DP^2: Reducing Transaction Overhead with Differential and Dual Persistency in Persistent Memory

Long Sun, Youyou Lu, and Jiwu Shu*
Department of Computer Science and Technology, Tsinghua University, Beijing, China
Tsinghua National Laboratory for Information Science and Technology, Beijing, China
sun-l12@mails.tsinghua.edu.cn, {luyouyou, shujw}@tsinghua.edu.cn

ABSTRACT

Emerging non-volatile memory (NVRAM) technologies, like phase change memory, envision persistent memory architectures. In case of power failure, operations on persistent memory should be in transactional semantics by adopting techniques such as WAL. To ensure consistency and atomicity, persist barriers are widely adopted, to prevent persistent memory controller from scheduling writes and exploiting bank-level parallelism of NVRAM devices. Besides, unified retention time for persistent writes, i.e., log and data writes, further reduces the performance of persistent memory system, while retention time for log writes does not need to be so long due to periodic truncation. In this paper, we study how NVRAM write latency affects the system throughput and propose DP^2, which consists of two main techniques: *differential persistency* and *dual persistency*. *Differential persistency* distinguishes log writes from data writes, and enhances the NVRAM memory controller to schedule log writes across persist barriers to fully utilize bank level parallelism. *Dual persistency* relaxes the retention time of log writes to reduce write latency and the iterations per write, which in turn enhances lifetime of NVRAM devices. Evaluation results show that our proposed techniques improve system throughput up by 43% on average and extend lifetime up by 47%, with 10^4-s retention time for log writes.

Categories and Subject Descriptors

D.4.2 [**Operating System**]: Storage Management; D.4.5 [**Operating System**]: Reliability; D.4.8 [**Operating System**]: Performance; B.8 [**Hardware**]: Performance and Reliability

*Corresponding author: Jiwu Shu (shujw@tsinghua.edu.cn).

CF'15 2015, May 18-21, 2015, Ischia, Italy

ACM 978-1-4503-3358-0/15/05...$15.00
http://dx.doi.org/10.1145/2742854.2742864.

General Terms

Performance, Design, Reliability

Keywords

Non-volatile memory; transaction; performance; retention time.

1. INTRODUCTION

Recently, byte-addressable non-volatile memory (NVRAM), a.k.a storage class memory (SCM), has been widely studied. In particular, phase-change memory (PCM), a kind of NVRAM, has many promising characteristics such as potential large capacity and low power consumption, while suffering from long write latency and limited write endurance. In addition, several multi-level cell (MLC) techniques have been studied to store two or four bits of data [12].

Computer architecture evolves with the adoption of NVRAM [5]. A lot of researchers rethink the traditional computer systems or design new storage architectures to take advantage of NVRAM. These designs can be classified into three categories. The first kind of work target NVRAM as primary storage, for the fast access speed compared to conventional persistent storage, such as [21, 2]. The second kind of work envision NVRAM as the drop-in replacement of DRAM, utilizing the byte-addressability and potential large density, including [21, 29, 12]. The third kind of work focus on building hybrid main memory. There are two design choices of this class. One alternative is using NVRAM as primary main memory, together with small volume DRAM serving as cache, such as [4, 23]. The other is to place NVRAM and DRAM on the same level, providing a uniform address space to operating system, including [3, 26, 6, 18, 11].

In addition to architectural design, several programming models about NVRAM have been proposed. BPFS [4] and PMFS [5] both are file systems that provide system API to applications to use underlying NVRAM hardware. NV-Heaps [3] and Mnemosyne [26] both are built as libraries that programmers can programming with NVRAM memory using special *malloc/free* functions. CDDS [24] allows programmers to safely exploit the low-latency and non-volatile aspects of new memory technologies, showing the performance gain on how data structures such as CDDS B-tree can take advantage of NVRAM.

When using NVRAM as persistent storage, *consistency* and *atomicity* must be maintained. To ensure consistency and atomicity, writes to NVRAM must be in linearizable

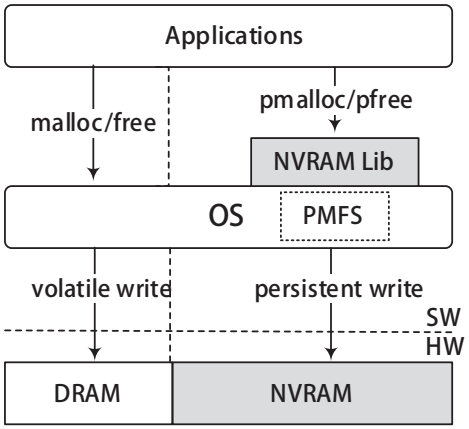

Figure 1: A Typical NVRAM-DRAM Hybrid System Architecture

order. However, most of the compilers and modern CPUs will reorder writes to improve performance, which is transparent to the operating system and applications. Thus, several special instructions must be used to prevent compilers and CPUs from reordering writes, such as *clflush*, *mfence* and so on. Several recent works [26, 5, 12] have noted that, those instructions are not enough to ensure the write ordering, because they do not guarantee that modified data actually reached the durability point; i.e., to persistent memory or some intermediate power-fail safe buffer [5]. Thus there must be architectural support for persist barriers [4, 21, 5]. Techniques to enhance atomicity include shadow-paging (SP) [8, 24, 4] and write ahead log (WAL) [16, 3, 26, 21, 6, 11, 5]. In this paper, we focus on the latter. WAL records redo or/and undo logs to provide atomicity, which results in additional writes to NVRAM. Logs must also be written to persistent memory and committed before in-place updates of persistent data structures. Both additional log writes and ordering constraints of WAL reduce the NVRAM system performance significantly.

Figure 1 shows a typical system architecture which architects DRAM and NVRAM to build hybrid main memory and provides applications with accessibility to NVRAM through libraries (i.e., Non-volatile Heaps). NVRAM libraries implement software transactional memory (STM) [9] to maintain consistency and atomicity. In this model, system performance is limited from several aspects. First, the performance of NVRAM is limited due to long write latency. Second, as persist barriers are widely adopted, NVRAM memory controller must respect every persist barrier to maintain ordering constraints, which prevents scheduling any writes across barriers. However, log writes and data writes are inherently different, and scheduling rules can be adopted by differentiating them. Third, STM introduces additional write loads (i.e., log writes) to NVRAM. Log writes add additional write latency beyond data writes and accelerate the wear-out speed of NVRAM devices. However, because log records are truncated periodically, retention time of logs needn't as long as that of data. Thus some sort of Retention Relaxation [12] techniques can be used to both speedup log writes and reduce average iterations for one write.

In this paper, we target PCM-DRAM hybrid main memory system in which DRAM serves as working memory and PCM serves as persistent in memory store. However, our proposed techniques apply to other NVRAM devices, as illustated in Section 2.2. We will analyse how PCM's long write latency and additional writes (i.e., log writes) to PCM will affect system performance. And then we propose DP^2, which consists of two techniques as follows:

(1) *Differential persistency*, distinguishes log writes from normal data writes and enhances the memory controller to adopt two rules to schedule log writes. With *differential persistency*, memory controller can fully utilize the bank level parallelism of PCM chips, while still respecting each of the persist barriers.

(2) *Dual persistency*, performs log writes and data writes on PCM devices with different retention guarantees. With *dual persistency*, log writes can be performed faster and need less iterations per write to PCM cells.

Evaluation results show that DP^2 can improve system throughput up by 43% on average and enhance lifetime up by 47%, with 10^4-s retention time for log writes.

The rest of the paper is organized as follows. We present the background and motivation of this paper in Section 2. Then, in Section 3, we introduce DP^2, including the proposed scheduling rules, optimizations on dual retention mode to reduce write latency and several design details. In Section 4, evaluations are performed to justify the effectiveness of our proposed techniques. And finally, we present related work in Section 5 and conclude the paper in Section 6.

2. BACKGROUND AND MOTIVATION

2.1 Non-volatile Heaps

In this paper, we focus on the library programming model, i.e., non-volatile heaps, with which programmers can manage NVRAM main memory through calls, i.e., *pmalloc/pfree*. There are two main designs in literature, NV-heaps [3] and Mnemosyne [26], both of which are built on hybrid main memory. Although the design and implementation details of them differ, the basic idea that constructing user level library with STM to exploit NVRAM, is the same. We take Mnemosyne as an example, and illustrate its design rules and some implementation details.

STM is a software method that provides concurrency control and supports transactional programming of synchronization operations on memory. Mnemosyne takes TinySTM [7], a light weight time-based STM implementation, to provide concurrency control on word granularity and to ensure consistent updates at all times. Mnemosyne implements lazy version management with write-ahead per-thread redo log, and eager conflict detection with encounter-time locking. To provide consistency and atomicity Mnemosyne assumes the atomic 8 bytes write from architectural support like BPFS [4] . In order to map consistency onto hardware, Mnemosyne relies on three hardware primitives: write-through stores (i.e., *movnti*), fences (i.e, *mfence*) and flushes (i.e, *clflush*). If a programmer wants to build persistent data structures, he needs to declare persistent variables, such as *pint*, claim memory using *pmalloc*, annotate a segment of codes with *atomic*, and compile the codes using compilers with STM support.

Table 1: Write speedup factors with different retention guarantees and average iterations for write operations

Non-Volatility(s)	Write Speedup	Average Iterations (one 64B row)
10^7	Baseline	5.7
10^6	1.2 ×	5.7 / 1.2
10^5	1.5 ×	5.7 / 1.5
10^4	1.7 ×	5.7 / 1.7
10^3	1.9 ×	5.7 / 1.9
10^2	2.1 ×	5.7 / 2.1

2.2 Retention Time vs. Write Latency

Retention time refers to the time period that during which the data stays valid. Persistent store requires a minimum retention time (e.g., of at least three months [12]). Generally speaking, long retention time requires a proportional long write latency. Previous works have presented the trade-off between retention time and write speed, mainly on STT-RAM-based caches and Retention Relaxation for NAND flash-based solid state drives (SSDs), refering to NVM Duet [12]. In contrast, NVM Duet focus on trading off retention time and write performance in the manner emphasized by Retention Relaxation, and quantify how PCM's non-volatility correlates with its write speed [12]. We omit the details of PCM modelling and refer to the modelling results directly shown in Table 1.

The first column in Table 1 shows the desired retention time, while the second column illustrates the potential write speedup when relaxing retention time. 10^7-s is the minimum threshold of durable retention and its write speed serves as baseline. One obvious conclusion we can draw is that NVRAM system can obtain a significant write latency reduction by lowering the retention guarantee. For example, the write speedup can achieve 1.7× for a PCM with 10^4-s retention capability. In order to utilize this property, NVM Duet differentiates writes of working memory from writes of persistent store. The retention time for working memory is relatively short, and can be kept valid through Smart Refresh [12], thus achieving fast write speed.

2.3 Motivation

According to [5], write speed of PCM (i.e., 100MB/s) is about 10× slower than DRAM (i.e., 1GB/s). Thus the performance of update-intensive workloads can be reduced by data persist operations significantly. We instrument Mnemosyne to analyse how write latency affects performance. We break down software stack into three parts: (1) log operations, including log writes, memory fences, etc.; (2) in-place updates, including write-backs of persistent data, flushing cache lines into PCM, etc.; (3) others, including STM instrumentation, application operations, etc. We collect the time that each part executes using experimental setup illustrated in Section 4.1 and show results in Figure 2. Figure 2(a) shows write overhead of different workloads. Persist overhead can reach 52.2%-66.2%, and log overhead can reach 27.8%-44.3%. The average log overhead of all workloads is 36.5%. Figure 2(b) shows the performance overhead with varying value sizes. The average log overhead of all value sizes is 38.4%. Based on the observations, reducing log overhead can improve system performance

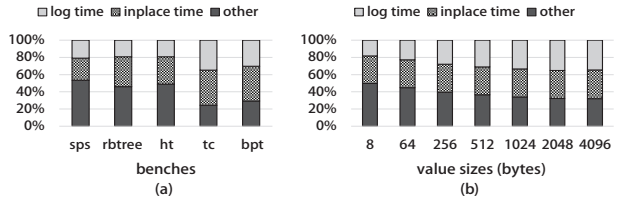

Figure 2: Persist overhead of both log writes and data writes to PCM with different benchmarks and varying value sizes of HashTable

significantly.

When architecting PCM and DRAM as hybrid main memory, writes arrive at PCM are log writes and normal data writes, both of which use persist-barriers to maintain ordering constraints. In order to maintain consistency and atomicity, at the hardware level, memory controller and other components must respect persist-barriers when performing write requests [12]. We note that the retention duration of log need not be as long as that of data, for logs can be truncated synchronously or periodically. Thus by differentiating log writes from data writes, system can gain several benefits.

3. DP^2 DESIGN

3.1 System Support

To distinguish log writes from data writes, DP^2 needs support from system hardware and software. Recently proposed NVM Duet [12] adds volatile AllocMap to memory controller and augments an additional bit into each write request. Bits in AllocMap indicating whether writes belong to working memory or persistent store, are updated by OS, and combined into write request before sent to write queue. The intuitive method is using NVM Duet to implement DP^2, by regarding log writes as working memory writes. However, we can take advantage of the existing software and hardware design without adding additional storage overhead to memory controller. In hybrid memory architecture, PCM is used to store both append-only log which needs small capacity, and persistent data which needs large volume. Thus the whole PCM space can be partitioned into log portion and data portion logically, differentiating writes to these two parts through specific address bits (e.g., the address range can be used as indication of the write type). We will show this in Section 3.3 later.

DP^2 needs the Dual-Retention PCM chips proposed by NVM Duet to achieve dual retention guarantees. However, we make some optimization for our particular use case, which will be shown in Section 3.4. To maintain consistency, several architectural support must be provided: (1) *movntq*, a non-temporal hint, implemented by using write combining memory [1]; (2) *mfence*, performing a serializing operation to avoid reordering and ensure cache coherency [1]; (3) some sort of persist barriers. We assume the *pm_wbarrier* [5] to be the basic system primitive and refer as barrier for short.

3.2 DP^2 Architecture

DP^2 system architecture is shown in Figure 3, which is based on a typical NVRAM-DRAM hybrid main memory

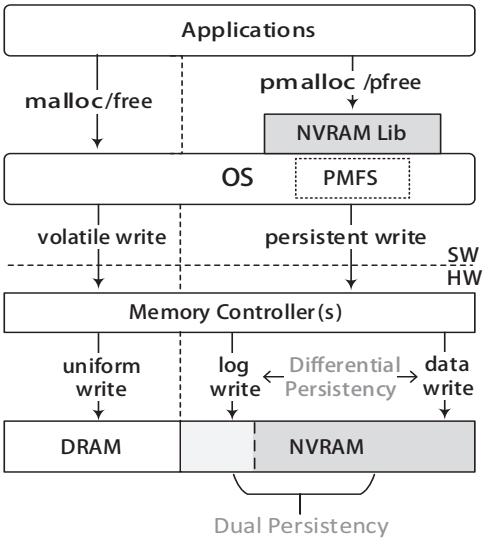

Figure 3: DP^2 System Architecture

architecture shown in Figure 1. Applications call *pmalloc/pfree* to utilize PCM to build persistent in memory data structures. NVRAM Lib is implemented as non-volatile heaps such as Mnemosyne to maintain consistency and atomicity. System interfaces should be provided such as *mmap* from traditional OS, or from the recently proposed PMFS [5]. DP^2 modifies the memory controller for PCM by adding two scheduling rules to differentiate log writes from data writes, and enhances the controller to fully utilize bank-level parallelism of PCM devices. We call this technique as *differential persistency*. As to PCM devices, two sets of retention parameters, i.e., the "normal" set and "short" set, will be used to perform writes. The "normal" set is used for data writes with long retention guarantee, while the "short" set is used for log writes with short durability. We call this technique as *dual persistency*. We will introduce these two techniques in detail and analyse how they improve system performance in the following sections.

3.3 Differential Persistency

Non-volatile heaps adopt STM to implement concurrency control and to maintain consistency, which in turn rely on some sort of WAL, such as redo log (both [3] and [26]). Thus a segment of application codes operates on PCM memory in transactional semantics. A transaction begins by creating read-sets and write-sets. Writes are redirected to write-sets after creating redo logs. Transaction commits by flushing redo logs first followed by a barrier, then flushing a commit record followed by a barrier, then updates writes in-place, and finally truncates log. Currently multi-threading programming is widely adopted to improve transaction throughput. As shown in Figure 4, we illustrate concurrent transactional write requests arriving at memory controller. Figure 4(a) shows a common case of write request sequence. All writes reaching persistent memory controller are treated as persistent writes separated by four persistent barriers. L0, C0 and D0 belong to transaction A, while L1, L2, C1, D1 belong to transaction B. Memory controller must respect every barrier in order to provide consistency. With two memory banks (i.e., Bank 1 and Bank2), a baseline

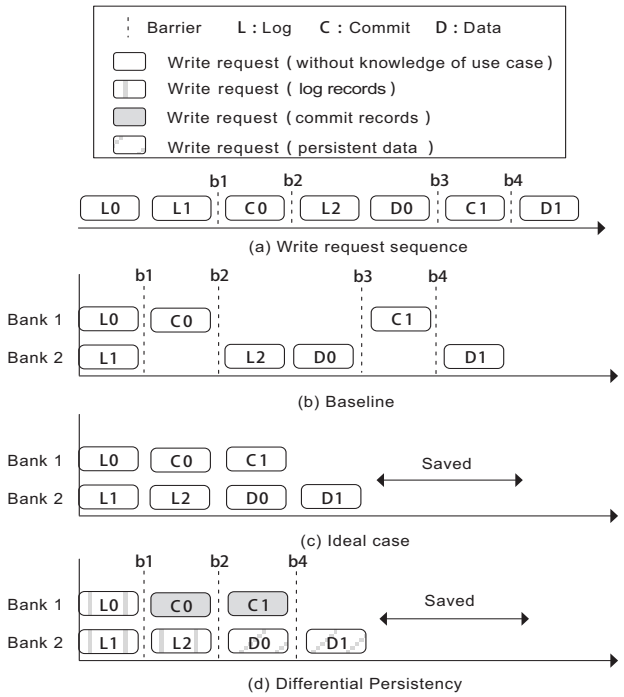

Figure 4: Write sequences and scheduling results for the proposed rule

scheduling is shown in Figure 4(b). Removing all persist barriers results in the ideal case, which is shown in Figure 4(c).

We propose *differential persistency*, which distinguishes log writes from data writes. Memory controller can take advantage of the differential information to schedule writes, to fully utilize bank-level parallelism. *Differential persistency* groups persistent writes into three categories: log records, commit records and persistent data, and adopts two scheduling rules:

Rule 1: Writes of log records can be scheduled before the persist barrier of its own commit record.

Most of the implementations of log are append-only, although some studies propose in-place updates to utilize byte-addressability of NVRAM (such as [11]). Thus, if only log records are persisted before commit record of the same transaction, consistency guarantee is obtained. In Figure 4(b), L2 can be scheduled to write concurrently with C0.

Rule 2: Writes of commit records can be performed one step former the pre-commit barrier.

From Figure 4(b), we can observe that C1 doesn't need to wait for D0. Rule 2 relaxes this constraint. Note that, commit records can't be scheduled arbitrarily because they must not persisted before any of log records of the same transaction.

The proposed scheduling rules can improve the bank-level parallelism utilization and don't violate the consistency guarantee. Figure 4(d) shows the potential performance gain when the two scheduling rules are adopted. After all, implementing this rule is simply adding the comparison logic without additional storage overhead like NVM Duet (i.e., AllocMap and additional bit in write request).

3.4 Dual Persistency

As illustrated in [12], PCM has several specific properties. For example, its retention capability is related to the width of target bands and shorter retention period leads to faster write speed. While architecting PCM as a unified main memory, writes can be classified into two categories: working memory writes, which needs fast write speed and no durability guarantee, and persistent store writes, which take the durability guarantee at the first place. Thus, dual-retention PCM chips (i.e., PCM chips include two sets of resistance parameters) can be used to support two kinds of write modes. In DRAM and PCM hybrid main memory, DRAM serves for working memory while PCM serves for persistent store. However, we note that persistent writes (i.e., writes to PCM) can also be differentiated in non-volatile heaps programming model, i.e., with differential persistency. To provide consistency and atomicity, log writes are necessary and they also need to be persisted before being truncated. Another fact is that logs that are stored in PCM do not live for a long time. Two log truncation modes are widely adopted: synchronous mode and asynchronous mode, to prevent the log from growing too large. In synchronous mode, logs are truncated immediately after data have been updated in-place. In asynchronous mode, logs are periodically truncated to avoid wrap-around because log buffer is usually small. In conclusion, the retention time of logs don't need to be the same as that of persistent data. We propose *dual persistency*, and will illustrate that performance improvement can be gained by relaxing the retention guarantee for log writes.

Log region in PCM is relative small and will spread across into memory banks evenly to fully utilize the bank-level parallelism. Log region is separated from data region. *Dual persistency* distinguishes log writes, including normal log entries and abort/commit entries, from data writes by comparing write address with pre-designated address ranges. Memory controller maintains two sets of resistance parameters: one set is "normal" (such as for 10^7-s retention capability), the other set is "short" (such as for 10^4-s retention capability), and uses "normal" set to perform data writes while "short" set for log writes. How short the retention time will be, is determined by what level durability guarantee that system requires. We will show the performance gain with different level durability guarantees in Section 4. Because logs are periodically truncated, PCM cells belong to log region are rewritten and invalidated frequently. Thus, *dual persistency* doesn't need DRAM-like refresh mechanism such as Smart Refresh [12].

The proposed *dual persistency* can improve system performance from many aspects. First, the amount of log writes are typically larger than persistent data writes (i.e., at least 2× with raw word log). If retention guarantee can be relaxed, log writes can be performed faster which speedup transaction processing significantly. Second, *dual persistency* doesn't refresh both data region and log region, avoiding extra energy consumption. Relaxed retention also results in less iterations when performing writes to PCM cells, thus reducing power consumption further. Third, the address space of log region design spread across all memory banks. System performance can be improved further by fully utilizing bank-level parallelism.

4. EVALUATION

Table 2: Description of workloads

Workloads	Description
SPS [3]	Random swaps of array entries
RBTree [3]	Insert/delete nodes in a red-black tree
HashTable [26]	Insert/delete entries in a hash table
KVStore [26]	Database benchmark on TokyoCabinet
B+Tree [15]	Insert/delete nodes in a B+ tree

4.1 Experimental Setup

In this evaluation, we take Mnemosyne [26] as the baseline system, and compare *NVRAM-DP* with it to show the effectiveness of our proposed techniques. All experiments are conducted on a server with a 2.4GHz Intel Xeon E5645 CPU and 64GB main memory. To emulate PCM's writes, we add additional latency to PCM writes. We use the same parameters as in [12], i.e., 250 ns per iteration. The average iterations to write a cell is listed in Table 1 and we take 10^7-s as the basic retention guarantee. The DRAM latency is measured by Intel Memory Latency Checker [25] as 85ns.

Workloads. Our experiment evaluates several workloads shown as Table 2. We evaluate several basic data structures widely used both in real application and in literature [3, 26], such as random swap in a large data array (SPS), a red-black tree and hashtable. We set the size of all keys and values in those data structures to be 64 bits by default. To evaluate the performance in real Key-Value systems, we evaluate a well-known KV-store, Tokyo Cabinet [10]. B+ tree is widely used both in database management systems and file systems. We implement a B+ tree, with a node size of 4KB and each key or value of 8 bytes size.

4.2 Experimental Results

4.2.1 Differential Persistency

To evaluate the effectiveness of the scheduling rules under *differential persistency*, we model a PCM memory controller with 4 ranks of 8 banks each, with 32-entry write queue per bank. We collect writes traces to PCM as well as persist barriers. The PCM memory controller takes these write-intensive traces to simulate write bandwidth under heavy writing workloads, and it respects every persist barrier to ensure consistency and atomicity. We implement the basic algorithm with no schedules across barriers and take it as baseline. And then we add rule 1 and rule 2 respectively. Figure 5 shows the bandwidth speedup results of HashTable with varying value sizes and different rules. In all cases, schedules with proposed rules perform better than the baseline. In particular, 1024 byte value size, rule 1 performs 41%, rule 2 performs 1% and rule 1+2 performs 44% better.

Figure 6 shows the effectiveness with *differential persistency* with varying number of threads. When value size is larger than 64 bytes, using the two rules can improve performance increasingly with the adding number of threads. Write sequences arriving at memory controller are affected by both the value size of a transaction and the number of threads of concurrent transactions. For example, with 4096 bytes, performance improvement grows with the number of threads, because the number of concurrent transactions affect the write sequence mainly. In contrast, with 8 byte, transaction size dominates the affection of write sequences and thus *differential persistency* shows

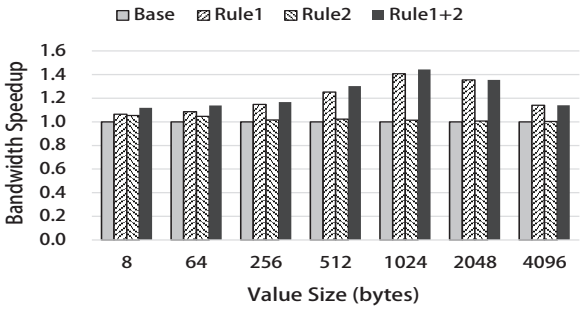

Figure 5: Bandwidth speedup with proposed scheduling rules under Differential Persistency

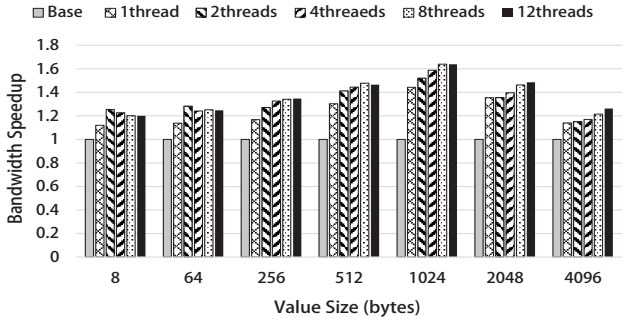

Figure 6: Bandwidth speedup with multithreads using proposed scheduling rules under Differential Persistency

unpredictable performance gains under multi-threads. In conclusion, 33% performance gain can be achieved by combining rule 1 and rule 2 on an average.

4.2.2 Dual Persistency

We take 10^7-s (i.e., three months) retention as the baseline retention guarantee for both log writes and data writes. Then we vary the retention guarantee for log writes to show the potential throughput speedup with *dual persistency*. Figure 7 shows the potential throughput speedup of HashTable. We take 10^4-s (i.e., about 2.8 hours) as an reasonable retention guarantee. Write speed of 10^4-s is 1.7x than the baseline according to Table 1. On average, *dual persistency* can improve 24% on throughput.

Differential persistency can benefit from *dual persistency* as well. In both cases, i.e., baseline schedule and *differential*

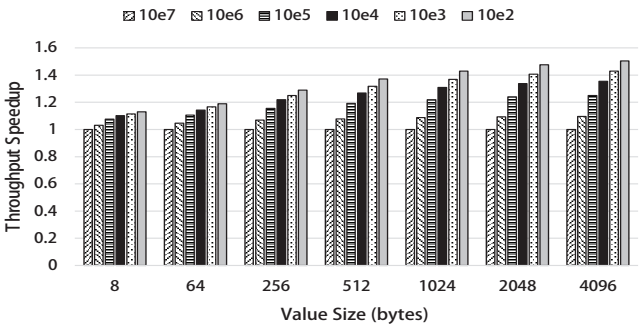

Figure 7: Throughput speedup using Dual Persistency with varying retention guarantees for logs

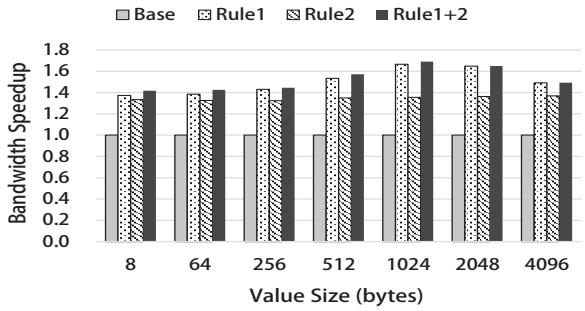

Figure 8: Bandwidth speedup with proposed scheduling rules under Dual Persistency

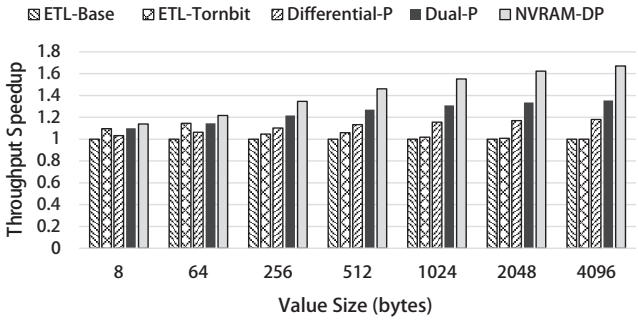

Figure 9: Throughput speedup comparison with all basic and proposed methods

persistency schedule, there can be writes to log region concurrently. Those concurrent writes will be performed faster than normal data writes with *dual persistency*. Figure 8 shows the bandwidth speedup of *differential persistency* with *dual persistency* support. The average improvement of rule 1 is 50%. The average improvement of rule 2 is 35%. And 53% improvement will be gained by combining the two rules.

4.2.3 NVRAM-DP:Put Them Together

In this section, we will put the proposed techniques together. We first show in Figure 9 the throughput speedup trends using HashTable with varying value size. And then we illustrate in Figure 10 the potential improvements on different workloads with all the value size set to 8 byte. In these two figures, ETL-Base and ETL-Tornbit represent the log techniques with unified write latency to PCM. Differential-P is with *differential persistency* and Dual-P is with *dual persistency* only. *NVRAM-DP* combines *differential persistency* and *dual persistency*.

As shown in Figure 9, ETL-Tornbit performs better than ETL-Base with small value size, i.e. 10% with 8 bytes and 14% with 64 bytes, because large value size will increment the computing overhead on tornbits. Dual-P performs 24% better than ETL-Base on average and still better than ETL-Tornbit with all value sizes. On average ETL-Tornbit improves 5% over ETL-Base and *NVRAM-DP* improves 43%.

Figure 10 illustrates the throughput speedup results of different workloads. We set the value size to 8 bytes. *NVRAM-DP* performs best on SPS, because a SPS transaction is simply swapping two values in an array and thus persistent

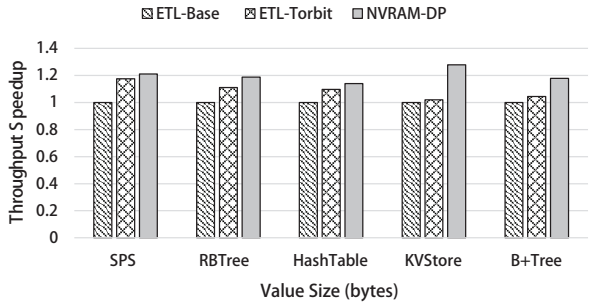

Figure 10: Throughput speedup comparison of different workloads with 8 byte value size, with basic methods and NVRAM-DP

writes dominate the executing time. *NVRAM-DP* performs better than both ETL-Base and ETL-Tornbit. On an average, *NVRAM-DP* performs 20% better than ETL-Base. Note that, more throughput improvement can be gained by enlarging the modifed value size of each transaction as analyzed before.

4.2.4 Endurance Analysis

We assume the underlying wear-leveling mechanism is ideal, and analyze the endurance enhancement with *NVRAM-DP* by an informal inference. As shown in Equation 1, we model endurance with three factors. NVRAM is divided into two parts: *LogPortion* and *DataPortion*. The factor α is the potential improvement on baseline by adopting different techniques. In ETL-Base, the number of log bytes is two folds of data bytes. Thus *LogPortion* is 2/3 and *DataPortion* is 1/3. α parameter of *NVRAM-DP* is obtained from former evaluation with 10^4-s retention guarantee for log writes,i.e, $\alpha = 1.7$. Dual-P can reduce the iterations per write to a NVRAM cell, while Differential-P has no effect on endurance. By taking Dual-P, *NVRAM-DP* can enhance the lifetime of NVRAM devices up by 47%.

$$Endurance = LogPortion \times \alpha + DataPortion \quad (1)$$

5. RELATED WORKS

Emerging non-volatile memory technologies have been used to reduce consistency overhead in storage systems based on flash memory [22, 19, 13, 14] and NVRAM [4, 5, 3, 26, 24]. We take a further step that optimize the log operations in order to improve system performance using NVRAM. Huang et. al. propose NVRAM-aware logging [11] to speed up OLTP, which store logs on NVRAM and data on disks. They give a cost-effective analysis about the proposed hybrid storage. While our mechanism is proposed based on hybrid memory where both log writes and normal persistent writes are stored to NVRAM, we are confident that with DP^2, logging optimization like [11] will achieve more performance gain.

Several works have been proposed to relax the write ordering or persist ordering to improve system throughput [20, 15, 27, 17], while still maintaining consistency and atomicity. Memory persistence[20] introduces three level of persistency, each of which relaxes NVRAM write constraint and allows high performance and currency data structures to different degrees. Loose ordering persistency [15] depends on modified cache structure and memory controller to provide enhanced performance. Kiln[27] uses non-volatile last level cache to removing the need of logging. The extreme design of durable memory is WSP [17], which assumes that the whole system hardware is non-volatile. Thus providing consistency and atomicity is trivial with WSP. In contrast to [20] and [15], our work investigates lower level ordering relaxation to exploit the bank-level parallelism of NVRAM device.

To optimize the write performance or write endurance of NVRAM systems, previous works have presented Retention Relaxation for NVRAM. Liu et. al. introduce NVM Duet in [12]. They differentiate working memory from persistent store even the whole main memory is PCM. By doing so, writes to working memory can be reordered to fully utilize the bank-level parallelism. Another benefit is that writes to working memory can be speedup by combing short latency write and period refresh. In contrast, we focus on improving system performance under hybrid main memory. We distinguish log writes to normal persistent writes, both of which arrive at NVRAM. According to the level of availability of power supply, less refresh or even no refresh is need. In addition, FIRM [28] maximizes system performance and fairness by balanced scheduling read and write requests of different types. We will exploit the potential benefits by considering both reads and writes in future work.

6. CONCLUSIONS

Byte-addressable NVRAM can be used to build hybrid main memory with DRAM, while DRAM serves as working memory and NVRAM as persistent store. In order to maintain consistency and atomicity in case of power failure, modification to NVRAM needs recovery techniques such as WAL. However, long write latency of NVRAM compared to DRAM results in expensive logging overhead. Besides long write latency, persist barriers should be respected by NVRAM memory controller to maintain consistency and atomicity. To mitigate logging overhead and enhance memory controller to schedule writes across persist barriers, this paper take advantage of retention relaxation techniques, which are used to improve write speed through relaxing the retention guarantee, and propose DP^2 with two main techniques: *differential persistency* and *dual persistency*. *Differential persistency* distinguishes log writes from normal data writes and schedules log writes with two rules to fully utilize the bank level parallelism of NVRAM chips. *Dual persistency* performs log writes and data writes with different retention guarantees. Evaluation results show that DP^2, which deploys *differential persistency* and *dual persistency*, can improve system throughput up by 43% on average and enhance lifetime up by 47%.

7. ACKNOWLEDGMENTS

We thank Volvos et. al. for their open source Mnemosyne library and their suggestions on tuning PCM bandwidth etc. This work is supported by the National Natural Science Foundation of China (Grant No. 61232003, 61327902), the National High Technology Research and Development Program of China (Grant No. 2013AA013201), Samsung Electronics, Huawei Technologies Co. Ltd., and Tsinghua University Initiative Scientific Research Program.

8. REFERENCES

[1] Intel architecture instruction set extensions programming reference, 319433-015. 2013.

[2] A. M. Caulfield, A. De, J. Coburn, T. I. Mollow, R. K. Gupta, and S. Swanson. Moneta: A high-performance storage array architecture for next-generation, non-volatile memories. In *Proceedings of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2010.

[3] J. Coburn, A. M. Caulfield, A. Akel, L. M. Grupp, R. K. Gupta, R. Jhala, and S. Swanson. Nv-heaps: Making persistent objects fast and safe with next-generation, non-volatile memories. *SIGPLAN Not.*, 46(3), Mar. 2011.

[4] J. Condit, E. B. Nightingale, C. Frost, E. Ipek, B. Lee, D. Burger, and D. Coetzee. Better i/o through byte-addressable, persistent memory. In *Proceedings of the 22nd ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*. ACM, 2009.

[5] S. R. Dulloor, S. Kumar, A. Keshavamurthy, P. Lantz, D. Reddy, R. Sankaran, and J. Jackson. System software for persistent memory. In *Proceedings of the Ninth European Conference on Computer Systems*. ACM, 2014.

[6] R. Fang, H.-I. Hsiao, B. He, C. Mohan, and Y. Wang. High performance database logging using storage class memory. In *Proceedings of the 27th IEEE International Conference on Data Engineering (ICDE)*, April 2011.

[7] P. Felber, C. Fetzer, and T. Riegel. Dynamic performance tuning of word-based software transactional memory. In S. Chatterjee and M. L. Scott, editors, *PPOPP*. ACM, 2008.

[8] J. Gray, P. McJones, M. Blasgen, B. Lindsay, R. Lorie, T. Price, F. Putzolu, and I. Traiger. The recovery manager of the system R database manager. *ACM Computing Surveys*, 1981.

[9] T. Harris, J. Larus, and R. Rajwar. *Transactional Memory, 2nd Edition*. Morgan and Claypool Publishers, 2nd edition, 2010.

[10] M. Hirabayashi. Tokyo cabinet: a modern implementation of dbm, 2010.

[11] J. Huang, K. Schwan, and M. K. Qureshi. Nvram-aware logging in transaction systems. *Proceedings of the VLDB Endowment*, 8(4), 2014.

[12] R.-S. Liu, D.-Y. Shen, C.-L. Yang, S.-C. Yu, and C.-Y. M. Wang. Nvm duet: Unified working memory and persistent store architecture. *SIGPLAN Not.*, 49(4), Feb 2014.

[13] Y. Lu, J. Shu, J. Guo, S. Li, and O. Mutlu. LightTx: A lightweight transactional design in flash-based SSDs to support flexible transactions. In *Proceedings of the IEEE 31st International Conference on Computer Design (ICCD)*. IEEE, 2013.

[14] Y. Lu, J. Shu, J. Guo, S. Li, and O. Mutlu. High-performance and lightweight transaction support in flash-based SSDs. *IEEE Transactions on Computers*, 2015.

[15] Y. Lu, J. Shu, L. Sun, and O. Mutlu. Loose-consistency ordering for persistent memory. In *Proceedings of the IEEE 32nd International Conference on Computer Design (ICCD)*. IEEE, 2014.

[16] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, and P. Schwarz. Aries: a transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. *ACM Transactions on Database Systems*, 17(1), 1992.

[17] D. Narayanan and O. Hodson. Whole-system persistence. In T. Harris and M. L. Scott, editors, *Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 2012.

[18] I. Oukid, D. Booss, W. Lehner, P. Bumbulis, and T. Willhalm. Sofort: A hybrid scm-dram storage engine for fast data recovery. In *Proceedings of the Tenth International Workshop on Data Management on New Hardware*. ACM, 2014.

[19] X. Ouyang, D. Nellans, R. Wipfel, D. Flynn, and D. K. Panda. Beyond block I/O: Rethinking traditional storage primitives. In *Proceedings of the 17th IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2011.

[20] S. Pelley, P. M. Chen, and T. F. Wenisch. Memory persistency. In *Proceeding of the 41st Annual International Symposium on Computer Architecture (ISCA)*. IEEE Press, 2014.

[21] S. Pelley, T. F. Wenisch, B. T. Gold, and B. Bridge. Storage management in the nvram era. *PVLDB*, 7(2), 2013.

[22] V. Prabhakaran, T. L. Rodeheffer, and L. Zhou. Transactional flash. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation (OSDI)*. USENIX, 2008.

[23] M. K. Qureshi, V. Srinivasan, and J. A. Rivers. Scalable high performance main memory system using phase-change memory technology. In *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA)*. ACM, 2009.

[24] S. Venkataraman, N. Tolia, P. Ranganathan, and R. H. Campbell. Consistent and durable data structures for non-volatile byte-addressable memory. In *Proceedings of the 9th USENIX Conference on File and Stroage Technologies (FAST)*. USENIX, 2011.

[25] T. W. Vish Viswanathan, Karthik Kumar. Intel memory latency checker. technical report.

[26] H. Volos, A. J. Tack, and M. M. Swift. Mnemosyne: Lightweight persistent memory. *SIGPLAN Not.*, 47(4), Mar. 2011.

[27] J. Zhao, S. Li, D. H. Yoon, Y. Xie, and N. P. Jouppi. Kiln: Closing the performance gap between systems with and without persistence support. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. ACM, 2013.

[28] J. Zhao, O. Mutlu, and Y. Xie. Firm: Fair and high-performance memory control for persistent memory systems. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2014.

[29] P. Zhou, B. Zhao, J. Yang, and Y. Zhang. A durable and energy efficient main memory using phase change memory technology. *SIGARCH Comput. Archit. News*, 37(3), June 2009.