Durable Write Cache in Flash Memory SSD for Relational and NoSQL Databases

Woon-Hak Kang
Sungkyunkwan University
Suwon, 440-746, Korea
woonagi319@skku.edu

Sang-Won Lee[*]
Sungkyunkwan University
Suwon, 440-746, Korea
swlee@skku.edu

Bongki Moon
Seoul National University
Seoul, 151-744, Korea
bkmoon@snu.ac.kr

Yang-Suk Kee
Samsung Semiconductor Inc.
Milpitas, USA, 95035
yangseok.ki@ssi.samsung.com

Moonwook Oh
Samsung Electronics
Hwasung, 445-701, Korea
mw.oh@samsung.com

ABSTRACT

In order to meet the stringent requirements of low latency as well as high throughput, web service providers with large data centers have been replacing magnetic disk drives with flash memory solid-state drives (SSDs). They commonly use relational and NoSQL database engines to manage OLTP workloads in the warehouse-scale computing environments. These modern database engines rely heavily on redundant writes and frequent cache flushes to guarantee the atomicity and durability of transactional updates. This has become a serious bottleneck of performance in both relational and NoSQL database engines.

This paper presents a new SSD prototype called *DuraSSD* equipped with *tantalum capacitors*. The tantalum capacitors make the device cache inside *DuraSSD* durable, and additional firmware features of *DuraSSD* take advantage of the durable cache to support the atomicity and durability of page writes. It is the first time that a flash memory SSD with durable cache has been used to achieve an order of magnitude improvement in transaction throughput without compromising the atomicity and durability. Considering that the simple capacitors increase the total cost of an SSD no more than one percent, *DuraSSD* clearly provides a cost-effective means for transactional support. *DuraSSD* is also expected to alleviate the problem of high tail latency by minimizing write stalls.

Categories and Subject Descriptors

H.2 [**DATABASE MANAGEMENT**]: Systems

[*]This work was done while the author was visiting Samsung Semiconductor Inc.

General Terms

Design; Reliability; Performance

Keywords

Atomicity; Durability; SSD; Durable Cache

1. INTRODUCTION

In the era of warehouse-scale computing, a large-scale application runs on hundreds or thousands of servers equipped with their own storage and networking subsystems. When an application is made up of many tasks running in parallel, completion of the application is often delayed by a few tasks experiencing a disproportionate amount of latency, thus affecting negatively the overall utilization of computing resources as well as the quality of services. This latency problem will be aggravated further with an increasing number of parallel tasks, because the variance of latencies in parallel tasks is always amplified by the system scale.

This latency concern, known as *high tail latency*, poses serious challenges for online service providers operating warehouse-scale computers and data centers [9]. Studies on the effect of increased server side delays show that users respond sharply to the speed of web services and a slower user experience affects long term behavior. For example, Amazon found every 100ms of latency cost them one percent in sales, and Google found an extra half second in search result generation dropped traffic 20 percent. Shopzilla found a five-second speed-up resulted in a 25 percent increase in page views, a 7 to 12 percent increase in revenue, a 50 percent reduction in hardware [13].

In order to meet the stringent requirements of low latency as well as high throughput, major web service providers have been replacing magnetic disk drives with flash memory SSDs in their data centers. Ebay witnessed 50% reduction in the rack space and 78% drop in power consumption with 100 TB of flash memory drives replacing disk-based systems [12]. Such companies as Amazon, Apple, Dropbox, Facebook and Google are also using solid-state storage in all the servers of their data centers or moving in that direction [20, 21]. This trend toward *all-flash* data centers has already begun and is expected to be accelerated.

Despite all these exciting developments with flash memory SSDs taking place in the warehouse-scale computing arenas, however, flash memory storage is not free of latency variability problem. One of the main causes of latency variability is the need of garbage collection. A flash memory SSD carries out garbage collection when it runs out of clean blocks, and this can increase read latency by a factor 100 with even a modest level of write activity [9]. Similarly, though it may not be as severe, a read request can be blocked by a varying number of write operations when the buffer pool is full of dirty pages. The read request must wait until some of the dirty pages are flushed to storage and a free buffer frame becomes available. This *free buffer wait* can occur more frequently with flash memory SSDs than disk drives due to the read-write speed disparity of flash memory [14]. Since it takes more time to write an evicted dirty page to flash memory than reading one into a free buffer frame, the buffer pool is more likely to be filled up with dirty pages quickly.

Evidently the problem of latency variability is aggravated in flash memory SSDs when the level of write activity is high. The level of write activity is often increased by the need of atomicity and durability support for transactional updates. The atomicity of an update cannot be guaranteed without preventing a partial page write problem, which is typically dealt with by writing pages redundantly to storage by a mechanism inspired by shadow paging (for example, double-buffer writes by the InnoDB storage engine of MySQL). To guarantee the durability and correct ordering of updates, most database servers rely on flushing writes from the device cache to the stable media (most commonly by executing `fsync` calls). Modern flash memory SSDs embed a great deal of DRAM write cache in order to hide relatively long write time. Consequently, flushing dirty pages from a large DRAM cache to flash memory chips frequently not only blocks an application longer but also increases the variability of read latency.

Backing up a DRAM cache with a capacitor is one of the most cost-effective ways of making the device cache durable, which in turn prevents a partial page write, supports an atomic update without redundant writes, and makes an update durable without force-writing it synchronously. The main contributions of this work is summarized as follows.

- It presents the durable cache technology embedded in a flash memory SSD and a prototype called *DuraSSD*, the DRAM cache of which is backed up with *tantalum capacitors* (also known as *tantalum-polymer capacitors*). The architecture and inner workings of *DuraSSD* are also described in detail.

- It demonstrates for the first time that *DuraSSD* is a quick and effective solution to fix the problem of a long write latency and the variability of a read latency, which are often experienced in OLTP and NoSQL applications requiring transactional updates. *DuraSSD* achieves more than an order of improvement in transaction throughput in relational and NoSQL workloads such as LinkBench and YCSB.

- *DuraSSD* provides support for atomicity, durability and write ordering at the device level. This opens up new opportunities for the leaner and more robust design of a database system.

- In addition to improved performance, *DuraSSD* prolongs the lifetime of a flash memory SSD significantly, because the absolute amount of data written to flash memory is reduced more than 50% by avoiding redundant writes and by utilizing a small page size.

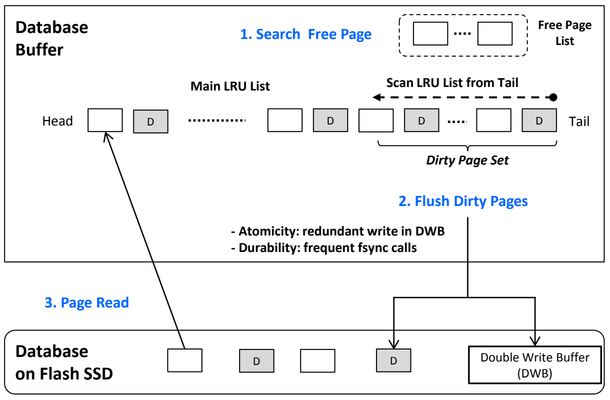


Figure 1: Reads blocked by writes

The rest of the paper is organized as follows. Section 2 motivates the development of *DuraSSD* and describes its benefits. Section 3 presents the overall architecture of *DuraSSD* and its internal firmware features introduced to guarantee the atomicity and durability of page writes. In Section 4, we evaluate the performance impact of *DuraSSD* on MySQL/InnoDB and Couchbase storage engines using two popular NoSQL benchmarks as well as TPC-C workload on a commercial relational database system. Section 5 reviews the existing work on durability by reliable memory, reliability of SSDs and atomic page write. Lastly, Section 6 summarizes the contributions of this work.

2. BENEFITS OF *DuraSSD*

As is discussed briefly in Section 1, the variability of read latency is harmful for reliable performance of OLTP database and NoSQL applications. The severity of the problem becomes worse when flash memory SSDs are used as storage devices.

Consider a database server that uses a shared buffer pool consisting of a number of page frames. The buffer pages are concurrently accessed by many user and system processes and are managed by a buffer replacement algorithm such as an LRU policy. In order for a process to read a page from storage, it has to obtain a free page frame from the buffer pool. If a free page is not available in the buffer pool, the read request will be blocked until a free page is made available for the process. The procedure for *making a free page available* involves non-trivial and potentially costly steps, as illustrated in Figure 1.

If the buffer manager runs out of free pages, it scans the LRU list of buffer frames from its tail to find a victim page. If the victim page is dirty, its content must be written to storage before the page being passed over to the requesting process. This implies that the total elapsed time of a single read operation felt in the process side will be at least the sum of a read latency and a write latency if the read is blocked by a write. Since a page write still takes a few times longer than a page read in most flash memory SSDs, the latency of a read operation measured at the process side will vary widely, sometimes as short as a page read but longer than a page write at some other times. This is the reason why the problem of latency variability will be aggravated with flash memory SSDs. Besides, the atomicity and durability requirements of database applications commonly require an update to be written to storage forcefully and redundantly. This amplifies the latency problem further.

There are not many options for addressing this problem other than avoiding unnecessary write operations and minimizing the average latency of a write operation itself. In the rest of this section, we present how the durable cache of a *DuraSSD* drive can alleviate the problem and summarize the benefits relational and NoSQL databases can take from it.

2.1 No Redundant Writes for Atomicity

A crucial assumption for database consistency and recovery is that individual pages should be written to storage atomically, which is unfortunately not supported by most secondary storage devices including disks and flash memory SSDs. If a system crashes, for example due to a power failure, while a page write is in progress, the page may be left with a mix of old and new data. With such a partially written page (*i.e.*, a shorn write), even the write-ahead logging (WAL) would not be enough to completely restore the consistent state of a database [22]. For this reason, modern database systems have developed several software-based methods to guarantee the atomicity of a page write. Many of the methods are inspired by shadow paging [18].

The InnoDB storage engine of MySQL deals with the partial page write problem by utilizing a redundant page update technique called *double-write* [2]. As is depicted in Figure 1, the InnoDB engine writes pages to a dedicated area in storage. It then re-writes each page to its original location in the database. All the redundant writes are done forcefully (using `fsync` calls if necessary) to ensure the pages are written persistently in the correct order. When the system recovers from a failure, it can always find consistent pages either in the database or in the double write buffer area.

PostgreSQL also takes a similar approach to avoid the partial page write problem. When the *full-page-write* option is on, the PostgreSQL server writes the entire content of a page (*i.e.*, before image) to the WAL log during the first modification of the page after a checkpoint. Storing the full page content guarantees that the page can be correctly restored but at the cost of increasing the amount of data to be written to the log [3]. Mobile database servers such as SQLite [1] and Sybase SQL Anywhere [31] are in the same vein in that they redundantly store before or after image of each data page being updated in a separate journal area.

With a durable internal cache, *DuraSSD* is capable of writing a page atomically. As soon as the content of a page write request is written to the internal cache of *DuraSSD*, atomic write of the page into flash memory is guaranteed from the moment on. In the event of a power failure, the tantalum capacitors of *DuraSSD* ensures that all the pages remaining in the internal write cache are written to dedicated flash memory blocks.

The device-level support of *DuraSSD* for atomic page writes provides a tremendous opportunity for performance improvement by relieving database servers of making redundant writes for atomic page updates. In fact, *DuraSSD* helps simplify the procedures of database updates and recovery, and improve the transaction throughput significantly, as is demonstrated in Section 4. Of course, guaranteeing atomic

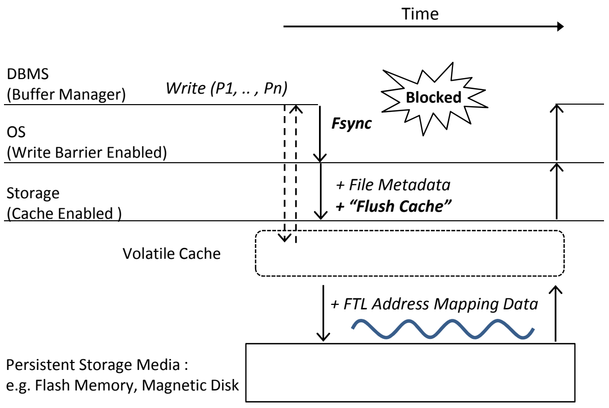


Figure 2: Durability by `fsync` and *flush-cache*

updates requires careful coordination across several software stacks from a database server to a file system to a kernel block layer and so on. For example, a database server and a file system should be configured to have the same page size as the granularity of page mapping in *DuraSSD*.

2.2 Write Barriers without Flushing

The `fsync` call provided by operating systems is a core mechanism both relational and NoSQL database systems use to achieve durability of write and preserve correct order of writes. As is shown in Figure 2, a `fsync` call causes a non-trivial delay in a database system, because it sends a *flush-cache* command to storage and the storage firmware flushes all the cached data to the non-volatile media upon the *flush-cache* command. A database system is usually blocked while a `fsync` call is being processed. This will be a significant overhead for a large relational or NoSQL system that processes a large number of small transactions, as `fsync` calls will be heavily used for the transactions.

With the durable cache of *DuraSSD*, database servers still need to use `fsync` calls to maintain correct order of writes. For example, when a database system needs to keep a strict order between data pages and log pages being written persistently ahead of the data pages, a `fsync` call will be used between the log pages and data pages. However, that does not necessarily mean that the device cache of *DuraSSD* need be flushed to flash memory just because a `fsync` call is invoked. Since it is guaranteed that all page write requests written to the internal cache of *DuraSSD* will be secured, the consistency of a database will never be disturbed even without flushing the device cache explicitly, as long as all the write requests reach *DuraSSD* in the correct order.

When write barriers are turned off in a file system, `fsync` call does not send a *flush-cache* command to storage. All the writes will then be written only to the device cache and be completed much more quickly. Therefore, with write barriers turned off, *DuraSSD* can avoid flushing its cache too often and improve database throughput considerably without sacrificing durability and correct order of writes.

In order to understand the effect of `fsync` frequency on the IO throughput, we measured the I/O throughput of a *DuraSSD* drive as well as two commercial SSDs and a disk drive. The frequency of `fsync` calls was varied from none to once every write to once every 256 writes. Each of the four devices was tested with the storage cache turned on and off. We also included a case where *DuraSSD* was run with write

Random Write IOPS		# of Writes per `Fsync`								
Device (Cache Size)	Storage Cache	1	4	8	16	32	64	128	256	no `fsync`
`HDD`¶ (16MB)	`OFF`	58	111	130	143	151	155	156	157	158
	`ON`	59	135	184	234	251	335	375	381	387
`SSD-A` (512MB)	`OFF`	168	332	397	441	463	479	480	490	494
	`ON`	256	759	1,297	2,219	3,595	5,094	6,794	8,782	11,681
`SSD-B` (128MB)	`OFF`	603	732	889	995	1,042	1,082	1,114	1,124	1,157
	`ON`	655	1,762	2,319	3,152	4,046	5,177	6,318	8,575	8,456
DuraSSD (512MB)	`OFF`	249	330	438	467	482	490	495	497	498
	`ON`	225	836	1,556	2,556	5,020	6,969	10,582	12,647	15,319
	`ON (NoBarrier)`	14,484	14,800	14,813	14,824	14,840	14,863	15,063	15,181	15,458

¶Disk: Seagate Cheetah 15K.6 146.8GB

Table 1: Effect of *fsync* and *flush cache* on 4KB page size random write IOPS

barrier turned off for comparison. The `fio` tool was used for the benchmark. The results summarized in Table 1 clearly show that the I/O throughput of both disk drives and SSDs is seriously affected by the frequency of `fsync` calls. This also demonstrates that if a storage device is not burdened with flushing its cache explicitly, which is feasible with a *DuraSSD* drive, the device will be able to deliver its best possible performance.

2.3 Magnified Write-Back Effect

Modern storage devices from disk drives to flash memory SSDs are equipped with an internal DRAM cache primarily for sequential prefetching and write buffering. Write buffering is particularly effective for random workloads. It has been shown that a write buffer as large as 0.1% of the storage can absorb write bursts and process them without stall for a variety of workloads [15, 29]. However, if the internal DRAM cache is volatile, there is a danger that data written only in the device cache will be lost upon a power failure or crash. So it is not safe to keep dirty pages in the write buffer for a long duration, and this becomes a limiting factor on the effective utilization of write cache. However, if the internal DRAM cache is durable (*i.e.*, non-volatile), as is the case with *DuraSSD*, write buffering can be fully exploited to minimize write stall and maximize the utilization of storage substrate.

The effect of write buffering is more significant with flash memory SSDs than disk drives.[1] Table 1 compares the random write throughput of *DuraSSD* with that of a magnetic disk drive and two commercial flash memory SSDs (denoted as `SSD-A` and `SSD-B`) with different amounts of internal cache. In the case of the three flash memory SSDs, the write throughput with no `fsync` was about approximately $13 \sim 68$ times higher than that with `fsync` for every write. On the other hand, the improvement ratio was no more than seven times with the disk drive. This is mainly because there is an ample opportunity of parallelism in modern flash memory SSDs [4, 16], while there is practically none in disk drives. Most flash memory SSDs have a multi-channel architecture where each channel connects two or more flash memory packages. A flash memory package contains typically two or four flash memory chips that can operate in parallel, each of which also supports parallel operations with two or more planes. For example, the theoretic upper bound on the degree of parallelism is 256 for an SSD with 8 channels, 4 packages per channel, 4 chips per package, and 2 planes per chip.

[1]Similar observations have been reported in the industry [10, 32].

A large durable cache of *DuraSSD* enables it to accommodate many concurrent writes without being limited by data protection need. Combined with the pure electronic performance of flash memory chips and the multi-channel architecture, *DuraSSD* can sustain high write throughput with no or less severe stall even in the presence of write bursts. It should also be noted that the burden of maintaining the mapping information persistently will be ameliorated with a durable cache in *DuraSSD*, because it does not have to flush (part of) the mapping table to flash memory for every single page write operation.

2.4 Effectiveness of a Small Page

The page size, as a unit of I/O operations, has steadily increased in the OLTP database engines for the past years. Many commercial database servers choose a page of 8KB as the default size. The most recent releases (versions 5.5 and 5.7) of an open source MySQL have increased the default page size to 16KB. This trend of using a large page size has been fueled and justified by the need to hide a long latency of mechanical disk drives.

As flash memory SSDs become more and more common in warehouse-scale computer systems as well as large OLTP systems in recent years, the issue of finding an optimal page size has been revisited. A few recent studies have found that a small page size can improve the performance of a flash memory based database engine for OLTP workloads where small random accesses are dominant [16, 26]. This is because a flash memory SSD, as a pure electronic device, exhibits extremely low latency and high I/O throughput and using a small page can minimize the amount of redundant data being read from and written to storage. The latter is even more relevant to improving I/O performance for OLTP applications, which usually update only a small fraction of a data page. Other notable positive effects of using a smaller page is that the page fault rate can be lower due to the buffer pool being polluted less by irrelevant data and the level of concurrency can be higher due to the smaller granularity of locking [6].

The size of a page, of course, cannot be smaller than the smallest unit of I/O supported by a storage device. The positive aspect of that is there is no more juggling for selecting a page size and one can simply pick the smallest unit of I/O. Since a page is the smallest unit of addressing in flash memory chips, an obvious choice is to set the size of a database page equal to the size of a flash memory page. The most common size of a flash memory page is 4KB for modern enterprise-class flash memory SSDs. However, there is a serious caveat with using a small page size when the cache in the storage device is flushed often by `fsync` calls.

Random IOPS	Page Size		
	16KB	8KB	4KB
Read-only (128 threads)	29,870	57,847	89,083
Write-only (1-fsync)	196	206	225
Write-only (256-fsync)	4,563	7,978	12,647
Write-only (128 no-barrier)	13,446	25,546	49,009

(a) *DuraSSD*

Random IOPS	Page Size		
	16KB	8KB	4KB
Read-only (128 threads)	516	528	538
Write-only (128 threads)	428	439	444

(b) Harddisk (Seagate Cheetah 15K.6 146.8GB)

Table 2: Effect of page size on IOPS

Table 2 shows the I/O throughput of an SSD and a disk drive measured by running the `fio` tool with different frequencies of `fsync` calls. When `fsync` calls were used infrequently (once in 256 writes), the I/O throughput increased about 170 percent by reducing the page size from 16KB to 4KB. A similar improvement (about 190 percent) was observed for read only operations, which did not require any `fsync` call. On the other hand, when `fsync` calls were used frequently (once every write), the increase in I/O throughput obtained by reducing the page size from 16KB to 4KB was only about 15 percent. This is an evidence that the overhead of flushing a device cache is a more dominant factor than the amount of data transfer in determining the I/O throughput.

This also implies that the throughput of a conventional flash memory SSD may be compromised by frequent `fsync` operations, when a smaller page is chosen, because flushing the cache cannot be postponed indefinitely. In this regard, *DuraSSD*, which does not require explicit cache flushing, can make the best use of a small page size without the danger of losing data. As is shown in Table 2, *DuraSSD* can achieve about three times higher I/O throughput by reducing the page size from 16KB to 4KB. On the other hand, in the case of disk drive, the improvement ratio is only about four percent.

3. ARCHITECTURE AND IMPLEMENTATIONS

We have developed a new type of flash memory SSD called *DuraSSD*. *DuraSSD* comes with a durable internal cache and helps achieve atomicity, durability and proper ordering of writes at the minimal cost. Figure 3 illustrates the internal architecture of *DuraSSD*. There are four major components in the *DuraSSD*: (1) durable cache, (2) atomic writer,

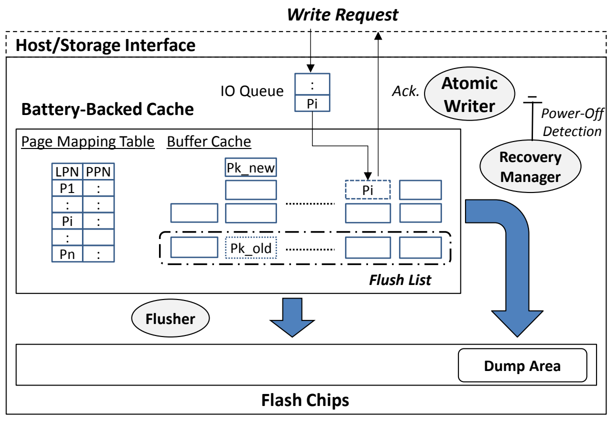


Figure 3: *DuraSSD* architecture

(3) cache manager (or flusher), and (4) recovery manager. This section details each of the components, and describes how *DuraSSD* can guarantee the key storage properties that database requires.

3.1 Durable Cache

The durable cache in *DuraSSD* refers to a DRAM-based buffer cache backed by small capacitors. The durable cache is composed of a set of DRAM chips of 512MB in total and fifteen tantalum capacitors. Figure 4 shows the front and back planes of a *DuraSSD* prototype. The tantalum capacitors are attached to its back plane. The retail price of fifteen tantalum capacitors is around five USD in total, which accounts for about one percent of the total *DuraSSD* price. We understand that new non-volatile memory technologies such as PCM, RRAM, STT-MRAM will be a more solid and versatile solution than a battery-backed approach, since they can provide more stable performance with a fewer thermal issues. However, we expect that such new memory technologies will need more time to mature and become economical.

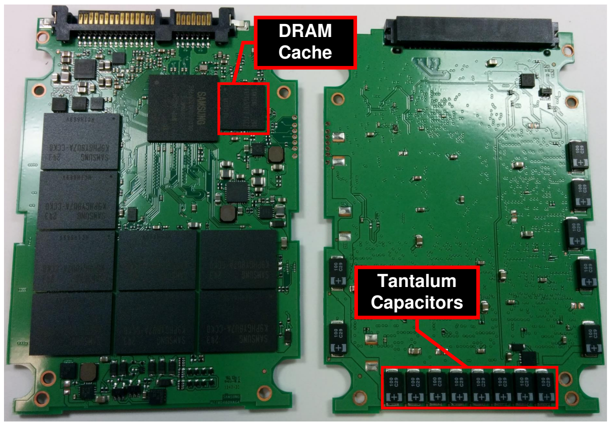


Figure 4: *DuraSSD* with cache and capacitors

DuraSSD maintains a pool of buffered pages and a page mapping table in the durable cache. These two data structures have to be kept consistent and secured persistently in the storage media. Thus the total capacity of tantalum capacitors must be large enough to flush all the data from the data structures to the storage media upon a power failure.

This will help preserve the device state consistent and implement idempotent recovery from a failure. The tantalum capacitor has a smaller form factor and lower cost overhead than the super capacitor. The time duration its back-up power can sustain is normally several milliseconds, so the fifteen tantalum capacitors of *DuraSSD* only allow flushing cached data of dozens of mega bytes.

3.1.1 Buffer Pool

The buffer pool is mainly used as write cache [29] to improve random write performance. The buffer pool should be able to handle the batch that the I/O command queue allows. The SATA NCQ (Native Command Queuing) can queue up to 32 commands at the same time. For fair resource sharing, each command may have a quota of the buffer pool. A typical enterprise class SSD has a high degree of internal parallelism. (See Section 2.3.) To maximize the throughput, all channels and data pipelines over parallel planes need to be filled as much as possible. For this purpose, the buffer pool is organized as a set of queues for buffered writes and schedules the write-backs considering the parallelism and wear-leveling.

NCQ allows multiple commands to be drained to leverage the internal parallelism. Therefore, the buffer pool needs a flow-control mechanism between the command queue and the buffer queues. The flusher of *DuraSSD* continuously pulls the write-backs from the buffer queues and writes them to flash memory in the FIFO manner, anticipating the buffer queues will be filled in the meantime. If a page is updated frequently, there can be multiple copies of the page in the buffer queue. When that happens, old copies except the latest one are discarded to improve the device endurance.

Then, how large should the buffer pool be to streamline data between the host and the device? Let us use the example presented in Section 2.3. The page size is 4KB and the maximum degree of parallelism is 256. The buffer pool thus needs 256× 4KB pages (or one megabyte) to fill all the pipelines up. Moreover, it is desired to support a *double buffering* scheme so that writing cached data to storage media and buffering data from the host in the buffer pool can be overlapped. In summary, even after adding some extra space to absorb bursty writes, a buffer pool of several megabytes would be good enough to achieve the maximum bandwidth of write traffics.

3.1.2 Mapping table

When a page is successfully written to the storage media, the *flusher* releases the buffer frame for recycling and updates the page mapping information with a new physical page address. As discussed in Section 2.4, a small database page can deliver better performance, and its proper alignment with a NAND page will be a clever way to optimizing the performance.

To leverage the benefits of a small page, *DuraSSD* emulates 4KB pages over 8KB NAND pages via a 4KB page mapping scheme. The disparity between physical page size (*i.e.*, 8KB) and mapping granularity (*i.e.*, 4KB) will introduce overhead to the device. For instance, a read or write of 4KB page from or to a NAND page still requires a 8KB page IO, and it needs to filter out unintended 4KB data. Under a heavy random write workload, however, the buffer pool should be able to find a pair of pages that can be combined and written together into a NAND page even under the 4KB mapping scheme.

The SSD capacity used in this paper was 480GB. With a 4KB page as a mapping unit, the page mapping table requires 120 million entries, each of which will take four bytes to represent a physical page in the address space of 480GB. This amounts to 480MB. As a rule of thumb, the moderate cache size of an SSD would be just about a megabyte per GB, that is, 0.1% of an SSD capacity. The reduction of a mapping unit from 8KB to 4KB doubles the DRAM requirement, which increases the total cost of a *DuraSSD* about one percent. In summary, the 4KB page mapping improves the raw I/O performance and the hit ratio of the buffer pool at the expense of one percent cost overhead.

3.2 Atomic Writer

In general, an I/O operation is considered complete, once a storage device sends an *ack* for the operation back regardless of it being delivered to the host or not. When a write command is drained from the I/O command queue, its data are streamed from the host to the durable cache of *DuraSSD*. The write command is then marked as complete when the data transfer is finished. With the durable cache, the atomicity of the write command is guaranteed from the moment when the command is marked as complete.

Consider a situation where a power failure happens after or while a write command is processed. On a power failure, all the write-backs are flushed from the durable cache to a persistent segment in the storage media called a *dump area*. When they are written to the dump area, their page mapping entries are not updated for fast flushing. At the restart time, for all the complete write commands, the recovery manager replays the write-backs stored in the dump area and reflects the changes to the page mapping table. While replaying write-backs will delay the system restart, it guarantees the atomicity and durability of complete write commands. For incomplete write commands, its write-backs will be simply discarded from the dump area. This guarantees the atomicity (or rollback) of incomplete write commands.

3.3 Flush-Cache Command in Durable Cache

The `T13` standard of the ATA storage interface supports a *flush-cache* command to flush the write cache to persistent storage media [24]. This command is generated from a `fsync` system call and is typically used to guarantee the durability of data and the ordering of writes. Especially, for an out-of-order I/O command queue like SATA NCQ, the *flush-cache* command can help enforce an ordering.

Since *DuraSSD* does not rely on write-barrier, there exists a risk of command reordering, which can violate the ordering semantics of database. In order to avoid the trouble, we have implemented an *ordered* version of NCQ so that the order of commands can be preserved as well as the persistence of them. Alternatively, we can enable write-barrier but implement the semantics of flush-cache in a different way to ensure all the commands that have arrived before the flush-cache command will be processed before it. We will explore this alternative for further optimization. Note that the order of flushing the write-backs from the buffer pool to the storage media does not cause a database inconsistency problem, because only the latest version of a page is written to the storage media.

3.4 Recovery Manager

3.4.1 On power failure

DuraSSD is equipped with a dedicated circuit logic to detect a power failure, upon which the recovery manager is invoked. Since the page mapping table accounts for most of the DRAM space, it will be both time and power consuming to flush the entire mapping table to the dump area. To reduce the amount of data to flush, an incremental backup technique can be used to track modified entries and flush only those entries to the dump area upon a power failure. On the other hand, the buffer pool and its in-memory data structures are relatively small and can be flushed entirely to the dump area.

Obviously, it is important to minimize the time taken to flush modified mapping entries and the buffer pool to the dump area upon a power failure. *DuraSSD* ensures that a group of clean flash memory blocks are always available for the dump area during normal processing time, so that the key data structures can be flushed as fast as possible without encountering a garbage collection. Finally, for recovery at the next boot-up, it is necessary to mark that the storage device was shutdown abnormally due to a power failure.

3.4.2 On reboot

At the reboot time, the first step taken by *DuraSSD* is to recharge the capacitors so that data can be protected from another power failure during the recovery process. If the flag for an emergent shutdown is set, the recovery manager is invoked. The recovery manager rebuilds the page mapping table by merging the most recent mapping table stored persistently in flash memory with the modified mapping entries dumped at a power failure. Then, it restores the buffer pool and its metadata from the dump area. Once the buffer pool is restored completely, the recovery manager clears the dump area and resets the emergent shutdown flag.

Note that the buffer pool stored in the dump area must be self-contained for the sake of recovery. The buffer pool should maintain the information about the logical page number for every valid page frame in the pool, which can be embedded or stored in a separate data structure. The information on the logical page numbers is used to update the page mapping table after the pages are propagated from the dump area to the storage media.

4. PERFORMANCE EVALUATION

This section presents the experimental analysis carried out to understand the impact of a durable cache on the performance of relational and NoSQL database systems. The experiments were done with transactional NoSQL workloads such as LinkBench and YCSB as well as the TPC-C benchmark [5, 7, 17]. We ran the database servers under different configurations so that the effect of a durable cache can be identified separately by redundant writes, `fsync` calls and page sizes.

4.1 Workloads

We ran the LinkBench and YCSB workloads on the open source systems MySQL and Couchbase, respectively. Couchbase is one of the most popular document-oriented key-value stores. The TPC-C workload was run on a commercial relational database system. As NoSQL workloads, transactions of LinkBench and YCSB are much smaller than those of a traditional TPC-C workload. At the same time, they are quite similar to a TPC-C workload in that both produces a large number of small random reads and writes. Below are described the characteristics of the two NoSQL workloads and the TPC-C benchmark program.

LinkBench LinkBench is a configurable open-source database benchmark for a large-scale social graph [5]. This benchmark reflects the characteristics of social graph data at Facebook where the majority of read requests are served by a caching layer. This reduces the temporal and spatial locality of read requests reaching the underlying database servers [19]. However, the overall workload reaching the database servers is still read intensive with just about 30% writes. The *force index* option was turned on in order to avoid I/O operations required additionally for query optimization. Also, the *buffer_flush_neighbors* option, which is to flush any neighbor pages together for a dirty victim page, was turned off in order to reduce unnecessary write overhead.

YCSB Yahoo! Cloud Serving Benchmark (YCSB) is a benchmark framework created for evaluating cloud systems [7]. YCSB consists of five workload types. Since all the other workloads except `Workload-A` are read-only, Workload-A was used to evaluate the durable write caching effect of *DuraSSD*. In comparison with TPC-C, YCSB mimics web applications running a huge number of simple queries, each of which touches a single record [34].

TPC-C We used the Benchmark Factory to evaluate the impact on the database throughput by *DuraSSD* and other storage devices. The Benchmark Factory is a database performance testing tool that allows you to conduct database workload replay and industry-standard benchmark testing [28]. The TPC-C benchmark was done with one of the most popular commercial database management system.

4.2 Experimental Setup

The experiments were done on a Linux platform with 3.5.0 kernel. It was equipped with four Intel Xeon(R) E5-4620 CPU sockets, eight cores per socket and 96GB DRAM per socket (384GB in total). The host machine had two *DuraSSD* drives of 480GB each, one of which was used as the main storage and the other as a database log device. Both *DuraSSD* drives were connected to the host via an SATA 3.0 interface that supports 6 Gbps.

The *XFS* file system was used for MySQL and Couchbase servers, and the *ext4* file system was used for the commercial database server. In both cases, the direct I/O option (`O_DIRECT`) was set on for all the storage devices to reduce the effect of file system buffer caching. The database log tail was set to flush by each committing transaction, but three log files of 4GB each were used to minimize the interference from logging. The benchmarking clients of LinkBench and YCSB were run with the database servers on the same hardware to avoid networking delay. The benchmarking clients of Benchmark Factory was run on a separate hardware connected to the servers via Gigabit Ethernet, because Benchmark Factory was not available on the Linux platforms.

4.3 Run-Time Performance

This section first demonstrates the effectiveness of durable cache in SSD by evaluating the LinkBench performance of MySQL InnoDB engine with and without a durable cache. Similarly, for the YCSB and TPC-C benchmark, the performance of Couchbase and the commercial relational database server were evaluated with and without a durable cache. Each performance measurement presented in this section was an average of three runs or more.

4.3.1 MySQL for LinkBench

The version of MySQL used in the experiments was the most recent 5.7.2-m12 development release. We created three LinkBench databases of 100GB with different page sizes of 4KB, 8KB, and 16KB to compare the tuning effect of page size. When the database was created, all tables were partitioned 32 ways to reduce the mutex contention, thus minimizing the system degradation from concurrency. Throughout all the LinkBench experiments, 128 client threads were concurrently run. For steady performance measurement, the LinkBench was pre-run for 600 seconds warm-up time to fill up the InnoDB buffer cache. A total of 6.4 million transactions (50,000 per each client) were run in each experiment.

Write barriers and Double write buffer

The first set of experiments was carried out to evaluate the impact of write barrier and double write buffer on database throughput. We ran the MySQL server with the LinkBench workload under four different configurations created by turning each of the two options on and off. In Figure 5 and the rest of the section, we will use the following notation to specify different configurations.

```
write-barrier / double-write-buffer
```

Figure 5 shows transaction throughput measured under the four configurations with three different page sizes, when the database buffer size was fixed to 10GB. Recall that under the ON/ON configuration, which is a default setting of MySQL/InnoDB, every `fsync` call sends a flush cache command to the storage device and every logical page write causes two physical pages writes for an atomic and durable update.

The most important observation made in the figure is that the largest gain in throughput was obtained when the write-barrier option was turned off. Transaction throughput increased about six times just by turning it off in the case of 4KB page, regardless of the double-write-buffer option.

On the other hand, when the double-write-buffer option was turned off, the gain in transaction throughput was about a factor of two with the write-barrier option on and about 25% with the write-barrier option off in the case of 4KB page. The latter was lower than but not too much off from the 40% improvement reported in the previous work carried out with FusionIO Atomic Write Extension [25]. The gap in throughput was wider when the write-barrier was on, because frequent flush-cache operations made it difficult to hide the latency of twice increased write operations by the double–write–buffer option.

Overall, the best throughput was obtained when the smallest page was used under the OFF/OFF configuration, while the worst was obtained when the largest page was used under the ON/ON configuration. Between these two cases, the gain in transaction throughput was more than 20 times.

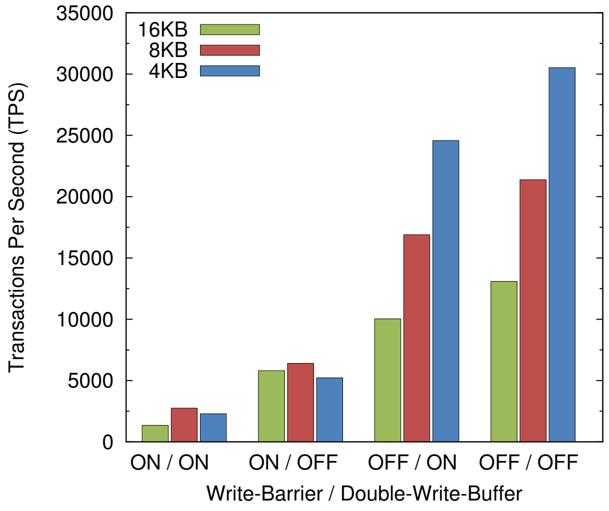


Figure 5: LinkBench transaction throughput

This indicates that *DuraSSD*, which can be used under the best OFF/OFF configuration with the smallest page size, is capable of yielding more than an order of magnitude improvement in transaction throughput without compromising the consistency of database.

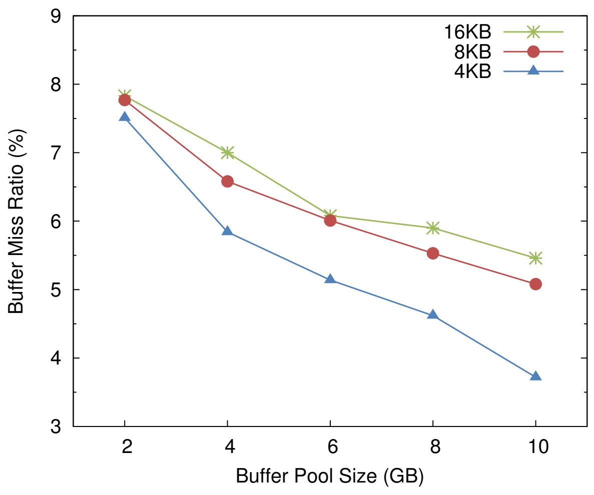
There is a little anomaly in the results shown in Figure 5. When the write-barrier option is on, the throughput of 4KB was a little lower than that of 8KB. This was because the depth of B$^+$-tree indexes increased when the page size was reduced from 8KB to 4KB, and the write IOPS was not affected much by different page sizes when `fsync` call was used frequently. The latter is shown in Table 2.

Effects of Page Size on **DuraSSD**

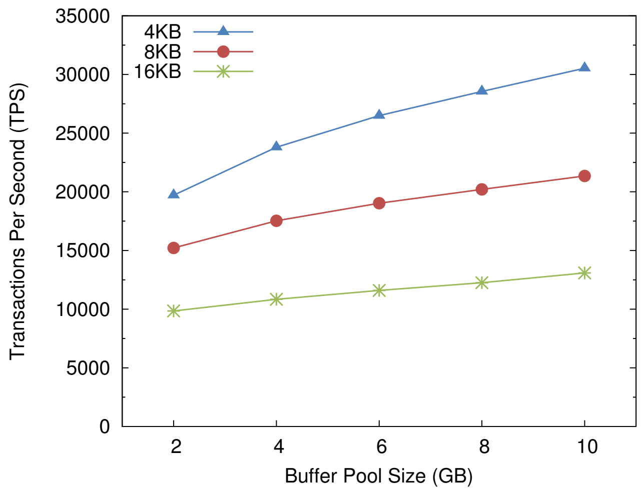
Except for the anomaly described above, a smaller page helped achieve higher transaction throughput than a large page. More importantly, when the write-barrier option was off, the performance gain obtained by using a small page was significant. More than two-fold increase in transaction throughput by reducing the page size from 16KB to 4KB was much higher than 30 to 50% increase reported previously in the literature [16, 26]. This is because a flash memory SSD can exploit the internal parallelism maximally when the write barrier is off.

Another benefit from using a small page is an improved cache effect. Figure 6(a) shows the cache misses observed from running the LinkBench workload with different page sizes. These measurements were obtained from MySQL runtime statistics reports. Though the miss ratios were measured under the OFF/OFF configuration, we believe they would represent all the other configurations as well, because cache misses happen in the buffer pool of a host system.

One thing that can be noticed in Figure 6(a) is that the miss ratio decreased more quickly with 4KB page size. This lower miss ratio, in combination with the higher IOPS with a small page shown in Table 2, contributes to the widening throughput gap among the three page sizes as the size of buffer pool increases, as shown in Figure 6(b). No saturation of throughput increase was observed in Figure 6(b) due to the lack of locality of accesses in the LinkBench workload.



(a) Buffer miss ratio (OFF/OFF)



(b) Varying buffer size (OFF/OFF)

Figure 6: LinkBench buffer miss ratio and TPS (100GB DB, OFF/OFF)

Transactions		ON/ON with 16KB Pages						OFF/OFF with 4KB Pages					
I/O Type	Name	Mean	P25	P50	P75	P99	Max	Mean	P25	P50	P75	P99	Max
Read	Get_Node	67.0	2	4	8	900	4238.1	1.5	0.9	2	2	7	533.1
	Count_Link	45.5	0.6	3	4	800	3274.6	1.2	0.6	0.9	2	5	42.8
	Get_Link_List	65.3	0.6	1	5	1000	14892.6	1.4	0.6	2	2	7	515.4
	Multiget_Link	67.6	0.6	2	5	1000	3047.2	1.3	0.6	0.9	2	7	44.1
Write	ADD_Node	51.6	6	8	9	1000	4891.1	8.9	8	9	10	16	523.0
	Delete_Node	82.2	8	10	16	1000	4267.4	9.6	9	10	11	17	539.1
	Update_Node	86.8	8	10	17	2000	5779.1	9.8	9	10	11	18	540.2
	Add_Link	214.9	11	22	300	2000	7051.0	11.2	9	11	13	23	540.3
	Delete_Link	115.4	3	7	93	2000	4841.2	5.4	0.9	3	10	20	522.1
	Update_Link	217.6	9	22	300	2000	9026.1	11.1	10	11	11	23	543.1

Table 3: Distribution of LinkBench transaction latency (in millisec)

Tail Tolerance of **DuraSSD**

As is mentioned in Section 1, the high tail latency poses serious challenges for online service providers, as even a small increase in latency may result in reduced traffic and revenue. Another set of experiments was carried out to evaluate the impact of *DuraSSD* on the latencies of read and write transactions. The latencies were measured when the buffer pool size was fixed to 10GB, and are summarized in Table 3.

This table compares the default configuration of MySQL (with 16KB pages under the ON/ON configuration) and the best configuration for *DuraSSD* (with 4KB pages under the OFF/OFF configuration) in terms of latencies at 25, 50, 75 and 99 percentiles as well as the minimum and maximum latencies for ten different types of transactions. The latency statistics were reported by the LinkBench script at the end of each benchmark run.

Table 3 clearly shows that the best configuration reduced the latency significantly in all measurements. Specifically, the mean latency was reduced by a factor of 5 to 45, and the maximum latency was reduced by a factor of 8 to 60. More importantly, the latency at 99 percentile was reduced roughly by two orders of magnitude across the board. Once again, this is clear evidence that *DuraSSD* can contribute greatly to the tail tolerance by lowering latencies at the tail of distribution.

We have observed that in LinkBench workload every other read request was blocked by writes on a page fault on average. This was the main reason *DuraSSD* could lower latencies considerably for both read and write requests by avoiding redundant writes and by flushing the device cache as infrequently as possible.

4.3.2 Commercial DBMS for TPC-C

We used the Benchmark Factory [28] to evaluate impact of *DuraSSD* on the TPC-C performance. The number of warehouses was set to 1,000 to create a TPC-C database in the same scale as the one used in the LinkBench experiment. The size of database was approximately 100GB and the size of the database buffer was set to 2GB for a commercial relational database server tested in the TPC-C experiment.

In this experiment, we used the *ext4* file system instead of XFS, because the commercial database server opened files with the `O_DSYNC` option expecting a write barrier to be requested for every page it wrote. This standard `O_DSYNC` semantics was supported by ext4 correctly but not by XFS.

With XFS not requesting a write barrier, the database consistency could be compromised.

TpmC	Page Size		
	16KB	8KB	4KB
Barrier On	4,291	4,845	7,729
Barrier Off	65,809	110,400	150,815

Table 4: TPC-C throughput measured in tpmC

The transaction throughput yielded by the commercial database server changed significantly by turning write barrier off. As is shown in Table 4, the throughput increased by a factor of 15.3 to 22.8 for different page sizes. This throughput increase was approximately three times higher than was observed in the LinkBench experiment, which was a factor of six in Figure 5. This is mainly because the commercial database server requested a write barrier for each page write request and the database buffer size was set to 2GB (five times smaller than MySQL). This made the caching effect of the durable cache in *DuraSSD* more pronounced. The trend in throughput increase across different page sizes was almost identical to the LinkBench results. The throughput increased by a factor of 1.8 to 2.3 by changing the page size from 16KB to 4KB regardless of the write barrier flag. This is consistent with the page size tuning effect observed in the MySQL LinkBench experiment.

4.3.3 Couchbase NoSQL

To analyze the effect of *DuraSSD* on NoSQL workloads, we ran a document-oriented NoSQL database, Couchbase, against the YCSB benchmark. Couchbase stores a JSON document in the value part of a key-value pair. Keys are maintained in a B^+-tree for efficient lookups. Updating a document changes all tree nodes on a path from the root to a leaf pointing to the document. Couchbase writes all the tree nodes to be updated as well as the new document to storage as a single unit. To support the atomicity and durability, the update is appended to storage without modifying existing data and flushed synchronously to storage by a `fsync` call.

The average size of key-value records in YCSB was 1KB. The depth of a B^+-tree was four, and the size of a tree node was 4KB by default. Thus, the size of each update was about 20KB including the tree nodes and a document. Couchbase can adjust the `fsync` frequency in order to trade durability for performance by changing the value of a `batch-size` parameter. If the `batch-size` is set to a positive integer, say k, a `fsync` call is executed every k updates.

The throughput of Couchbase was measured in operations per second (OPS), where an operation is much like a transaction in an OLTP system. The experiment was done with `workload-A` of YCSB benchmark [7] against a 100GB database. The workload-A was chosen, because it was the only one including write operation among all five workloads of YCSB benchmark. The workload-A consists of 50% read and 50% update operations by default. We also measured the throughput when the workload was 100% updates. The replication option of Couchbase was not used, because we were interested in evaluating the impact of *DuraSSD* on performance of an individual node.

Table 5 summarizes the throughput of Couchbase measured under different configurations created by turning the write barrier on or off and changing the batch size from one to 100. The throughput was measured by running 200,000 operations in a single thread.

`batch-size`	1	2	5	10	100
Update 100%	206	398	988	1,954	4,692
Update 50%	195	390	1400	2,041	4,921

(a) With write barriers on

`batch-size`	1	2	5	10	100
Update 100%	2,404	3,464	3,826	4,959	5,101
Update 50%	2,406	3,464	4,209	5,461	6,208

(b) With write barriers off

Table 5: Throughput of Couchbase for YCSB

When write barrier was on, the effect of `fsync` frequency was shown clearly in the throughput measurements. The throughput of batch-size one was more than 20 times lower than that of batch-size 100 in both cases of 100% updates and 50% updates. On the other hand, when write barrier was off, the gap in throughput between batch-size one and batch-size 100 became much narrower to a factor of 2.12 to 2.58. Evidently Couchbase can achieve an order of magnitude improvement in throughput by force-writing updates less frequently. This also provides a clear evidence that *DuraSSD* can help Couchbase improve its throughput significantly while it guarantees the consistency of asynchronous writes as if they were done synchronously.

5. RELATED WORK

By making the DRAM write cache durable with the help of tantalum capacitors and embodying several firmware algorithms inside it, *DuraSSD* can reliably support the atomicity of writes at the storage level against unexpected power failures. By running it in write-back mode, OLTP applications can take advantage of its full potentials without the danger of losing data. This section briefly reviews and compares with *DuraSSD* existing work in durable memory, reliability of SSDs, device level support for atomic write, and page size tuning.

5.1 Durability by Reliable Memory

The idea of making a cache embedded in a storage device durable is not new. When it was proposed the track buffer of a disk drive could be used as a write cache, the track buffer was assumed to be backed up by battery [29]. The durable write cache in a disk drive was not successful, because the size of a track buffer was relatively small and a large expensive capacitor was needed to flush the write cache due to the mechanical characteristics of a disk drive. Moreover, as is shown in Table 1, the performance gain by the durable write cache in a disk drive is limited.

There have been several attempts to make a database system deliver high throughput and low-latency by making the main memory of a host system durable entirely or partially. For example, it has been shown that high transaction throughput can be achieved by storing write-ahead log data in a small amount of host memory backed up by battery and thus by avoiding the high latency of log writes on a slow disk drive upon transaction commits [8, 30]. The use of reliable memory was also suggested for simplifying the design of recovery logics in main memory database systems as well as removing the overhead of check-pointing and write-ahead log writes to a disk drive [11]. Similarly, it has been shown that reliable memory can be used as a replacement of volatile buffer cache in a disk-based database system [23].

Those earlier attempts assumed that DRAM could be made durable either by a battery-backed memory board or uninterruptable power suppliers. Due to the high cost, however, neither of these methods has been adopted widely except for expensive high-end storage appliances. On the other hand, the write cache in a flash memory SSD can be backed up by an inexpensive tantalum capacitor, because much smaller power is required to flush the cached data to flash memory chips than disk drive tracks. Moreover, the benefit of a durable write cache in an SSD is, as is shown in Table 1, significant enough to justify the marginal cost for battery backup.

5.2 Reliability of SSDs

Even well-prepared and experienced data center operators such as Amazon are still suffering from frequent power loss [27, 33]. Recently, an interesting experiment was carried out to understand how flash memory SSDs behave against power failure [33]. In the experiment, various power faults were injected directly to fifteen commercial SSDs and two disk drives using a hardware specially designed for the purpose. The reported results show that thirteen out of the fifteen SSDs tested exhibit various anomalies under power faults such as shorn writes, metadata corruption, and unserializable writes.

Although it is not known how exactly these anomalies would influence a database system with respect to maintaining the consistency, it is not difficult to see they will make the design of a database system more sophisticated so that it can be prepared for all those anomalies. Considering that data centers are steadily moving towards all-flash, it will be extremely beneficial to have flash memory SSDs prepared properly against power failure. The *DuraSSD* presented in this paper guarantees the atomicity, durability and correct write ordering of data pages even in the event of sudden power failure. We believe that this will be an essential and critical feature required to support database consistency and recovery in the warehouse-scale computing and traditional database environments.

5.3 Atomic Page Write

To the best of our knowledge, FusionIO Atomic Write Extension is the only device available in the market that supports atomicity of page writes at the storage level [25]. It supports atomicity of page writes by altering its firmware or flash translation layer so that all sectors belonging to an atomic write are appended to a contiguous location of flash memory. Besides, a bit flag is associated with each of the sectors to detect whether a page write is done completely or partially.

Moreover, the atomic write feature of FusionIO is available only through its vendor-specific Virtual Storage Layer (VSL) interface, and need to be enabled at boot time via a special *ioctl()* command. Therefore, their approach is less portable and not widely adopted in practice despite its promising performance benefit. In contrast, *DuraSSD* does not require any change in the storage interface and provides a guarantee of atomic and durable page writes without write barriers and frequently flushing the device cache.

5.4 Page Size Tuning

A few recent studies show that a smaller page size is more effective for flash memory SSDs running OLTP applications [16, 26]. This is consistent with what we observed in the LinkBench and TPC-C benchmarks carried out with *DuraSSD*. However, the performance gain from using *DuraSSD* was much more significant. This is because *DuraSSD* was able to reduce write latencies more dramatically than the flash memory SSDs used in the previous work. Since *DuraSSD* reduces the latency of a random write of any size commonly used in practice, it can take advantage of using a small page without any serious negative effect.

6. CONCLUSION

In order to meet the strict requirements of ACID transactions, many database systems rely on redundant writes and frequently flushing device caches to stable storage (by `fsync` calls). This approach has several disadvantages, especially with modern flash memory SSDs. First, redundant writes halve the update throughput as well as the lifespan of SSDs. Second, frequent `fsync` calls deprive the write cache of the opportunity to exploit the massive parallelism inside an SSD for better random write throughput. In order to tackle these problems, we have proposed *DuraSSD*, which provides a storage level guarantee that a page write is atomic and durable once it is transferred to the DRAM cache backed up by cost-effective tantalum capacitors. The architecture and elaborate firmware algorithms for supporting atomic write, cache management, and recovery are also presented.

DuraSSD offers many benefits. First, a database system need not write the same page twice just for the sake of its atomicity any longer. Second, *DuraSSD* can be deployed as a write-back storage device without compromising the durability and write ordering so that its internal parallelism can be fully utilized for maximum write throughput. Third, with a durable cache, *DuraSSD* does not have to be forced to flush its cache, and hence can manage its device cache more efficiently and independently of application logics. Fourth, with no need to flush its cache frequently, *DuraSSD* can take advantage of using a small page size for higher throughput.

We have evaluated a *DuraSSD* prototype in the experiments with the LinkBench and YCSB benchmarks as well as the TPC-C benchmark, and have shown that *DuraSSD* can improve transactional database throughput by more than an order of magnitude. We have also shown that *DuraSSD* successfully addresses the problem of high tail latency, which is a crucial issue in modern data centers, by significantly shortening the write latency and limiting the variability of read latency.

Acknowledgments

This research was supported in part by National Research Foundation of Korea(NRF)(2012R1A1A2A10044300) and the IT R&D program of MKE/KEIT [10041244, SmartTV 2.0 Software Platform]. This work was also supported by Research Resettlement Fund of SNU.

7. REFERENCES

[1] Atomic Commit In SQLite. `http://www.sqlite.org/atomiccommit.html`, 2012.

[2] MySQL 5.7 Reference Manual. `http://dev.mysql.com/doc/refman/5.7/en/`, 2013.

[3] PostgreSQL 9.3.1 Documentation. `http://www.postgresql.org/docs/9.3/`, 2013.

[4] Nitin Agrawal, Vijayan Prabhakaran, Ted Wobber, John D. Davis, Mark Manasse, and Rina Panigrahy. Design Tradeoffs for SSD Performance. In *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, ATC'08, pages 57–70, 2008.

[5] Timothy G. Armstrong, Vamsi Ponnekanti, Dhruba Borthakur, and Mark Callaghan. LinkBench: a Database Benchmark based on the Facebook Social Graph. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, pages 1185–1196, 2013.

[6] Shimin Chen, Phillip B. Gibbons, Todd C. Mowry, and Gary Valentin. Fractal Prefetching B+-Trees: Optimizing Both Cache and Disk Performance. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, SIGMOD '02, pages 157–168, 2002.

[7] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking Cloud Serving Systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, pages 143–154, 2010.

[8] G. Copeland, T. Keller, R. Krishnamurthy, and M. Smith. The Case for Safe RAM. In *Proceedings of the 15th International Conference on Very Large Data Bases*, VLDB '89, pages 327–335, 1989.

[9] Jeffrey Dean and Luiz André Barroso. The Tail at Scale. *Communications of the ACM*, 56(2):74–80, February 2013.

[10] Doug Crowthers. What's the Deal With Write-Cache Buffer Flushing? `http://www.tomshardware.com/reviews/ssd-performance-tweak,2911-15.html`, June 2011.

[11] H. Garcia-Molina and K. Salem. Main Memory Database Systems: An Overview. *IEEE Transactions on Knowledge and Data Engineering*, 4(6):509–516, dec 1992.

[12] Samuel Greengard. eBay Bets On Flash. `http://www.baselinemag.com/storage/eBay-Bets-On-Flash`, August 2011.

[13] James Hamilton. The Cost of Latency. `http://perspectives.mvdirona.com/2009/10/31/TheCostOfLatency.aspx`, October 2009.

[14] Guy Harrison. Flash Tablespace vs. DB Flash Cache. `http://guyharrison.squarespace.com/blog/2010/1/24/flash-tablespace-vs-db-flash-cache.html`, Jan 2010.

[15] W. W. Hsu and A. J. Smith. The Performance Impact of I/O Optimizations and Disk Improvements. *IBM Journal of Research and Development*, 48(2):255–289, 2004.

[16] Sang-Won Lee, Bongki Moon, and Chanik Park. Advances in Flash Memory SSD Technology for Enterprise Database Applications. In *Proceedings of the 35th SIGMOD international conference on Management of data*, pages 863–870, 2009.

[17] Scott T. Leutenegger and Daniel Dias. A Modeling Study of the TPC-C Benchmark. In *Proceedings of ACM SIGMOD*, pages 22–31, 1993.

[18] Raymond A. Lorie. Physical Integrity in a Large Segmented Database. *ACM Transactions on Database Systems*, 2(1):91–104, Mar 1977.

[19] Mark Marchukov. TAO: The Power of the Graph. `http://goo.gl/DBpCrZ`, june 2013.

[20] Cade Metz. Flash Drives Replace Disks at Amazon, Facebook, Dropbox. `http://www.wired.com/wiredenterprise/2012/06/flash-data-centers`, June 2012.

[21] Cade Metz. Apple and Facebook Flash Forward to Computer Memory of the Future. `http://www.wired.com/wiredenterprise/2013/03/flash-fusion-io-apple-facebook/`, March 2013.

[22] C. Mohan. Disk Read-Write Optimizations and Data Integrity in Transaction Systems Using Write-Ahead Logging. In *Proceedings of ICDE*, pages 324–331, 1995.

[23] Wee Teck Ng and Peter M. Chen. Integrating reliable memory in databases. *The VLDB Journal*, 7(3):194–204, August 1998.

[24] Marc Noblitt. Proposal for New Flush Cache Command. `http://www.t10.org/t13/technical/e01126r0.pdf`, June 2001.

[25] Xiangyong Ouyang, David W. Nellans, Robert Wipfel, David Flynn, and Dhabaleswar K. Panda. Beyond Block I/O: Rethinking Traditional Storage Primitives. In *Proceedings of International Conference on High-Performance Computer Architecture*, HPCA '11, pages 301–311, 2011.

[26] Ilia Petrov, Robert Gottstein, Todor Ivanov, Daniel Bausch, and Alejandro P. Buchmann. Page Size Selection for OLTP Databases on SSD Storage. *Journal of Information and Data Management*, 2(1):11–18, 2011.

[27] Robert McMIllan. Amazon Blames Generators for Blackout That Crushed Netflix. `http://www.wired.com/wiredenterprise/2012/07/amazon_explains/`, July 2012.

[28] Quest Software. Benchmark Factory for Databases. `http://www.quest.com/benchmark-factory/`.

[29] Jon A. Solworth and Cyril U. Orji. Write-Only Disk Caches. In *Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, SIGMOD '90, pages 123–132, 1990.

[30] Michael Stonebraker. The Design of the POSTGRES Storage System. In *Proceedings of the 13th International Conference on Very Large Data Bases*, VLDB '87, pages 289–300, 1987.

[31] Sybase. SQL Anywhere I/O Requirements for Windows and Linux. A Whitepaper from Sybase, an SAP Company, March 2011.

[32] Vadim Tkachenko. SSD, XFS, LVM, Fsync, Write Cache, Barrier and Lost Transactions. `http://goo.gl/dlpjNa`, November 2009.

[33] Mai Zheng, Joseph Tucek, Feng Qin, and Mark Lillibridge. Understanding the Robustness of SSDs under Power Fault. In *Proceedings of USENIX conference on File and Storage Technologies*, FAST '13, pages 271–284, 2013.

[34] Roberto V. Zicari. Measuring the Scalability of SQL and NoSQL Systems. `http://goo.gl/ai3nE1`, May 2011.