# Towards Efficient NVDIMM-based Heterogeneous Storage Hierarchy Management for Big Data Workloads


Renhai Chen[†]
renhai.chen@tju.edu.cn

Zili Shao[*‡]
shao@cse.cuhk.edu.hk

Duo Liu[§]
liuduo@cqu.edu.cn

Zhiyong Feng[†]
zyfeng@tju.edu.cn

Tao Li[¶]
taoli@ece.ufl.edu

[†]College of Intelligence and Computing, Shenzhen Research Institute of Tianjin University, Tianjin University
[‡]Department of Computer Science and Engineering, The Chinese University of Hong Kong
[§]College of Computer Science, Chongqing University
[¶]Department of Electrical and Computer Engineering, University of Florida



## ABSTRACT

In this paper, we propose a holistic solution to address several important and challenging issues in storage data management in light of emerging NVDIMM-based architecture: namely, new performance modeling, NVDIMM-based migration, and architectural support for NVDIMMs on migration optimization. In particular, a novel NVDIMM-based heterogeneous storage performance model is proposed to effectively address bus contention issues caused by placing NVDIMMs on the memory bus. We also develop an NVDIMM-based lazy migration scheme to effectively minimize adverse effects caused by memory traffic interferences during storage data management processes. Finally, the NVDIMM-based architectural support for migration optimization is proposed to increase channel parallelism in the destination NVDIMMs and bypass buffer caches in the source NVDIMMs, so that the impact of memory traffic can be alleviated. We present detailed evaluation and analysis to quantify how well our techniques can enhance the I/O performances of big workloads via efficient heterogeneous storage hierarchy management. Our experimental results show that overall the proposed techniques yield up to 98% performance improvement over the state-of-the-art techniques.



## CCS CONCEPTS

• **Computer systems organization** → **Secondary storage organization**; • **Hardware** → *Memory and dense storage*.

## KEYWORDS

NVDIMM, heterogeneous storage, bus contention, machine learning



[*]Zili Shao is the corresponding author.




*MICRO-52, October 12–16, 2019, Columbus, OH, USA*



ACM ISBN 978-1-4503-6938-1/19/10…$15.00
https://doi.org/10.1145/3352460.3358266

**ACM Reference Format:**
Renhai Chen, Zili Shao, Duo Liu, Zhiyong Feng, and Tao Li. 2019. Towards Efficient NVDIMM-based Heterogeneous Storage Hierarchy Management for Big Data Workloads. In *The 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-52), October 12–16, 2019, Columbus, OH, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3352460.3358266


## 1 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. These workloads typically are executed on virtualized data centers with various storage resources. In virtualized data centers, storage resources are abstracted as data stores, and I/O is operated on multiple VMDKs (Virtual Machine Disks) that distribute across physical devices. Various workloads with different storage devices may lead to imbalanced performance, and the objective of the storage management is to achieve a balanced I/O performance with low latency and high throughput via better resource utilization.

NVDIMM (Non-Volatile Dual In-line Memory Module) is becoming a popular storage media due to its low I/O latency, superior capacity scalability and saving in hardware cost, power/cooling, and floor space [5, 17]. There are three types of JEDEC-defined NVDIMMs, i.e., NVDIMM-N, NVDIMM-P, and NVDIMM-F [39]. NVDIMM-N has limited capacity, as it is mainly used to back up data from DRAM into NAND flash on a power loss. Both NVDIMM-P and NVDIMM-F map the non-volatile memory into the memory address space, thereby significantly increasing the capacity. NVMDIMM devices have been integrated into storage systems and various products have been provided by leading IT companies such as IBM and Lenovo [17]. However, NVDIMMs post several new challenges for storage management.

First, it is more difficult to predict the I/O performance of NVDIMM devices. Different from traditional storage devices that transfer data via separated I/O paths, NVDIMM modules are installed into DDR slots and data are delivered via shared memory channels. Therefore, its I/O performance modeling should consider not only I/O workloads but also memory traffic, which cannot be solved by existing methods based on traditional storage architectures. Note that accurately predicting the I/O performance of NVDIMM devices is critical for optimizing initial data placement, imbalance detection, and data migration. Without it, VMDKs may be initially placed on

the overloaded NVDIMM devices and unnecessary data migrations from/to NVDIMM devices may be triggered, resulting in significant system overheads. Second, the density of VMDKs increases rapidly with big data workloads, which makes online balancing I/O performance across data stores more critical. Specifically, NVDIMM devices transfer I/O traffic via shared memory channels, and huge amount of I/O data in NVDIMM devices transferred via memory bus inevitably brings adverse effects on memory traffic, which can further deteriorate the performance of server nodes involved in data migration so as to influence normal business operations. Thus, it is important to minimize such adverse effects. Third, the enormous amount of data in VMDKs migrated from/to NVDIMMs present significant challenges for the NVDIMM performance. Therefore, new NVDIMM architecture techniques are urgently needed to exploit the characteristics of NVDIMMs for data migration in storage data management.

Existing storage management solutions cannot effectively solve these problems. Thus, we propose new NVDIMM-based storage management techniques with architectural optimization to address these challenges. In particular, we present three techniques to address initial data placement with load imbalance detection, data migration process, and NVDIMM architectural support, respectively. For the initial data placement with load imbalance detection, an NVDIMM-based heterogeneous storage performance model is presented to address bus contention caused by NVDIMM devices. Considering the data migration process, a lazy migration scheme is developed to minimize adverse effects caused by memory traffic interferences via reducing unnecessary data transfer. The NVDIMM-based architectural optimization technique is designed to increase channel parallelism in destination NVDIMMs and bypass NVDIMM buffer caches in source NVDIMMs, by which adverse effects caused by data migration can be alleviated.

We have conducted experiments with various big data benchmarks on a simulation environment and demonstrated the effectiveness of the proposed holistic techniques from various aspects. Our experimental results show that overall the proposed techniques yield up to 98% performance improvements over the state-of-the-art techniques [9, 10, 52].

This paper makes the following key contributions:

- To the best of our knowledge, this is the first work to design new NVDIMM-based storage data management techniques with architectural optimization to address data migration for storage resource management in virtualized data centers.
- We for the first time proposed a novel NVDIMM-based heterogeneous storage performance model to effectively solve bus contention caused by NVDIMM devices.
- We developed a lazy migration scheme to manage data migration processes so as to effectively minimize adverse effects caused by memory traffic interferences.
- The novel architectural optimization techniques have been designed to alleviate adverse effects for NVDIMM to serve as either source or destination in data migration.

## 2 BACKGROUND AND PRIOR ART

### 2.1 Server with NVDIMM-based Storage Device

Figure 1 shows a server node with the NVDIMM-based storage device for big data applications. Traditional storage devices are

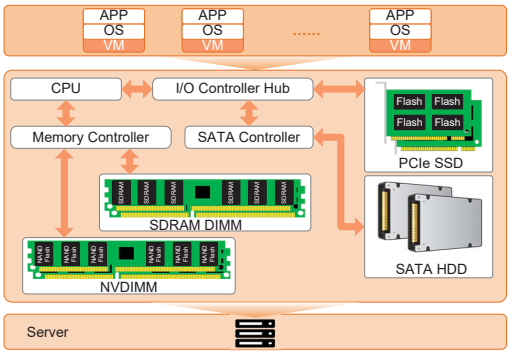


**Figure 1: A server node with NVDIMM.**

connected via the I/O controller hub with slow speed interfaces, such as SATA (serial ATA) or PCIe (PCI Express). On the contrary, NVDIMM modules are installed into DDR slots and transfer data through fast memory channels [5]. Specifically, NVDIMMs and DRAM-based DIMMs share the same memory channels, and applications transparently utilize NVDIMM techniques through a block I/O interface.

Note that NVDIMMs and DRAM-based DIMMs are placed on the same memory channel. If they were separated, channel/slot resources would not be fully utilized with limited channel parallelism of DRAM DIMMs and NVDIMMs. For example, there are only 6 memory channels with 12 slots in an IBM/Lenovo System x3530 M4 7160 server [18]. If these channels/slots are dedicated to one type of DIMMs, the channel/slot resources will not be useful for severing the other type of requests even when they are idle. Therefore, NVDIMMs and DRAM DIMMs share channels in practice (e.g., eXFlash from IBM/Lenovo) [14, 25].

| Attributes | NVDIMM | PCIe SSD | SATA HDD |
|---|---|---|---|
| Read latency | ~150 *us* | ~400 *us* | ~5 *ms* |
| Write latency | ~5 *us* | ~15 *us* | ~5 *ms* |
| Capacity | 400GB | 512GB | 3072GB |
| Price | ~420$ | ~177$ | ~82$ |
| Cost ($/GB) | ~1.05 | ~0.35 | ~0.027 |

**Table 1: Comprehensive comparison of different storage devices [16, 19, 40, 42].**

NVDIMM devices can greatly alleviate I/O latencies but with limited capacity. On the other hand, the size of a virtual machine disk image is typically huge. When multiple VMDKs are consolidated to a storage system, to achieve a balanced I/O performance (low latency, high throughput) and utilization (effective capacity), it is critical to intelligently distribute data (including initial placement and dynamic migration) across the heterogeneous storage devices that consist of NVDIMMs, SSDs, and HDDs with different performance/cost tradeoffs (as shown in Table 1).

Note that the capacity of NVDIMM is comparable to SSD/HDD and the price of NVDIMM will be lowered with the development of this technology. Therefore, it is reasonable to store VMDKs in NVDIMMs. In addition, regardless of the usage scenario (e.g., using expensive NVDIMMs for caching hot data), if many workloads (e.g., hot data) are consolidated into one NVDIMM device, the response time for these workloads may become ultra-long. Therefore, to achieve a balanced I/O performance (low latency, high throughput) and utilization (effective capacity), this paper intelligently distributes data (including initial placement and dynamic migration) across the heterogeneous storage devices that consist of NVDIMMs, SSDs, and HDDs with different performance/cost tradeoffs.

## 2.2 Data Management in the Heterogeneous Storage System

VMDK management is an important technique to improve the overall storage utilization across multiple physical devices. A storage manager such as vSphere and vMotion [43] performs data placement and migration on a cluster of storage devices. It triggers data relocation when the imbalanced device utilization is detected, such as the overloaded SATA HDD and underloaded NVDIMM as shown in Figure 2 (a). Note that each server node can be equipped with different kinds of storage devices as illustrated in Figure 1. In this section, we use the following storage architecture as an example to present the storage management in the multiple server nodes.

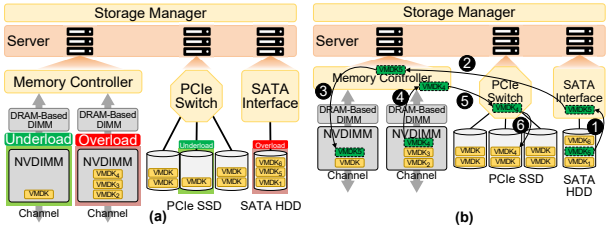


**Figure 2: (a) Imbalanced load in the storage system. (b) Data migration for load balancing.**

To perform data migration, VMDKs are required to be moved from source storage devices to destination storage devices (e.g., the steps ❶ ❷ ❸ and ❹ ❺ ❻ as shown in Figure 2 (b)). To accomplish this procedure, the first issue is how to effectively detect the overload conditions and react by migrating VMDK images. This is challenging since the overload condition heavily depends on the characteristics of I/O workloads and underlying storage devices.

To address the above issue, several state-of-the-art migration schemes are proposed. BASIL [9] is a software system that automatically manages I/O placement and performs load balancing across devices. BASIL supports online device modeling and I/O load balancing, but it lacks cost-benefit analysis, which may lead to poor decisions. As a result, Pesto [10] is proposed to completely automate I/O load balancing management for data centers with cost-benefit analysis. LightSRM [52] further leverages I/O mirroring to redirect the I/O requests without moving the virtual disk, thereby mitigating the storage migration cost.

These schemes can effectively manage the traditional storage system. However, to accomplish efficient data management in the NVDIMM-based heterogeneous storage hierarchy, we need to conquer several new challenges introduced by the NVDIMM architecture. As shown in Figure 2 (b), different from traditional storage devices that transfer data via separate I/O paths, NVDIMM modules are installed onto DDR slots and data are delivered via shared memory channels. Prior art [5] that focuses on balancing the memory bus resource allocation between NVDIMMs and DRAM DIMMs has demonstrated that the bus contention plays an important role in the NVDIMM performance. Therefore, the I/O performance modeling in the NVDIMM-based storage hierarchy must consider not only I/O workloads but also memory traffic, which cannot be solved by existing methods that assume traditional storage architectures. On the other hand, in NVDIMM modules, as I/O traffic is transferred via shared memory channels, it is inevitable to bring interferences to original memory traffic. As a result, during VMDK migrations, the performance of server nodes can be further deteriorated by memory traffic interferences. Thus, how to mitigate such adverse effects in VMDK migrations needs to be investigated.

# 3 MOTIVATION

In this section, we illustrate the interference issue introduced by the mixed NVDIMM and DRAM DIMM traffic, and its impact on data management efficiency in the heterogeneous storage hierarchy.

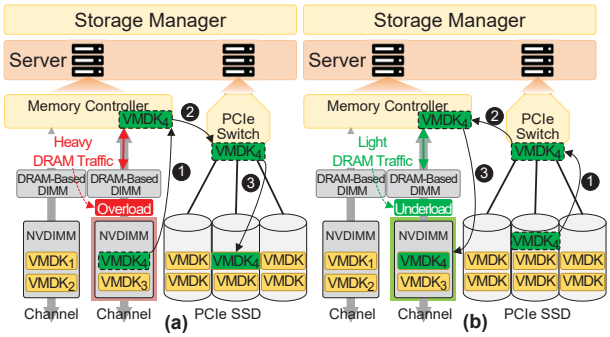


**Figure 3: Unnecessary data migration introduced by prior art [9, 10, 52]. (a) Data migrated from the NVDIMM to the PCIe SSD. (b) Data migrated from the PCIe SSD to the NVDIMM.**

Since prior storage management schemes (e.g., Pesto) do not take the bus contention into consideration, the unnecessary data migration may be triggered (e.g., the steps ❶ ❷ ❸ as shown in Figures 3 (a) and (b)). When the DRAM-based memory traffic heavily occupies the memory bus, the NVDIMM performance is throttled by the bus contention. As a result, traditional schemes such as Pesto may mistakenly detect that the NVDIMM is overloaded, and trigger the unnecessary data migration from the NVDIMM to the PCIe SSD as shown in Figure 3 (a). Later, when the DRAM-based memory traffic is reduced, it is recognized that the NVDIMM is underloaded. Then, the VM disk image previously migrated may be moved from the PCIe SSD to the NVDIMM as shown in Figure 3 (b). Similarly, unnecessary VMDK migrations may also occur because of the initial misplacement introduced by prior works. For instance, when the memory bus traffic is heavy, a newly created VMDK will be placed in other devices even with underloaded NVDIMM. Later, when the bus traffic becomes light, this VMDK may be migrated to the NVDIMM, thereby resulting in unnecessary data transfer.

To further analyze the overhead introduced by the prior schemes, we conduct experiments based on Hibench [13] and SPEC CPU2006 [11]. Hibench is a representative big data benchmark suite and for the illustration purpose, eight 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. The experimental environment in this section is the same as we used in Section 6.1. The default NVDIMM buffer cache is configured as 400MB with the LRFU algorithm [8]. To illustrate the interference impact, two sets of experiments are conducted by running each one of the big data applications separately or concurrently with *429.mcf* based on single node and multiple server nodes, respectively. The normalized I/O latency is adopted to quantitatively analyze the system performance. We first obtain the average I/O latency of each workload, and then calculate the average statistics across all workloads. The normalized I/O latency of each workload is obtained based on the average statistics across all workloads.

The interference impact on NVDIMM devices can greatly influence the efficiency of storage data management. To demonstrate this, we first conduct experiments by running the eight big data applications together on a heterogeneous storage system (the detailed configuration is described in Section 6.1). Three state-of-the-art load balancing schemes, BASIL [9], Pesto [10], and LightSRM [52], are adopted to manage data migrations, respectively. Each of the three schemes is implemented as the storage management mechanism to manage the NVDIMM, SSD, and HDD. We then conduct experiments by running the eight big data applications concurrently with *429.mcf*.

Table 2 shows that the bus contention can introduce significant storage data management overheads across three state-of-art load balancing schemes, ranging from 39% to 91%. The main reason is that by assuming traditional storage architectures, existing methods only consider I/O workloads in their performance modeling. As a result, these schemes fail to accurately estimate the NVDIMM performance and thus frequently trigger data migrations from/to the NVDIMM devices. Therefore, we need to investigate new performance modeling for NVDIMM devices and corresponding optimization schemes to alleviate the interference impact for the NVDIMM-based heterogeneous storage architecture.

| Environment | Schemes | Overhead |
|---|---|---|
| Single node | BASIL | 91% |
| | Pesto | 77% |
| | LightSRM | 50% |
| Multiple nodes | BASIL | 86% |
| | Pesto | 63% |
| | LightSRM | 39% |

**Table 2: Migration overhead compared with or without memory interference.**

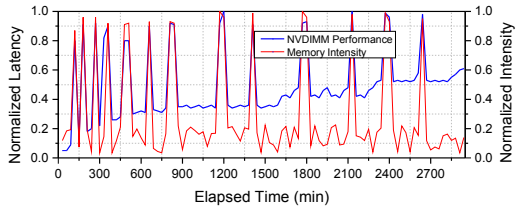


**Figure 4: The memory traffic effect on the NVDIMM performance.**

We further examine memory behavior and its effect on the NVDIMM performance. We track the latency of the NVDIMM and the memory intensity (the number of memory reads and writes) every 30 minutes. It can be observed that the NVDIMM performance periodically fluctuates with the memory traffic variation. The essential reason is that the memory access and CPU computation are interleaving in most applications. This behavior can be reflected by RPKI and WPKI (i.e., memory reads/writes per kilo instructions) of the workloads, as described in Section 6.1. Thus, if we can identify this behavior, we can not only avoid the unnecessary data migration but also reduce the memory bus contention between the migrated data and the DRAM-based memory traffic.

# 4 STORAGE DEVICE PERFORMANCE AND BUS CONTENTION MODELING

In an NVDIMM-based heterogeneous storage environment, in order to reduce unnecessary data migration, it is critical to construct new performance modeling so as to effectively and efficiently predict the performance of NVDIMM devices considering the memory bus contention issue. Next, we first utilize several motivational examples to illustrate the relationship between the device performance and workloads, and then propose our performance modeling, bus contention modeling, machine-learning-based implementation, and model verification.

## 4.1 Relationship Between Device Performance and Workload Characteristics

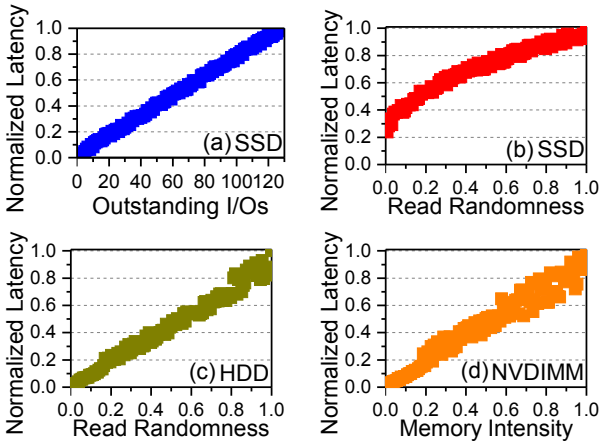


**Figure 5: The performance variation with different data access characteristics and devices.**

We first examine the key factors that influence the performance of PCIe SSDs. To this end, several synthetic traces based on *dfsioe_r* and *sort* big data applications are utilized to analyze the effects of outstanding I/Os and read randomness. Figures 5 (a) and (b) show the results obtained by directly running these traces on a PCIe SSD. Note that the I/O latency arises linearly as the number of outstanding I/O operations increases as shown in Figure 5 (a). Figure 5 (b) reveals that the more random the read operations are, the worse the I/O latencies become. Different from the outstanding I/Os, the relationship between the device latency and read randomness is non-linear, especially for the workloads that manifest high randomness.

Figure 5 (c) shows the relationship between the I/O latency and read randomness on the HDD device. Note that different from SSD, the latency varies linearly as read operations become more random. The reason is that as the randomness of the read operations increases, the overall rotational delay in HDDs will increase linearly.

For NVDIMM devices, I/O latency tends to vary linearly with memory intensity as shown in Figure 5 (d). This is because with NVDIMM devices, the bus contention delay increases linearly with more memory traffic. This is different from HDD and SSD since the data are moved into/out the NVDIMM device by using the DDR interface while the HDD/SSD device uses the SATA/PCIe interface to transfer data. Thus, for NVDIMM devices, the bus contention should be an important factor for predicting the NVDIMM performance and guiding load balancing.

These parameters are some representative ones to reflect the relationship between the workload characteristics and device performance. Figures 5 (a) and (b) show the different workload characteristics play different roles in determining the device performance. Figures 5 (b) and (c) illustrate that the same workload characteristic may play different roles in determining the performance of different

devices. Figure 5 (d) shows NVDIMM's performance is influenced by memory intensity, which is unique and considered in this paper. These results motivate us to use a black-box model to model the relationship between the workload characteristics and device performance.

## 4.2 Performance Modeling

Since the device performance is affected by many factors and even the same factor may have different impacts on different devices, a static model cannot well predict the device performance.

Thus, we propose a machine learning method to dynamically study and predict the relationship between the workload characteristics and the device performance. Specifically, we use a black-box model to predict the storage device performance as a function ($f$) of workload characteristics. This model takes the workload characteristics ($WC$) as input parameters and outputs the predicted performance metric ($PP$), as follows:

$$PP = f(WC), \tag{1}$$

$$WC = < wr\_ratio, OIOs, IOS, wr\_rand, rd\_rand, free\_space\_ratio > . \tag{2}$$

Equation (1) shows how to calculate the storage device performance. $WC$ is used to represent the workload characteristics that are measured with six factors as shown in Equation (2). Here, $wr\_ratio$ represents the read and write ratio that is defined as the percentage of reads and writes among all requests; $OIOs$ represents the number of outstanding requests; $IOS$ is the I/O request size; and $rd\_rand/wr\_rand$ counts the percentages of random read and write accesses in the I/O request stream, respectively. We deduce random reads or writes by checking the accessed addresses. If two requests access the adjacent addresses, these two requests are sequential reads or writes. Otherwise, these two requests are random accessed requests.

$free\_space\_ratio$ in Equation (2) represents the free space ratio and is specifically used to reflect the GC (Garbage Collection) effect for flash-based storage devices. When invalid flash space is reclaimed by GC, the write-cliff problem [15] may occur and seriously degrades the flash-based storage device performance. While the GC effect can be eliminated most of time with the well-designed implementation and preemptive approaches [5, 15, 24], when there is limited free space, GC will be triggered frequently and the write cliff can still seriously affect the performance. Thus, $free\_space\_ratio$ is introduced to reflect the GC effect on NVDIMM devices. Unlike the previous work in modelling the GC effect of SSDs [7], where the internal information such as the over-provisioning ratio and the GC policy is required, our black model does not need prior knowledge and $free\_space\_ratio$ will be obtained by training. Other FTL activities such as wear leveling have not been incorporated in our model. How such FTL activities affect the model and performance will be investigated in our future work.

To construct a black-box model, we first collect the training data that consist of workloads characteristics and its corresponding performance on a storage device. Note that the training data shall be representative (to span a wide spectrum) and sufficient (to have an adequate number of the tests). We then use statistical machine learning algorithms to capture the mapping between workload characteristics (independent variables) and performance metrics (dependent variables). In our approach, given the training data as the input, the regression algorithm is applied to calculate a predictive function that maps the input to the desired output.

## 4.3 Bus Contention Modeling

The bus contention effect is estimated by subtracting the predicted device performance from the measured device performance as follows:

$$BC_d = MP_d - PP_d, d \in NVDIMM. \tag{3}$$

In Equation (3), $MP_d$ is used to present the measured device performance that is the latency containing both the device response time and the delay caused by the bus contention; $PP_d$ is the predicted device performance that reflects the device performance without considering the bus contention and is calculated based on Equation (1).

## 4.4 Machine-learning-based Implementation

To apply the performance and bus contention modeling in Equation (1), we need to build the relationship between the workload characteristics and device performance. Linear regression is an approach for modeling the relationship between a scalar dependent variable and one or more explanatory variables (or independent variables) [29, 41]. In this paper, we focus on using more than one explanatory variable, which is called multiple linear regression. After using the regression tree to obtain the performance model, the linear regression can make a performance prediction based on a given set of workload characteristics.

Note that the linear regression is an approach for modeling the relationship between a scalar dependent variable and one or more explanatory variables (or independent variables). This model can well satisfy our requirement, which needs to predict the device performance with multiple workload characteristics. The results shown in Section 4.5 (Model Verification) demonstrate the accuracy of the proposed model. As this model is simple, it does not introduce noticeable computation and space overhead. We also tried other models, such as the aggregation model [10]. However, the aggregation model is based on the outstanding IOs only while the linear regression model considers all the key and non-key factors. Thus, the linear regression model is selected.

We build a regression tree from the regression function, which is generated by recursively splitting the input independent variables into leaf nodes using a binary sequence. The leaf nodes of the tree provide the predicted values for dependent variables as a constant function of independent variables. Trees are recursively built in a top-down manner, beginning with a root node. At each step in the recursion, the building algorithm determines which predictor variable in the training data best splits the current node into leaf nodes. The best split is the one that can minimize the difference (e.g., the root mean square deviation (RMSD)) among the samples in the leaf nodes, where the error is the difference between each sample and the average of all samples in the leaf node. In other words, a good split produces leaf nodes with samples that contain similar values.

| wr_ratio | IOS | free_space_ratio | Latency |
|---|---|---|---|
| 25% | 4 $KB$ | 10% | 65 $us$ |
| 25% | 8 $KB$ | 60% | 40 $us$ |
| 50% | 4 $KB$ | 60% | 42 $us$ |
| 50% | 8 $KB$ | 10% | 85 $us$ |
| 75% | 4 $KB$ | 60% | 32 $us$ |
| 75% | 8 $KB$ | 10% | 80 $us$ |

**Table 3: An example of training samples.**

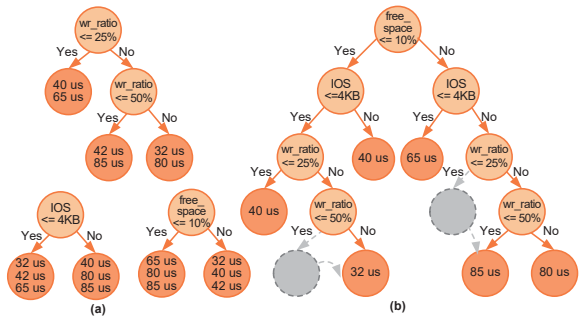


**Figure 6: An example of the steps in building a regression tree from the samples in Table 3.**

Figure 6 shows one example. Considering the training samples in Table 3, there are three predictor variables (i.e., write ratio, request size, and free space ratio) and one predicted device latency. Note that there are six predictor variables in Equation (2), and here we select three for illustrative purposes. Splitting on a single factor with the write ratio, request size, and free space ratio results in the tree shown in Figure 6 (a). However, splitting on the free space ratio, as shown in Figure 6 (a), yields the lowest RMSD in the leaf nodes (i.e., the values in each leaf are more homogeneous). Therefore, the *free_space_ratio* is the best first split and selected as the root node. The algorithm then recursively performs on each subset. The same predictor variable may appear at different levels in the tree, as it may be best to split multiple times on that variable. In this case, the I/O size (*IOS*) is the next best split, resulting in the tree shown in Figure 6 (b).

## 4.5 Model Verification

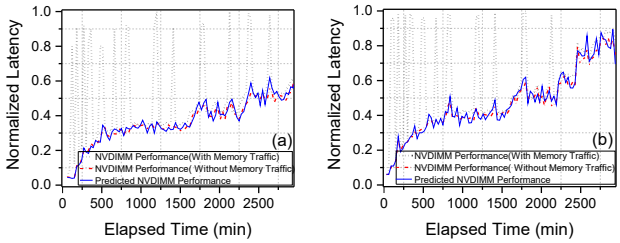


**Figure 7: Comparison of the predicted NVDIMM performance and the real NVDIMM response time with (a) 100% free space, and (b) 10% free space initially.**

To verify the accuracy of the proposed bus contention modeling, we conduct experiments based on Hibench [13] and SPEC CPU2006 [11]. *429.mcf* in SPEC CPU2006 is chosen as a representative for memory-intensive applications. The experiments are conducted with the same environment as described in Section 3. The training data are generated by a synthetic I/O workload generator [20] and the big data workloads [13]. By taking the workload characteristics as input parameters, we generates a series of I/O requests that are sent to the NVDIMM devices. We create the I/O workloads under above five types of access patterns and one particular storage condition (i.e., *free_space_ratio*). In the experiment, we use 800GB data, which is representative (to span a wide spectrum) and sufficient (to have an adequate number of the tests), to train our model.

Figure 7 shows our experimental results. The curve with the dotted line shows the measured NVDIMM performance with mixing the memory traffic. With the same workloads, the curve with the green color represents the predicted NVDIMM performance by using the models described in Equations (1) and (2), and the one with the red color denotes the measured NVDIMM performance without mixing the memory traffic. In the experiments, we track the latency of the NVDIMM with/without mixing memory traffic and record the predicted NVDIMM response time every 30 minutes. From Figure 7, it can be observed that the predicted NVDIMM latency is close to the NVDIMM performance without the mixed memory traffic, and our model only introduces 5% loss of accuracy even with the limited free space (resulting in frequent garbage collection). The error of the proposed model is negligible compared with the huge performance deviation caused by the bus contention on the NVDIMM performance that can be observed from Figure 7. Thus, by subtracting measured and predicted NVDIMM performance values, the predicted bus contention status can be closely estimated.

# 5 NVDIMM-BASED HETEROGENEOUS STORAGE MANAGEMENT

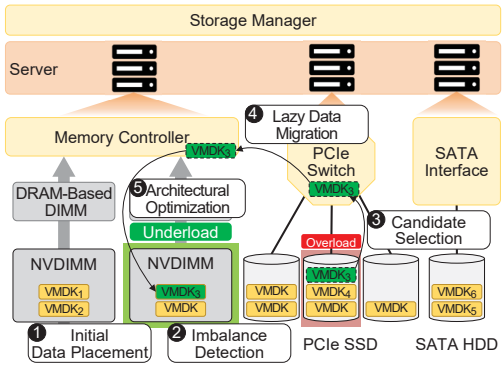


**Figure 8: An overview of the proposed schemes.**

In this section, we propose a holistic solution to optimize NVDIMM-based heterogeneous storage hierarchy management. An overview of the proposed schemes is shown in Figure 8. Next, in Section 5.1, we will present the bus contention aware migration management that addresses three critical steps in a VMDK management, namely, initial data placement (Step ❶ ), imbalance detection (Step ❷ ), and candidate selection (Step ❸ ). Then, in Sections 5.2 and 5.3, two optimization schemes are proposed to alleviate migration overheads from the data migration process (Step ❹ ) and architectural support (Step ❺ ) perspectives, respectively.

## 5.1 Bus Contention Aware Management

*5.1.1 Initial Data Placement.* Initial storage data placement is vitally important to reduce potential imbalance issues in future. A well planned workload placement can effectively exploit the advantages of storage devices and eliminate unnecessary data migration. In our initial data placement scheme, we aim to place a new VMDK onto a storage device where it can introduce the minimum overall system overhead and does not trigger data migration. This is achieved by placing the VMDK onto a device with the minimum overhead as follows.

$$min(\sum_{\kappa=1}^{i-1} P_{d_\kappa} + PP_{d_i} + \sum_{\kappa=i+1}^{n} P_{d_\kappa})/n. \quad (4)$$

In Equation (4), $n$ is the number of devices; $d_i$ is the device where the VMDK is placed; $PP_{d_i}$ represents the predicted performance of $d_i$ if the VMDK is placed on $d_i$; and $P_{d_\kappa}$ represents the performance of the device $d_\kappa$ calculated according to Equation (5).

Using Equation (4), we can obtain the average system performance when placing VMDK onto each device, and the device with the minimum average system performance is selected as a candidate. However, simply placing VMDK on the candidate device may introduce the data migration. To avoid data migration, we further check the imbalance threshold $\tau$ described in Section 5.1.2. If placing VMDK on the selected candidate triggers data migration which can be predicted by using Equations (1) and (2), we remove this candidate and redo the above procedures until no migration occurs.

*5.1.2 Imbalance Detection and Candidate Selection.* We aim to distribute the load proportionally to each storage device. To achieve this goal, we need to detect imbalanced devices and select candidate workloads to perform load balancing. The measured performance can directly reflect the current device status (e.g., overload) and thus has been widely used for imbalance detection and candidate selection in the traditional load balancing design. However, this method is not suitable for the NVDIMM-based storage architecture since directly using the measured performance will attribute the bus contention latency to the device performance. For this reason, we use the predicted workload performance to calculate the NVDIMM performance as follows.

$$P_{d_j} = \begin{cases} MP_{d_j} = (\sum_{w_i \ on \ d_j}^{n} MP_{w_i})/n, \ d_j \notin NVDIMM. \\ PP_{d_j} = (\sum_{w_i \ on \ d_j}^{n} PP_{w_i})/n, \ d_j \in NVDIMM. \end{cases} \quad (5)$$

Here, $n$ is the number of workloads on a device, and $w_i$ represents the $i$th workload. The device performance $P_{d_j}$ can be calculated using Equation (5), where $MP_w$ and $MP_d$ denote the measured workload and device performance, respectively. $PP_w$ represents workload performance, which can be obtained according to Equations (1) and (2). Based on $PP_w$, we calculate the predicted NVDIMM performance $PP_d$.

Let max($P_d$) and min($P_d$) be the maximum performance and the minimum performance, respectively. The imbalance fraction $\Delta$ is defined as $\Delta$ ($P_d$) = max($P_d$) - min($P_d$). In our imbalance detection scheme, if $\frac{\Delta(P_d)}{max(P_d)}$ is greater than a predefined threshold $\tau$, data migration will be triggered. Two candidate devices that are with the minimum and maximum loads, respectively, are selected to perform data migration, in which a VMDK is migrated from the device with the maximum load to the one with the minimum load.

Note that when considering the NVMDIMM performance, bus contention latency is disregarded in our techniques. The reason is as follows: Once migrated, a VMDK will be operated in a relatively long time. Thus, the performance influence on NVDIMM devices from memory traffic fluctuations in short-time periods can be cancelled out considering the relatively long total execution time.

## 5.2 Lazy Data Migration

In this section, we present the lazy data migration, which features with the I/O mirroring technique and the cost/benefit function, to address the on-time balancing I/O performance issue and reduce the memory bus contention.

Different from the state-of-the-art migration schemes that directly copy data from the old location to the new location, the I/O mirroring technique is proposed to lazily redirect upcoming requests to the new location. By adopting this technique, all upcoming write requests are redirected to the new locations, so that we can reduce unnecessary data transfer. For other data, we propose to use the cost/benefit function to guide the migration. With this function, data are only migrated when the benefit is larger than the cost, and thus we can guarantee that the data migration can enhance the overall storage performance. During a migration process, a bitmap table is utilized to record the location of a data block (i.e., 0 for not-migrated, and 1 for migrated). This mechanism introduces the space and performance overhead. The space consumption is acceptable, e.g. for a NVDIMM device with 400 GB space and 4KB of each data block, it requires 400GB/4K*1bit=12.5MB. In addition, this space is reclaimed when the migration is finished. The performance overhead is negligible, since the mechanism does not introduce complicate data lookup by only checking one bit to determine the data location.

**Cost**: Data migration induces non-trivial performance overhead on the source and destination during the dominant copy phase. For each data movement, the proposed scheme calculates the performance cost in the source and destination using the models described in Equation (1). Specifically, for data movement between two NVDIMM devices, the cost can be calculated as follows:

$$Cost = Q_{mig} \times (t_{PP_{r_{src}}} + t_{PP_{w_{dst}}} + t_{BC_{d_{src}}} + t_{BC_{d_{dst}}}). \quad (6)$$

In Equation (6), $t_{PP_{r_{src}}}$ and $t_{PP_{w_{dst}}}$ are used to represent the time consumed by migrating one unit of data (i.e., 4KB block) in the source and destination, respectively. $t_{BC_{d_{src}}}$ and $t_{BC_{d_{dst}}}$ are used to represent the time consumed on the bus contention of moving one data unit in the source and destination NVDIMMs, respectively. $Q_{mig}$ is used to represent the migrated data volume. Thus, the total time spent to migrate $Q_{mig}$ can be obtained by Equation (6).

**Benefit**: For each data movement, we calculate the possible benefits in the source and destination based on the device performance and the fraction of the load that is being moved. If the destination has no load, the migrated workload is used for the calculation at the destination. Specifically, we use the following equation to compute the benefit:

$$Benefit = Q_{live} \times ((t_{P_{d_{src\_b}}} + t_{P_{d_{dst\_b}}}) - (t_{PP_{d_{src\_a}}} + t_{PP_{d_{dst\_a}}})). \quad (7)$$

Here, "b" and "a" represent before and after migration, respectively. The performance enhancement $\triangle t$ can be calculated by subtracting the predicted source/destination device latency of processing one data unit after data migration ($(t_{PP_{d_{src\_a}}} + t_{PP_{d_{dst\_a}}})$) from the current source/destination device latency of processing one data unit ($(t_{P_{d_{src\_b}}} + t_{P_{d_{dst\_b}}})$). Then, the benefit is calculated by multiplying the device performance enhancement $\triangle t$ and the data volume of live data migration $Q_{live}$.

## 5.3 Architectural Optimization

We propose a novel mechanism to perform architectural optimization so as to maximally utilize NVDIMM devices on data migration. Specifically, we explore the unique features of migrated data to eliminate the bus contention of destination NVDIMMs and cache pollution of source NVDIMMs.

*5.3.1 Migration-Aware Bus Scheduling for Destination NVDIMMs.* We first propose migration-aware scheduling polices to fully explore the parallelism (e.g., channel-level parallelism) of flash-based NVDIMMs. As a storage device, NVDIMM devices require data consistency, i.e., the system needs to stay from one valid state to another valid state after any write request is served for persistent store [4, 22, 27, 30, 31, 33, 36, 47]. Consistency is vital for persistent store to guard against data corruption due to system crashes or power failures [26, 35, 50]. The primary consistency techniques

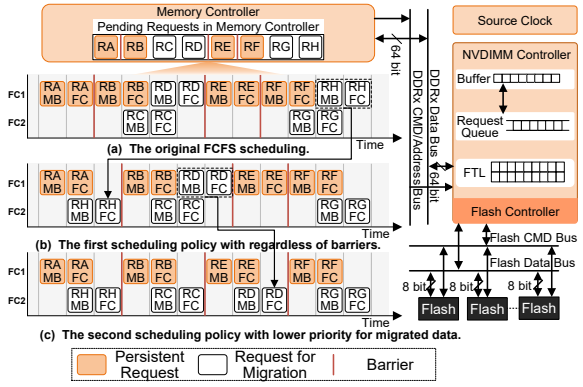


**Figure 9: The proposed scheduling polices.**

are based on persistent update mechanisms at the software level, such as journaling [6, 12, 44, 48, 51] and shadow updates [28, 44, 45, 51], and enforcing ordering writes at the hardware level. To guarantee the consistency of flash-based NVDIMMs, at the hardware level, the cache controller, write buffer, memory controller, and other components must respect the barriers when performing write requests. However, the barriers with ordered persistent data may sacrifice the potential parallelism in NVDIMM.

An example is given in Figure 9 to illustrate the persistent data effect on parallelism. The baseline is the First-Come-First-Serve (FCFS) scheduling policy [37]. Eight write requests (RA,RB, ..., and RH) and three barriers are sent to the memory controller; RA, RB, RD, RE, RF, and RH go to the first flash channel (FC) of the NVDIMM after passing the memory bus (MB), and RC and RG go to the second flash channel.

Figure 9 (a) illustrates a scheduling algorithm that respects the barriers. In this scheduling scheme, RB, RC, and RD cannot be issued until RA is finished because of the first barrier. Similarly, RE is pending until RB, RC, and RD complete, and RF, RG, and RH should wait until RE is finished. From the example, it can be observed that keeping consistency prohibits the scheduler from fully exploiting the available channel parallelism, which is a critical performance factor in current NVDIMM architectures.

Migrated data present different features compared with the persistent data. A mirror stored in another data store keeps the original migrated data, so the migrated data can be easily recovered from the mirror even suffering power failure. This means we can treat the migrated data as non-persistent data and break the ordering constraint. Figure 9 (b) shows a scheduling algorithm that we can use if the barriers do not restrict the access sequence of migrated data. This phenomenon leads to an optimal design if the NVDIMM system needs to serve the dual roles of migrated data and persistent store simultaneously. For example, suppose that among the eight write requests, RA, RB, RE, and RF belong to persistent store, and the others belong to migrated data or non-persistent store. As shown in Figure 9 (b), we can observe that RH can be served concurrently with RA since RH belongs to the migrated request.

Based on the above observation, we propose two scheduling policies. The first one is that migrated data can be scheduled regardless of barriers. Because barriers are only intended to restrict the order of the write requests to persistent store, the memory controller can schedule migrated write requests free from the barrier constraints. The second policy is that writes to persistent store are prioritized over writes to migrated data. The idea is to prioritize write requests on dependency chains. A barrier delimits multiple writes into two epochs and creates dependency between the two epochs, i.e., the following epoch cannot begin execution until the preceding epoch completes. Therefore, by scheduling writes to persistent store first in the presence of barriers, we can make subsequent persistent writes available to the memory scheduler earlier, thereby exposing more opportunities for exploiting channel-level parallelism. An example of the second scheduling policy is shown in Figure 9 (c). With our scheduling policy, when we reorder migrated and normal writes, we will check if a migrated write will access the same location as these normal writes involved in the reordering; if yes, the migrated write will be directly discarded.

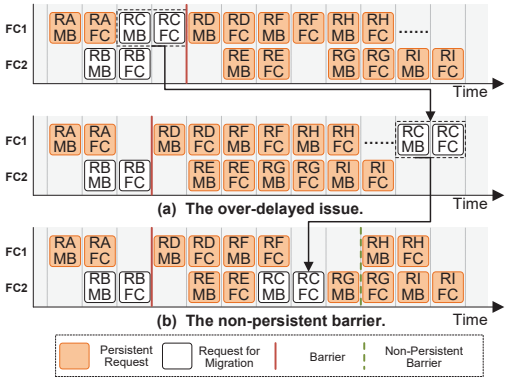


**Figure 10: The non-persistent barrier used to solve the over-delayed issue.**

The second scheduling policy may introduce the over-delayed issue: the upcoming persistent requests may delay the response of migrated write requests to an undetermined time. An example to illustrate this issue is shown in Figure 10 (a). To address this issue, we propose a non-persistent barrier mechanism. With the non-persistent barrier, migrated write requests are issued to NVDIMMs immediately. The insertion of the non-persistent barrier is performed by the memory controller. The memory controller monitors the requests in the transaction queue. If a migrated write request arrives at a predefined earlier time period compared with all the persistent writes, memory controller inserts a non-persistent barrier to first respond the migration write request as shown in Figure 10 (b).

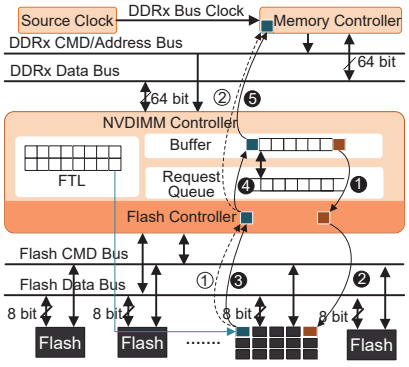


**Figure 11: Bypassing the buffer cache for migrated data.**

*5.3.2 Buffer Cache Bypassing for Source NVDIMMs.* Data migration for load balancing can affect the hit ratios of NVDIMM

buffer caches, which has huge impact on the NVDIMM performance. Figure 11 illustrates the effect on buffer caches. Traditional designs have no knowledge about the accessed requests, and all accessed data are first cached in the buffer cache (the steps ❸ ❹ ❺ shown in Figure 11) for further fast accessing. Once a migrated read request is issued to the NVDIMM controller, it deems this data to be accessed in the near future and kicks out other data if there is no space. Such data eviction may trigger data write-back (the steps ❶ ❷ shown in Figure 11) between the buffer cache and flash memory, which not only introduces extra data write operations but also incurs the lengthy garbage collection process.

However, the migrated data will be moved and further accesses to the data will be redirected to other devices. Thus, the enormous amount of data in a VMDK migration will heavily pollute the buffer cache and the NVDIMM performance is severely degraded. To address this issue, we propose a bypassing mechanism, which classifies the data type (i.e., migrated data or normal data requests) and directly transfers migrated data from flash memory to the memory controller (the steps ① ② shown in Figure 11). This can effectively reduce the deleterious impact on the buffer cache and thus greatly improves the NVDIMM performance.

***Implementation:*** To implement the proposed architectural optimization schemes, the request type (i.e., migrated data or normal data requests) should be delivered from the storage manager to the NVDIMM controller. At the software layer, one bit in the I/O request header is added and utilized to represent the type information (e.g., migrated data request), and then we encode this information in the virtual address when converting IO requests to NVDIMM memory accesses. To covert the bit in the I/O request header into something that the memory controller logic can identify, we pass this information to the physical address during address translation. Specifically, OS is modified to have the knowledge of the connotation of most significant bit (MSB) in virtual space and passes this connotation to the physical address space. We use the MSB of the 64-bit address to denote whether the request is a migrated data request. At the hardware layer, we augment the memory controller with a new command signal line that can transfer the request type to the NVDIMM controller.

# 6 EVALUATION

## 6.1 Experimental Setup

We use Marssx86 [34], DRAMSim2 [38], and NANDFlashSim [21] to achieve the experiment. Marssx86 is a cycle-accurate full-system simulator that uses PTLsim [49] for CPU simulation on top of the QEMU emulator [2]. DRAMSim2 is a cycle-accurate simulator of the main-memory system. Based on NANDFlashSim, we develop an SSD and NVDIMM simulator and adopt the page level FTL [1] in the both the SSD and NVDIMM controller. We first collect the memory traces by using Marssx86 with integrated DRAMSim2 and the I/O traces by running the big data workloads. Then, by injecting the mixed memory and I/O traffic into the simulator with the integration of DRAMSim2 and NANDFlashSim, we simulate the NVDIMM integrated heterogeneous storage management.

Note that there is another simulation method by using DRAM to emulate NVDIMM with adding extra latency. However, two major factors impede the design of this simulation methodology. First, the internal structure of NVDIMM (e.g., the parallel components) could not be reflected by DRAM. In addition, the physical layout of DRAM DIMM and NVDIMM, which reflects the bus contention, could not be well simulated. Therefore, this simulation method is not adopted in this paper.

| | |
|---|---|
| CPU | 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 | 256 GB, 2-bit MLC, 16 flash channels of 4 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, 12800 MB/s DDR3 1600MHz interface |
| SSD | 512 GB, 2-bit MLC, 16 flash channels of 4 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, 4096MB/s PCIe 2.0 x8 interface |
| HDD | 1TB, 7200rpm rotational speed, 600MB/s SATA 6Gbps interface |

**Table 4: The system configuration.**

Table 4 presents the configurations of the simulated 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 the NVDIMM is configured to 256GB with 16 flash channels. The parameters of the SSD and HDD devices used in this simulator are described in Table 4. In the SSD simulator, the page-level FTL [1] is adopted. The Ethernet connection parameters used in this paper are based on the standard NE2000 NIC device model.

| 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% |

**Table 5: The configuration of the mixed workloads.**

Table 5 summarizes workload combinations in the evaluation. A typical server should simultaneously provide computation, main memory, and storage resources, and these resources are shared by different applications (e.g., I/O intensive big data workloads or CPU/memory intensive SPEC applications). So, jointly running the big data workloads can cover a wide range of typical real-world workload behaviors. We combine these big data workloads to one of the three SPEC CPU2006 [11] workloads, namely (i.e., 429.mcf,

470.lbm, and 433.milc). The three SPEC CPU2006 workloads are chosen based on the memory intensity. The RPKI and WPKI (i.e., memory reads/writes per kilo instructions) of the workloads are listed in Table 5. In our evaluation, we adopt both single node (with one server node) and multiple server nodes (with three server nodes) tests. This paper focuses on the distributed storage architecture adopted in Hadoop clusters, where storage and computing are integrated in a server node [3, 23, 32]. Each server node is equipped with SSD, HDD, and NVDIMM. In this kind of systems, NVDIMM devices can be deployed with server nodes to speed up VM execution. Under this architecture, normal IO data are moved from PCIe devices to local memory/cache, while for NVDIMM devices, data are delivered via shared memory channels. We compare the proposed techniques with various representative heterogeneous storage management mechanisms, including BASIL [9], Pesto [10], and LightSRM [52].

## 6.2 Experimental Results

We initially assign workloads to servers randomly, but in a greedy manner so as to keep a space-balanced arrangement. This allows us to focus on evaluating performance management algorithms. After running the workloads in the new device for four hours, the baseline schemes recommended three additional workload movements.

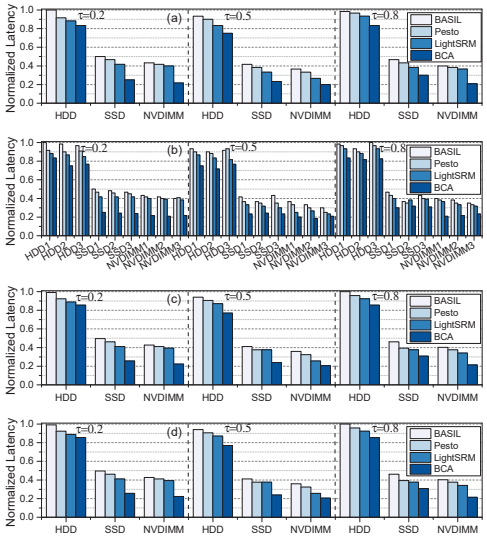


**Figure 12: The device performance of (a) the big data workloads mixed with 429.mcf (single node), (b) the big data workloads mixed with 429.mcf (multiple nodes), (c) the big data workloads mixed with 470.lbm (single node), and (d) the big data workloads mixed with 433.milc (single node).**

*6.2.1 Bus Contention Aware (BCA) Management.* Figure 12 illustrates the normalized device latency. These results are normalized to the slowest device latency. The initial placement of each run is not shown, since nothing of interest was observed during this warm-up period. The plotted data cover the time periods that contain all data migrations from the beginning to the end. The performance improvement is up to 33% (28% on average), up to 28% (23% on average), and up to 24% (16% on average) compared with BASIL, Pesto, and LightSRM, respectively, when the big data workloads run in a single node. The experimental results for the multiple nodes test exhibit the similar trend with an average performance improvement of 25% compared all the baseline schemes. Note that BCA is effective for HDD/SSD. This is because BCA can reduce the unnecessary data migration from/to HDD/SSD, thus enhancing the performance of HDD/SSD.

We further vary the migration threshold $\tau$ (a parameter in Section 5.1.2) from 0.2 to 0.8 to evaluate its impact. With the increase of $\tau$, the total migration overhead decreases. However, not all the device performance becomes better, since the device performances become less balanced when $\tau$ is large. In the rest of this section, $\tau$ is set to be 0.5.

It can also be observed that with the decrease of the memory intensity, the benefits of the proposed scheme decrease from 26% (mixed with 429.mcf shown in Figure 12 (a)) to 17% (mixed with 433.milc shown in Figure 12 (d)) on average. The reason is that lower memory intensity can make the state-of-the-art schemes more accurately predict the NVDIMM performance, thus resulting in reducing unnecessary data movements. This also demonstrates the bus contention plays an important role in the NVDIMM management. The impact of the bus contention on big data workloads mixed with 429.mcf is discussed in Section 6.2.4.

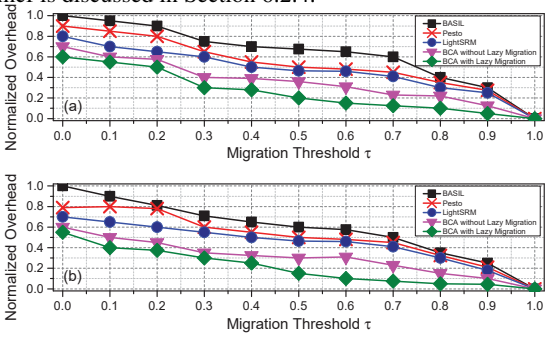


**Figure 13: The migration overhead with a single node ((a)) and multiple nodes ((b)) tests.**

*6.2.2 Lazy Data Migration.* This section shows the results to evaluate our lazy data migration. The total normalized migration time is shown in Figure 13. In a single server node with the proposed bus contention aware scheme (BCA without Lazy Migration), the migration overhead can be reduced by 44%, 33%, and 24% on average compared with BASIL, Pesto, and LightSRM, respectively. These baselines ignore the bus contention, which leads to inaccurately predict the NVDIMM performance and thus results in enlarging the data migration. As expected, increasing the migration threshold can reduce the migration overhead since it greatly reduces the triggering frequency of data migration. However, as discussed in Section 6.2.1, when the threshold becomes larger, devices performances are more unbalanced; thus, the overall system performance may not be improved.

Our lazy migration schemes can help reduce the migration time. As shown in Figure 13, the migration time can be reduced by 27% on average compared with the bus contention aware design without lazy migration (BCA without Lazy Migration). These achievements benefit from the proposed I/O mirroring migration and cost-benefit guided migration. The I/O mirroring technique can effectively reduce the data copy, thus reducing the migration time. The cost-benefit guided migration can help eliminate the resource contention (e.g., memory bus contention) and thus accelerate the migration procedure.

*6.2.3 Architectural Optimization.* In Figure 14, we compare the baseline with a system that adopts the first scheduling policy (Policy One without barriers), a system that adopts the second scheduling policy (Policy Two with lower priority scheduling for migrated data), and a system that adopts both Policy One and Policy Two. We use the normalized speedup [51] as the metric to show the performance improvement for big data workloads. We obtain the I/O throughput that is the number of executed I/O operations per cycle for each workload, and calculate the average I/O throughput across all the workloads. The normalized speedup is obtained by dividing the I/O throughput with the average throughput.

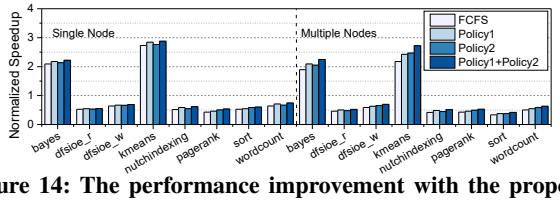


**Figure 14: The performance improvement with the proposed scheduling policies.**

We first focus on evaluating the benefits of the first scheduling policy. It can be observed that the average performance improvement of the big data workloads is up to 16% (8% on average) compared with the baseline. With *Policy Two*, the performance is improved by up to 17% (7% on average). If *Policy One* and *Policy Two* are both adopted, the performance is improved by up to 25% (14% on average). These results demonstrate that the proposed two scheduling policies play different roles in accelerating the performance of big data workloads and interact with each other.

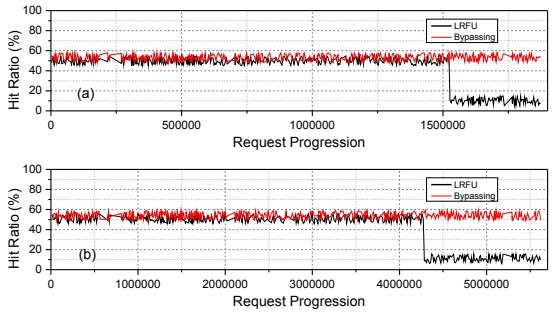


**Figure 15: The cache bypassing effect with a single node ((a)) and multiple nodes ((b)) tests.**

The experimental results in Figure 15 show the NVDIMM buffer cache hit ratio. Note that with the LRFU [8] cache management, the cache hit ratio drops dramatically with lower than 18% after the NVDIMM receives 150000 requests with a single server node test. This is caused by the data migration. The huge amount of data migrated from the NVDIMM leads to valid data being evicted from the buffer cache. On the other hand, with our bypassing mechanism, even though the data migration occurs, the cache hit ratio is stable.

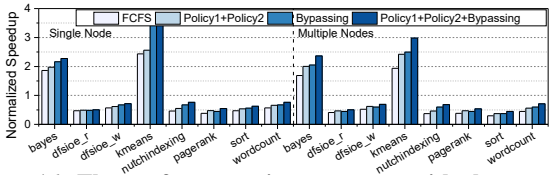


**Figure 16: The performance improvement with the proposed scheduling policies and bypassing scheme.**

Note that the migration-aware scheduling policy and bypassing cache management benefit the NVDIMM architecture from different aspects. The former increases the parallelism, while the later enhances the cache hit ratio. Combing these two techniques should archive more benefits. Figure 16 shows that the overall performance can be enhanced by up to 45% (32% on average).

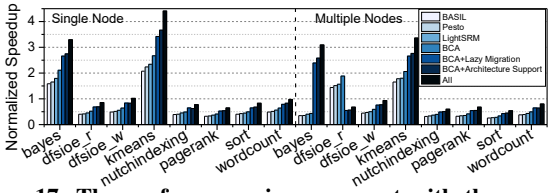


**Figure 17: The performance improvement with the proposed schemes.**

*6.2.4 Putting It All Together.* Bus contention aware management, lazy migration, and architectural optimization for load balancing benefit the NVDIMM architecture from different aspects. Figure 17 demonstrates that integrating all proposed techniques significantly improves the performance by up to 98% (87% on average) compared with BASIL. Putting all techniques together also leads to a performance gain of 59% compared with only employing the proposed BCA technique. Since the proposed schemes can effectively reduce unnecessary data migration caused by the bus contention, the proposed design can reduce the migration impact on the memory-intensive SPEC workload.

## 7 CONCLUSIONS

This paper presents novel NVDIMM-based storage management techniques with architectural optimization to overcome I/O bottleneck for big data workloads by maximally optimizing data management under the heterogeneous storage environment. We observed that naively adding NVDIMMs in servers yields suboptimal results due to inaccurately predicting NVDIMM performance and unnecessary data migrations from/to NVDIMM devices. In order to fully utilize NVDIMMs, we proposed three techniques, namely, bus contention aware management, lazy data migration, and architecture support for migration. Experiments based on big data workloads demonstrate that the proposed design achieves up to an 98% performance improvement over the baseline schemes. We expect better results can be obtained by implementing our techniques on Linux with DAX [46] in which the NVDIMM performance is enhanced with the native memory support.

## 8 ACKNOWLEDGMENTS

The work described in this paper is partially supported by the grants from the National Natural Science Foundation of China (61702357 and 61672116), Shenzhen Science and Technology Foundation (JCYJ20170816093943197 and JCYJ20170817100300603), Natural Science Foundation of Tianjin (18JCQNJC00300), Peiyang Scholar Foundation of Tianjin University (2019XRG-0004), the Research Grants Council of the Hong Kong Special Administrative Region, China (GRF 15222315, GRF 15273616, GRF 15206617, GRF 15224918), and Direct Grant for Research, The Chinese University of Hong Kong (Project No. 4055096).

# REFERENCES


[1] Amir Ban. 1998. Flash-memory translation layer for NAND flash (NFTL). *M-systems* (1998).

[2] Fabrice Bellard. 2005. QEMU, A fast and portable dynamic translator. In *USENIX Annual Technical Conference*. 41–46.

[3] Eric Boutin, Paul Brett, Xiaoyu Chen, Jaliya Ekanayake, Tao Guan, Anna Korsun, Zhicheng Yin, Nan Zhang, and Jingren Zhou. 2015. JetScope: Reliable and interactive analytics at cloud scale. *Proceedings of the VLDB Endowment* 8, 12 (2015), 1680–1691.

[4] Renhai Chen, Zhiwei Qin, Yi Wang, Duo Liu, Zili Shao, and Yong Guan. 2015. On-demand block-level address mapping in large-scale NAND flash storage systems. *IEEE Trans. Comput.* 64, 6 (2015), 1729–1741.

[5] Renhai Chen, Zili Shao, and Tao Li. 2016. Bridging the I/O performance gap for big data workloads: A new NVDIMM-based approach. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 421–432.

[6] Jeremy Condit, Edmund B. Nightingale, Christopher Frost, Engin Ipek, Benjamin Lee, Doug Burger, and Derrick Coetzee. 2009. Better I/O through byte-addressable, persistent memory. In *Proceedings of the ACM SIGOPS 22th Symposium on Operating Systems Principles (SOSP)*. 133–146.

[7] Peter Desnoyers. 2014. Analytic models of SSD write performance. *ACM Transactions on Storage* 10, 2 (2014), 8:1–8:25.

[8] Donghee Lee, Jongmoo Choi, Jong-Hun Kim, Sam H. Noh, Sang Lyul Min, Yookun Cho, and Chong Sang Kim. 2001. LRFU: A spectrum of policies that subsumes the least recently used and least frequently used policies. *IEEE Trans. Comput.* 50, 12 (2001), 1352–1361.

[9] Ajay Gulati, Chethan Kumar, Irfan Ahmad, and Karan Kumar. 2010. BASIL: Automated IO load balancing across storage devices. In *8th USENIX Conference on File and Storage Technologies (FAST)*, Vol. 10. 169–182.

[10] Ajay Gulati, Ganesha Shanmuganathan, Irfan Ahmad, Carl Waldspurger, and Mustafa Uysal. 2011. Pesto: Online storage performance management in virtualized datacenters. In *Symposium on Cloud Computing (SOCC)*. 19:1–19:14.

[11] John L. Henning. 2006. SPEC CPU2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News* 34, 4 (2006), 1–17.

[12] Jian Huang, Karsten Schwan, and Moinuddin K. Qureshi. 2014. NVRAM-aware logging in transaction systems. *VLDB Endowment* 8, 4 (2014), 389–400.

[13] Shengsheng Huang, Jie Huang, Jinquan Dai, Tao Xie, and Bo Huang. 2010. The HiBench benchmark suite: Characterization of the MapReduce-based data analysis. In *International Conference on Data Engineering Workshops (ICDEW)*. 41–51.

[14] IBM. 2012. SAP In-Memory computing on IBM eX5 systems. *http://www-07.ibm.com/au/hana/pdf/redp4814.pdf* (2012).

[15] IBM. 2014. IBM flash system technical whitepaper. *http://www-01.ibm.com/support/docview.wss?uid=tss1wp102489&aid=1* (2014).

[16] IBM/Lenovo. 2014. Benefits of IBM/Lenovo eXFlash memory-channel storage in enterprise solutions. *Lenovo Press* (2014).

[17] IBM/Lenovo. 2016. eXFlash DDR3 storage DIMMs. *https://lenovopress.com/tips1141.pdf* (2016).

[18] IBM/Lenovo. 2017. The System x3530 M4 7160 server. *http://systemx.lenovofiles.com/help/index.jsp?topic=%2Fcom.lenovo.sysx.7160.doc%2Fintroduction.html* (2017).

[19] IBM/Lenovo. 2018. exFlash 400GB DDR3 DIMM. *https://www.amazon.com/Exflash-400GB-DDR3-Storage-00FE005/dp/B00IY651DS/ref=sr_1_2?ie=UTF8&qid=1491052510&sr=8-2&keywords=exflash* (2018).

[20] Intel. 2017. Open storage toolkit from Intel labs. *https://sourceforge.net/p/intel-iscsi/wiki/Home/* (2017).

[21] Myoungsoo Jung, Ellis H. Wilson, David Donofrio, John Shalf, and Mahmut T. Kandemir. 2012. 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)*. 1–12.

[22] Aasheesh Kolli, Steven Pelley, Ali Saidi, Peter M. Chen, and Thomas F. Wenisch. 2016. High-performance transactions for persistent memories. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 399–411.

[23] Marcel Kornacker, Alexander Behm, Victor Bittorf, Taras Bobrovytsky, Alan Choi, Justin Erickson, Martin Grund, Daniel Hecht, Matthew Jacobs, Ishaan Joshi, Lenni Kuff, Dileep Kumar, Alex Leblang, Nong Li, Henry Robinson, David Rorke, Silvius Rus, John Russell, Dimitris Tsirogiannis, Skye Wanderman-milne, and Michael Yoder. 2015. Impala: A modern, open-source sql engine for hadoop. In *Biennial Conference on Innovative Data Systems Research (CIDR)*.

[24] Junghee Lee, Youngjae Kim, Galen M. Shipman, Sarp Oral, and Jongman Kim. 2013. Preemptible I/O scheduling of garbage collection for solid state drives. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 32, 2 (2013), 247–260.

[25] Lenovo. 2017. In-memory computing with SAP HANA on Lenovo X6 systems. *https://lenovopress.com/sg248086-in-memory-computing- with-sap-hana-on-lenovo-x6-systems* (2017).

[26] Duo Liu, Kan Zhong, Tianzheng Wang, Yi Wang, Zili Shao, Edwin H. Sha, and Jingling Xue. 2017. Durable address translation in PCM-based flash storage systems. *IEEE Transactions on Parallel and Distributed Systems* 28, 2 (2017), 475–490.

[27] Ren-Shuo Liu, De-Yu Shen, Chia-Lin Yang, Shun-Chih Yu, and Cheng-Yuan Michael Wang. 2014. NVM Duet: Unified working memory and persistent store architecture. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 455–470.

[28] Youyou Lu, Jiwu Shu, Long Sun, and Onur Mutlu. 2014. Loose-ordering consistency for persistent memory. In *IEEE 32nd International Conference on Computer Design (ICCD)*. 216–223.

[29] Douglas C. Montgomery, Elizabeth A. Peck, and G. Geoffrey Vining. 2015. Introduction to linear regression analysis. (2015).

[30] Abdullah Muzahid, Shanxiang Qi, and Josep Torrellas. 2012. Vulcan: Hardware support for detecting sequential consistency violations dynamically. In *45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 363–375.

[31] Dushyanth Narayanan and Orion Hodson. 2012. Whole-system persistence. In *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 401–410.

[32] Shadi A. Noghabi, Kartik Paramasivam, Yi Pan, Navina Ramesh, Jon Bringhurst, Indranil Gupta, and Roy H. Campbell. 2017. Samza: Stateful scalable stream processing at LinkedIn. *VLDB Endowment* 10, 12 (2017), 1634–1645.

[33] Heejin Park, Jaeho Kim, Jongmoo Choi, Donghee Lee, and Sam H. Noh. 2015. Incremental redundancy to reduce data retention errors in flash-based SSDs. In *Symposium on Mass Storage Systems and Technologies (MSST)*. 1–13.

[34] Avadh Patel, Furat Afram, Shunfei Chen, and Kanad Ghose. 2011. MARSS: A full system simulator for multicore x86 CPUs. In *Proceedings of the 48th Design Automation Conference (DAC)*. 1050–1055.

[35] Zhiwei Qin, Yi Wang, Duo Liu, Zili Shao, and Yong Guan. 2011. MNFTL: An efficient flash translation layer for MLC NAND flash memory storage systems. In *48th ACM/EDAC/IEEE Design Automation Conference (DAC)*. 17–22.

[36] Jinglei Ren, Jishen Zhao, Samira Khan, Jongmoo Choi, Yongwei Wu, and Onur Mutlu. 2015. ThyNVM: Enabling software-transparent crash consistency in persistent memory systems. In *Proceedings of the 48th International Symposium on Microarchitecture (MICRO)*. 672–685.

[37] Scott Rixner, William J Dally, Ujval J Kapasi, Peter Mattson, and John D Owens. 2000. Memory access scheduling. 28, 2 (2000).

[38] Paul Rosenfeld, Elliott Cooper-Balis, and Bruce Jacob. 2011. DRAMSim2: A cycle accurate memory system simulator. *Computer Architecture Letters* 10, 1 (2011), 16–19.

[39] ARTHUR SAINIO. 2016. NVDIMM-changes are here so what's next? *In-Memory Computing Summit* (2016).

[40] Samsung. 2018. Samsung 970 PRO 512GB - NVMe PCIe M.2 2280 SSD. *https://www.amazon.com* (2018).

[41] George AF Seber and Alan J Lee. 2012. *Linear regression analysis*. Vol. 936. John Wiley & Sons.

[42] Toshiba. 2018. Toshiba P300 3TB desktop 3.5 inch SATA 6Gb/s 7200rpm internal hard drive. *https://www.amazon.com/* (2018).

[43] VMware. 2009. Live migration for virtual machines without service interruption. *Product Datasheet* (2009), 1–2.

[44] Haris Volos, Andres Jaan Tack, and Michael M. Swift. 2011. Mnemosyne: Lightweight persistent memory. In *Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 91–104.

[45] Chundong Wang, Qingsong Wei, Jun Yang, Cheng Chen, and Mingdi Xue. 2015. How to be consistent with persistent memory? an evaluation approach. In *International Conference on Networking, Architecture and Storage*. 186–194.

[46] Matthew Wilcox. 2014. DAX: Page cache bypass for filesystems on memory storage. (2014).

[47] Michael Wu and Willy Zwaenepoel. 1994. eNVy: A non-volatile, main memory storage system. In *Proceedings of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 86–97.

[48] Jun Yang, Qingsong Wei, Cheng Chen, Chundong Wang, Khai Leong Yong, and Bingsheng He. 2015. NV-Tree: Reducing consistency cost for NVM-based single level systems. In *13th USENIX Conference on File and Storage Technologies (FAST)*. 167–181.

[49] Matt T. Yourst. 2007. PTLsim: A cycle accurate full system x86-64 microarchitectural simulator. In *IEEE International Symposium on Performance Analysis of Systems Software (ISPASS)*. 23–34.

[50] Chi Zhang, Yi Wang, Tianzheng Wang, Renhai Chen, Duo Liu, and Zili Shao. 2014. Deterministic crash recovery for NAND flash based storage systems. In *Design Automation Conference (DAC)*. 148:1–148:6.

[51] Jishen Zhao, Onur Mutlu, and Yuan Xie. 2014. FIRM: Fair and high-performance memory control for persistent memory systems. In *47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 153–165.

[52] Ruijin Zhou, Huixiang Chen, and Tao Li. 2015. Towards lightweight and swift storage resource management in big data cloud era. In *Proceedings of the 29th ACM on International Conference on Supercomputing (ICS)*. 133–142.