# Bridging the I/O Performance Gap for Big Data Workloads: A New NVDIMM-based Approach

Renhai Chen†, Zili Shao*, and Tao Li‡
†*Embedded Systems and CPS Laboratory, Department of Computing, The Hong Kong Polytechnic University
‡Department of Electrical and Computer Engineering, University of Florida
†csrchen@comp.polyu.edu.hk, *cszlshao@comp.polyu.edu.hk, and ‡taoli@ece.ufl.edu

***Abstract*—The long I/O latency posts significant challenges for many data-intensive applications, such as the emerging big data workloads. Recently, the NVDIMM (Non-Volatile Dual In-line Memory Module) technologies provide a promising solution to this problem. By employing non-volatile NAND flash memory as storage media and connecting them via DIMM (Dual In-line Memory Module) slots, the NVDIMM devices are exposed to memory bus so the access latencies due to going through I/O controllers can be significantly mitigated. However, placing NVDIMM on the memory bus introduces new challenges. For instance, by mixing I/O and memory traffic, NVDIMM can cause severe performance degradation on memory-intensive applications. Besides, there exists a speed mismatch between fast memory access and slow flash read/write operations. Moreover, garbage collection (GC) in NAND flash may cause up to several millisecond latency.**

**This paper presents novel, enabling mechanisms that allow NVDIMM to more effectively bridge the I/O performance gap for big data workloads. To address the workload heterogeneity challenge, we develop a scheduling scheme in memory controller to minimize the interference between the native and the I/O-derived memory traffic by exploiting both data access criticality and resource utilization. For NVDIMM controller, several mechanisms are designed to better orchestrate traffic between the memory controller and NAND flash to alleviate the speed discrepancy issue. To mitigate the lengthy GC period, we propose a proactive GC scheme for the NVDIMM controller and flash controller to intelligently synchronize and transfer data involving in forthcoming GC operations. We present detailed evaluation and analysis to quantify how well our techniques fit with the NVDIMM design. Our experimental results show that overall the proposed techniques yield 10%∼35% performance improvements over the state-of-the-art baseline schemes.**

## I. Introduction

Big data applications demand high I/O performance since enormous amount of data need to be transferred between memory and storage devices for processing. Although the performance of I/O devices has been steadily improved, I/O storage systems still remain as one of the major system bottlenecks [1], [2], [3], [4], [5], [6], [7]. In particular, long I/O latency posts significant challenges for data-intensive applications such as real-time data analytics. Therefore, how to alleviate I/O latency becomes a critical issue for big data applications.

NVDIMM (Non-Volatile Dual In-line Memory Module) technologies provide a promising solution to this problem. By leveraging non-volatile NAND flash memory as storage media and connecting them via DIMM (Dual In-line Memory Module) slots, NVDIMM devices are exposed to memory bus through which they can communicate directly with CPUs and main memory. Compared with traditional storage devices connected via slower I/O interfaces such as SATA (serial ATA), SAS (Serial Attached SCSI), and PCI Express, residing on memory bus enables NVDIMM devices to alleviate the access latency caused by I/O controllers. For example, compared with SATA SSDs (Solid State Drives), the eXFlash [8], a NVDIMM system provided by IBM, could improve read latency and write latency by more than 5 times and 13 times, respectively [1]. Moreover, NVDIMM techniques manifest additional benefits such as superior capacity scalability and saving in terms of hardware cost, power/cooling, and floor space compared with conventional SSDs [9], [10], [3], [11], [12], [13].

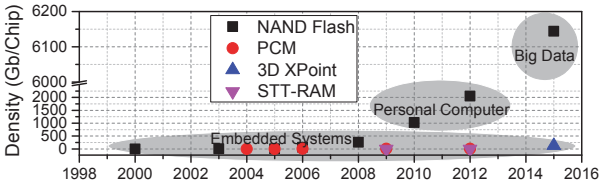



Fig. 1. The emerging non-volatile memories.

Note that there are alternative designs available that leverage emerging NVM technologies, such as the PCM-based DIMM [14], [15], [16], [17], [18], [19], [20], 3D XPoint [21], and STT-RAM [22], [23], [24]. These solutions, however, either are not as mature as the flash technology, or target application domains other than emerging big data workloads. Fig. 1 summarizes the evolution of several promising NVM technologies in terms of density and suitable application domain. For instance, currently, the leading industry vendors can provide a 6Tb single NAND flash chip [25]. However, the PCM, 3D XPoint, and STT-RAM technologies from industry are still in their infancy stage. As Big data workloads demand much more storage capacity and fast I/O access speed, the flash-based DIMM is a more feasible and cost-effective solution.

With the above advantages, NVDIMM techniques are being adopted by industries. Nevertheless, existing efforts largely focus on NVDIMM specification and fabrication issues. To maximally unleash its potential on big data applications, several new challenges have to be addressed. First, I/O traffic present salient differences compared with memory traffic. By

*Zili Shao is the corresponding author.

978-1-5090-3508-3/16/$31.00 © 2016 IEEE

migrating I/O traffic to memory bus, the I/O-derived memory traffic will severely interfere with native memory traffic. Second, the high-speed bursty I/O-derived memory traffic post new issues in NVDIMM design. Particularly, we need to carefully manage and coordinate read/write traffic while considering the slow read/write/erase operations in NAND flash. Third, in NAND flash, when garbage collection (GC) occurs, it takes up to several milliseconds to respond read/write requests, which hinders efficient data transfer.

In this paper, we develop novel, enabling mechanisms to address these challenges. In the memory controller, heterogeneous memory traffic is separated and placed into two different transaction queues so as to eliminate traffic congestion caused by the I/O-derived requests with slow data transfer speeds from NVDIMM. Based on this, to minimize the average access latency, a channel allocation mechanism is developed for the bus arbitrator to dynamically schedule requests from both queues by exploiting data access criticality (i.e. spatial/temporal locality, reuse, etc.) and resource utilization (i.e. queue length). To address the speed mismatch issue, we propose to augment the memory controller with simple modification so data can be transferred based on an asynchronous timing mechanism. Moreover, we propose a coordination scheme for the NVDIMM controller to intelligently manage the flash page-level read/write buffers with prefetching and pre-eviction so data requested by the memory controller can be prepared in advance. This is achieved with the assist of a novel cross-layer scheduling strategy that can group and reorder transactions in the NVDIMM transaction queue in the memory controller. To mitigate the lengthy GC period, a proactive GC scheme is proposed so the NVDIMM controller can judiciously transfer data that will be blocked in a forthcoming GC operation. Thus, the lengthy garbage collection overhead can be removed from the critical path of read/write operations.

We have conducted experiments with various big data benchmarks on a Marssx86 [26] driven full simulation environment and demonstrated the effectiveness of the proposed schemes from various aspects. Our experimental results show that overall the proposed techniques yield 10%∼35% performance improvements over the state-of-the-art baseline schemes [27], [28], [29], [30].

The main contributions of this paper are:

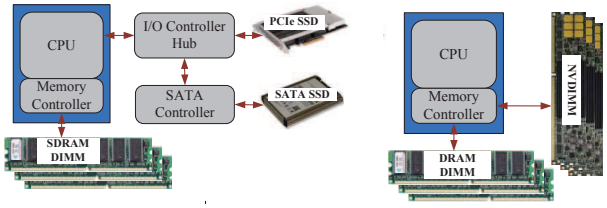
- To the best of our knowledge, this is the first work to study the I/O-derived memory traffic mixed with native memory traffic in the NVDIMM technologies.
- We propose a harmonic memory scheduling policy that can effectively solve the interference problem caused by heterogeneous memory traffic by exploiting data access criticality and resource utilization.
- We propose a novel coordination mechanism that can intelligently pass the information of data requests in the transaction queue to the NVDIMM controller in advance without explicitly transferring extra address information. This helps greatly alleviate the speed mismatch problem as data requested can be prepared beforehand in the read/write buffer in the NVDIMM controller.
- We for the first time propose a proactive garbage collection scheme that can effectively eliminate the lengthy garbage collection effect from the critical path.

The rest of this paper is organized as follows: Sections II and III present the background and the motivation, respectively. Section IV characterizes the mixed I/O and DRAM related memory traffic and their implications. Section V presents the proposed techniques. The experimental results are presented and discussed in Section VI. In Section VII, we conclude the paper.

## II. Background

### A. System Architecture

In a system equipped with traditional I/O storage architectures, storage devices are connected via the I/O controller hub with low speed interfaces, such as SATA (serial ATA) and PCIe (PCI Express). On the contrary, NVDIMM modules are installed into DDR slots and transfer data through fast memory channels. Fig. 2 illustrates the differences in data access between the NVDIMM and the traditional storage organization, such as PCIe SSD and SATA disk.




(a) Traditional I/O storage. (b) NVDIMM.

Fig. 2. The system architecture with (a) traditional I/O storage, and (b) NVDIMM.

TABLE I
THE I/O LATENCY FOR DIFFERENT ARCHITECTURES [1].

| Attributes | NVDIMM | PCIe SSD | SATA SSD |
|---|---|---|---|
| Interface | DDR3 1600 MHz | PCIe2.0 X 8 | SATA 6 Gbps |
| Read latency | ∼150 $us$ | ∼400 $us$ | ∼800 $us$ |
| Write latency | ∼4.66 $us$ | ∼15 $us$ | ∼65 $us$ |

Table I summarizes the I/O latency with the NVDIMM and the traditional storage architectures. Using NAND flash storage media, NVDIMM, such as IBM eXFlash 400GB DDR3 Storage DIMM devices, can provide ∼5 us write latency and ∼150 us read latency with the 1600 MHz operation frequency [1]. The NVDIMM technique also manifests good capacity scalability and significant improvement in implementation cost, power/cooling save, and floor space compared with conventional SSDs [31], [32], [33]. All these attractive features make NVDIMM a promising solution to replace the traditional storage devices.

### B. Utilizing NVDIMM with Different Hardware/Software Methods

*1) Hardware:* In general, there are two possible approaches to integrate NVDIMM modules with DRAM-based DIMMs.

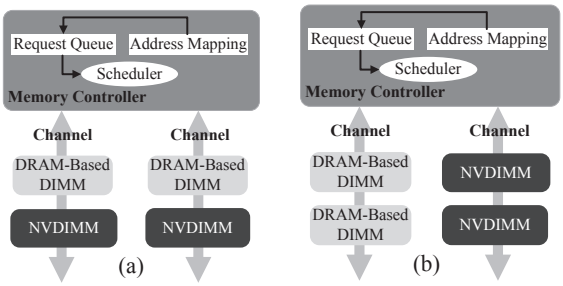



Fig. 3. Two NVDIMM placement approaches.

As shown in Fig. 3 (a), the first method is to place NVDIMMs and DRAM-based DIMMs in the same memory channel. The second method is to allocate them into different memory channels as shown in Fig. 3 (b). One disadvantage of the second approach is that channel resources cannot be fully utilized, as NVDIMMs are slower than DRAM-based DIMMs. Therefore, the first memory storage architecture is generally adopted in industries, such as in the IBM exFlash DIMM technique [34], [35]. Thus, the architecture in Fig. 3 (a) is adopted in this paper.

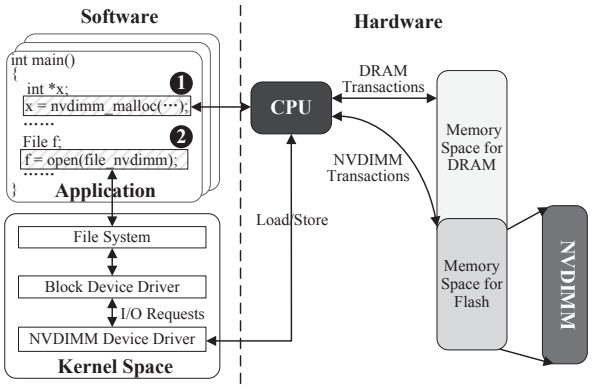



Fig. 4. Two different software management methods to utilize NVDIMM resources.

*2) Software:* Similarly, there are two possible methods to utilize the NVDIMM devices at the software level. In the first method, applications are aware of NVDIMM devices and can utilize them through the memory space they provide. On the contrary, the second method is to transparently utilize NVDIMM techniques through a block I/O device driver interface.

To support the first method, an operating system (OS) should provide a new interface to applications, such as *nvdimm _malloc(...)* as shown in Fig. 4. This method also requires that software developers modify their programs in order to utilize NVDIMMs. For the second method, a NVDIMM device driver can be added into the OS kernel to convert I/O requests to memory requests, so application programs do not need to be changed. In this paper, we adopt the second method so applications can directly utilize NVDIMM devices without any modification.

## III. Motivation

In this section, we illustrate the interference impact introduced by the mixed NVDIMM and DRAM DIMM traffic and the speed mismatch issue caused by fast memory access and slow flash read/write operations in NVDIMM.

We conduct experiments based on Hibench [36] and SPEC CPU2006 [37]. Hibench is a representative big data benchmark suite and for the illustration purpose, typical big data applications, namely, *bayes*, *dfsioe_r*, etc., are selected. In SPEC CPU2006, *429.mcf* is chosen as a representative for memory-intensive applications. All results are obtained by concurrently running *429.mcf* with one of the big data applications on an integrated simulation environment consisting of Marssx86 [26] (the CPU simulator), DRAMSim2 [38] (the DRAM DIMM simulator), and a NVDIMM simulator we develop based on NANDFlashSim [39] (the flash simulator). For each pair of *429.mcf* and one big data application, both offline test with a single node and online test with multiple slave nodes are conducted. The default NVDIMM buffer cache is configured to 400MB with the LRU algorithm [40]. The normalized speedup is adopted to quantitatively analyze the system performance [30]. We first obtain the instruction throughput that is the number of executed instructions per cycle for each workload, and then calculate the average instruction throughput across all the workloads. The normalized speedup is obtained by dividing the instruction throughput with the average throughput.

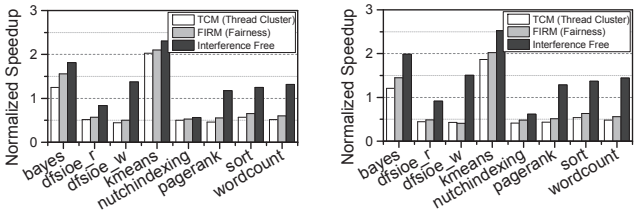
TABLE II
The Queue Division Effect.

| Environment | Workloads | Speedup |
|---|---|---|
| Offline test with a single node | Memory intensive workload (429.mcf) | 91.96% |
| | Big data workloads (others) | 9.26% |
| Online test with three slave nodes | Memory intensive workload (429.mcf) | 89.68% |
| | Big data workloads (others) | 9.08% |

We first analyze the queue division effect. There are two scenarios. "Unified Queue" is the scenario in which all NVDIMM and DRAM DIMM traffic is fed into one queue, while "Split Queue" is one in which NVDIMM and DRAM DIMM traffic is separated into two different queues in DRAMSim2. In Table II, it is observed that the queue division scheme has a significant impact on the performance of *429.mcf*, while it only slightly affects the big data applications. The main reason is that within one queue, since the processing of I/O-derived memory requests is slower than that of native memory requests, the I/O-derived traffic generally occupies the queue and the native memory requests are blocked.

An interference-free scheme and two state-of-the-art memory scheduling schemes, TCM [28] and FIRM [30], are utilized to show the performance degradation introduced by the bus contention. In the interference-free scheme, we assume an oracle case without any bus conflicts. TCM, as an optimized memory scheduling scheme, can dynamically classify applications into two clusters, namely low and high memory-intensity, and employs heterogeneous scheduling policies across the clusters to optimize for both system throughput and fairness.

FIRM is a memory scheduling policy recently proposed for persistent memory systems with the consideration of the bus turnaround overhead. Each of the three schemes is implemented as the memory scheduling mechanism to manage the unified queue with both NVDIMM and DRAM DIMM traffic in DRASMSim2.




(a) Offline test with a single node. (b) Online test with three slave nodes.

Fig. 5. Performance of the TCM [28], FIRM [30] and, interference free schemes.

In Fig. 5(a), we observe that the performance of the interference-free scheme outperforms that of the other schemes. The average performance improvements of the interference free scheme are 69% and 51%, respectively, compared with the TCM and FIRM. The similar results can be observed in Fig. 5 (b). These results exhibit that even the state-of-the art memory scheduling schemes cannot handle the mixed NVDIMM and DRAM DIMM traffic. Therefore, new scheduling policies need to be developed.

In summary, the above analysis implies that directly placing NVDIMM on the memory bus could not fully utilize the NVDIMM resources. Next, we perform detailed investigation on the underlying characteristics of mixed NVDIMM and DRAM DIMM memory traffic.

## IV. Characterization

In this section, we first present the state-of-the-art memory controller design with DRAM DIMMs and NVDIMMs. Then, we characterize the heterogeneous memory traffic.

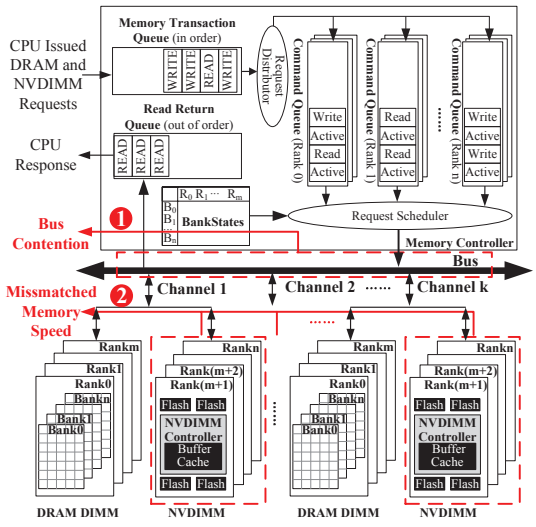



Fig. 6. The memory controller with DRAM DIMMs and NVDIMMs.

Fig. 6 elaborates the system architecture of the memory controller with NVDIMMs and DRAM DIMMs. The memory controller is equipped with one memory transaction queue and multiple command queues. The memory controller accepts requests (i.e. DRAM and NVDIMM transactions) from CPU and processes them sequentially. Once a transaction wins arbitration, it is decomposed into a sequence of memory commands and mapped to a command queue. The command queues are arranged in such a way that there is one queue per bank or per rank. Then, commands are scheduled to the memory devices through the signaling interface depending on the command scheduling policy. NVDIMMs bring several new challenges for this architecture. The first issue is the bus contention introduced by NVDIMM. Second, it is a challenging task to match the slow asynchronous NAND flash access speed with the fast synchronous memory controller speed. We utilize representative big data workloads to further analyze the mixed NVDIMM and DRAM DIMM memory traffic. To this end, we modify memory controller in DRAMSim2 to monitor the status of memory transaction queue on the fly.

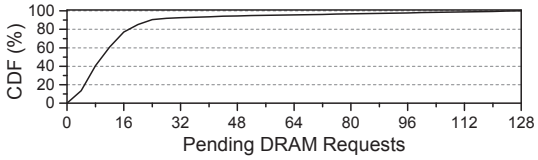



Fig. 7. The CDF of DRAM requests in the memory transaction queue with 429.mcf and dfsioe_r.

To characterize the bus contention, we conduct experiments by concurrently executing 429.mcf with *dfsioe_r* in the offline mode with a single node. The memory transaction queue depth is configured to 128. Fig. 7 shows the cumulative distribution function (CDF) of the total number of DRAM requests in the queue at each moment during the whole execution period. It can be observed that there are over 90% cases when the number of NVDIMM transactions is larger than 104 (the number of DRAM transaction is less than 24) in the queue. This manifests slow I/O-derived memory traffic prevents fast native memory traffic from entering into the queue. Note that this issue may not occur in design where NVM (e.g. PCM) is used as the main memory. In such hybrid main memory system, the memory requests can be either placed in PCM DIMM or DRAM DIMM. In case that the PCM DIMM (with high write latency) is blocked, memory requests can be placed or migrated to the DRAM DIMM. Compared with PCM DIMM, the flash DIMM is used as massive storage, which is capable of providing up to 400GB space [1]. The I/O requests to the flash DIMM could not be migrated or placed to the DRAM DIMM even if the flash DIMM is blocked. As a result, the slow I/O requests will wait in the transaction queue and can block the upcoming DRAM requests within the same transition queue.

To examine queue utilization with different workloads, we run the *dfsioe_r* application and the *sort* application in the offline mode with a single node, respectively. Fig. 8 (a) shows

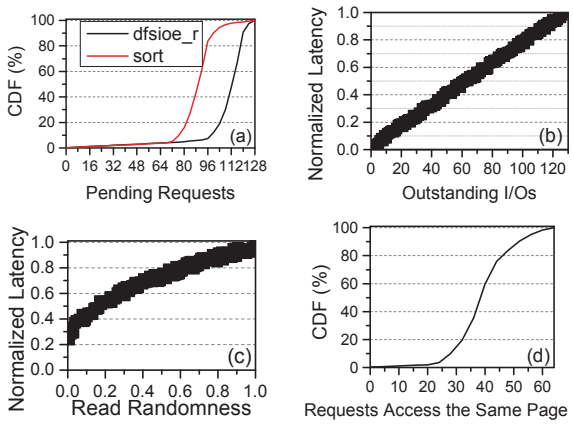



Fig. 8. The resource utilization and performance variation.

the CDF functions of the queue depth for *dfsioe_r* and *sort*. It can be observed that *dfsioe_r* demonstrates more intensive queue utilization. For instance, there are over 80% cases in *dfsioe_r* when the queue depth is over 104, while less than 5% cases in *sort*. With more transactions in the queue, we should adopt more aggressive scheduling schemes if resources such as channels are available.

We further develop several synthetic traces based on *dfsioe_r* and *sort* big data applications to analyze the effects of outstanding I/Os and read randomness. Figures 8 (b) and (c) show the results obtained by directly running these traces on DRAMSim2. The I/O latencies arise with the number of outstanding I/O operations increase, as shown in Fig. 8 (b). Fig. 8 (c) reveals that the more random the read operations are, the worse the I/O latencies become. These results motivate us to dynamically schedule channel resources according to workload characteristics.

Fig. 8 (d) manifests that a lot of I/O-derived memory transactions (by combining all I/O-derived memory traffics of *dfsioe_r*, *dfsioe_w*, and *sort* together) access the same NAND flash page and result in good spatial locality in general. This is because the block I/O requests with large size (i.e. 4 KB) are transmitted to the I/O-derived memory traffic with small-sized memory transactions (i.e. 64 bytes). Therefore, if we can identify those requests for the same NAND flash page in NVDIMM buffer cache management, the hit ratio can be greatly improved.

## V. The Proposed Schemes

In this section, we propose several schemes to address the workload heterogeneity challenge, the speed mismatch challenge, and the lengthy GC period in Sections V-A, V-B and V-C, respectively.

### A. Harmonic Memory Scheduling in Memory Controller

Since slow NVDIMM requests will block fast DRAM requests in the same memory transaction queue, we propose to divide the memory transaction queue into two parts: DRAM transaction queue and NVDIMM transaction queue. By placing different requests into different queues, we can avoid the interference problem due to mixed heterogeneous memory traffic. However, this raises a new challenge, namely, how to schedule requests from the two queues to minimize the average access latency.

To address this issue, we propose a novel bus arbitrator to schedule the NVDIMM and DRAM DIMM memory traffic. The bus arbitrator monitors the channel states (busy or idle) and records the issued request states (ready or waiting). In case the same kind of requests, such as NVDIMM requests or DRAM DIMM requests, are ready, the bus arbitrator schedules these requests based on the request issued time. The earlier issued request will be scheduled prior to the later ones within the same kind of requests. On the other hand, when two different kinds of requests, one NVDIMM request and one DRAM DIMM request, are ready, the bus arbitrator will arbitrate the channel resources according to the harmonic bus scheduling policy. The policy is based on the DRAM and NVDIMM bus scheduling ratio, $SR_d$ and $SR_n$. If $SR_d > SR_n$, the bus arbitrator will schedule the DRAM request, vice versa. $SR_d$ and $SR_n$ can be calculated as follows:

$$SR_d = w(\frac{P_d}{P_{d\_max}} - \frac{P_n}{P_{n\_max}}) + (1-w)(\frac{P_{dq}}{L_{dq}} - \frac{P_{nq}}{L_{nq}}), \quad (1)$$

$$SR_n = -SR_d, \quad (2)$$

where $0 \leq w \leq 1$, $L_{dq} \neq 0$, $L_{nq} \neq 0$. Here, $P_d$/$P_n$ stands for the performance of DRAM/NVDIMM. $P_d\_max$/$P_n\_max$ is used to represent the maximum access speed of DRAM/NVDIMM. The ratio of $P_d$ to $P_d\_max$ ($P_n$ to $P_n\_max$) stands for the performance criticality of DRAM requests (NVDIMM requests). $P_{dq}$/$P_{nq}$ represents the number of pending requests in the DRAM/NVDIMM transaction queue. $L_{dq}$/$L_{nq}$ is the length of the DRAM/NVDIMM transaction queue. The ratio of $P_{dq}$ to $L_{dq}$ ($P_{nq}$ to $L_{nq}$) represents the utilization ratio of DRAM (NVDIMM) transaction queue.

The key issue is to predict the performance of NVDIMM and DRAM DIMM according to different workloads. To this end, a black-box model is used to predict the performance as a function ($f$) of workload characteristics. This model takes the workload characteristics ($WC$) as input parameters and outputs the predicted performance metric ($P$), as follows:

$$P_d = f(WC_d), \quad (3)$$

$$WC_d =< q\_dep, stride, reuse > . \quad (4)$$

The performance of DRAM DIMM can be represented in Equations 3. To analyze applications for memory locality, we present effective and measurable metrics for quantifying the outstanding requests ($q\_dep$), spatial locality ($stride$), and temporal locality ($reuse$) as described in Equation 4.

$$P_n = f(WC_n), \quad (5)$$

$$WC_n =< wr\_ratio, q\_dep, wr\_rand, rd\_rand > . \quad (6)$$

The performance of NVDIMM can be calculated through Equation 5, and the performance is characterized with four factors as shown in Equation 6. The workload can be characterized by the read and write ratio ($wr\_ratio$) that is defined as the percentage of writes among all requests. $q\_dep$ represents the number of outstanding NVDIMM requests in the queue.

$rd\_rand/wr\_rand$ represents the percentage of read/write random accesses in the I/O request stream.

Directly implementing the above performance models will incur noticeable computation overheads. However, the memory controller has the limited computational capacity. To solve this discrepancy, we use the approximating computing approach to implement the proposed algorithms. Specially, we divide the performance and queue utilization into several stages, and use different thresholds to replace the division operations in Equation 1. A mapping table is then used with the number of pending requests in the transaction queue as the input, and the output is the normalized queue utilization. Another mapping table is employed with the data characteristics as the input, and the output is the normalized performance stage. By comparing the normalized queue utilization and performance stage, the harmonic scheduling scheme can determine the performance criticality and the resource utilization status. Then, the scheduling ratio can be obtained.

Assuming each characteristic has $M$ thresholds and the queue utilization is divided into $N$ stages, the memory overhead can be calculated as $(4 \times N + 4 \times M^3 + 5 \times M^4) \times S$, where S is the size of an entry in the mapping table. In practice, we can achieve good results with very small N and M. For example, in our experiment, the queue utilization is divided into 6 stages. The thresholds of the $q\_dep$, $stride$, and $reuse$ are assigned to 6, 4, and 2, respectively. The thresholds of the $wr\_ratio$, $q\_dep$, $wr\_rand$, and $rd\_rand$ are configured to 6, 6, 4, and 4 , respectively. An entry in the mapping table occupies one byte. The memory overhead is $(4 \times 6 + 4 \times 6 \times 4 \times 2 + 5 \times 6^2 \times 4^2) = 3096$ Bytes.

### *B. Asynchronous Data Transfer and Intelligent Buffer Management*

In this section, we first present an asynchronous timing mechanism for data transfer by augmenting the memory controller with simple modification. We then propose a coordination scheme for the NVDIMM controller to intelligently manage the flash page-level read/write buffers with prefetching and pre-eviction so data requested by the memory controller can be prepared in advance.

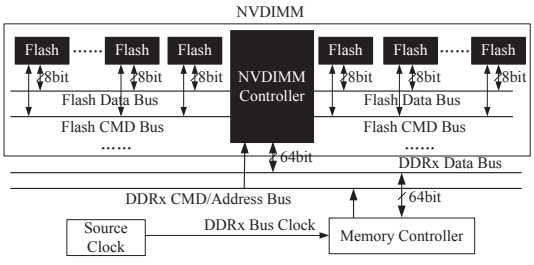



Fig. 9. NVDIMM structure.

*1) Data Access Pattern with NVDIMM:* Figures 9 and 10 illustrate the structures of a NVDIMM module and the NVDIMM controller, respectively. The state-of-the-art memory controller employs the synchronization timing sequence to access data. Due to slow, asynchronous data transfer from the flash memory, the traditional synchronization timing sequences could not well satisfy the NVDIMM data access pattern.

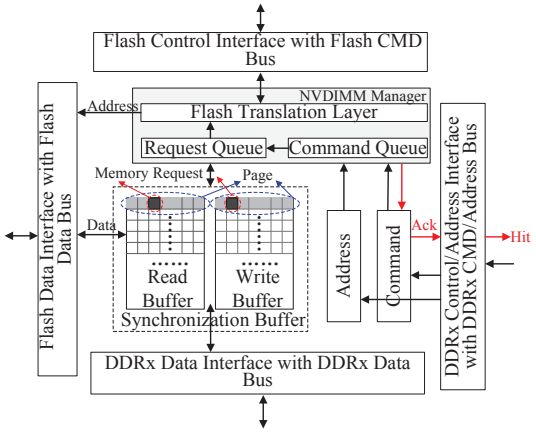
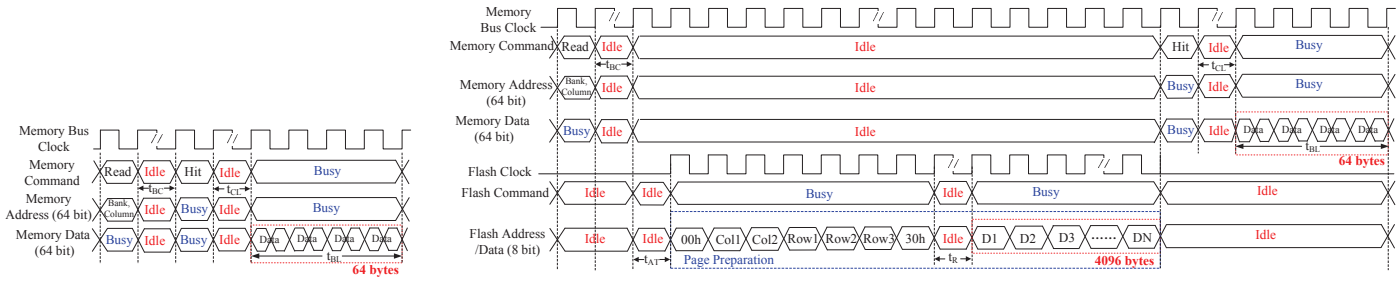



Fig. 10. NVDIMM controller.

To solve this issue, we propose to augment the traditional memory controller by adding a new *ack* command signal line as shown in Fig. 10. When the accessed data are stored in the synchronization buffer or the buffer has space to store new data, the *ack* command line will return a hit signal or high voltage to the memory controller. Meanwhile, the address lines will return the accessed request address. Based on the request address, the memory controller can detect the request type (read or write request), and thus can further determine whether the request data is ready or the synchronization buffer has space to accept the write data. Note that the *ack* signal is augmented to the memory controller and does not affect the compatibilities of the current DRAM DIMMs. This design can use one of the multiple NC (Not Connected) pins in the current DIMM slots to connect the *ack* signal. For example, the 79 pin in the RDIMM is a NC pin [41], and the *ack* signal can be connected to this pin. With this simple enhancement, the memory controller can accurately predict the accessed data storage locality and adopt different access timing sequences.

We also design a synchronization buffer in the NVDIMM controller as shown in Fig. 10. The synchronization buffer not only helps match the different bus speeds, but also convert requests with different sizes and connect different hardware interfaces. For instance, data read from or written to NAND flash memory are performed on a page basis (one page contains multiple bytes, i.e. 4096 bytes), while the memory controller issues memory requests according to the cache line size (i.e. 64 bytes). Thus, the synchronization buffer serves an interface for data assembly and disassembly.

We use the read operation as an example to show the timing sequences. As shown in Figures 11 (a) and (b), a read command with the memory bank and column address is first delivered from the memory controller to NVDIMM with one memory clock cycle. After the NVDIMM controller receives the read command from the memory controller, the NVDIMM controller will consult the flash translation layer (FTL) [42], [43] about the data locality with the $t_{BC}$ waiting




(a) Data in buffer cache. (b) Data in flash memory.

Fig. 11. The read timing sequence.

time. Then, the flash translation layer checks the data locality by comparing the translated address with the page cache address. If the data are stored in flash memory, the flash memory read operations is triggered as shown in Fig. 11 (b). $t_{AT}$ represents the flash chip activation time, and $t_R$ is the data preparation time from the flash cells to the buffer inside NAND flash memory. When the accessed data are ready in the synchronization buffer, the NVDIMM controller notifies the memory controller via the proposed *ack* command signal line. Finally, after $t_{CL}$ waiting time, data are transferred from NVDIMM to the memory controller during the $t_{BL}$ time period.

*2) Intelligent Buffer Management:* To maximally utilize the synchronization buffer, we should not only improve the buffer hit ratio but also prefetch or pre-evict data based on data access patterns by exploiting parallel channels in NAND flash. Although I/O-derived memory traffic manifests strong spatial locality as shown in Fig. 8 (d), it is challenging to pass the information of data requests in the NVDIMM transaction queue to the NVDIMM controller without explicitly transferring extra address information.

To address this issue, we propose a novel coordination scheme to orchestrate the NVDIMM controller and the memory controller. On the memory controller side, we propose a page-aware scheduling (PAS) strategy that groups and reorders transactions in the NVDIMM transaction queue. With this strategy, we aim to strike a balance by adaptively prioritizing the scheduling of two groups of requests. Basically, the requests in the same flash page are used to assist the NVDIMM controller for improving the hit ratio, while the requests from different flash pages are for prefetching or pre-evicting by exploiting flash channel parallelism. Correspondingly, on the NVDIMM controller side, by monitoring the address information of the requests in the command queue, data can be either prefetched or pre-evicted so data or space can be prepared in advance in the read and write buffers.

With our page-aware scheduling strategy, in the beginning, the scheduling of the requests in different flash pages has higher priority compared with the requests in the same flash page. When scheduling the requests in different flash pages, we adopt a round-robin approach so each time one request from one page will be scheduled in a first-in-first-out (FIFO) manner. Since the I/O-derived memory traffic usually demonstrates strong spatial locality, in this way, the NVDIMM controller can maximally exploit channel parallelism while maintaining high hit ratios. However, due to the limited buffer size, buffer miss may occur. In that case, the response latency will be significantly degraded compared with the cases with buffer hit. Therefore, this can be detected in the memory controller and at that moment, we convert to schedule the requests in the same flash page in the FIFO manner.

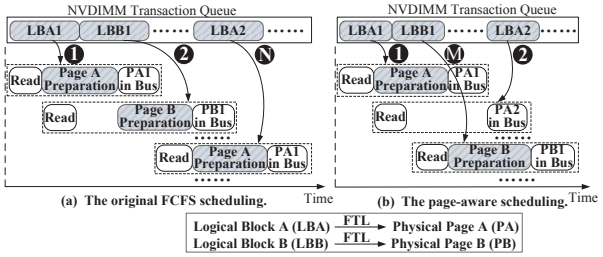



Fig. 12. Comparison of two schedules generated by FCFS and PAS in which the requests in the same page are scheduled.

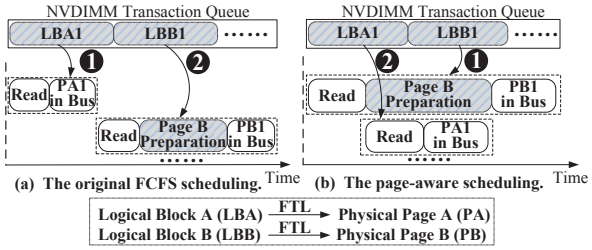



Fig. 13. Comparison of two schedules generated by FCFS and PAS in which the requests in different pages are scheduled.

Figures 12 and 13 show two examples to compare our page-aware scheduling strategy with the First-Come-First-Serve (FCFS) scheduling policy [44]. Fig. 12 (b) illustrates the case when the page-aware scheduling strategy is in the process of scheduling the requests in the same flash page. Assume $N$ $(N > 2)$ memory read requests are issued to NVDIMM, and LBA1 and LBA2 belong to the same physical page A. FCFS schedules requests in a FIFO manner according to the requests arrival time. Suppose that all the other $(N-2)$ requests from LBA1 to LBA2 do not access the physical page A, and the read buffer is small so it has to evict page A. Then when LBA2 is scheduled, as shown in Fig. 12 (a), page A needs to be fetched again from the NAND flash. On the other hand, since LBA2 and LBA2 are in the same flash page, LBA2 will be scheduled earlier; thus, as shown in Fig. 12 (b), page A in the read buffer can be reused and the hit ratio can be improved.

Fig. 13 (b) illustrates the benefit by scheduling the requests in different pages. Assume LBA1 and LBB1 belong to two physical flash pages, A and B, respectively. Suppose that page A has been cached in the read buffer in the NVDIMM controller. Using FCFS, as shown in Fig. 13 (a), after LBA1 has been finished, LBB1 will be scheduled. However, if the data are not in the buffer, we have to wait until the data are ready. On the contrary, with our page-aware scheduling strategy, when entering the process of scheduling the requests in different flash pages, LBB1 will be scheduled earlier than LBA1 as shown in Fig. 13 (b). As a result, the memory bus can utilize the LBB1 data preparation time to transfer the data of LBA1. Furthermore, the waiting time of LBB1 can be reduced compared with FCFS.

With the above scheduling techniques, the order of requests may be changed. This may result in the data inconsistence issue when NVDIMM suffers the power loss or other interruptions. Note that NVDIMM is used as the storage and communicates with the file system. Most modern file systems (e.g. the ext4 file system) equip journaling mechanism to ensure file system consistency. Similar to the I/O scheduling in the block device driver, the lost data (due to power loss or other interruptions) without being responded by NVDIMM will not affect the file system consistency. In case of a power failure and there exists non-persistent data in write buffer, the NVDIMM can flush these data to the flash storage with the help of the ultra-capacitor or battery.

### C. Proactive Garbage Collection Design

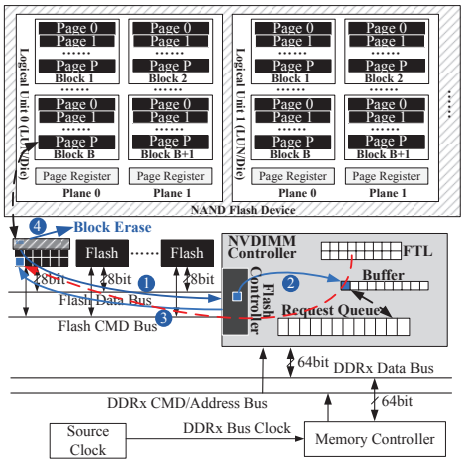



Fig. 14. The proactive garbage collection.

The garbage collection operation usually contains two stages, including valid page copies and block erases. For the first stage, the NVDIMM controller copies the valid pages from flash memory, and then writes valid pages to free pages. Traditional designs will discard the copied pages after the pages are written back to flash memory. The proactive garbage collection caches future accessed pages in the synchronization buffer according to the pending request information as the second step shown in Fig. 14. This can reduce the data transfer between the flash memory and the NVDIMM controller, and thus improve the performance of NVDIMM. In addition, the valid page copies will block the flash access bus, resulting in all the flash chips in the same flash bus being blocked. The proactive garbage collection will coordinate the flash bus traffic and memory bus traffic to reduce the valid page copy overhead. The NVDIMM response time can be represented as follows:

$$T_{nvdimm\_response\_c} = N_{buffer} \times T_{memory\_bus} + N_{pending\_c\_r} \times T_{flash\_r} + N_{pending\_c\_w} \times T_{flash\_w}. \quad (7)$$

$T_{nvdimm\_response\_c}$ denotes the effective NVDIMM response time without being interrupted by the valid page copy operation. $N_{buffer}$ is the number of effective requests in the buffer cache, which can directly respond to the memory controller. $N_{pending\_c\_r}$ and $N_{pending\_c\_w}$ represent the number of effective read and write requests to the other flash channels, respectively. $T_{memory\_bus}$ represents the time period of transferring one data request in the memory bus. $T_{flash\_r}$ and $T_{flash\_w}$ stand for the time period of storing one page data from the memory controller and reading one page data from NAND flash memory, respectively.

The valid page copy operation can be refined into several steps, and only one page data are transferred between the memory controller and flash memory for each step. If $T_{nvdimm\_response\_c}$ is larger than the blocked access latency or no request accesses data via the blocked flash channel, the valid page copy operation can be triggered without introducing the timing overhead. Otherwise, the proactive garbage collection will first respond the requests in the blocked channel.

For the second stage, the block erase operation will block the response of the logical unit (LUN) in a flash chip. A logical unit (LUN) is the minimum unit that can independently execute commands and report status [45]. In order to eliminate the block erase effect, the pending read requests will be prefetched to the synchronization buffer prior to the block erase operation.

$$T_{lun\_block} = T_{block\_e}. \quad (8)$$

$$T_{nvdimm\_response\_l} = N_{buffer} \times T_{memory\_bus} + N_{pending\_l\_r} \times T_{flash\_r} + N_{pending\_l\_w} \times T_{flash\_w}. \quad (9)$$

$$N_{pre} = \frac{T_{lun\_block} - T_{nvdimm\_response\_l}}{T_{memory\_bus}}. \quad (10)$$

$T_{lun\_block}$ is the timing overhead caused by the block erase operation. $T_{nvdimm\_response\_l}$ denotes the effective NVDIMM response time without being blocked by the block erase operation. $N_{pending\_l\_r}$ and $N_{pending\_l\_w}$ represent the number of read and write requests without being blocked by the block erase operation, respectively. The number of prefeched request can be calculated in Equation 10. If $T_{nvdimm\_response\_l} \geq T_{lun\_block}$ , the block erase operation can be directly triggered without introducing the timing overhead. Otherwise, at least $N_{pre}$ requests data are needed to be prefetched from the blocked LUN.

Note that the proposed proactive garbage collection scheme does not introduce endurance degradation. This scheme is triggered when a block has limited available spaces or very few

free spaces in NVDIMM (i.e. the similar triggering condition compared with most of the state-of-the-art garbage collection schemes). However, the proactive garbage collection selects a more appropriate time to reclaim these spaces with the minimum performance overhead to the NVIDMM response. Therefore, the proposed scheme yields similar endurance compared with the existing methods.

## VI. Evaluation

### A. Experimental Setup

We build up a simulation framework by integrating Marssx86 [26], DRAMSim2 [38], and NANDFlashSim [39]. Marssx86 is a cycle-accurate full-system simulator that uses PTLsim [46] for CPU simulation on top of the QEMU emulator [47]. DRAMSim2 is a cycle-accurate simulator of the main-memory system. Based on NANDFlashSim, we develop a NVDIMM simulator and adopt the page level FTL [42] in the NVDIMM controller. In order to utilize the NVDIMM resources, the PMBD [48] device driver is integrated into the Linux kernel 3.2.1. The PMBD device driver registers as a standard block device with configurable block capacity and can translate I/O requests to load/store memory access instructions. After the block device is mounted to a specific directory, applications and benchmarks can access the NVDIMM device via general purpose file operations, such as *read()*.

TABLE III
THE SYSTEM CONFIGURATION.

| | |
|---|---|
| CPU | 4-core, 2 GHz, 4 issue, out-of-order |
| L1 I/D | 32KB/32KB, 4-way, private, 64B line |
| L2 | 2MB, 4-way, private, 64B line |
| L3 | 8MB, 4-way, private, 64B line |
| Memory Controller | 4 memory channels, 128 DRAM DIMM transaction queue depth, 128 NVDIMM transaction queue depth |
| DRAM DIMM | 8 GB DDR3-1600 chips, 4 ranks of 8 banks each, 13.75ns time period taken from the active command to the read/write command, 18.75ns time period taken from the read/write command to the precharge command, 13.75ns time taken for the precharge operation, 64ms refresh period, 110ns refresh time period for each row |
| NVDIMM | 32 GB, 2-bit MLC, 4 flash channels of 2 NAND flash chips each channel, 128 pages per block, 4KB page size, 50us page read latency, 650us page write latency, 2ms block erase latency, 52ns synchronization buffer access latency, 4096 request queue depth, 4096 command queue depth |

Table III presents the configurations of the simulated four-core CMP system. The memory controller connects with 4 memory bus channels, and each channel contains one DRAM DIMM and one NVDIMM. The DRAM DIMM and NVDIMM queue depth is configured to 128. The DRAM DIMM main memory consists of 8 GB in total and is organized as four ranks (each rank contains eight banks). Each rank has a 32-entry command queue in the memory controller to buffer pending command requests. The total capacity of NVDIMM is configured to 32GB with 4 flash channels.

Table IV summarizes the evaluated workload combinations that we use to evaluate the NVDIMM system. We combine each big data workload (i.e. bayes, dfsioe_r ) to one of the three SPEC CPU2006 [37] workloads, namely, 429.mcf, 470.lbm, and 433.milc. The three SPEC CPU2006 workloads are chosen based on memory intensity. The RPKI and WPKI (i.e. main-memory reads/writes per kilo instructions) of the workloads are listed in Table IV. In our evaluation, we adopt both offline test mode (with one node) and online test mode (with one master node and multiple slave nodes). Particulary, three slave nodes are adopted to evaluate the proposed schemes. We demonstrate the effectiveness of the proposed scheme by comparing it with various representative memory scheduling mechanisms, including PAR-BS [27], TCM [28], NVMDuet [29], and FIRM [30].

TABLE IV
THE CONFIGURATION OF THE MIXED WORKLOADS.

| **Benchmarks** | **Description** | | |
|---|---|---|---|
| bayes | 100,000 pages, 100 classes | | |
| dfsioe_r | 2,500 files, 10MB file size | | |
| dfsioe_w | 2,500 files, 10MB file size | | |
| kmeans | 300,000 samples, 20 dimensions | | |
| nutchindexing | 100,000 pages | | |
| pagerank | 500,000 pages | | |
| sort | 2,400,000 data size | | |
| wordcount | 3,200,000 data size | | |
| | **RPKI** | **WPKI** | **WPKI/PRKI** |
| 429.mcf | 40.58 | 15.42 | 38.00% |
| 470.lbm | 22.68 | 13.28 | 58.55% |
| 433.milc | 1.82 | 1.44 | 79.12% |

### B. Experimental Results

*1) Harmonic Memory Scheduling:* Figures 15 (a) and (b) illustrate the normalized speedup of the offline and online test, respectively. The performance is improved by 13% on average compared with the baseline schemes in the offline test mode. The experimental results for the online test are similar to the offline test with an average performance improvement of 11%. We further vary the weight $w$ (a parameter in Equation 1) from 0.2 to 0.8 to evaluate the performance impact of the criticality and resource utilization. For the I/O intensive workloads, such as *dfsioe_r*, the performance improves from 15% to 22% as $w$ increases from 0.2 to 0.8. On the contrary, the performance gain reduces from 28% ($w = 0.2$) to 10% ($w = 0.8$) on *sort* workload, which exhibits computation intensive behavior. These results demonstrate that both the I/O access patterns and the transaction queue utilization play important roles in our harmonic memory scheduling (HMS) technique. We setup the value of the weight $w$ as 0.5 for the rest of evaluation. Figures 15 (c) and (d) show that bus traffic is reduced as the memory intensity of workload decreases. For instance, the average performance improvement from 13% to 4% when replacing the mixed workloads from *429.mcf* to *433.milc*. In the rest of this paper, we focus on studying the NVDIMM performance improvement to show the benefits of our techniques.

*2) Page-Aware Scheduling:* The NVDIMM response time is another important evaluation metric [49], [50]. In NVDIMM, the response time is mainly affected by the syn-

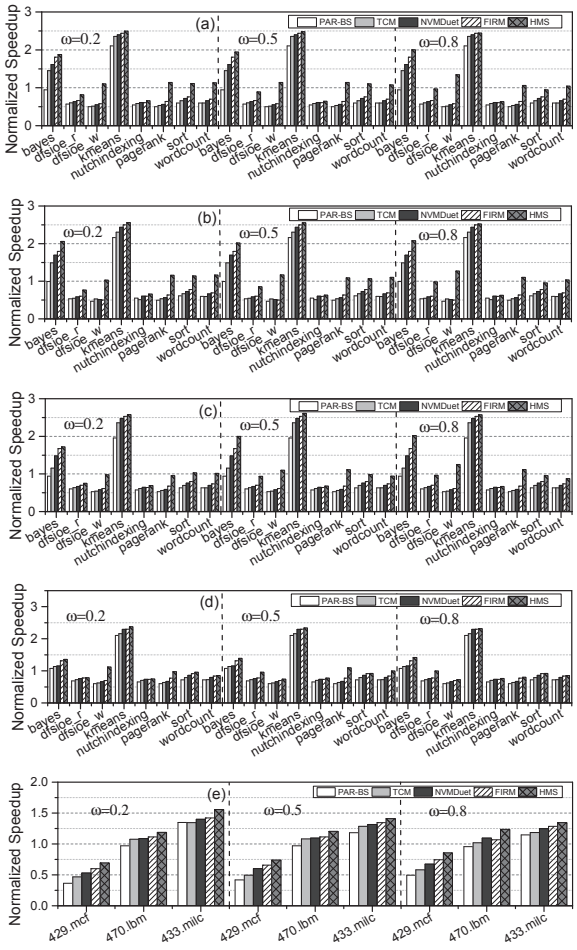



Fig. 15. The system performance of (a) the big data workloads mixed with 429.mcf (offline test with a single node), (b) the big data workloads mixed with 429.mcf (online test with multiple slave nodes), (c) the big data workloads mixed with 470.lbm (offline test with a single node), (d) the big data workloads mixed with 433.milc (offline test with a single node), and (e) the SPEC CPU2006 workloads mixed with the big data workloads (offline test with a single node).

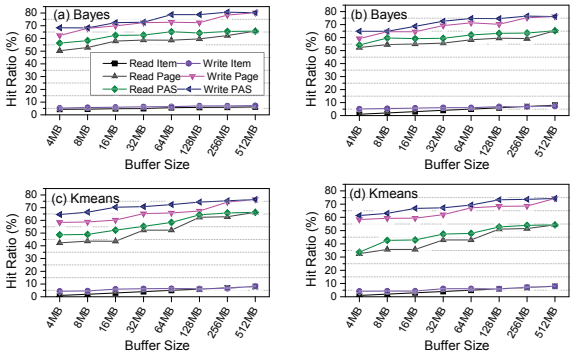



Fig. 16. The cache hit ratio with the offline ((a), (c)) and online ((b), (d)) tests.

chronization buffer, and the synchronization buffer hit ratio plays a vital role in determining the NVDIMM response time.

Fig. 16 illustrates the synchronization buffer cache hit ratio with different read and write buffer cache sizes. The item-based scheme represents the memory block size (64 bytes) based caching mechanism, while the page-based scheme is utilized to cache one flash page data. The experimental results in Fig. 16 show that with a constrained buffer size, the page-based scheme with the FCFS scheduling policy outperforms the item-based scheme with the same scheduling policy, and the page-based scheme with PAS even presents better cache hit ratio compared with the page-based scheme with the FCFS scheduling policy. Taking bayes as an example, the average read cache hit ratio for the page-based scheme with the FCFS scheduling policy is about 10.88 times and 10.75 times better than that of the item-based scheme with the FCFS scheduling policy, and the page-based scheme with PAS can further improve the read/write cache hit ratio by 5% and 4% on average, respectively, compared with the page-based scheme with the FCFS scheduling policy. The cache ratio improvement mainly benefits from the presence of page locality in memory transactions and the proposed page-aware scheduling policy.

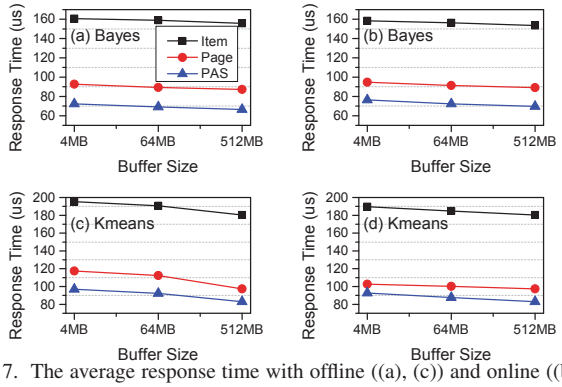



Fig. 17. The average response time with offline ((a), (c)) and online ((b), (d)) tests.

The cache hit ratio improvement can reduce the NVDIMM response time. The average system response time across different big data workloads is shown in Fig. 17. It can be observed that the average response time of NVDIMM can be greatly reduced with the page-based scheme (e.g. from $158.42us$ to $89.73us$ compared with the item-based scheme on bayes in the offline test). Moreover, the proposed PAS scheduling policy can achieve 10% performance improvement compared with the page-based scheme with the FCFS scheduling policy.

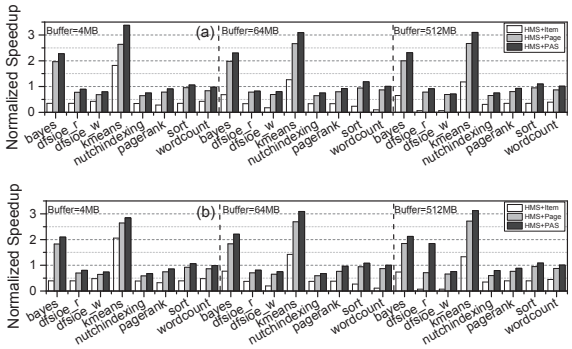



Fig. 18. The performance improvement with PAS. (a) Offline test with a single node. (b) Online test with multiple slave nodes.

The response time improvement can enhance the NVDIMM performance and eventually benefits the overall system performance. Figures 18 (a) and (b) plot the performance improvement with the offline and online tests, respectively.

When the buffer cache size is 4MB, the average performance improvement of the page-based scheme with PAS is 12% for both offline and online tests compared with the page-based scheme with the FCFS scheduling policy. With the buffer cache increased from 4MB to 512MB, the PAS performance improvement increases from 12% to 15% for both the offline and online cases.

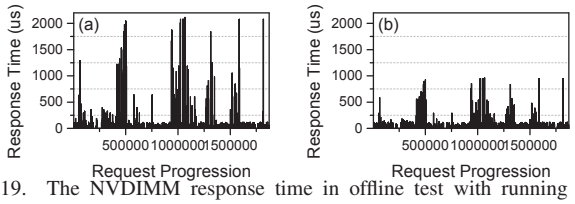



Fig. 19. The NVDIMM response time in offline test with running bayes (a) without proactive garbage collection, and (b) with proactive garbage collection.

*3) Proactive Garbage Collection:* Fig. 19 plots the proactive garbage collection effect with running bayes in the offline test mode. With requests continuously issued to NVDIMM, the worst-case response time can be effectively eliminated with the proactive garbage collection technique. For instance, when garbage collection occurs, the average NVDIMM response time can be reduced by 52%.

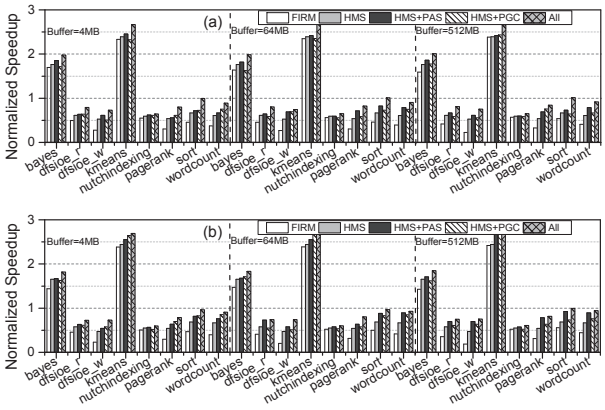



Fig. 20. The performance improvement by integrating all techniques. (a) Offline test with a single node. (b) Online test with multiple slave nodes.

*4) Putting It All Together:* The harmonic memory scheduling, page-aware scheduling, and proactive garbage collection schemes benefit the NVDIMM architecture from different aspects. Specifically, the harmonic memory scheduling scheme eliminates bus contention for heterogeneous DRAM DIMM and NVDIMM memory traffic while the page-aware scheduling and proactive garbage collection techniques reduce the NVDIMM response time to well match the fast memory bus speed. Therefore, combining these mechanisms can lead to greater performance gains. Figures 20 (a) and (b) demonstrate that the integrating of all proposed techniques significantly improves the performance by up to 35% (27% on average) compared with FIRM. Putting all techniques together also leads to a performance gain of 13% compared with only employing the proposed harmonic memory scheduling technique.

## VII. Conclusions

In this paper, we present a NVDIMM-based approach to solve the lengthy I/O speed problem for big data applications. We observe that naively adding NVDIMM to the memory channel yields suboptimal results due to bus contention and speed mismatching issues. In order to fully utilize NVDIMM, we propose three techniques, namely, the harmonic memory scheduling scheme, the page-aware scheduling policy, and the proactive garbage collection technique. Experiments based on full-system simulations, and big data workloads demonstrate that the proposed design achieves up to a 35% (27% on average) performance improvement over the baseline schemes. To the best of our knowledge, this is the first work to characterize and explore the optimizations for the mixed I/O-derived and native memory traffic in the light of emerging NVDIMM technologies.

## Acknowledgment

The work described in this paper is partially supported by the grants from the Research Grants Council of the Hong Kong Special Administrative Region, China (GRF 152138/14E and GRF 15222315/15E), National Natural Science Foundation of China (Project 61272103 and Project 61373049), National 863 Program 2013AA013202, the Hong Kong Polytechnic University (4-ZZD7,G-YK24, G-YM10, and G-YN36), and NSF Grants (1527535, 1423090, and 1320100).

## References

[1] IBM, "Benefits of IBM eXFlash memory-channel storage in enterprise solutions," *http://www.redbooks.ibm.com/abstracts/redp5089.html?Open* , 2015.

[2] A. Awad, B. Kettering, and Y. Solihin, "Non-volatile memory host controller interface performance analysis in high-performance I/O systems," in *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS '15)*, pp. 145–154, 2015.

[3] A. M. Caulfield and S. Swanson, "QuickSAN: A storage area network for fast, distributed, solid state disks," in *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA '13)*, pp. 464–474, 2013.

[4] O. Kocberber, B. Grot, J. Picorel, B. Falsafi, K. Lim, and P. Ranganathan, "Meet the walkers: Accelerating index traversals for in-memory databases," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '13)*, pp. 468–479, 2013.

[5] Z. Jia, J. Zhan, L. Wang, R. Han, S. A. McKee, Q. Yang, C. Luo, and J. Li, "Characterizing and subsetting big data workloads," in *2014 IEEE International Symposium on Workload Characterization (IISWC '14)*, pp. 191–201, 2014.

[6] H. Jeon, K. El Maghraoui, and G. B. Kandiraju, "Investigating hybrid ssd ftl schemes for hadoop workloads," in *Proceedings of the ACM International Conference on Computing Frontiers (CF '13)*, pp. 20:1–20:10, 2013.

[7] A. M. Caulfield, L. M. Grupp, and S. Swanson, "Gordon: Using flash memory to build fast, power-efficient clusters for data-intensive applications," in *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '09)*, pp. 217–228, 2009.

[8] IBM, "eXFlash DDR3 storage DIMMs," *http://www.redbooks.ibm.com /abstracts/tips1141.html*, 2015.

[9] F. Chen, D. A. Koufaty, and X. Zhang, "Understanding intrinsic characteristics and system implications of flash memory based solid state drives," in *Proceedings of the 11th International Joint Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '09)*, pp. 181–192, 2009.

[10] J. Coburn, T. Bunker, M. Schwarz, R. Gupta, and S. Swanson, "From ARIES to MARS: Transaction support for next-generation, solid-state drives," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP '13)*, pp. 197–212, 2013.

[11] Y. Zhang, L. P. Arulraj, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "De-indirection for flash-based SSDs with nameless writes," in *10th USENIX Conferen on File and Storage Technologies (FAST '12)*, pp. 1–16, 2012.

[12] T. Luo, S. Ma, R. Lee, X. Zhang, D. Liu, and L. Zhou, "S-CAVE: Effective SSD caching to improve virtual machine storage performance," in *Proceedings of the 22Nd International Conference on Parallel Architectures and Compilation Techniques (PACT '13)*, pp. 103–112, 2013.

[13] G. Sun, Y. Joo, Y. Chen, D. Niu, Y. Xie, Y. Chen, and H. Li, "A hybrid solid-state storage architecture for the performance, energy consumption, and lifetime improvement," in *2010 IEEE 16th International Symposium on High Performance Computer Architecture (HPCA '10)*, pp. 1–12, 2010.

[14] 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," in *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '11)*, pp. 105–118, 2011.

[15] W. Wei, D. Jiang, S. A. McKee, J. Xiong, and M. Chen, "Exploiting program semantics to place data in hybrid memory," in *2015 International Conference on Parallel Architecture and Compilation (PACT '15)*, pp. 163–173, 2015.

[16] A. Awad, P. Manadhata, S. Haber, Y. Solihin, and W. Horne, "Silent shredder: Zero-cost shredding for secure non-volatile main memory controllers," in *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '16)*, pp. 263–276, 2016.

[17] A. Kolli, S. Pelley, A. Saidi, P. M. Chen, and T. F. Wenisch, "High-performance transactions for persistent memories," in *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '16)*, pp. 399–411, 2016.

[18] R. Wang, L. Jiang, Y. Zhang, and J. Yang, "SD-PCM: Constructing reliable super dense phase change memory under write disturbance," in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '15)*, pp. 19–31, 2015.

[19] R. Wang, L. Jiang, Y. Zhang, L. Wang, and J. Yang, "Exploit imbalanced cell writes to mitigate write disturbance in dense phase change memory," in *Proceedings of the 52Nd Annual Design Automation Conference (DAC '15)*, pp. 88:1–88:6, 2015.

[20] X. Zhang, Y. Zhang, and J. Yang, "Tristate-set: Proactive set for improved performance of mlc phase change memories," in *2015 33rd IEEE International Conference on Computer Design (ICCD '15)*, pp. 659–665, 2015.

[21] Micron, "Breakthrough nonvolatile memory technology," *https://www.micron.com/about/emerging-technologies/3d-xpoint-technology*, 2016.

[22] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "Energy reduction for STT-RAM using early write termination," in *Proceedings of the 2009 International Conference on Computer-Aided Design (ICCAD '09)*, pp. 264–268, 2009.

[23] C. Smullen, V. Mohan, A. Nigam, S. Gurumurthi, and M. Stan, "Relaxing non-volatility for fast and energy-efficient STT-RAM caches," in *2011 IEEE 17th International Symposium on High Performance Computer Architecture (HPCA '11)*, pp. 50–61, 2011.

[24] M. T. Chang, P. Rosenfeld, S. L. Lu, and B. Jacob, "Technology comparison for large last-level caches (L3Cs): Low-leakage sram, low write-energy STT-RAM, and refresh-optimized eDRAM," in *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA '13)*, pp. 143–154, 2013.

[25] Micron, "MT29F6T08ETHBBM5-3R," *https://www.micron.com/products/nand-flash/3d-nand*, 2015.

[26] A. Patel, F. Afram, S. Chen, and K. Ghose, "MARSS: A full system simulator for multicore x86 cpus," in *Proceedings of the 48th Design Automation Conference (DAC '11)*, pp. 1050–1055, 2011.

[27] O. Mutlu and T. Moscibroda, "Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared DRAM systems," in *Proceedings of the 35th Annual International Symposium on Computer Architecture (ISCA '08)*, pp. 63–74, 2008.

[28] Y. Kim, M. Papamichael, O. Mutlu, and M. Harchol-Balter, "Thread cluster memory scheduling: Exploiting differences in memory access behavior," in *43rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '10)*, pp. 65–76, 2010.

[29] 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," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '14)*, pp. 455–470, 2014.

[30] J. Zhao, O. Mutlu, and Y. Xie, "FIRM: Fair and high-performance memory control for persistent memory systems," in *47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '14)*, pp. 153–165, 2014.

[31] M. Jung, W. Choi, J. Shalf, and M. T. Kandemir, "Triple-A: A non-SSD based autonomic all-flash array for high performance storage systems," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '14)*, pp. 441–454, 2014.

[32] M. Jung and M. Kandemir, "Sprinkler: Maximizing resource utilization in many-chip solid state disks," in *IEEE 20th International Symposium on High Performance Computer Architecture (HPCA '14)*, pp. 524–535, 2014.

[33] G. Sun, Y. Joo, Y. Chen, D. Niu, Y. Xie, Y. Chen, and H. Li, "A hybrid solid-state storage architecture for the performance, energy consumption, and lifetime improvement," in *IEEE 16th International Symposium on High Performance Computer Architecture (HPCA '10)*, pp. 1–12, 2010.

[34] IBM, "SAP in-memory computing on IBM eX5 systems," *http://www.redbooks.ibm.com/redpapers/pdfs/redp4814.pdf*, 2015.

[35] IBM, "In-memory computing with SAP HANA on lenovo X6 systems," *http://www.redbooks.ibm.com/redpapers/pdfs/redp4814.pdf*, 2015.

[36] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The HiBench benchmark suite: Characterization of the MapReduce-based data analysis," in *IEEE 26th International Conference on Data Engineering Workshops (ICDEW '10)*, pp. 41–51, 2010.

[37] J. L. Henning, "SPEC CPU2006 benchmark descriptions," *ACM SIGARCH Computer Architecture News*, vol. 34, no. 4, pp. 1–17, 2006.

[38] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A cycle accurate memory system simulator," *Computer Architecture Letters*, vol. 10, no. 1, pp. 16–19, 2011.

[39] M. Jung, E. Wilson, D. Donofrio, J. Shalf, and M. Kandemir, "NANDFlashSim: Intrinsic latency variation aware NAND flash memory system modeling and simulation at microarchitecture level," in *IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST '12)*, pp. 1–12, 2012.

[40] S. Jiang and X. Zhang, "LIRS: An efficient low inter-reference recency set replacement policy to improve buffer cache performance," in *Proceedings of the 2002 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '02)*, pp. 31–42, 2002.

[41] Micron, "DDR3 SDRAM RDIMM," *https://www.micron.com/products/dram-modules/rdimm/DDR3%20SDRAM#*, 2015.

[42] A. Ban, "Flash-memory translation layer for NAND flash (NFTL)," *M-systems*, 1998.

[43] J. Kim, J. M. Kim, S. Noh, S. L. Min, and Y. Cho, "A space-efficient flash translation layer for CompactFlash systems," *IEEE Transactions on Consumer Electronics*, vol. 48, no. 2, pp. 366–375, 2002.

[44] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, *Memory access scheduling*, vol. 28. 2000.

[45] M. T. Intel Corporation *et al.*, "Open NAND flash interface specification," 2014.

[46] M. Yourst, "PTLsim: A cycle accurate full system x86-64 microarchitectural simulator," in *IEEE International Symposium on Performance Analysis of Systems Software (ISPASS '07)*, pp. 23–34, 2007.

[47] F. Bellard, "QEMU, A fast and portable dynamic translator.," in *USENIX Annual Technical Conference, FREENIX Track*, pp. 41–46, 2005.

[48] F. Chen, M. Mesnier, and S. Hahn, "A protected block device for persistent memory," in *30th Symposium on Mass Storage Systems and Technologies (MSST '14)*, pp. 1–12, 2014.

[49] Y. Chen, S. Alspaugh, and R. Katz, "Interactive analytical processing in big data systems: A cross-industry study of mapreduce workloads," *Proc. VLDB Endow.*, vol. 5, no. 12, pp. 1802–1813, 2012.

[50] S. Chen, "Cheetah: A high performance, custom data warehouse on top of mapreduce," *Proc. VLDB Endow.*, vol. 3, no. 1-2, pp. 1459–1468, 2010.