Dawid: A Cost-effective I/O Scheduling Scheme for SSDs under Capacitance Constraints

Sanghyun Nam
Soongsil University

Seungmin Shin
Soongsil University

Eunji Lee
Soongsil University

Abstract

Enterprise-class SSDs realize a persistent buffer within device using the capacitors. This approach reached a limit as the SSD density outpaces the scaling capability of the capacitors. This paper presents a novel buffer architecture for SSDs under capacitance constraints. We re-order the write requests in a way of minimizing the memory footprint that needs protection. This design reduces the required capacitance to ensure persistence in SSDs at the modest performance loss. We implemented `Dawid` in an open-source SSD development framework. Performance evaluation shows that `Dawid` has only 10% more flash writes than full-protection SSD, while the required capacitence is reduced by 70%.

1 Introduction

The SSD has increased significantly in density for the past decade. As an example, in 2011, a typical 2.5-inch SSD had 256GB capacity, but by 2018, a high-capacity SSD boasted a 30TB, expanding by 100× over the past ten years [**?**, **?**]. This remarkable growth of the device-capacity is thanks to the advanced scaling technologies such as nanoscale fabrication [**?**] and multi-layer stacking [**?**].

Unfortunately, not all components of the SSDs have kept up with the scaling rate. The capacitor, which is adopted in enterprise-class SSDs for power-loss protection (PLP), fails to proceed at the pace. Historically, storage devices have been equipped with a small size of volatile buffer in front of the persistent disk. By using them as a read cache and a write buffer, they hide a long latency of the physical storage medium as well as mitigating an endurance limitation of the worn-out devices. However, the volatile buffer loses all data in the event of power crash. To prevent a data loss or corruption by this, enterprise-class SSDs rely on the capacitors; it reserves energy to persist data in volatile buffer in the unforeseen event of a power crash. In addition, the adoption of capacitors enables an SSD to ignore the `FLUSH` command that explicitly requests all data in the volatile buffer to be made durable. This property increases the buffering effect in SSD significantly, leading to both less write traffic and a shorter operation latency.

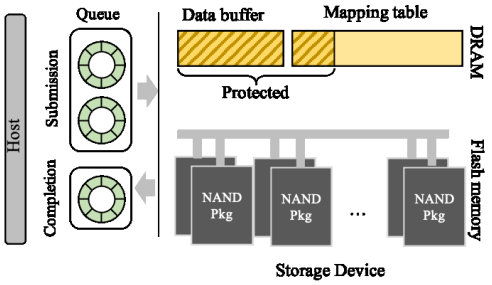


Figure 1: **SSD architecture with `Dawid` buffer.**

The reliance on capacitors, however, has reached its limit. Although the capacitance density has also steadily improved, it is not as rapid as the SSD scaling speed. Al(aluminum) and Ta(tantalum)-electrolytic capacitors used in SSDs have increased in density by tenfold from 1960 to 2005. This is approximately 50x slower than the SSD density increase rate. Given that the internal buffer size increases in proportion to the storage capacity (typically 0.1% of storage capacity [**?**, **?**]), the density gap between capacitance and memory technologies imposes an intrinsic limitation on the current architecture wherein the entire buffer is protected by capacitors.

This paper presents a device-internal buffer architecture called `Dawid` for the SSDs under capacitance constraints. Fig. 1 shows the SSD architecture targeted in our study. The data maintained in the buffer can be classified into two types: the actual user data and the metadata for SSD management (i.e, mapping table). When the buffer is partially protected, the number of dirty pages is limited to the maximum amount of data that the on-board

capacitance can protect. If the number of dirty pages goes beyond the limit, changes should be flushed to the flash memory immediately to meet the durability constraint for SSDs. `Dawid` applies this compromise only to the metadata, while protecting the user data entirely. The data write is not only synchronous with the user request, which hampers user experiences seriously when delayed, but also unrecoverable in the event of the power crash.

For this architecture, the problem boils down to how to reduce the write traffic to persist the mapping table to flash memory under capacitance constraints. Reducing the synchronization overhead of the mapping table is a well-known problem and has been extensively studied for a past decade [**?**, **?**]. However, they mostly focus on the SSDs without PLP, which have different properties to the PLP-SSDs with capacitance constraints. As opposed to the SSDs without capacitors, (1) the PLP-SSDs ensure data persistency immediately, and (2) there is no need to write-back the buffered data when protected. In this regard, the `Dawid` buffer aims at minimizing the *dirty memory footprint* of the mapping table at any point in time. To this end, `Dawid` processes outstanding requests in the order that least increases the number of dirty pages in the mapping table. This scheme not only reduces the number of flushes for the mapping table partially protected but also increases the efficiency of flush operation by aggregating more translation updates into the smaller translation pages.

The `Dawid` buffer is built upon the current trend of increasing queue depth of the storage interfaces. SATA and SAS support a single queue with 32 and 245 commands, but NVMe has up to 65,535 queues with as many as 65,536 commands per queue. This extension allows SSDs to further optimize the internal activities by taking advantage of the outstanding request information. By re-scheduling the requests in a way of reducing the write-back cost of the mapping table, the `Dawid` buffer achieves high IOPS and low latency of SSDs at a small amount of capacitance.

To evaluate the effectiveness of `Dawid`, we implement the proposed buffer design in `FEMU`, which is an open-source SSD development framework [**?**]. The performance evaluation with various workloads shows that `Dawid` reduces the write traffic by up to 78% and provides 25% higher IOPS compared to the FIFO scheduling scheme when only 10% of the mapping table is protected. Compared to the full-protection architecture, `Dawid` has has 20% more writes and 9% of performance overhead, while reducing the required capacitance by 90%.

The remainder of this paper is organized as follows. In section 2, we briefly review the requirements and constraints for SSD with respect to reliability. Section 3 explains the design of SpartanSSD and Section 4 describes the implementation and performance evaluation results. Section 5 quantitatively analyzes the energy consumption of SSDs. Section 6 discusses our propoed design in relation to prior work, and Section 7 concludes.

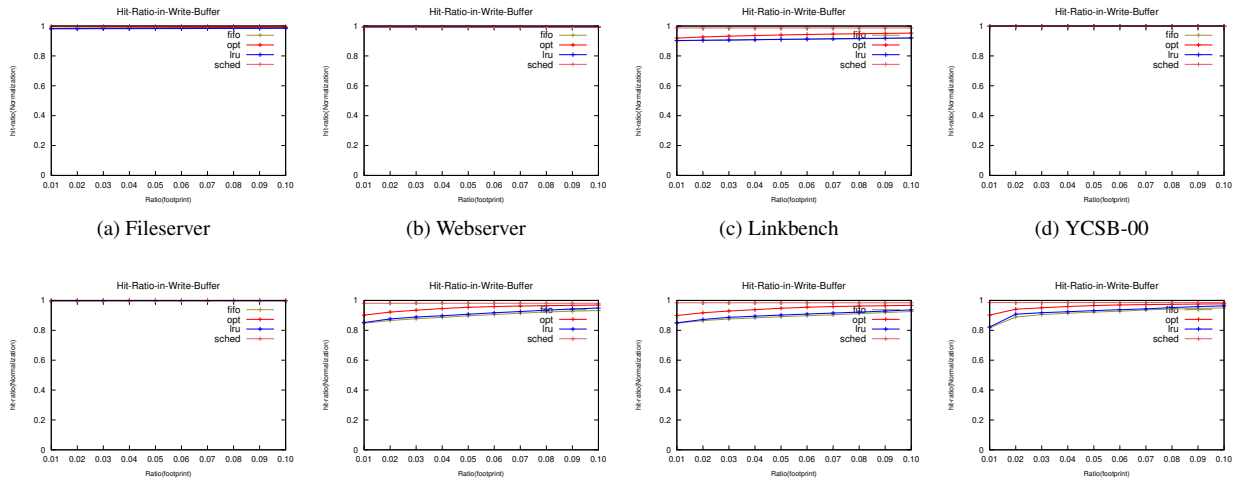
Notes

Workload	Reqs.	Footprint(D)	Footprint(M)	Total I/O(D)	Total I/O(M)	Map-data-ratio(Percentage)
fileserver	4965791	1.36GB	22.27MB	18.94GB	346.95MB	1.79
webserver	15264309	27.81GB	445.70MB	58.22GB	448.15MB	0.75
linkbench	3412657	4.00GB	174.61MB	13.02GB	1314.69MB	9.86
YCSB-00	70682260	162.571GB	332.03MB	269.63GB	552.15MB	0.20
YCSB-01	58037200	123.82GB	252.73MB	221.39GB	453.60MB	0.20
Systor-16LUN3	1200345	1.97GB	89.06MB	4.58GB	715.70MB	15.26
Systor-16LUN4	1000701	1.62GB	61.33MB	3.82GB	596.50MB	15.26
Systor-18LUN3	1464747	2.19GB	83.20MB	5.59GB	1044.59MB	18.26

Table 1: **Workload Characteristics(fifo policy, protected 0.01).**

Workload	Reqs.	Footprint(D)	Footprint(M)	Total I/O(D)	Total I/O(M)	Map-data-ratio(Percentage)
random	5000000	GB	MB	19.07GB	18.88GB	98.99
fileserver-300s	1143106	1.14GB	2.41MB	4.36GB	9.98MB	0.22
fileserver-60s	413613	1.04GB	2.24MB	1.58GB	3.65MB	0.23
linkbench-load	3456185	3.19GB	6.62MB	13.18GB	48.72MB	0.36
linkbench-run	3272243	3.41GB	23.11MB	12.48GB	1.14GB	9.11

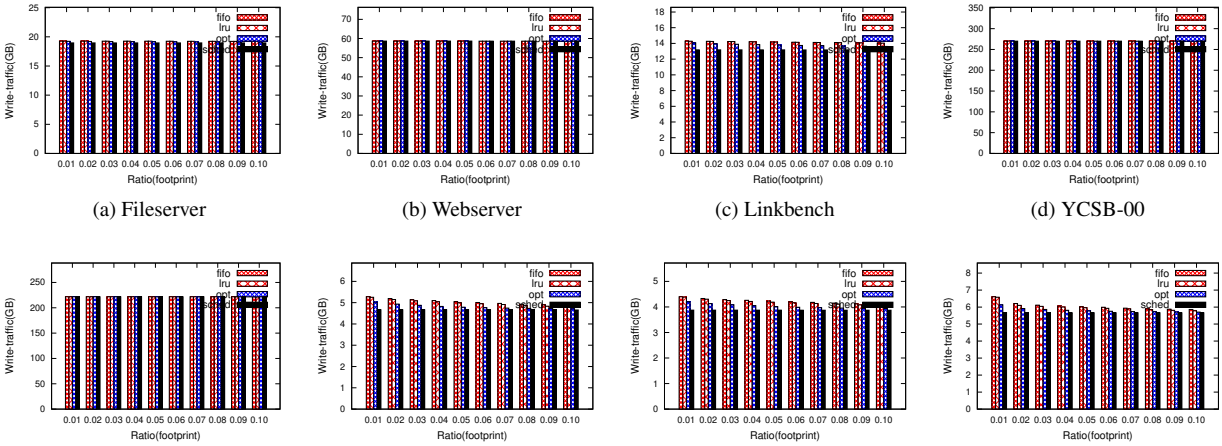
Table 2: **Workload Charateristics(filesystem aging).**



(a) Fileserver (b) Webserver (c) Linkbench (d) YCSB-00

(e) YCSB-01 (f) Systor-16LUN3 (g) Systor-16LUN4 (h) Systor-18LUN3

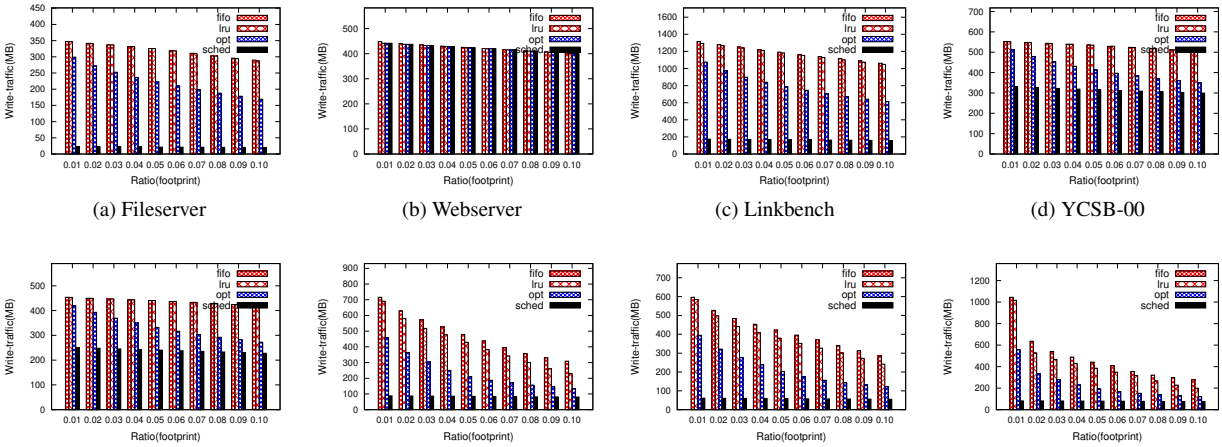
Figure 2: **Hit ratio in protected mapping table pages.**



(a) Fileserver (b) Webserver (c) Linkbench (d) YCSB-00

(e) YCSB-01 (f) Systor-16LUN3 (g) Systor-16LUN4 (h) Systor-18LUN3

Figure 3: **Write-back traffic of total data.**



(a) Fileserver (b) Webserver (c) Linkbench (d) YCSB-00

(e) YCSB-01 (f) Systor-16LUN3 (g) Systor-16LUN4 (h) Systor-18LUN3

Figure 4: **Write-back traffic of mapping table.**