Wonderful : A Terrific Application and Fascinating Paper

Sanghyun Nam
Soongsil University

Eunji Lee
Soongsil University

Abstract

Enterprise-class SSDs realize a persistent internal buffer 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, called Dawid. Assuming that a portion of buffer Dawid schedules the write requests such that the number of dirty pages within buffer is minimized at a time window. This design significantly mitigates the additional flash writes that are inevitable under capacitance constraints. 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

With the growing popularity of data-intensive applications, the SSD density has increased rapidly in recent years. 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 various scaling technologies such as nanoscale fabrication [**?**] and multi-layer stacking [**?**].

However, 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. In general, storage devices have been equipped with a small size of volatile buffer. By using them for read caching and/or write buffering, 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.

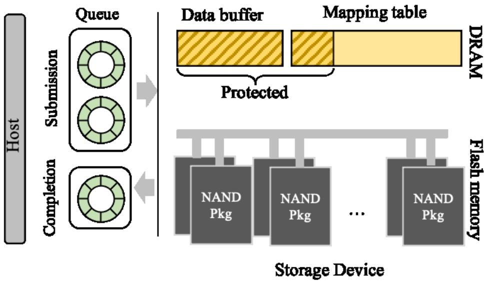


Figure 1: **SSD architecture with Dawid buffer.**

The reliance on capacitors, however, has reached its limit. Although the capacitance density has also steadily improved, it is not as rapid as 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 that fully protects the entire buffer data with capacitors.

With this motivation, this paper presents an SSD-internal buffer architecture, called Dawid, which operates under capacitance constraints. Dawid essentially limits the number of dirty pages within buffer to the level the maximum capacitance can protect. If the number of dirty pages goes beyond the limit, they are flushed to flash memory. Note that the data maintained in buffer can be classified into two types: the actual user data and the metadata needed for their management in SSD (e.g., mapping table). In Dawid, the capacitance constraint is applied only to the metadata, while protecting the user data entirely. This is because the storage suffers from serious performance degradation when the user data is not fully protected and should be entirely flushed upon a FLUSH command.

Based on this underlying architecture, Dawid performs an I/O scheduling for the write requests within storage queue to reduce the write traffic caused by the capacitance limitation. The scheduling algorithm aims to minimize the dirty page footprint of the internal buffer at a given time window. This behavior can reduce the frequency at which the number of dirty pages exceeds the threshold and the modified pages are forced to flash memory. To this end, Dawid prioritizes the write request with the least increase of the number of dirty pages in the mapping table. With this policy, the write request of which the associated mapping page is already dirty has a top priority because it does not add the dirty page count. For other cases, Dawid groups the write requests that modify the same mapping page and process them in batch in the order of group size. This policy enhances buffering and coalescing of the changes to the mapping table in the buffer, reducing a flush overhead significantly under capacitance constraints compared to the FIFO (First-In First-Out) scheduling policy currently used in SSDs.

To evaluate the effectiveness of Dawid, we implement the proposed buffer design in 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 policy 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.

2 Related work

2.1 Reducing Capacitance Requirement

The need for reducing the energy consumption needed for power-loss protection arises in different contexts. A few studies reduce the total energy consumption by speeding up the back-up process at a power failure using the fast media. Guo et al. reduce the capacitance requirement by writing back the volatile buffer data into PRAM (Phase Change Random Access Memory), which is faster and uses lower power than NAND flash [?]. They argue that this reduction enables to replace the supercapacitors that are suffering from serious aging problems with the regular capacitors, which have more reliable characteristics [?]. This consequently enhances the robustness of storage device. As a similar approach, Smartbackup [?] proposes dynamic NAND channel allocation and SLC (single-level cell) mode programs to make the dump process shorter at sudden power-off. It makes full use of available SSD channels and dynamically adjusts these channels based on the available power of the capacitor to exploit the nature of high parallelism on NAND flash arrays. In addition, as the SLC mode program shows significantly shorter time than subsequent MLC, TLC, or QLC mode, it programs the target page to dump in SLC mode to achieve shorter time required for dumping process.

Configuration	**Size**
Channel	8x
Way	4x
Die	4x
Plane	4x
Page Size	4KB
SSD Capacity	512GB

Table 1: **SSD Configuration.**

Data Type	**Size**
User Data Buffer	4MB
Mapping Table	512MB
Mapping Table Directory	512KB
Metadata for Allocation	1MB
Metadata for GC(Garbage Collection)	9MB
Total Buffer Memory	526.5MB

Table 2: **Components of the SSD-internal buffer.**

Another approach to reducing the capacitor size is protecting a part of the volatile buffer. DRWB (Dual-Region Write Buffer) divides the internal-SSD buffer into small protected region (backed by a capacitor) and large unprotected region and when the data on unprotected region is updated, the delta for the page is logged in the protected region [?]. With this differential logging, DRWB logically realizes the non-volatile buffer using a small size of capacitor. However, the proposed technique only regards the user data, having no consideration on the metadata such as mapping table, despite that it actually accounts for most of the internal buffer of SSDs. Furthermore, commercial SSDs typically do not cache read data in the buffer because the host memory can serve as a cache memory of the storage device. For these reasons, the effectiveness of DRWB may be limited in practical environment.

In line with this, Kang et al. present an SSD prototype with durable cache, called DuraSSD, to enhance the write performance in database and NoSQL systems. They observe that frequent cache flushing of SSD which is requested to guarantee the atomicity and durability of transactions, makes a long write latency and imposes a serious performance degradation [?]. To resolve the problem, they maintain the internal-SSD cache durable by using the partially protected DRAM with a small size of tantalum capacitors. DuraSSD maintains a group of user data pages and a page mapping table in DRAM cache; on the power-failure, it flushes all of the user data and dirty mapping entries into the dump area in flash memory, because flushing the entire mapping table requires excessive time and energy. On recovery, DuraSSD re-writes the user data in dump area to their permanent location in NAND flash with mapping table updates, and it merges dirty mapping entries with its permanent copy.

Spartan-SSD shares similarities with DuraSSD in that they both selectively protect the user data, but as shown in the performance evaluation, it has notable differences. When the working-set size goes beyond the protected buffer size by capacitors, DuraSSD incurs a serious performance decrease, while Spartan-SSD invariantly provides excellent performance across the various workloads, even without any additional capacitors.

2.2 Maintaining Capacitor Aging

Another group of studies attack the capacitor aging problem. Alcicek et al. demonstrate the ultracapacitor aging according to the temperature through experimental measurements [?] and Hannonen et al. present a method to detect the capacitor degradation using the variations of the output voltage at the dc-dc converter [?]. Gao et al. detect the current available capacitence and bound the number of dirty pages under the limit so as to prevent a data loss against the capacitor aging [?, ?]. They run a periodic background write-back process to detect the dirty page budget dynamically, and activate the write-back process when the number of dirty pages approaches the budget closely. Spartan-SSD assumes an overly-charged capacitor enough to provide a sufficient capacitence during the SSD lifetime, which is like a commercial SSD.

2.3 Write Buffer in Scalable Storage

Some studies explore ways of using the internal write buffer efficiently in scalable SSDs. Chen et al. project that even the high capacity of SSDs will use the small size of write buffer because the capacitor that protects the buffer does not scale well due to the cost, size, and reliability constraints [?]. Nevertheless, they observe that the small sized write buffer can be effective for reducing write traffic in particular applications that perform journaling heavily. Motivated by this observation, they present the application-SSD co-design to reduce the data writes buffered for heavy logging/journaling applications. They propose to protect write-hot log/journal data with capacitors while the log/journal data being durable. In addition, they propose NVMe interface extension for host to notify SSDs the ranges of write-hot LBAs for more efficient protection by capacitors with reduced complexity of hot/cold separation. It reduced substantial amount of flash memory write traffic with few megabytes of capacitor-powered write buffer, but it is specific to heavy log/journal applications and requires change of application code to benefit from its scheme.

Workload	OPs	Footprint(Data)	Footprint(Index)
sequential	2M	0,000	00,000
random	2M	0,000	00,000
webserver	2M	5.2G	5.8G
fileserver	2M	1.9G	2.2G
linkbench run	2M	5.0G	14.8G
tpcc	2M	2.7G	5.8G

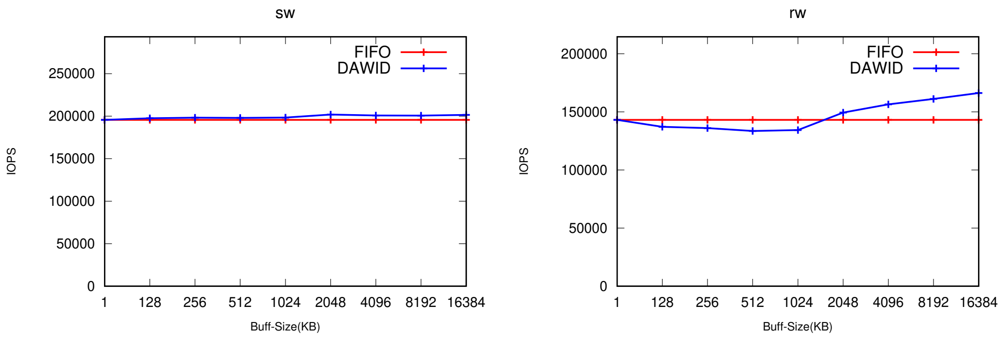
Table 3: **Number of cache references and misses. Ref(R)** is the number of cache references when running RocksDB; **Miss(R)**, the number of cache misses with RocksDB; **Ref(J)**, references with JFDB; and **Miss(J)**, misses with JFDB.

3 Performance Evaluation

3.1 Implementation

3.2 Performance results

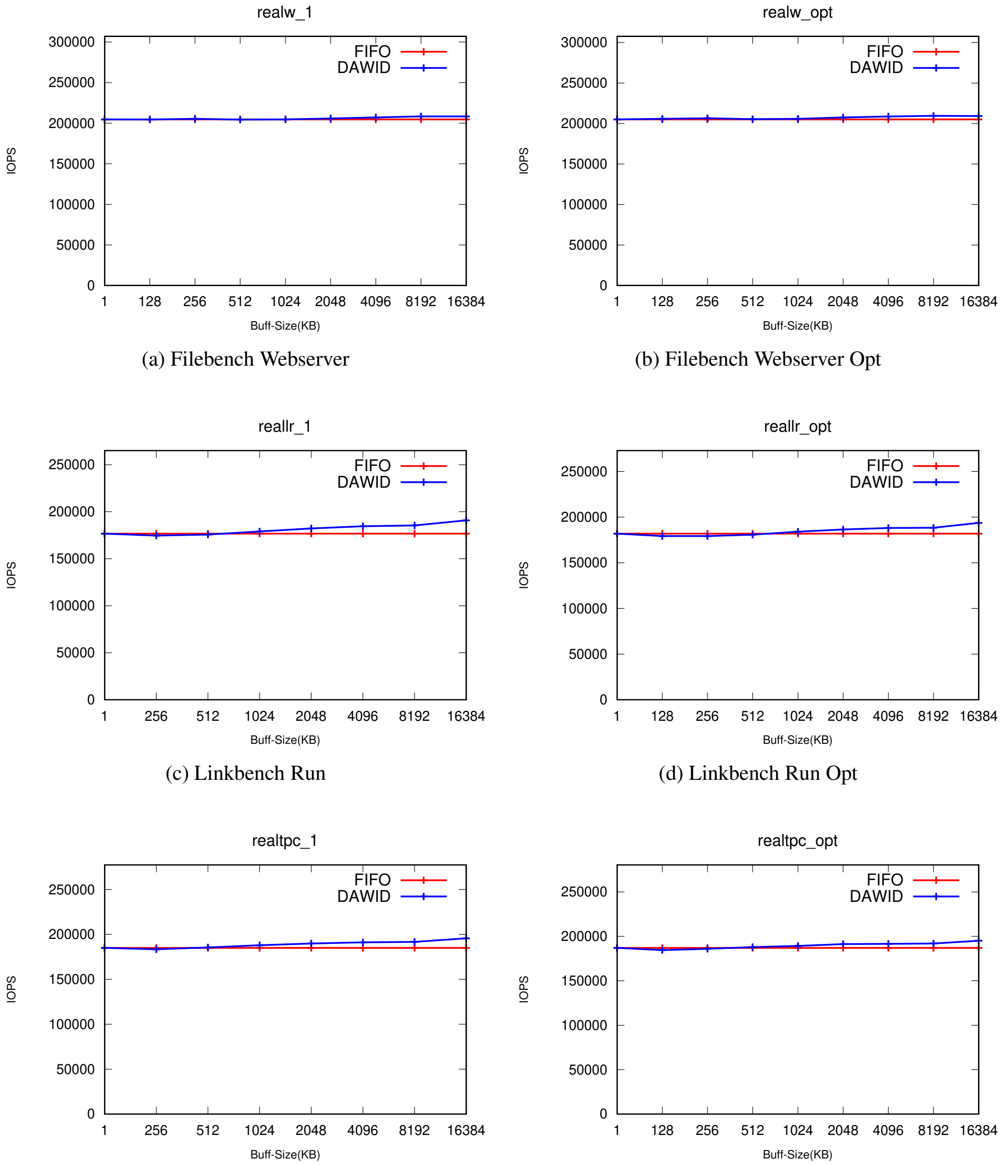
Notes



(a) Sequential write

(b) Random write

Figure 2: **IOPS for micro benchmarks.**



(a) Filebench Webserver

(b) Filebench Webserver Opt

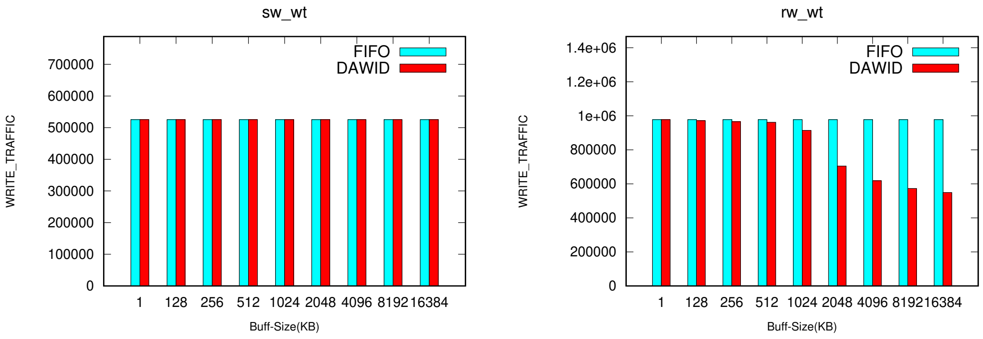
(c) Linkbench Run

(d) Linkbench Run Opt

(e) TPCC

(f) TPCC Opt

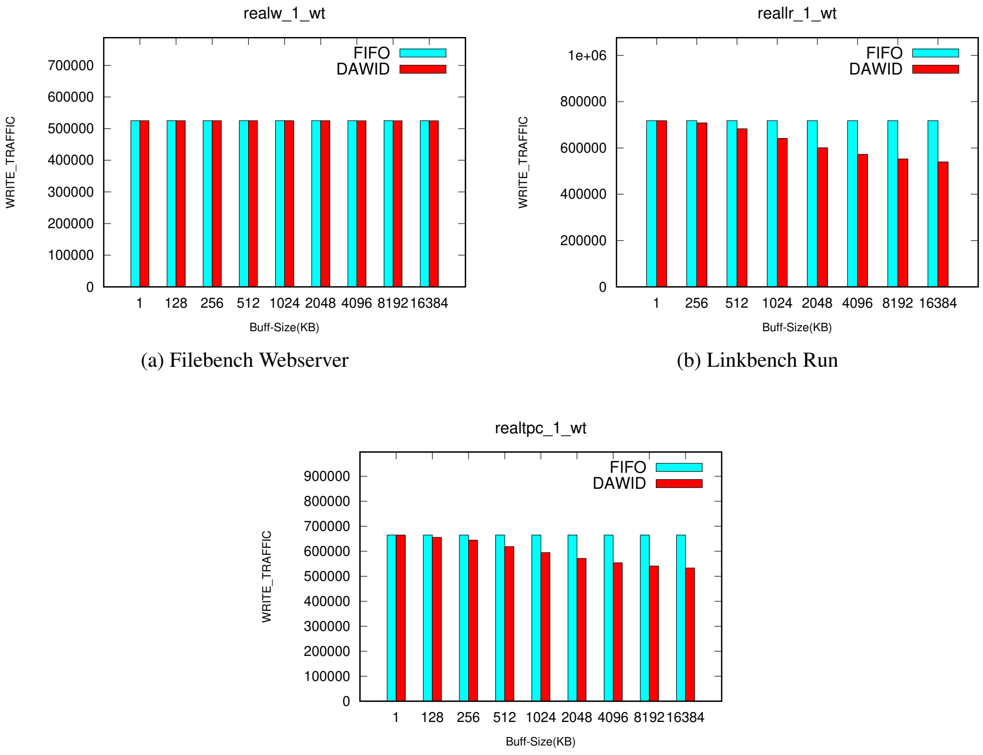
Figure 3: **IOPS for macro benchmarks.**



(a) Sequential write

(b) Random write

Figure 4: **Write traffic for micro benchmarks.**

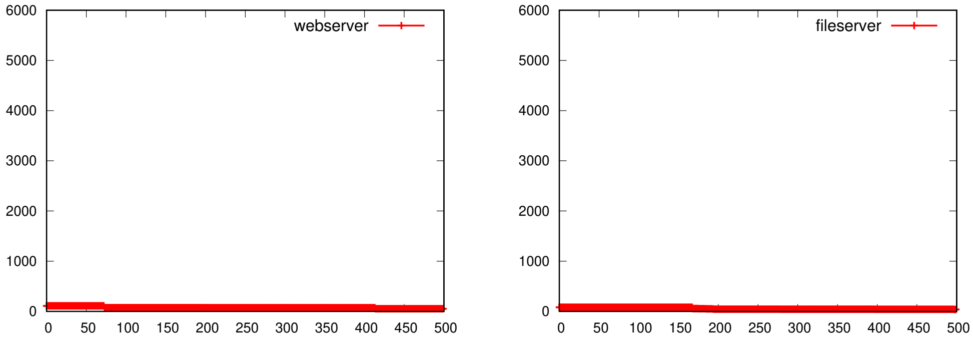


(a) Filebench Webserver

(b) Linkbench Run

(c) TPCC

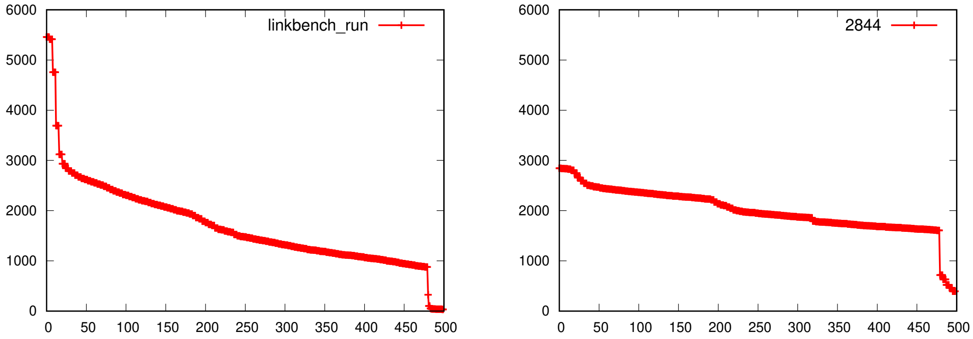
Figure 5: **Write traffic for macro benchmarks.**



(a) Filebench Webserver

(b) Filebench Fileserver

Figure 6: **Workload Characteristic.**



(a) Linkbench Run

(b) TPCC

Figure 7: **Workload Characteristic.**