RackBlox: A Software-Defined Rack-Scale Storage System with Network-Storage Co-Design*

Benjamin Reidys† Yuqi Xue† Daixuan Li† Bharat Sukhwani‡
Wen-mei Hwu† Deming Chen† Sameh Asaad‡ Jian Huang†
University of Illinois Urbana-Champaign† IBM T. J. Watson Research Center‡

Abstract

Software-defined networking (SDN) and software-defined flash (SDF) have been serving as 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. However, it suffers from suboptimal end-to-end performance, due to the lack of coordination between SDN and SDF.

In this paper, we co-design the SDN and SDF stacks by redefining the functions of their control plane and data plane, and splitting them up within a new architecture named RackBlox. RackBlox has three major components: (1) coordinated I/O scheduling, to coordinate the effort of I/O scheduling across the network and storage stack to achieve predictable end-to-end performance; (2) coordinated garbage collection (GC), to coordinate the GC activities across the SSDs in a rack to minimize their impact on incoming I/O requests; (3) rack-scale wear leveling, to enable global wear leveling among SSDs in a rack by periodically swapping data, for achieving improved device lifetime for the entire rack. We implement RackBlox using programmable SSDs and a programmable switch. Our experiments demonstrate that RackBlox can reduce the tail latency of I/O requests by up to 5.8× over state-of-the-art rack-scale storage systems.

1 Background and Motivation

Software-defined infrastructure has become the new standard for managing data centers, as it provides the flexibility to customize hardware resources for applications [3]. As the backbone technology, software-defined networking (SDN) allows network operators to manage network resources through programmable switches. Since SDN has demonstrated its benefits, software-defined flash (SDF) [2, 3] has also been developed. Similar to SDN, SDF enables upper-level software to manage the low-level flash chips for improved performance and resource utilization [2, 3].

Both SDN and SDF have their own control plane and data plane, and provide programmability for developers to define and implement their policies for resource management and scheduling. However, SDN and SDF are managed separately in modern data centers. At first glance, this is reasonable by following the rack-scale hierarchical design principles. However, it suffers from suboptimal end-to-end performance, due to the lack of coordination between SDN and SDF.

Although SDN and SDF make the best effort to achieve their quality of service, they do not share their states and lack global information for storage management and scheduling, making it challenging to achieve predictable end-to-end performance. Prior works [1] proposed various software techniques such as token bucket and virtual cost to enforce performance isolation across the rack-scale storage stack. However, they treat the underlying SSDs as black boxes, and cannot capture hardware events, such as garbage collection (GC) and I/O scheduling. Thus, it is challenging to achieve predictable I/O performance across the entire rack.

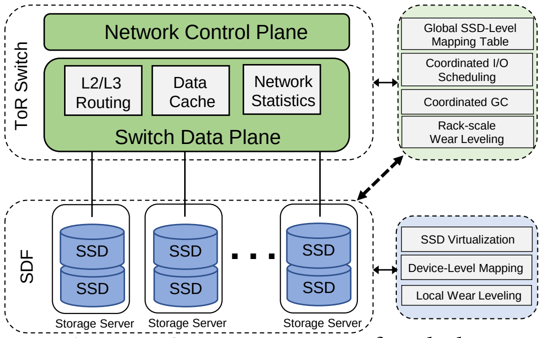

Figure 1. System overview of RackBlox.

2 Design and Implementation

In this paper, we rethink the software-defined network and storage hierarchy, and propose a new software-defined architecture, RackBlox as shown in Figure 1. We first decouple the storage management functions (i.e., flash translation layer) of SSDs, and integrate the 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, RackBlox exploits the idle data replicas in the rack and redirects I/O requests that would be delayed by GC. Fourth, RackBlox 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 virtualized 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, RackBlox 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, RackBlox defines a custom network packet format based on regular network protocols. The packet has one field for the operation type, one for the vSSD ID, and one for the measured network latency as the packets are transferred through the network. The payload is filled according to the operation type.

Coordinated I/O Scheduling. RackBlox tracks I/O requests across the entire stack: (1) Net_{time}: the elapsed time in the

*This work was published at SOSP 2023 [4]

(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. RackBlox tables placed in the ToR switch.

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 to transfer the response back over the network. To manage I/O scheduling in SDF, RackBlox uses $Prio_{sched}$ = (Net_{time} + $Storage_{time}$+ $Predict_{time}$) as the scheduling priority. As RackBlox 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, RackBlox considers the network latency to help reduce the end-to-end latency.

RackBlox tracks the Net_{time} with In-band Network Telemetry in programmable switches and determines the $Storage_{time}$ using the queuing delay for each I/O request in the queue of the storage stack. RackBlox 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 incoming 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, RackBlox queries the replica table to get the GC status and replica for the vSSD. The destination table is queried 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. 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.

RackBlox also opportunistically utilizes idle cycles to free blocks. RackBlox predicts the idle time for a vSSD based on the last interval between I/O requests. 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

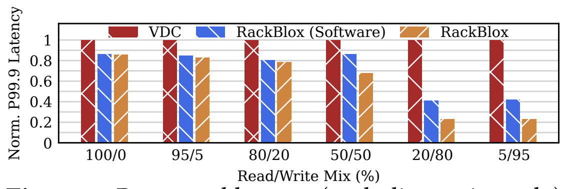

Figure 3. P99.9 read latency (excluding write-only).

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 [2]. RackBlox 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.

Implementation. We implement RackBlox using a Tofino switch and programmable SSD, whose controller allows read-/write/erase operations against raw flash resources. The switch tables are implemented using P4. The custom packet format is implemented with DPDK.

Evaluation. We compare RackBlox with state-of-the-art software-defined storage architectures : (1) *VDC*: Virtual datacenter [1], which enables end-to-end isolation between flows sharing the network and storage with a multi-resource token-bucket; (2) *RackBlox (Software)*: We extend VDC with software-based coordinated I/O scheduling and GC.

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

References

[1] 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.

[2] 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.

[3] 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.

[4] Benjamin Reidys, Yuqi Xue, Daixuan Li, Bharat Sukhwani, Wen-Mei Hwu, Deming Chen, Sameh Asaad, and Jian Huang. Rackblox: A software-defined rack-scale storage system with network-storage co-design. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, page 182–199. Association for Computing Machinery, 2023.