Towards Software Defined Persistent Memory: Rethinking Software Support for Heterogenous Memory Architectures

Swaminathan Sundararaman
SanDisk Corporation

Nisha Talagala *
Parallel Machines

Dhananjoy Das
Sandisk Corporation

Amar Mudrankit*
Yahoo, Inc.

Dulcardo Arteaga*
Florida International University

Abstract

The emergence of persistent memories promises a sea-change in application and data center architectures, with efficiencies and performance not possible with today's volatile DRAM and persistent slow storage. We present Software Defined Persistent Memory, an approach that enables applications to use persistent memory in a variety of local and remote configurations. The heterogeneity is managed by a middleware that manages hardware specific needs and optimizations. We present the first ever design and implementation of such an architecture, and illustrate the key abstractions that are needed to hide hardware specific details from applications while exposing necessary characteristics for performance optimization. We evaluate the performance of our implementation on a set of microbenchmarks and database workloads using the MySQL database. Through our evaluation, we show that it is possible to apply Software Defined concepts to persistent memory, to improve performance while retaining functionality and optimizing for different hardware architectures.

1. Introduction

For decades, application performance has been defined by the use of two resources, memory and storage. Memory was "fast", accessed directly via CPU load/store semantics that bypassed the OS, and was volatile. Storage was "slow", accessed via the OS through system calls such as read() and write(), and was persistent. Applications brought their data in from storage, operated on it in memory, and put the result back in storage.

The advent of flash began the blurring of storage and memory through an intermediate tier that was substantially faster than disk but also much slower than DRAM. The expected arrival of media such as Phase Change Memory, Resistive RAM and Memristor, will drive further convergence of storage and memory [40]. Since these memories are expected to have latencies low enough to enable direct CPU load/store access, software can now access storage with memory semantics combined with persistence [34, 39]. With such memories, applications can expect substantial performance benefits [37].

However, these new media tiers further complicate system configurations since they do not directly replace existing tiers of DRAM, Flash, or HDDs. Persistent Memory (PM) is expected to be slower and cheaper than DRAM, and faster but possibly more expensive than flash. Given this expansion of media tiers, software now faces greater design complexity. Applications will need to answer questions such as: which tiers should be optimized for under what circumstances, and what will performance be when the physical machine running the application does not have memory/storage of the expected type or size, etc.? Prior to the arrival of PM, the storage industry has driven the *Software Defined Storage* initiative that uses middleware to enable optimal application behavior on varied hardware configurations of flash, disk and networking. We argue that the arrival of PM will require a similar **Software Defined Persistent Memory (SDPM)** focus, and describe an initial approach to this problem.

We believe that PM will need middleware support due to the expected variants of hardware. At present, the industry appears to be converging on several key defining

* Work done while at SanDisk Corporation

INFLOW'15, October 4, 2015, Monterey, CA.
Copyright © 2015 ACM 978-1-4503-3945-2/15/10…$15.00.
http://dx.doi.org/10.1145/2819001.2819004

characteristics of PM: direct CPU load/store access, byte grained updates and extremely low latency [7, 11, 29]. Beyond these elements, however, there are many variables. Exact performance, capacity, and endurance characteristics are unclear, with different studies highlighting a range of expectations [30, 45, 48]. Furthermore, to accommodate a variety of cost/capacity/performance trade-offs, tiering between PM and flash may be required. Attach points are also open for debate, with memory bus access being proposed by some and I/O bus access (like PCIe) or remote access (over RDMA) being proposed by others [3, 13, 40].

For our SDPM architecture, we show how six hardware configurations of PM that all meet the defining criteria above can be seamlessly accessed by an application with no custom changes for each hardware configuration. Our architecture enables DDR attached local PM, PCIe attached local PM, RDMA or Ethernet accessed remote DDR attached PM, and RDMA or Ethernet accessed remote PCIe attached PM. Our architecture also supports dynamic tiering between all of these forms of PM and Flash. For our prototype and benchmarking, we use PM hardware, based on power fail protected DRAM, that is commercially available today in both DDR attach (Type N NVDIMMs) and PCIe attached variants [3, 6].

The contributions of this paper are as follows. First, to the best of our knowledge, we present the first middleware architecture that manages multiple PM hardware types with different attach points and performance characteristics. Second, we outline the design trade-offs required for making PM "Software Defined", i.e., how to abstract key hardware details while enabling applications to optimize for others. Third, we describe a cost-effective, pragmatic, and immediately deployable architecture that tiers PM with flash while retaining a seamless data management model. Fourth, we optimize a database application (MySQL) to use SDPM through a single unified API and describe the impact of different hardware attach points (including an efficient PCIe attach) on both base latency and application level performance – to our knowledge, the first study to do so comprehensively.

The rest of the paper is as follows. Section 2 describes related work. Sections 3 and 4 describes the assumptions that drove our architectural decisions and our SDPM design, respectively. Section 5 describes a case study with MySQL, a widely deployed database. Section 6 evaluates the performance of our SDPM implementation with different PM architectures. Sections 7 and 8 discusses our observations and possible architectural extensions and summarizes our learnings, respectively.

2. Related Work

The Software Defined approach is now a popular trend, having already been applied to networking, storage, and data center design. In the storage context, all major storage vendors provide products to decouple storage management from hardware specifics [42]. Our work, though similar in spirit, is complementary to these initiatives since we focus on managing PM and driving decoupling of PM hardware and attach points from end applications. With respect to flash specifically, caching software enables seamless management of flash/disk hybrids while appliances support tiering of both flash and disk [4].

PM is expected to have access latencies from 100ns-500ns and support byte addressability [29]. Researchers have proposed file systems that exploit PM [24, 26, 43, 47] and optimizations to existing file systems [21]. Recent work in the Linux community includes DAX [44] to export PM devices as virtual memory via a file. Also, whole system persistence has also been explored [33].

Recently a variety of hardware architectures have emerged to connect persistence across DDR, with Type N NVDIMMs providing the load/store PM access using DRAM with flash backing. Multiple vendors provide these devices and various studies have used this hardware as a placeholder for emerging memories. PMs using PCIe attach have been explored by researchers and also demonstrated by Fusion-io [3].

New data structures, programming models, and libraries have also been proposed to combine persistence with memory semantics [19, 20, 23, 43]. New CPU instructions have been proposed in several studies [23, 24] and Intel has announced the availability of new instructions in upcoming processors for PM [10]. Underlying most of this work is the fundamental assumption that user space applications will be able to access PM through a memory interface and bypass the OS. I/O models are also being supported for legacy backward compatibility [11, 12]. The SNIA Non Volatile Memory Programming Technical Working Group has formalized these two access models as part of their 1.0 NVM Programming Specification [7]. These efforts add to earlier work in recoverable virtual memories that was based on DRAM and disk architectures [27, 36].

Previous studies have shown the performance potential of PM in varied capacities [29, 32, 46]. Some studies focus on the benefits of fast persistence while others focus on the DRAM displacement possible with slower but cheaper PM.

3. The Need for Software Defined Persistent Memory

Beyond the complexity caused by the addition of a new storage/memory tier, other factors necessitate a software defined approach for practical wide scale deployment of PM. Here we discuss these factors and our assumptions.

Seamless Storage Management: We believe that data stored on PM will require the same management (such as compliance, audit, etc.) as data stored on other persistent media. Such needs are driven by the nature of the data, domain specific requirements on security and retention, and are not necessarily impacted by media type or performance. Unlike classic volatile memory, PM requires a persistent namespace to recover data after a reboot. The memory map (mmap) model has appealed to industry and researchers alike because it enables the ability to access the data with memory operations while retaining a familiar persistent file namespace (the most commonly used data management model by data center administrators). We therefore assume that the PM storage management should be done via a File System (FS) namespace.

Varied Attach Points and Form Factors: Since PM is still emerging technology, much is unknown about the media and the ideal attach point for different server platforms. While memory bus attaches will likely have optimal latencies, storage deployment history has shown that attach points have practical concerns unrelated to performance. For example, while a DDR attach can be ideal from a performance perspective, depending on the number of available DDR slots and the required mix of DRAM and PM [14], a server may not be able to support enough of both. Similarly, some systems have minimal PCIe slots [8], making the PCIe interface undesirable for large amounts of PM. Other desirable characteristics, such as dual port, are more realistically achievable in PCIe than in DDR. While hot-plug DDR technologies exist [22], other attach points are better able to support this function [35]. Finally, the emerging trend of disaggregation may drive top of rack style memory/storage solutions which optimize efficiency at rack level rather than server node level [2]. Given these factors, we assume that an ideal SDPM architecture should be extensible to any local or remote physical attach points.

Byte Addressability: Byte addressability in PM can enable new usages not possible with block based storage [21, 34, 39]. Some attach protocols, such as NVM Express, may not support byte operations but for other practical reasons may be the only available access mode for some servers. As such, we believe it is valuable for an SDPM architecture to support the levels of byte addressability possible with each hardware architecture and emulate this capability where needed.

Support Local and Network Attached Hardware: The emerging disaggregation trend suggests that server configurations may have locally attached or top of rack persistence. The top of rack persistence may be accessed via a range of options from PCIe to networking. An ideal SDPM architecture should support both local and remote access to PM and optimize under the covers for each.

Varied Cost/Performance Configurations: While it is appealing to think of all data in PM, data growth rates are so substantial that many deployments require both high performance and low cost media. Therefore, an ideal SDPM architecture should accommodate tiering between PM and flash. Moreover, PM specifications suggest performance lower than DRAM, so we support caching of volatile data in DRAM as needed [29].

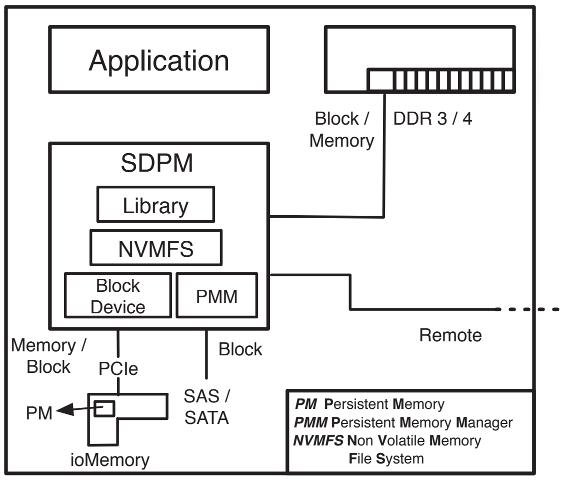


Figure 1. SDPM Architecture

4. Software Defined PM

In this section we discuss design goals and describe our architecture and prototype implementation.

4.1 Design Goals

Based on the needs of practical deployments as outlined in Section 3, we define our goals for SDPM as follows:

Goal 1: Support both memory and I/O access, traditional storage management, and persistence guarantees to combine the best of memory and storage worlds.

Goal 2: Support a variety of local and remote attach points with differing performance but identical functionality and semantic guarantees. In particular, our goal is to provide byte addressability at the application level and optimal performance for each hardware configuration. The low latencies of PM require that software be as thin as possible. Given this, SDPM should provide load/store access to PM whenever possible and minimal overhead approaches otherwise.

Goal 3: Enable tiering of data between PM and flash, with caching in DRAM, to enable different cost/performance configurations.

Goal 4: Provide a single programming model to the application. Applications cannot reasonably be expected to implement different code for each possible server hardware configuration. In SDPM, applications should be able to choose a single API, which is adapted optimally for each hardware configuration with identical persistence and semantic guarantees.

4.2 Design Overview

Figure 1 describes the high level design of SDPM. The design has four key elements: a FS for namespace and tiering, layered programming language libraries for unified but flexible APIs, hardware and location abstraction for hiding PM attach point and location details.

4.2.1 Non Volatile Memory File System (NVMFS)

The SDPM architecture has an FS at its center. For our prototype implementation, we select NVMFS, a flash-optimized POSIX compliant Linux FS [9]. We extended NVMFS to support both PM and flash devices. Given our focus on PM, we do not describe the flash aspects of NVMFS (details available here [25, 28]). We now describe how NVMFS meets many of our design goals.

NVMFS provides a unified and persistent namespace to both PM and flash. Through this namespace, NVMFS transparently provides both memory and I/O access to PM, enabling applications to flexibly and seamlessly switch between different access modes to data in PM. The unified persistent namespace also enables transparent application acceleration where existing applications can benefit from the FS tiering of PM and flash.

We build upon historical mmap to provide direct (or raw) mmap to PM via NVMFS. All FSes (including NVMFS) provide load/store access through conventional mmap where data is first cached in DRAM before it is persisted to the storage media. In contrast, direct mmap directly accesses data in PM without caching in DRAM [7]. With direct mmap, a user space application can map a file into its virtual address space and perform memory operations which translate into loads and stores directly to the PM. On crash/restart, applications can remap the file using direct mmap and resume access.

We support tiering in NVMFS by maintaining the media location of each data block within the FS inodes. Data access via a unified namespace combined with enclosing data location within an inode enables NVMFS to transparently move data between PM and flash. We also implement several well known dynamic tiering algorithms that optimize for common access patterns (such as log files) and data liveliness (i.e., retention requirements).

We also support a single programming model via a combination of application specific libraries over direct mmap and transparent access via POSIX APIs that support both memory and block I/O access to data. This approach has the added benefit that applications that currently use a FS do not need to be modified to work with PM. On the flip side, it does force these application to go through the OS to get to the FS. This additional code traversal can reduce the performance benefits [19, 39] for transparent acceleration as compared to direct PM access.

It is worth noting that we have selected a FS namespace over a variety of other possible models such as object namespaces, key value stores, compiler supported persistent variables etc. We believe that the file namespace is a good candidate as a base namespace for several reasons. First, it is by far the most common namespace used by system administrators, implying that existing data management practices that rely on files can adopt PM with minimal or no changes. Second, using direct mmap, any other type of namespace can be built atop the FS using direct load/store access (the file namespace enables and does not replace all other namespaces). Third, since FSes manage disk and flash today, PM can be added to the mix seamlessly with the benefits of both traditional APIs and direct mmap used as needed.

4.2.2 Programming Libraries

Programming libraries can provide a unified access API to applications. Such libraries can be implemented in user space and communicate with NVMFS for management. Many APIs are possible and the intent of this paper is not to advocate for a specific API but rather for the layering approach itself. Moreover, libraries can provide optimized data structures, language support, memory management, error checking and other runtime functions. Example libraries are Oracle's NVM-Direct which provides transactional and data structure support for PM [20]. We also implemented a small log acceleration library which is used in our experiments (see Section 5).

NVMFS enables optimal implementation of such libraries by providing direct mmap() to map the PM directly to virtual address space, msync() to flush data to the persistence domain, and two additional operations to optimize CPU cache flush (barrier()), and to control memory mapping types. NVMFS also provides options to libraries to guide file data placement across the two media types.

4.2.3 Hardware Abstraction

All PM hardware is discovered and managed by the Persistent Memory Manager (PMM). PMM is a dedicated kernel module that presents both memory pages and data blocks for use by NVMFS. In our prototype SDPM system, PMM supports both DDR and PCIe attaches for PM. The PM is exposed to the user space libraries via NVMFS as virtual memory space. PMM is also responsible for setting memory caching types, guaranteeing persistence, and free space management for PM.

Memory Mapping Types. PM can have different memory mappings depending on the hardware. For example, the OS supports write back, write through, write combining, and uncached for DDR attached PM but only a subset are supported for PCIe MMIO attached PM. SDPM is responsible for ensuring that each memory type is default mapped to the optimal model possible for its physical attach (in our prototype, write combining is the default mode). We also enable FS operations that allow the application to control the per file memory mapping.

Guaranteeing Persistence. Given the different ways to map persistent memory by applications, we need mechanism(s) to guarantee all in-flight data (such as in CPU caches, registers, etc.) have reached the PM device independent of its attach point. For example, depending on the memory mapping model, certain CPU instructions will en-

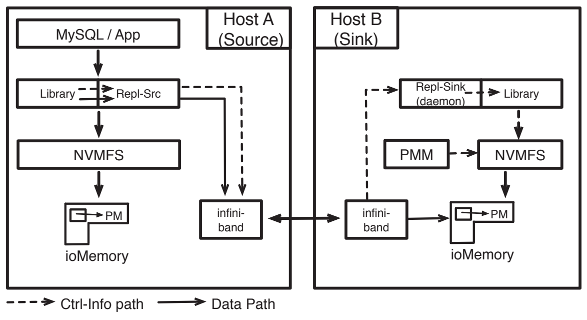


Figure 2. Remote attached Persistent Memory

sure flushes of the intermediate CPU caches. The further mechanism for ensuring persistence for DDR is Advanced Dynamic Refresh (ADR) [5]. ADR ensures that data flushed from CPU caches through CLFLUSH and other methods is driven from any intermediate memory buffers to PM at power cut. Since ADR support does not extend to PCIe MMIO based PM, we defined and implemented a custom protocol to ensure similar support with the cooperation of the PCIe device. In direct mmap mode, at msync(), NVMFS is responsible for guaranteeing persistence. PMM provides a barrier() operation to NVMFS and user space libraries that employ their own CPU flush operations. When barrier() is invoked, SDPM will ensure data is moved to the persistence domain as needed for the attach point.

4.2.4 Abstracting Local and Remote Attach Points

PM can be attached either locally or on a remote machine. SDPM abstracts different attach points that can be used within a server. As long as PM is both discovered and mapped, it can be used by local or remote applications.

Figure 2 shows the remote attach model for our SDPM prototype. Two instances of the library are required. We refer to the local instance (where the application is running) as the *s*ource, and the remote instance (where the PM is located) as the *s*ink. The client (source) exposes the same APIs as that in the local attach case. The library instance on source translates these API calls into a network transport (either Ethernet or Infiniband in our implementation) to a remote instance of the library on the sink. The sink hosts a local instance of NVMFS which can support all of the capabilities and hardware configurations outlined in the above design sections. Note that data replication such as in Mojim is also possible using our SDPM architecture [47].

When SDPM is used over Ethernet, we employ sockets over TCP/IP to communicate data between nodes. When used over Infiniband, direct OS-bypass communication is used with RDMA write operations for the data. SDPM enables additional optimizations such as direct transfer of data between memory and network through third party PCIe DMA when PCIe based PM is used. Our architecture enables these operations to occur dynamically as possible for each hardware configuration without the application needing to do specific optimizations.

There are several other SDPM considerations that apply specifically for remote attaches. For example, different memory types exist at the destination (sink) node as in the local attach case. When DDIO is enabled, data may need to be explicitly flushed on the sink to before the PM update can be positively acknowledged [13]. In the case of PCIe MMIO, a direct PCIe to PCIe transfer is possible from NIC to PM hardware. This avoids any traversal of data to the CPU/memory complex on the sink node.

In all cases, the RDMA write operation only guarantees that data has been sent from the receiving NIC (on the sink) to the memory location. To meet the persistent store guarantee, we need to ensure that the data was in fact written to the ADR boundary on the sink node. This requires a subsequent RDMA read to ensure that the data was fully flushed to the end memory location.

5. Case Study: MySQL

We use MySQL (a database application) to show the applicability and capability of our SDPM architecture [1]. We chose MySQL as it is the most popular and fastest growing open source database today.

Prior studies have shown that databases can expect substantial performance improvements by using PM and that the degree of improvement will depend on the amount of PM available [38, 41]. Since PM hardware based on new memory technologies like PCM are not yet widely available, we focus our application example on small amounts of PM that can be purchased in hardware today (such as Type N NVDIMMs). For a small amount (1-10GB) of such memory, the ideal database use case is log acceleration. Log acceleration is also an important focus for database researchers since transaction performance can be limited by log write (i.e., log persistence) latencies.

MySQL allows for pluggable storage engines. InnoDB (and its variant XtraDB) is the most widely used production storage engine due to its reliability and performance [17]. The InnoDB storage engine uses two logs, the Transaction Log (TxLog, also sometimes referred to as the Redo Log) and the Binary Log (BinLog) [15, 16]. Updates to both types of logs gate the performance of MySQL transaction commits. Hence, our PM integration focuses on placing log updates in low latency PM for automatic later destage to flash if needed. The logs are stored as normal files on NVMFS and the SDPM architecture is responsible for tiering parts of the files into flash when they are not being actively updated. This enables SDPM to use small amounts of PM for our experiments and have log files of arbitrary size (which is particularly useful for BinLogs whose size is not pre-determined).

To integrate SDPM with MySQL, we wrote a small library that uses the direct mmap() mechanism to access part

Config	Persistent Memory Attach	Local or Remote
config-1	DDR NVDIMM	Local
config-2	PCIe MMIO	Local
config-3	DDR NVDIMM	Remote Ethernet
config-4	PCIe MMIO	Remote Ethernet
config-5	DDR NVDIMM	Remote Infiniband
config-6	PCIe MMIO	Remote Infiniband

Table 1. Persistent Memory configuration.

local	**config-1, config-2**
System Configuration	Local: HP DL380 96GB DDR
MySQL version	Percona Server 5.5
Flash(PCI-e)	Fusion-io Gen 2 ioMemory 1.2TB
Persistent Memory	PCI-e ioMemory Persistent Memory
	DDR Viking NVDIMM
Remote	**config-3, config-4, config-5, config-6**
System Configuration	Source: HP DL380 96GB DDR
	Sink : SuperMicro 2U 64GB DDR
MySQL version	Source: Percona Server 5.5
Flash(PCI-e)	Fusion-io Gen 2 ioMemory 1.2TB
Persistent Memory	Source: DDR / PCI-e ioMemory PM
	Sink: DDR Viking NVDIMM,
	PCI-e ioMemory Persistent Memory
Network	Infiniband: ConnectX-3 56 Gbit IB
	Ethernet: Intel 82599ES 10-Gigabit

Table 2. System configuration Local and Remote.

of the log file as direct load/store PM. Our library exposes POSIX-like APIs (such as append() and read()) that enable MySQL to place data at the end of the log file and to read from the file anywhere as needed for recovery or normal operation. Within our programming library, the append() is translated into a memory copy operation into PM (for local attach) or an RDMA write operation into PM (for remote attach). The InnoDB engine operates on the logs mostly as normal except at the point of write() where the library operations are used instead of file write() operations. In this model, the database can perform very small low latency updates to the logs, which enables faster and finer grained transaction commits.

6. Evaluation

We evaluate the performance of our SDPM prototype through a series of microbenchmarks and application level performance studies. Prior studies have shown comprehensive results on the performance of individual PM configurations under a range of workloads [29, 38, 47]. Hence, we focus our evaluation on the Software Defined aspect of our prototype, i.e., we take a workload and evaluate how well SDPM performs across different hardware configurations. For workloads, we use both microbenchmarks and application level experiments. Since our application focus is database log acceleration, we also focus our performance studies on write (or CPU store) operations and the associated persistence guarantees.

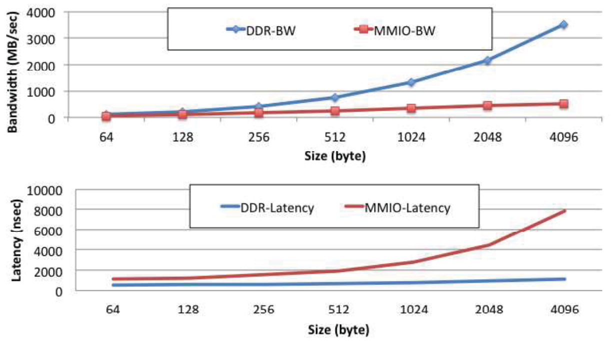


Figure 3. Microbenchmark: Local PCIe MMIO (config-2) Vs. DDR4 (config-1) (a) Bandwidth (b) Latency

Table 1 lists the supported configurations and Table 2 provides the details of our machine configurations. We explore two primary dimensions and one secondary dimension. The primary dimensions are local vs. remote and the attach point for the PM itself (DDR or PCIe). Second, within the remote dimension, we explore both Infiniband and Ethernet transports. These combinations help us to assess whether we can adequately deliver application performance across a range of hardware architectures.

Note that for DDR attached PM, we use a single Type N NVDIMM. It is possible to also test with DRAM as a placeholder for PM, if the areas being evaluated are not persistence related. However, there are subtle performance differences between using a single Type N NVDIMM and generically allocating system DRAM - which can then come from an interleaved set of volatile DIMMs. Rather than dissect the details of such DRAM configurations, we focus on the PM hardware available today (ie NVDIMM) and use actual NVDIMMs.

6.1 Local Attach: Microbenchmarks

Figure 3 shows the latency and bandwidth of updates to PM through both the local PCIe MMIO and DDR attach. In each case, the data is written to the virtual memory location which is direct mmap'ed by NVMFS to the physical PM. The data is then committed through a SDPM barrier() operation (using hardware specific micro-operation(s)).

As the data shows, the cost of the barrier operation dominates the latency for small updates regardless of whether the update is over PCIe MMIO or DDR. For small (64B) updates, the latency of PCIe MMIO (with barrier()) is within 2x of DDR. As the transfer size increases, the gap in bandwidth between the DDR and the PCIe attach becomes more clear in higher latencies for the PCIe solution. It is worth noting that our PCIe hardware uses Gen2 PCIe which has a maximum lane bandwidth of 1GB/s while the DDR attach hardware is using DDR4. We have observed lower latencies (roughly by about 2x for large transfers) for PCIe MMIO when using Gen3 PCIe.

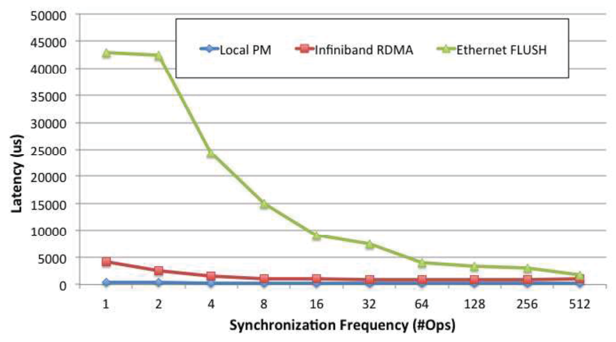


Figure 4. Microbenchmark: 4K write + barrier (config-1, config-3, and config-5)

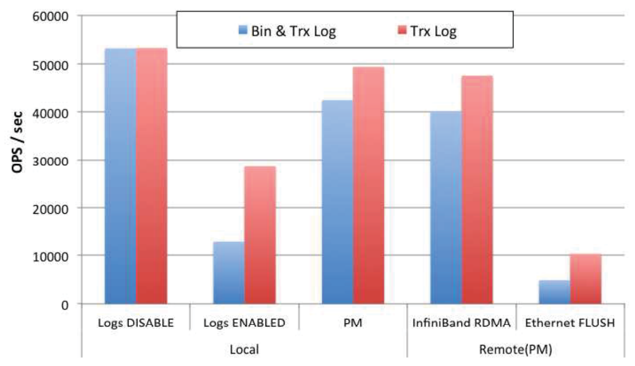
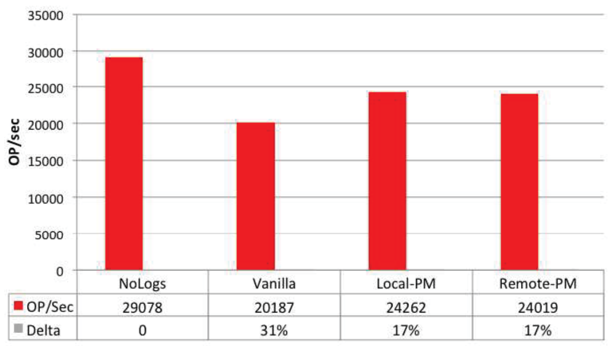


Figure 5. MySQL - insert heavy workload (config-1, config-3, and config-5)



	NoLogs	Vanilla	Local-PM	Remote-PM
🟥 OP/Sec	29078	20187	24262	24019
■ Delta	0	31%	17%	17%

Figure 6. LinkBench 10x workload (config-1 and config-5)

Based on these results, we expect PCIe MMIO to be a reasonable solution for access patterns that involve small updates, and we have observed this in application level tests. However, due to lack of space, we focus the rest of the paper on presenting the results for the DDR attach.

6.2 Remote Attach: Microbenchmark

Figure 4 shows microbenchmarks for simple write operations from user space through the library. In each case, 4KB writes are performed into PM with a synchronization operation performed after a number of 4KB operations (displayed on X-axis). As discussed in Section 4, when synchronization is invoked, it is performed as needed for each hardware configuration. DDR would imply a cache flush and network would imply a network sync following whatever is required on the remote node for local sync. As Figure 4 shows, for small sync operations, as expected there is a notable difference in latency between TCP/IP and the other configurations TCP/IP is about 20x slower than Infiniband for small transfers. Between Infiniband and local DDR PM there is about a 4x difference. Since the destination in each case is PM, the bulk of the overhead is due to the synchronization operation. As the number of operations between synchronizations increases, the performance becomes closer. Since different applications will use different update sizes, this gives us an indication of how workloads will respond when used with different configurations under SDPM. While this section only presented DDR attached PM, the results for PCIe MMIO attached PM are similar in the tradeoffs.

6.3 MySQL: Transactional Workload

Figure 5 shows the performance of MySQL when PM is used for log acceleration using an insert heavy workload (100% inserts) when run with and without PM. For comparison, the Logs DISABLE bars shows theoretical peak performance when logs are disabled (i.e., performance possible with infinite log acceleration). The Logs ENABLED bars shows what happens with no PM (i.e., all logs are written directly to flash). The remainder are the PM configurations where the PM is local DDR, remote DDR over infiniband, or remote DDR over Ethernet. In each case, the data shows two columns, one for only accelerating the Txlog and one for accelerating both TxLog and BinLog.

The difference in transaction performance between Logs DISABLE and the baseline Logs ENABLED is almost 2x, indicating the opportunity for PM log acceleration. We see that the local PM over DDR achieves nearly all of the achievable acceleration. However, the Infiniband option does nearly as well, suggesting that the small added latency results in a measurable but not substantial loss in performance at the application level. Even when remote, the PM configuration still delivers better performance than local flash when used over Infiniband. However, as latency increases substantially with Ethernet, the application performance of remote PM drops dramatically to become far less than that of local flash.

Given the extreme drop in application performance for the Ethernet configurations, we focus only on Infiniband/RDMA based remote configurations for the remainder of the paper.

6.4 MySQL: Social Graph Workload

LinkBench was created by Facebook to simulate the activity pattern of a social graph [18]. Linkbench operations includes a mix of insert, delete, update, and lookup database operations and is approximately 70% access heavy (forms of reads) and only 30% modification heavy (updates and deletes). The default (1x) LinkBench dataset comprises 10 million node ids with a dataset size of 10GB on media. A dataset with 100 million nodes is referred to as 10x workload. Since the 10x workload allows only a partial amount of the data to fit in the buffer pool, it is a more realistic test of the I/O subsystem. Figure 6 shows the results for this workload for local and remote DDR attached PM where the remote attach is over Infiniband. Since this workload is read intensive, the difference between the Vanilla (all data on flash) and the NoLogs (infinite log acceleration) is 31%. Both the local and remote PM solutions achieve a reasonable fraction of the available performance gain, delivering 17% acceleration over Vanilla. The performance of local and remote is very similar (less than 3% difference) showing again that the latency achieved by Infiniband for small updates is sufficient to deliver good performance for applications.

7. Discussion

Our experience developing SDPM has led to several insights. There are many possible hardware configurations that an application can optimize for, ranging from different CPU mapping strategies to different attach points. The need to work with remote memory only further exacerbates the complexity involved. Some hardware specific optimizations (such as direct PCIe-PCIe transfer for remote attach), may be difficult for applications to be aware of or to control. Fortunately, our results demonstrate that it is possible to abstract the bulk of this complexity within an SDPM architecture without substantial loss of performance at microbenchmark level and nearly identical performance at application level for some hardware configurations.

For TCP/IP over Ethernet however, the slowdown presented is substantial and very visible at application level. Ethernet based measurements discussed in the paper are using the standard networking stack. An offload network device can potentially help reduce the operation latencies as reported by the application. This is an area of future work for us. Even within the Infiniband implementation, further optimizations are possible. Infiniband provides a DMA efficient method, send-immediate, which is ideal for small data payloads since it avoids an additional DMA. Using send-immediate, we have measured update operation latencies of less than 2 usec (including barrier assertion on remote) for sync interval of 3 or more updates. This observation leads us to expect that small application log updates will benefit from this optimization.

The potential of software defined concepts applied to PM is vast and we have only explored an initial subset of its possibilities in this paper. A more comprehensive architecture would include clustering of PM and possibly a distributed file system. We have focused on the specific challenges involved in PM since that is the new media technology. We believe a distributed SDPM architecture can build on our SDPM work and also incorporate existing concepts from distributed in-memory file systems and data stores [31].

In a similar vein, we focused on DDR and byte addressable MMIO PCIe for our prototype hardware attaches. Given the high latencies of SAS/SATA attaches, it is not clear whether PM will ever be attached using these transports. NVMe is however a possibility given its low latency for block accesses. It is possible to extend our SDPM architecture to include NVMe PM devices in two ways. In a first approach, we could replace our flash layer with these devices. A second, and perhaps better approach, would be to use read-modify-write operations and DRAM caching to emulate byte addressable PM using block addressable PM devices over NVMe [21].

Intel has announced three new CPU instructions in future platforms to support PMs (CLFLUSHOPT, PCOMMIT, CLWB) [10]. Since these instructions are not yet available, our SDPM prototype does not use them. However, it would be trivial to implement barrier() and msync() operations to leverage these instructions when they become available. This in fact illustrates the power of the software defined approach. As new instructions become available with subsequent platforms, applications do not have to have platform specific code. The optimizations can be abstracted under the generalized persistence operations and semantics offered by SDPM.

8. Conclusions

We presented SDPM, the first instance of a software defined approach to PM that can bring the benefits of new non-volatile memory media to wide ranging practical deployments. By abstracting heterogeneous memory hardware, we are able to support both local and remote attaches as well as different local attach points likely to be available in servers. By carefully selecting the characteristics to expose and abstract, we enable applications to get optimal performance across a range of configurations or optimize aggressively for a specific configuration. By tiering PM with flash, we enable both existing applications and applications written to exploit PM to operate with a wide range of PM capacities. Our prototype and performance evaluation demonstrate that the architecture provides good performance and near optimal acceleration for a range of local and remote PM configurations.

9. Acknowledgments

We thank the anonymous reviewers for their insightful feedback. We thank the many developers of the SDPM project and also the members of the advanced development group at SanDisk for their help and feedback.

References

[1] MySQL. https://www.mysql.com, 2010.

[2] OpenCompute. http://www.opencompute.org/, 2011.

[3] Fusion-io Breaks One Billion IOPS Barrier. http://www.fusionio.com/press-releases/fusion-io-breaks-one-billion-iops-barrier, 2012.

[4] ioCache. http://www.fusionio.com/products/iocache, 2012.

[5] JEDEC DDR3 Specification. http://www.jedec.org/standards-documents/docs/jesd-79-3d, 2012.

[6] NVDIMM. http://www.micron.com/products/dram-modules/nvdimm, 2012.

[7] NVM Programming Model (NPM). http://www.snia.org/tech_activities/standards/-curr_standards/npm, 2013.

[8] Blade servers: An introduction and overview. http://searchdatacenter.techtarget.com, 2014.

[9] In a Battle of Hardware, Software Innovation Comes Out On Top. http://itblog.sandisk.com/in-a-battle-of-hardware-software-innovation-comes-out-on-top/, 2014.

[10] Intel Architecture Instruction Set Extensions Programming Reference. https://software.intel.com/sites/default/files/-managed/0d/53/319433-022.pdf, 2014.

[11] Persistent Memory Programming. http://pmem.io, 2014.

[12] Providing Atomic Sector Updates in Software for Persistent Memory. http://events.linuxfoundation.org/sites/events/files/-slides/vault-btt_0.pdf, 2014.

[13] RDMA with byte-addressable PM. http://downloads.openfabrics.org/WorkGroups/-ofiwg/dsda_rqmts/RDMA_with_PM.pptx, 2014.

[14] What is SAP HANA? http://www.saphana.com/community/about-hana, 2014.

[15] InnoDB: The Binary Log. http://dev.mysql.com/doc/refman/5.7/en/binary-log.html, 2015.

[16] Redo Logging in InnoDB. `https://blogs.oracle.com/mysqlinnodb/entry/redo_logging_in_innodb`, 2015.

[17] The InnoDB Storage Engine. http://dev.mysql.com/doc/refman/5.7/en/innodb-storage-engine.html, 2015.

[18] ARMSTRONG, T. G., PONNEKANTI, V., BORTHAKUR, D., AND CALLAGHAN, M. Linkbench: A database benchmark based on the facebook social graph. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2013), SIGMOD '13, ACM, pp. 1185–1196.

[19] BAILEY, K., CEZE, L., GRIBBLE, S. D., AND LEVY, H. M. Operating system implications of fast, cheap, non-volatile memory. In *Proceedings of the 13th USENIX Conference on Hot Topics in Operating Systems* (Berkeley, CA, USA, 2011), HotOS'13, USENIX Association, pp. 2–2.

[20] BRIDGE, B. NVM-Direct. https://github.com/oracle/NVM-Direct, 2015.

[21] CAMPELLO, D., LOPEZ, H., KOLLER, R., RANGASWAMI, R., AND USECHE, L. Non-blocking writes to files. In *13th USENIX Conference on File and Storage Technologies (FAST 15)* (Santa Clara, CA, Feb. 2015), USENIX Association, pp. 151–165.

[22] CHEN, T. Introduction to acpi-based memory hot-plug. In *LinuxCon/CloudOpen Japan* (2013), LinuxCon '13.

[23] COBURN, J., CAULFIELD, A. M., AKEL, A., GRUPP, L. M., GUPTA, R. K., JHALA, R., AND SWANSON, S. 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* (New York, NY, USA, 2011), ASPLOS XVI, pp. 105–118.

[24] CONDIT, J., NIGHTINGALE, E. B., FROST, C., IPEK, E., LEE, B., BURGER, D., AND COETZEE, D. Better i/o through byte-addressable, persistent memory. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles* (New York, NY, USA, 2009), SOSP '09, ACM, pp. 133–146.

[25] DAS, D., ARTEAGA, D., TALAGALA, N., MATHIASEN, T., AND LINDSTRÖM, J. Nvm compression—hybrid flash-aware application level compression. In *2nd Workshop on Interactions of NVM/Flash with Operating Systems and Workloads (INFLOW 14)* (Broomfield, CO, Oct 2014).

[26] DULLOOR, S. R., KUMAR, S., KESHAVAMURTHY, A., LANTZ, P., REDDY, D., SANKARAN, R., AND JACKSON, J. System software for persistent memory. In *Proceedings of the Ninth European Conference on Computer Systems* (New York, NY, USA, 2014), EuroSys '14, ACM, pp. 15:1–15:15.

[27] GUERRA, J., MÁRMOL, L., CAMPELLO, D., CRESPO, C., RANGASWAMI, R., AND WEI, J. Software persistent memory. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference* (Berkeley, CA, USA, 2012), USENIX ATC'12, pp. 29–29.

[28] JOSEPHSON, W., BONGO, L., LI, K., AND FLYNN, D. Dfs: A file system for virtualized flash storage. In *Usenix Conference on File and Storage Technologies* (February 2010).

[29] KIM, H., SESHADRI, S., DICKEY, C. L., AND CHIU, L. Evaluating phase change memory for enterprise storage systems: A study of caching and tiering approaches. In *Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST 14)* (Santa Clara, CA, 2014), USENIX, pp. 33–45.

[30] LEE, B., IPEK, E., MUTLU, O., AND BURGER, D. Architecting phase change memory as a scalable dram alternative. In *International Symposium on Computer Architecture (ISCA)* (June 2009).

[31] LI, H., GHODSI, A., ZAHARIA, M., SHENKER, S., AND STOICA, I. Tachyon: Reliable, memory speed storage for cluster computing frameworks. In *Proceedings of the ACM Symposium on Cloud Computing* (New York, NY, USA, 2014), SOCC '14, ACM, pp. 6:1–6:15.

[32] MÜHLBAUER, T., RÖDIGER, W., SEILBECK, R., REISER, A., KEMPER, A., AND NEUMANN, T. Instant loading for main memory databases. *Proc. VLDB Endow. 6*, 14 (sep 2013), 1702–1713.

[33] NARAYANAN, D., AND HODSON, O. Whole-system persistence with non-volatile memories. In *Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2012)* (March 2012), ACM.

[34] OIKAWA, S. Virtualizing storage as memory for high performance storage access. In *Proceedings of the 2014 IEEE International Symposium on Parallel and Distributed Processing with Applications* (Washington, DC, USA, 2014), ISPA '14, IEEE Computer Society, pp. 18–25.

[35] PIOTROWSKI, A., AND MAKOWSKI, D. Pciexpress hot plug mechanism in linux-based atca control systems. *International Journal of Microelectronics and Computer Science Vol. 1, nr 2* (2010), 201–204.

[36] SATYANARAYANAN, M., MASHBURN, H. H., KUMAR, P., STEERE, D. C., AND KISTLER, J. J. Lightweight recoverable virtual memory. *ACM Trans. Comput. Syst. 12*, 1 (feb 1994), 33–57.

[37] STRUKOV, D. B., SNIDER, G. S., STEWART, D. R., AND WILLIAMS, R. S. The missing memristor found. *Nature 453*, 7191 (2008), 80–83.

[38] SUZUKI, K., AND SWANSON, S. The non-volatile memory technology database (nvmdb). Tech. Rep. CS2015-1011, Department of Computer Science & Engineering, University of California, San Diego, May 2015. http://nvmdb.ucsd.edu.

[39] SWANSON, S., AND CAULFIELD, A. Refactor, reduce, recycle: Restructuring the i/o stack for the future of storage. *Computer 46*, 8 (Aug. 2013), 52–59.

[40] TALAGALA, N. One Billion IOPS: Auto Commit Memory Blurs the Line Between Enterprise Storage and Memory. http://www.fusionio.com/blog/one-billion-iops-auto-commit-memory-blurs-the-line-between-enterprise-storage-and-memory, 2012.

[41] THATTE, S. M. Persistent memory: A storage architecture for object-oriented database systems. In *Proceedings on the 1986 International Workshop on Object-oriented Database Systems* (Los Alamitos, CA, USA, 1986), OODS '86, IEEE Computer Society Press, pp. 148–159.

[42] THERESKA, E., BALLANI, H., O'SHEA, G., KARAGIANNIS, T., ROWSTRON, A., TALPEY, T., BLACK, R., AND ZHU, T. Ioflow: A software-defined storage architecture. In *SOSP'13: The 24th ACM Symposium on Operating Systems Principles* (November 2013), ACM.

[43] VOLOS, H., TACK, A. J., AND SWIFT, M. M. Mnemosyne: Lightweight persistent memory. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 2011), ASPLOS XVI, ACM, pp. 91–104.

[44] WILCOX, M. DAX: Page cache bypass for filesystems on memory storage. http://lwn.net/Articles/618064/, 2014.

[45] WILLIAMS, R. How we found the missing memristor. *IEEE Spectr. 45*, 12 (Dec. 2008), 28–35.

[46] YANG, J., WEI, Q., CHEN, C., WANG, C., YONG, K. L., AND HE, B. Nv-tree: Reducing consistency cost for nvm-based single level systems. In *13th USENIX Conference on File and Storage Technologies (FAST 15)* (Santa Clara, CA, Feb 2015), USENIX Association, pp. 167–181.

[47] ZHANG, Y., YANG, J., MEMARIPOUR, A., AND SWANSON, S. Mojim: A reliable and highly-available non-volatile memory system. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 2015), ASPLOS '15, ACM, pp. 3–18.

[48] ZILBERBERG, O., WEISS, S., AND TOLEDO, S. Phase-change memory: An architectural perspective. *ACM Comput. Surv. 45*, 3 (July 2013), 29:1–29:33.