# Building Next-Generation Software-Defined Data Centers with Network-Storage Co-Design

Benjamin Reidys
University of Illinois Urbana-Champaign

Jian Huang
University of Illinois Urbana-Champaign

## Abstract

Software-defined networking (SDN) and software-defined flash (SDF) have been becoming the backbone of modern data centers. They are managed separately to handle I/O requests. At first glance, this is a reasonable design by following the rack-scale hierarchical design principles. But it suffers from suboptimal end-to-end performance, due to the lack of coordination between SDN and SDF.

In this paper, we take an initial effort towards building the next-generation software-defined data center by co-designing the SDN and SDF stacks. We redefine the functions of their control plane and data plane, and split them up with a new architecture named NetFlash. NetFlash has three major components: (1) coordinated I/O scheduling, to coordinate the effort of I/O scheduling across the network and storage stack achieve predictable end-to-end performance; (2) coordinated garbage collection (GC), to coordinate the GC across the SSDs in a rack to minimize their impact on incoming I/O requests; (3) rack-scale wear leveling, which enables global wear leveling among SSDs in a rack by periodically swapping data, for achieving improved device lifetime for the rack.

## 1 Background and Motivation

The software-defined infrastructure has become the new standard for managing data centers, as it provides flexibility and agility for platform operators to customize hardware resources [6, 12, 24]. As the backbone technology, software-defined networking (SDN) allows operators to manage network resources through programmable switches [5, 11, 19]. Similarly, software-defined storage (SDS) [24, 28, 31] has also been developed. A typical example is software-defined flash (SDF) [21, 24], which enables upper-level software to manage the low-level flash chips for improved performance and resource utilization [9, 16, 24]. Since the cost of flash chips has dramatically decreased while offering orders of magnitude better performance than hard disk drives (HDDs), they are becoming the mainstream choice in large-scale data centers [8, 13, 14]. And naturally, SDF has been under intensive study and wide deployment [9, 14, 20, 27].

Both SDN and SDF have their own control plane and data plane, and provide programmability for developers to implement policies for resource management and scheduling. However, SDN and SDF handle I/O requests separately across the network and storage stacks in modern data centers. This suffers from suboptimal end-to-end performance and misses the opportunities offered by the software-defined rack. Ideally, as we forward I/O requests in SDN with advance knowledge of the storage status (e.g., busy, idle, or predicted performance), it can make smarter decisions (e.g., early redirection to data replicas). Similarly, as SDF schedules the received I/O requests, the measured network latency can help the SDF adjust I/O scheduling to meet the service-level objective.

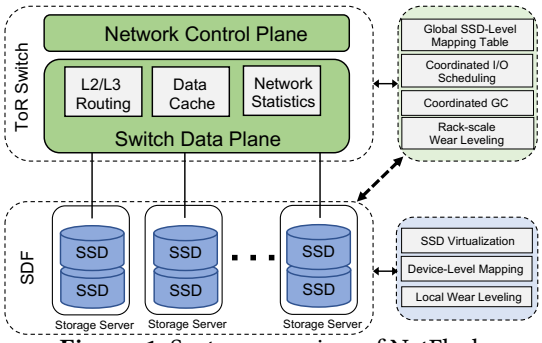


**Figure 1.** System overview of NetFlash.

## 2 NetFlash Design and Implementation

In this paper, we rethink the software-defined network and storage hierarchy, and propose a new software-defined architecture, NetFlash, shown in Figure 1. We first decouple the storage management functions (i.e., flash translation layer) of SSDs, and integrate appropriate functions such as GC and wear-leveling into the SDN to enable SDN/SDF state sharing and global resource management. Second, we use state sharing in the data plane to coordinate I/O scheduling. Third, to alleviate the GC overhead, NetFlash exploits the idle data replicas in the rack and redirects I/O requests that would be delayed by GC. Fourth, NetFlash enables rack-scale wear-leveling using a two-level wear leveling mechanism.

**Decoupling the Storage Management.** As the ToR switch has limited hardware resources, we keep the essential functions for the virtual SSD (vSSD) management locally on storage servers. They include SSD virtualization, device-level mapping, and local wear leveling for SSDs in a server.

To make the ToR switch aware of the states of vSSDs in a rack, NetFlash maintains two tables as shown in Figure 2: (1) replica table, which tracks the GC status and replica of each vSSD; (2) destination table, which tracks the server IP and GC status of each vSSD. For state communication between the ToR switch and storage servers in the rack, NetFlash has its own network packet format based on regular network protocols. The packet has one field to indicate different operations, one field for the target vSSD ID, and one field for the measured network latency for the packets transferred through the data center network. The payload will be filled according to the operation specified in NetFlash header.

**Coordinated I/O Scheduling.** NetFlash tracks I/O requests across the entire stack: (1) $Net_{time}$: the elapsed time in the network since the I/O request is issued until it reaches the storage server; (2) $Storage_{time}$: the delayed time in the I/O queue of the storage stack; (3) $Predict_{time}$: the predicted time

(a) Replica Table

| vSSD_ID | GC Status | Replica vSSD_ID |
|---|---|---|
| vSSD1 | 1 | vSSD12 |
| ... | ... | ... |

(b) Destination Table

| vSSD_ID | GC Status | Server IP |
|---|---|---|
| vSSD1 | 1 | 10.0.0.16 |
| ... | ... | ... |
| vSSD12 | 0 | 10.0.0.20 |

**Figure 2.** NetFlash tables placed in the ToR switch.

to transfer the response back over the network. To manage I/O scheduling in SDF, NetFlash uses $Prio_{sched}$ = ($Net_{time}$ + $Storage_{time}$+ $Predict_{time}$) as the scheduling priority. As NetFlash issues I/O requests from the queue in the storage stack with the maximum $Prio_{sched}$ value. Unlike state-of-the-art storage I/O scheduling schemes, NetFlash considers the network latency to help reduce the end-to-end latency.

NetFlash tracks the $Net_{time}$ with In-band Network Telemetry (INT) in programmable switches [1] and determines the $Storage_{time}$ using the queuing delay for each I/O request in the queue of the storage stack. NetFlash predicts the time it will take to return the response ($Predict_{time}$) with a simple yet effective sliding window algorithm that uses the average network latency measured from the most recent requests.

**Coordinated Garbage Collection.** Since the ToR switch forwards each request entering the rack, it is natural to utilize the switch to coordinate the GC events across these SSDs.

For each read request in the switch, NetFlash queries the replica table to get the GC status and replica for the vSSD. The destination table is queried for for the GC status of the replica. If the vSSD is not performing GC or if both the vSSD and its replica are performing GC, we forward the packet as is. Otherwise, we redirect the request to the replica vSSD.

Since all replicas receive the same writes, they may perform GC concurrently [18]. Thus, we empower the switch to delay a replica's GC by configuring a relaxed *soft_threshold*. Instead of having the SDF notify the switch when it must do GC (at the *hard_threshold*), it requests GC once its free block ratio falls below the earlier *soft_threshold*. The switch can therefore delay GC in one replica until it reaches the *hard_threshold* or until the other replica finishes GC.

NetFlash also opportunistically utilizes idle cycles to free blocks. NetFlash predicts the idle time for a vSSD based on the last interval between I/O requests [3, 22, 25]. Once the predicted interval exceeds a set threshold, the storage server executes the GC and updates the GC status in the switch.

**Rack-Scale Wear Leveling.** To extend the lifetime of a rack of SSDs, we propose a two-level wear leveling mechanism: a local (intra-server) wear balancer processes the local wear balance between SSDs in a server, and a global (inter-server) wear balancer reduces the wear variance for SSDs in a rack. These balancers cooperate to ensure rack-scale wear leveling.

For the local wear balancer, to obtain the uniform lifetime among SSDs in a storage server, we track the average erase count for an SSD and periodically swap the SSD that has incurred the maximum wear with the SSD that has the minimum rate of wear, following the relaxed wear leveling approach developed in [9]. NetFlash can achieve uniform lifetime for SSDs in a storage server by swapping once per 12 days in the worst case. Similarly, we quantify the wear imbalance between storage servers by using the wear of a server (average erase count of its SSDs). However, the swapping cost between servers is more expensive than within a server, due to the networking overhead. Therefore, we relax the swapping frequency to every 8 weeks by default.

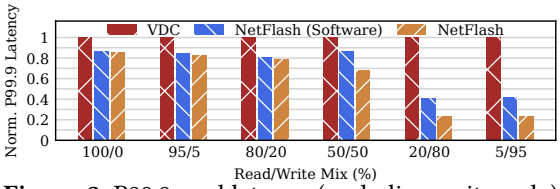


**Figure 3.** P99.9 read latency (excluding write-only).

**Implementation.** We implement NetFlash using a Tofino switch [23] and programmable SSD, whose controller allows read/write/erase operations against raw flash resources. The switch tables are implemented using P4 [4]. The custom packet format is implemented with DPDK [10].

**Evaluation.** We compare NetFlash with state-of-the-art software-defined storage architecture designs: (1) *VDC*: Virtual datacenter (VDC) [2], which enables end-to-end isolation between each flow sharing the same physical network and storage with multi-resource token-bucket rate limiting; (2) *NetFlash (Software)*: We extend VDC by adding software-based coordinated I/O scheduling and GC, instead of using a programmable switch and SSDs.

Our evaluation shows that: (1) NetFlash reduces the tail latency of I/O requests by up to 5.8× for data-center applications (see Figure 3); (2) NetFlash is robust to various storage and network scheduling policies; (3) NetFlash benefits various SSD devices and network latency distributions; and (4) NetFlash ensures the lifetime of an entire rack of SSDs.

## 3 Conclusion and Future Work

We present NetFlash, a new rack-scale storage system by co-designing the software-defined networking and storage stack. As software-defined data centers envision virtualization and pooling of compute, memory, and storage resources for flexibility and agility [29], these resources are increasingly disaggregated over the network [7, 17, 26]. The trend towards disaggregation provides opportunities for co-designing with SDN. As we show with NetFlash, network/storage co-design can improve end-to-end performance through cross-stack state sharing and coordination between SDN and SDF. Accordingly, NetFlash represents an important step towards building next-generation software-defined data centers. Similarly, prior work has identified co-design opportunities between SDN and disaggregated memory [15, 30]. As future work, we wish to exploit potential co-design opportunities between the compute, memory, and storage coordinately in the disaggregated setting, while treating programmable switches as the new operating system for managing hardware resources at rack scale.

## References


[1] In-band Network Telemetry (INT) Dataplane Specification. https://github.com/p4lang/p4-applications/blob/master/docs/INT.pdf.

[2] Sebastian Angel, Hitesh Ballani, Thomas Karagiannis, Greg O'Shea, and Eno Thereska. End-to-end performance isolation through virtual datacenters. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, Broomfield, CO, October 2014.

[3] Eitan Bachmat and Jiri Schindler. Analysis of methods for scheduling low priority disk drive tasks. In *Proceedings of the 2002 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'02)*, 2002.

[4] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3), 2014.

[5] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. P4: Programming Protocol-Independent Packet Processors. *ACM SIGCOMM Computer Communication Review*, 44(3), July 2014.

[6] David Clark, Jennifer Rexford, and Amin Vahdat. A Purpose-Built Global Network: Google's Move to SDN. *Communications of ACM*, 59(3), March 2016.

[7] Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang G. Shin. Efficient memory disaggregation with infiniswap. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI'17)*, Boston, MA, 2017. USENIX Association.

[8] Mingzhe Hao, Huaicheng Li, Michael Hao Tong, Chrisma Pakha, Riza O. Suminto, Cesar A. Stuardo, Andrew A. Chien, and Haryadi S. Gunawi. MittOS: Supporting Millisecond Tail Tolerance with Fast Rejecting SLO-Aware OS Interface. In *Proceedings of the 26th Symposium on Operating Systems Principle (SOSP'17)*, 2017.

[9] Jian Huang, Anirudh Badam, Laura Caulfield, Suman Nath, Sudipta Sengupta, Bikash Sharma, and Moinuddin K. Qureshi. Flashblox: Achieving both performance isolation and uniform lifetime for virtualized ssds. In *Proceedings of the 15th USENIX Conference on File and Storage Technologies (FAST'17)*, Santa Clara, CA, 2017.

[10] Intel. Intel data plane development kit (dpdk), 2022. http://dpdk.org/.

[11] Xin Jin, Xiaozhou Li, Haoyu Zhang, Robert Soule, Jeongkeun Lee, Nate Foster, Changhoon Kim, and Ion Stoica. Netcache: Balancing key-value stores with fast in-network caching. In *SOSP'17*, Shanghai, China, 2017.

[12] Joey Benamy. Software-defined infrastructure: What is it, and why is it the future of it? https://stratacloud.com/blog/software-defined-infrastructure-what-is-it-\and-why-is-it-the-future-of-it, 2016.

[13] Daehyeok Kim, Amirsaman Memaripour, Anirudh Badam, Yibo Zhu, Hongqiang Harry Liu, Jitu Padhye, Shachar Raindel, Steven Swanson, Vyas Sekar, and Srinivasan Seshan. Hyperloop: Group-based nic-offloading to accelerate replicated transactions in multi-tenant storage systems. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM'18)*, 2018.

[14] Laura Caulfield. Project denali to define flexible ssds for cloud-scale applications. https://www.opencompute.org/files/2018-03-OCP-Denali.pdf, 2018.

[15] Seung-seob Lee, Yanpeng Yu, Yupeng Tang, Anurag Khandelwal, Lin Zhong, and Abhishek Bhattacharjee. Mind: In-network memory management for disaggregated data centers. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP'21))*. Association for Computing Machinery, 2021.

[16] Sungjin Lee, Ming Liu, Sangwoo Jun, Shuotao Xu, Jihong Kim, and Arvind. Application-Managed Flash. In *Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST'16)*, Santa Clara, CA, February 2016.

[17] Huaicheng Li, Daniel S. Berger, Lisa Hsu, Daniel Ernst, Pantea Zardoshti, Stanko Novakovic, Monish Shah, Samir Rajadnya, Scott Lee, Ishwar Agarwal, Mark D. Hill, Marcus Fontoura, and Ricardo Bianchini. Pond: Cxl-based memory pooling systems for cloud platforms. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'23)*. Association for Computing Machinery, 2023.

[18] Huaicheng Li, Martin L. Putra, Ronald Shi, Xing Lin, Gregory R. Ganger, and Haryadi S. Gunawi. loda: A host/device co-design for strong predictability contract on modern flash storage. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP '21)*. Association for Computing Machinery, 2021.

[19] Ming Liu, Liang Luo, Jacob Nelson, Luis Ceze, Arvind Krishnamurthy, and Kishore Atreya. Incbricks: Toward in-network computation with an in-network cache. In *ASPLOS'17*, Xian, China, 2017.

[20] Ruiming Lu, Erci Xu, Yiming Zhang, Fengyi Zhu, Zhaosheng Zhu, Mengtian Wang, Zongpeng Zhu, Guangtao Xue, Jiwu Shu, Minglu Li, and Jiesheng Wu. Perseus: A Fail-Slow detection framework for cloud storage systems. In *21st USENIX Conference on File and Storage Technologies (FAST'23)*, Santa Clara, CA, 2023. USENIX Association.

[21] Matias Bjorling and Javier Gonzalez and Philippe Bonnet. LightNVM: The Linux Open-Channel SSD Subsystem. In *Proc. USENIX FAST'17*, Santa Clara, CA, February 2016.

[22] Ningfang Mi, Alma Riska, Qi Zhang, Evgenia Smirni, and Erik Riedel. Efficient management of idleness in storage systems. *ACM Trans. Storage*, 5(2), 2009.

[23] EdgeCore Networks. DCS801 6.4T Programmable Data Center Switch, 2022.

[24] Jian Ouyang, Shiding Lin, Song Jiang, Yong Wang, Wei Qi, Jason Cong, and Yuanzheng Wang. SDF: Software-Defined Flash for Web-Scale Internet Storage Systems. In *Proc. ACM ASPLOS*, Salt Lake City, UT, March 2014.

[25] Benjamin Reidys, Peng Liu, and Jian Huang. Rssd: Defend against ransomware with hardware-isolated network-storage codesign and post-attack analysis. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'22)*, Lausanne, Switzerland, 2022.

[26] SNIA. Rethinking Software Defined Memory (SDM) for large-scale applications with faster interconnects and memory technologies, 2022.

[27] Steve Helvie and Rajeev Sharma. Software defined container-based storage solution has arrived on ocp storage hardware platforms! https://www.opencompute.org/blog/software-defined-container-based-storage-solution-has-arrived-on-ocp-storage-hardware-platforms, 2020.

[28] 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 *Proceedings of the SOSP'13*, Farmington, PA, November 2013.

[29] VMWare. Software-Defined Data Center (SDDC) Solutions, 2023.

[30] Qing Wang, Youyou Lu, Erci Xu, Junru Li, Youmin Chen, and Jiwu Shu. Concordia: Distributed shared memory with In-Network cache coherence. In *19th USENIX Conference on File and Storage Technologies (FAST'21)*. USENIX Association, 2021.

[31] Ning Zhang, Junichi Tatemura, Jignesh M. Patel, and Hakan Hacigumus. Re-evaluating Designs for Multi-Tenant OLTP Workloads on SSD-based I/O Subsystems. In *Proc. SIGMOD'14*, Snowbird, UT, June 2014.