Summit*, 2017. https://www.flashmemorysummit.com/English/Collaterals/Proceedings/2017/20170809_SIT6_Petersen.pdf.