A Survey of Software Techniques for Using Non-Volatile Memories for Storage and Main Memory Systems

Sparsh Mittal, *Member, IEEE* and Jeffrey S. Vetter, *Senior Member, IEEE*

Abstract—Non-volatile memory (NVM) devices, such as Flash, phase change RAM, spin transfer torque RAM, and resistive RAM, offer several advantages and challenges when compared to conventional memory technologies, such as DRAM and magnetic hard disk drives (HDDs). In this paper, we present a survey of software techniques that have been proposed to exploit the advantages and mitigate the disadvantages of NVMs when used for designing memory systems, and, in particular, secondary storage (e.g., solid state drive) and main memory. We classify these software techniques along several dimensions to highlight their similarities and differences. Given that NVMs are growing in popularity, we believe that this survey will motivate further research in the field of software technology for NVMs.

Index Terms—Review, classification, non-volatile memory (NVM) (NVRAM), flash memory, phase change RAM (PCM) (PCRAM), spin transfer torque RAM (STT-RAM) (STT-MRAM), resistive RAM (ReRAM) (RRAM), storage class memory (SCM), solid state drive (SSD)

1 Introduction

For all computing systems ranging from hand-held embedded systems to massive supercomputers, memory systems play the primary role in determining their power consumption, reliability, and, unquestionably, application performance. The ever-increasing data-intensive nature of state-of-the-art applications demands significantly higher performance within the constraints of power and cost, beyond what conventional memory technologies can provide. For example, more than 100 hours of video and 250 K photos are uploaded every minute to YouTube [1] and Facebook [2], respectively. Similarly, it is expected that future extreme-scale systems will deal with several exabytes of data; using hard disk to support checkpoint/restart in these systems can degrade their performance by more than 50 percent [3]. Further, these systems will require at least 100 petabytes of main memory, which, if designed with today's DDR3 DRAM, would consume 52 MW power, which is far more than the 20 MW power budget mandated for the entire system [4], [5], [6].

These requirements have motivated the researchers to explore novel memory technologies. Non-volatile memory (NVM) devices, such as Flash, phase change memory (PCM), spin transfer torque RAM (STT-RAM), and resistive RAM (ReRAM) can provide many benefits over existing devices for designing different components of the memory hierarchy, from caches to main memory to secondary storage. The potential benefits of these NVMs stem from their projected physical properties that may allow them to both consume very low power and provide much higher density than projected traditional technologies. For example, the size of a typical SRAM cell is in the range of 125–$200F^2$, while that of a PCM and Flash cell is in the range of 4–$12F^2$ and 4–$6F^2$, respectively, where F denotes the smallest lithographic dimension in a given technology node [3], [7]. Due to their features, NVMs are increasingly being deployed in products. For example, Intel TurboMemory uses a small amount of Flash memory as a cache to buffer disk data [8]. Similarly, many large-scale enterprises are using Flash in their data centers [9] and supercomputers [10], [11]. In fact, it has been recently announced that US Department of Energy's next-generation supercomputer, named Summit, will have 800 GB of NVM in each of its 3,400 nodes; this NVM can be configured as either a burst buffer or as extended memory [12].

On the other hand, these NVMs also have some limitations. For example, their write endurance is several orders of magnitude lower than conventional memories, and write operations typically have high latency and energy costs. It is clear that until innovations to offset these limitations are demonstrated at device and architecture level, the community will need software and system-level techniques to integrate NVM technologies in future memory systems.

1.1 Contributions

In this paper, we present a survey of software and system-level techniques that have been proposed to exploit the advantages and mitigate the disadvantages of NVMs when used in the memory hierarchy, and, in particular, secondary storage (e.g., solid state drive) and main memory. We classify these software techniques along several dimensions to highlight their similarities and differences. We also discuss those papers which compare or combine multiple non-volatile or conventional memory technologies.

- *S. Mittal is with the Future Technologies Group, Oak Ridge National Laboratory, Oak Ridge, TN 37830. E-mail: mittals@ornl.gov.*
- *J.S. Vetter is with the Future Technologies Group, Oak Ridge National Laboratory, Oak Ridge, TN 37830, and the Georgia Institute of Technology, GA. E-mail: vetter@ornl.gov.*

Manuscript received 13 Nov. 2014; revised 27 May 2015; accepted 2 June 2015. Date of publication 8 June 2015; date of current version 13 Apr. 2016.
Recommended for acceptance by R. Brightwell.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2015.2442980

TABLE 1
Approximate Device-level Properties of Memory Technologies [3], [7], [14]

	Cell size	Access Granularity	Read Latency	Write Latency	Erase Latency	Endurance	Standby Power
HDD	N/A	512 B	5 ms	5 ms	N/A	$> 10^{15}$	1W
SLC Flash	4–6F^2	4 KB	25 μs	500 μs	2 ms	$10^4 - 10^5$	0
DRAM	6–10F^2	64 B	50 ns	50 ns	N/A	$> 10^{15}$	Refresh power
PCM	4–12F2	64 B	50 ns	500 ns	N/A	$10^8 - 10^9$	0
STT-RAM	6–50F^2	64 B	10 ns	50 ns	N/A	$> 10^{15}$	0
ReRAM	4–10F^2	64 B	10 ns	50 ns	N/A	10^{11}	0

1.2 Terminology and Scope

Following other researchers (e.g. [13]), we use SCM to refer to byte-addressable non-volatile memories such as STT-RAM, PCM and ReRAM and use NVM to refer to all the SCMs and Flash. We discuss techniques proposed for both single-level cell (SLC) and multi-level cell (MLC) memories. Unless otherwise mentioned, Flash refers to the NAND Flash and not the NOR Flash, and SSD refers to Flash-based SSD and not PCM-based SSD. Also, some techniques proposed for other memory technologies (e.g. HDDs or DRAM) may also be applied to NVMs, however, we include only those techniques which have been proposed in context of NVMs. Since different techniques have been evaluated in different contexts, we only present their main idea and do not show the quantitative results.

A few previous papers focus on the device-level properties of Flash memory [15] or review use of SCMs for cache and main memory [7], [16]. By comparison, this paper surveys system- and software-level techniques proposed for all NVMs for addressing several important aspects, such as design of persistent memory systems, lifetime enhancement, reliability, cost efficiency, energy efficiency, design of hybrid memory systems etc. We classify the techniques based on several key features/characteristics to highlight their similarities and differences. By providing a synthetic overview of existing frontiers of NVM management techniques, we aim to provide clear directions for future research in this area. This survey paper is expected to be useful for researchers, OS designers, computer architects and others.

The rest of the paper is organized as follows. Section 2 presents a background on the characteristics and limitations of NVMs. Section 3 classifies the research projects on NVMs based on several parameters and then discusses a few of them. Section 4 discusses research projects which study integration or comparison of multiple memory technologies. Finally, Section 5 presents the conclusion and future challenges.

2 A Brief Overview of Memory Technologies

In this section, we briefly summarize the properties and challenges of different memory technologies. For sake of comparison we also discuss conventional memory technologies such as hard disk drive (HDD) and DRAM. Table 1 presents the device level properties of different memory technologies. Note that these values should be taken as representatives only since ongoing research may lead to changes in these parameters. For more details on the device-level properties of memory technologies, we refer the reader to previous works [3], [7], [17], [18], [19], [20], [21].

In Table 1, access granularity refers to the minimum amount of data that are read/written in each access and endurance refers to the number of writes a memory block can withstand before it becomes unreliable. The property common to all NVMs is that their write latency/energy are significantly higher than that of read latency/energy. Also, under normal conditions, they retain data for several years without the need of any standby power. The specific properties of different NVMs are discussed below.

2.1 Flash

Flash memory has three types of operations, namely read, program (write), and erase. A write operation can only change the bits from 1 to 0, and hence, the only way to change a bit in a page from 0 to 1 is to erase the block that contains the page which sets *all* bits in the block to 1. Since erase operations are significantly slower than the write operations, Flash SSDs use Flash translation layer (FTL) [22] which helps in 'hiding' the erase operations and thus exposing only read/write operations to the upper layers. FTL maintains a mapping table of virtual addresses from upper layers to physical addresses on the Flash and uses this to perform wear-leveling.

The pages in Flash are classified as valid, invalid (containing dead data) and free (available for storing new data). To hide the latency of erase operations, FTL performs out-of-place writes whereby it writes to a free page and invalidates the previous location of the page and udpates the mapping. Read and write operations are done at page granularity, while erase operation is done at block granularity. Typically, the size of a page and a block are 4 and 256 KB, respectively. FTLs also perform garbage collection (GC), whereby invalid pages are reclaimed by erasing the blocks and relocating any valid pages within them to new locations (if required). Clearly, given the crucial impact of FTL on the performance of Flash SSDs and presence of large number of factors involved in its design (e.g. GC policies, page v/s block mapping, size and storage location of its mapping table etc.), FTL requires discussion of its own and hence, we refer the reader to previous works for more details [22], [23].

The cell-size of Flash is 4-6F^2, while that of SRAM is 120-200F^2 [7]. Clearly, due to its high density and latency and low write endurance, Flash is generally suitable for use as a storage device or a caching layer between DRAM and HDD. Flash is a mature NVM technology and is being developed and deployed by several commercial manufacturers [9].

Compared to the rotating media (viz. HDD), Flash is based on semiconductor chips which leads to compact size, low power consumption and better performance for random data accesses. Also, SSDs have no moving parts, no mechanical wearout, and are resistant to heat and shock. However, the relative performance advantage of SSD over HDD highly depends on the workload characteristics. For example, it has been shown that for many write-intensive scientific workloads, SSDs may provide only marginal gain over HDDs [24]. Also, due to its higher-cost, SSD cannot completely replace HDD [25]. Thus, the research challenges for Flash which need to be addressed at system level include lifetime enhancement by minimizing the number of write/erase operations, wear-leveling, managing faulty blocks, and performance improvement by retention relaxation and design of hybrid memory systems.

2.2 PCM, STT-RAM and ReRAM

The most important feature of these three memory technologies (referred to as SCMs) which distinguishes them from Flash, is that they are byte-addressable. Computer systems have traditionally used DRAM as a volatile memory and HDD and Flash as persistent storage. The difference in their latencies has, however, led to large differences in their interfaces [26], [27]. These SCMs offer the promise of storage capacity and endurance similar to or better than Flash while providing latencies comparable to DRAM. For these reasons, the SCMs hold the promise of being used as universal memory technologies. Although, compared to Flash and DRAM, the SCMs are less mature, yet significant amount of research has been done in recent years for developing and utilizing them [7], for example, a 16 Gb ReRAM prototype has been recently demonstrated [28] which features an eight-bank concurrent DRAM-like core architecture and 1 GB/s DDR interface. The system-level techniques proposed for these memories address several key research challenges such as integrating them in memory/storage hierarchy to design persistent memory systems and complement conventional memory technologies, and enhancing lifetime, reliability and performance etc.

Sections 3 and 4 discuss the techniques proposed for addressing these issues.

3 NVM Management Techniques

Table 2 classifies the research projects based on the NVM used and the level of memory hierarchy at which the NVM is used. Many studies involving secondary storage require long execution time and/or modeling of operating system (OS) operations, which is not provided by the typical user-space simulators. The simulators may also use simplistic models and thus, miss crucial real-world details. On the other hand, real hardware platforms do not provide experimentation flexibility like that of a simulator and given the emerging nature of SCMs, real hardware platforms with SCM-based storage are also generally unavailable. This presents a challenge in the study of NVMs and makes the choice of a suitable evaluation platform crucially important. For this reason, Table 2 further classifies the techniques based on whether they have been evaluated using a simulator or real hardware (e.g. a CPU or an FPGA) to help the readers gain insight.

Table 2 also classifies the techniques based on their optimization objectives and the essential approach. We now discuss some of these techniques in bottom-to-top order of abstraction levels across the software stack, beginning with architecture, firmware, middleware (I/O, operating system etc.) up to programming models/APIs (application programming interfaces), although note that several of these techniques span across these ‘boundaries’.

3.1 Write/Erase Overhead Minimization

NVMs in general have low write-endurance and due to write-variation introduced by workloads, a few blocks may receive much higher number of writes than the remaining blocks. This issue can be addressed by both minimizing the writes/erasures and uniformly distributing them over all the blocks (called wear-leveling). We now discuss the techniques based on these approaches.

Huang et al. [43] propose a technique to reduce write-traffic to SSDs, which also increases their lifetime. Their technique employs delta-encoding for semantic blocks and deduplication to data blocks. For every block write request, it decides whether it is semantic or data block. For data block write, it computes MD5 digest to determine whether it is a duplicate block write. For duplicate blocks, their technique returns existing block number in the found hash entry and thus avoids the need of allocating hash table memory. If it is a semantic block (which include super-blocks, group descriptors, data block bitmap etc.), their technique calculates the content delta relative to its original content, and then appends the delta to a delta-logging region. Since semantic blocks are visited much more frequently than data blocks, with each update bringing very minimal changes; their technique reduces the number of writes.

Papirla and Chakrabarti [84] note that the write latency and energy in a Flash memory is data-dependent, for example, in a four-level Flash, writing ‘01’ and ‘10’ patterns incurs higher latency and energy than writing ‘00’ and ‘11’ patterns. They propose a data-encoding technique to reduce the number of ‘01’ and ‘10’ patterns that are written in memory. Also, their technique does not harm the error performance of the application. Further, in a four-level NOR Flash memory, ‘11’ state is called erase state since it does not require programming the memory cell while the remaining states are called programmed state. By reducing the ‘01’ and ‘10’ patterns, their technique reduces the number of programming cycles which improves the lifetime of Flash memory.

Grupp et al. [52] use WOM (write-once memory) coding scheme to increase the lifetime of Flash memory and also save energy. By using extra bits, WOM code allows multiple logical value to be written even if physical bits can transition only once. By virtue of this, WOM code allows writing data to a block twice before erasing it, which reduces the number of erasures required.

In embedded systems, Flash can be used as main memory, however, its large data access granularity (e.g. 4 KB v/s 64 B used in cache) and small endurance present challenges. Shi et al. [82] present a technique to reduce the number of writes on Flash main memory to improve its lifetime and also bridge the gap between access granularities. They observe that due to data locality, the data accessed in consecutive writes come from a limited number of pages. Based on this, they use victim cache, along with a write-buffer to

TABLE 2
A Classification of Techniques

Classification	References
Non-volatile memory used	
Flash	[8], [11], [18], [19], [20], [22], [23], [24], [25], [27], [29], [30], [31], [32], [33], [34], [35], [36], [37], [38], [39], [40], [41], [42], [43], [44], [45], [46], [47], [48], [49], [50], [51], [52], [53], [54], [55], [56], [57], [58], [59], [60], [61], [62], [63], [64], [65], [66], [67], [68], [69], [70], [71], [72], [73], [74], [75], [76], [77], [78], [79], [80], [81], [82], [83], [84], [85], [86], [87], [88], [89], [90], [91], [92], [93], [94], [95], [96], [97], [98], [99], [100]
PCM	[3], [13], [14], [26], [27], [31], [51], [68], [75], [85], [90], [99], [101], [102], [103], [104], [105], [106], [107], [108], [109], [110], [111], [112], [113], [114], [115], [116], [117], [118], [119], [120], [121], [122], [123], [124], [125], [126], [127], [128], [129], [130], [131], [132]
STT-RAM	[13], [26], [27], [101], [110], [112], [113], [125], [128], [133], [134]
ReRAM	[13], [26], [112], [125], [126], [127], [130], [132]
Level in memory hierarchy where NVM is used	
Secondary storage (and its cache etc.)	[8], [18], [19], [20], [22], [23], [24], [25], [31], [32], [33], [34], [39], [43], [44], [45], [46], [47], [48], [49], [50], [54], [55], [57], [58], [59], [60], [63], [64], [65], [66], [67], [68], [71], [72], [73], [74], [75], [76], [77], [78], [79], [80], [81], [85], [86], [87], [88], [90], [91], [92], [93], [94], [97], [98], [100], [102], [104], [106], [107], [124], [129]
Main memory	[13], [14], [62], [82], [85], [99], [101], [102], [103], [106], [110], [111], [115], [116], [117], [118], [119], [120], [121], [122], [125], [128], [131], [133], [134]
On-chip cache	[128], [133]
Evaluation platform	
Simulator	[3], [19], [22], [23], [24], [26], [30], [31], [33], [36], [39], [42], [45], [46], [48], [49], [56], [57], [58], [59], [60], [63], [64], [68], [70], [71], [72], [73], [74], [75], [76], [79], [80], [81], [82], [84], [85], [86], [91], [92], [93], [94], [96], [98], [99], [100], [101], [102], [103], [104], [110], [111], [114], [116], [117], [118], [120], [121], [128], [129], [130], [131], [133]
Real hardware	[8], [13], [18], [24], [26], [27], [34], [38], [42], [43], [44], [46], [52], [53], [55], [62], [64], [65], [67], [68], [69], [74], [83], [87], [89], [90], [95], [96], [103], [105], [106], [109], [110], [112], [113], [115], [122], [125], [126], [127], [132], [134], [135]
Study/optimization approach/objective	
Performance improvement	[3], [8], [11], [13], [18], [19], [22], [24], [25], [27], [31], [32], [39], [40], [42], [43], [44], [52], [55], [56], [57], [58], [59], [60], [63], [64], [65], [66], [67], [68], [69], [71], [72], [73], [74], [75], [76], [77], [79], [80], [83], [84], [85], [86], [87], [88], [89], [90], [91], [92], [94], [95], [97], [98], [99], [103], [105], [106], [107], [108], [109], [110], [112], [115], [126], [131], [132], [133], [134], [135]
Energy efficiency	[25], [31], [40], [42], [52], [84], [86], [90], [91], [99], [101], [120], [123], [131], [133]
Lifetime improvement	[19], [23], [31], [32], [33], [34], [36], [39], [43], [44], [46], [47], [48], [49], [50], [51], [52], [53], [57], [59], [65], [69], [72], [73], [75], [78], [81], [82], [84], [85], [88], [91], [93], [99], [100], [106], [108], [116], [117], [118], [119], [121], [122], [123], [124], [125]
Wear-leveling	[19], [23], [30], [31], [45], [46], [47], [48], [49], [50], [53], [91], [93], [99], [107], [108], [111], [117], [124]
Write/erase overhead minimization	[43], [44], [57], [59], [64], [65], [71], [73], [75], [82], [85], [91], [99], [100], [111], [120], [123], [129], [133]
Salvaging faulty blocks and scrubbing	[48], [51], [80], [81], [85], [116], [118], [119], [120], [121], [122]
NVM retention relaxation	[56], [72], [80], [92], [102]
Persistency and consistency	[13], [13], [62], [76], [96], [102], [110], [113], [114], [128], [130], [132], [133], [134], [136]
Checkpointing, reliability and error-correction	[3], [33], [34], [35], [45], [53], [56], [60], [72], [74], [76], [78], [79], [81], [84], [85], [91], [96], [98], [103], [112], [115], [116], [117], [118], [120], [122], [128], [129]
Data-value dependent optimization	[51], [84]
Cost efficiency	[20], [25], [39], [68], [71], [87], [97]

perform write-coalescing, since for multiple last level cache write-back operations, only a single (or few) write takes place to Flash main memory.

Qureshi et al. [131] note that only the SET operation in PCM writes is slow while the RESET operation is nearly as fast as the reads. Hence, by proactively performing SET for all bits much before the anticipated write, the time consumed in write can be reduced which also leads to saving of energy. Based on this, their technique initiates SET operation as soon as the line becomes dirty in the cache.

3.2 Wear-Leveling

Chang et al. [47] propose a static wear-leveling algorithm. In their algorithm, the blocks are divided into sets and each set is associated to a bit in the Block Erasing Table (BET). Initially, all bits in the BET are ’0’. If a member of a set is erased within the interval, its associated bit is transitioned to 1. The total number of erasures in any interval is also recorded. If the ratio of the number of erasures over the number of 1s in the BET reaches a predefined threshold, a set whose corresponding bit is still 0 is randomly selected. Afterwards, all valid data in this set are moved to a free block set, and the former set is erased for future use.

Wang and Wong [46] observe that since OS has the knowledge about files at a higher level of abstraction (e.g. the file type data belong to, the applications that are using them etc.), this information can be used to provide hints to lower-level FTL. The FTL uses these hints to deduce the update frequency and recency of files and thus performs better wear-leveling, since block allocation is done in a manner that young blocks are allocated to hot data and old blocks are allocated to cold data.

Chang and Huang [23] present a wear-leveling algorithm which aims to reduce the wear of elder (i.e., a block with high erase count) blocks. Their technique tracks the erase recency of blocks and whenever the erase recency of an elder block becomes higher than the average by a pre-determined threshold, the logical blocks with a low update recency are remapped to these elder blocks. The algorithm dynamically tunes the threshold to strike a balance between the benefit and cost of wear-leveling.

Wang and Wong [49] propose observation wear-leveling (OWL), which aims to proactively avoid the unevenness of erasures through monitoring temporal locality and block utilization. Their technique uses a table named the Block Access Table (BAT), in which a block is a logical block (and not a Flash block). The BAT stores access frequencies of logical blocks that have been recently rewritten. Using BAT, the OWL algorithm ranks data of logical blocks, and allocates Flash blocks accordingly. The ranking is used to predict a logical block's relative access frequency in the near future. In this way, OWL puts data into suitable blocks in a proactive way to achieve wear-leveling.

Jeong et al. [93] show that by slowly erasing a Flash block with lower erase voltage, the endurance of Flash can be improved. Based on this, they provision multiple write and erase modes with different operation voltages and speeds to extend the lifetime of Flash memory with minimal effect on the application throughput. At software level, garbage collector and wear-leveler are modified to utilize these write/erase modes. For example, instead of using the same endurance for all the blocks (as in a baseline Flash), their wear-leveler uses the effective endurance (as enabled by their lifetime extension approach) to evenly distribute the effective wearing among Flash blocks.

Im and Shin [124] present a wear-leveling technique for PCM-based storage. Their technique counts the write-counts on PCM pages and if a logical page is frequently updated, their technique allocates larger number of physical pages to it to balance the writes on all pages. Thus, the logical pages have a different number of physical pages allocated to them based on the update frequency. They have also shown that their technique keeps the number of additional writes due to wear-leveling small.

3.3 Salvaging Faulty Blocks and Tolerating Failures

The 'raw lifetime' of a memory system is decided by the first failure of a block. This lifetime can be significantly enhanced by tolerating the failure of a few blocks and/or salvaging faulty blocks. Several techniques have been proposed for this and we now discuss a few of them.

Liu et al. [78] present fault-tolerance based techniques to improve the lifetime of Flash-based SSD caches. Since the erroneous data in write-through SSD caches can be recovered by accessing the HDD, one of their technique converts the uncorrectable errors into cache misses that bring in valid data from HDD. Another technique utilizes a fraction of SSD cache capacity to increase the ECC (Error-correcting code) strength when Flash reaches wearout thresholds, thus the SSD cache continues operating with reduced capacity.

Cai et al. [81] present a technique to improve the Flash lifetime by reducing the raw bit error rate, even when Flash memory has endured high P/E cycles far beyond its nominal endurance. Their technique periodically reads each page in Flash memory, corrects its errors using simple ECC, and either remaps the page to a different location or re-programs it in its original location, before the page accumulates more errors than can be corrected with simple ECC. Thus, the lifetime of Flash memory is improved by addressing retention errors which form the most dominant type of errors in Flash memory [34], [35].

Wang and Wong [48] propose a method to salvage bad blocks in Flash to extend its lifetime. Their method works on observation that many pages in a bad block may still be healthy and hence, discarding a block on failure of a few pages leads to wastage and small lifetime. Their method combines the healthy pages of a set of bad blocks together to form a smaller set of virtually healthy blocks which can be used to store cold data.

Maddah et al. [51] present a physical block sparing scheme that delays the retirement of a faulty block when the Flash or PCM memory exhibits failure due to write-endurance limitation. On reaching its write-endurance limit, a PCM or Flash block shows "stuck-at" fault, which means that it gets stuck at either 0 or 1, and can still be read but not reprogrammed [35]. Thus, the occurrence of errors becomes data dependent; an error manifests only when a different bit value is written to a faulty cell than what it is stuck at. Based on this, on an unsuccessful write to a block, their scheme does not immediately retire a block. Instead, a spare block is temporarily borrowed from the spare pool of blocks. Later on, write operation is again attempted on the original (faulty) block which is likely to succeed if the data are same as the existing data on the block. In such a case, the spare block borrowed above is returned to the pool. When the faulty block shows failures more than a threshold, it is finally classified as bad and retired. In effect, for the same lifetime improvement, their scheme reduces the requirement of spare blocks or achieves higher lifetime for the same spare pool capacity.

Yoon et al. [119] present a fine-grained remapping technique to protect NVM against errors. Conventionally, when a block accumulates more wear-out failures than can be corrected, it gets disabled and remapped. Their technique utilizes the still-functional cells of worn-out memory blocks to store the redirection address using heavily redundant code. The spare area is created dynamically by the OS within the main memory. When a remapped block itself fails, it will be remapped further, which may create chained remapping. To avoid this, their technique writes the final address to the original failed block. The benefit of their fine-grained remapping technique is that failure of a 64 B block does not lead to remapping or disabling of an entire 4 KB page.

Gao et al. [122] present a hardware-software cooperative approach to tolerate failures in NVM main memory. Their approach makes error handling transparent to the application by using the memory abstraction offered by garbage-collected managed languages such as Java, C#, JavaScript etc. The runtime ensures that memory allocations never use the failed lines and moves data when the memory lines fail during program execution. Conventional hardware-only schemes which use wear-leveling, delay a single failure, however, their limitation is that due to wear-leveling, after a large number of writes, failures happen uniformly throughout the memory which causes fragmentation. By contrast, their technique uses a low-cost "failure clustering hardware" which logically remaps failed lines to top or bottom edge of the region to maximize the contiguous space available for object allocation. On the first failure, this hardware also installs a pointer to the boundary between normal and failed lines. Thus, their technique reduces fragmentation and improves performance under failures.

Zhao et al. [74] note that the raw BER of Flash memory varies dramatically under different P/E cycles with different retention time. Hence, the ECC used to ensure reliability over-protects the Flash memory, since most pages show better-than-worst-case reliability. To utilize the residual error-correction strength, they propose error-prone over-clocking of Flash memory chip I/O links which translates into higher performance, and the ECC becomes responsible for the errors caused by both Flash memory storage and controller-Flash data transfer. Based on the study of over-clocking on data read path versus over-clocking on data write path, they show that the former is much more effective and favorable than the latter. This is because, the over-clocking on write-path can lead to permanent data storage errors, while the impact of over-clocking on read-path can be more easily tolerated.

Ipek et al. [116] present a method to allow graceful degradation of PCM capacity on occurrence of hard failures. Their technique works by replicating a single physical page over two faulty PCM pages, as long as there is no byte position that is faulty in both pages. Using this, every byte of physical memory can be served by at least one of the two replicas. Since even in pages with hundreds of bit failures, the probability of finding such a pair of pages is high, their technique leads to large improvement in lifetime of PCM memory over conventional error-detection techniques.

Sampson et al. [85] present techniques which trade-off data accuracy for gaining performance, lifetime and density in NVMs for applications which can tolerate data inaccuracies. Their first technique allows errors in MLC memories by reducing the number of programming pulses used to write them, which provides higher performance and density. The second technique uses the blocks that have exhausted their hardware error correction resources to store approximate data which increases the memory lifetime. Further, to reduce the effect of failed bits on overall result, correction of higher-order bits is prioritized.

Several multimedia and graphics applications can tolerate minor errors since these errors are not perceived by human end-users [5], [137]. Fang et al. [123] leverage this property to reduce the write-traffic to PCM. In their technique, if the originally stored data are very close to the new data to be written, the write operation is canceled and originally stored data are taken as the new data. This also reduces the energy consumption of the memory.

3.4 Retention Relaxation

Flash memory allows trading-off the retention period with write speed and P/E (program/erase) cycles. For example, the Flash memory can be programmed faster but with shorter retention time guarantee and thus, the write-operations can be made faster. This property also applies to other NVMs. Several techniques utilize this property for optimization of NVM memory systems. We now discuss a few of them.

Pan et al. [80] present an approach to relax the retention period of Flash for improving P/E cycling endurance and/or programming speed. To avoid losing data at small retention periods, data refresh operations are used. Further, to minimize the impact of refresh operations on normal I/O requests, they propose a scheduling strategy which performs several optimizations, such as giving higher priority to I/O requests, preferring issue of refresh operations when the SSD is idle, etc.

Shi et al. [72] present a technique to improve the endurance and write-performance of Flash memory. Based on the error model of Flash, their technique applies different optimizations at different stages of Flash lifetime. In the first (i.e., early lifetime) stage, the retention time and P/E cycles are traded-off to optimize performance by maximizing the write-speed. In the second (i.e., middle lifetime) stage, the write operations are differentiated into hot writes and cold writes and the hot writes are speeded up while the cold writes are slowed down to improve performance with little effect on endurance. In the last stage, the write speeds are slowed down to extend the endurance. They also propose a smart refresh approach which refreshes only the data approaching the retention time.

Liu et al. [56] analyze different datacenter workloads and observe that for most workloads, the data are frequently updated and hence, they require only few days of retention. This is much shorter than the retention time typically specified for NAND Flash. This gap in retention time can be exploited to optimize the operation of SSD. Also, with retention relaxation, fewer retention errors need to be tolerated and hence, ECC of lower complexity can be used to protect data which reduces the ECC overhead [138].

Huang et al. [79] present a technique to improve the performance of a Flash-based SSD. Their technique selectively replaces ECC with EDC (error detection code) based on the consistency and reliability requirements of the pages. Specifically, when writing fresh data which have no backup in the next storage layer, their technique uses ECC, otherwise it uses EDC. Since the decoding latency of EDC is much smaller, read access to EDC-protected pages is speeded up, which improves the performance. On data corruption, a page is accessed from lower storage layer, thus the reliability is not sacrificed.

Wu and Zhang [92] present a technique to reduce the response time of SSD. Their technique monitors the write-intensity of the workload, and when the write request queue contains several overlapped write transactions, it increases the memory programming speed at the cost of shorter data retention time. Later on when the write intensity again becomes low, the short-lifetime data are rewritten to ensure data integrity. They also develop a scheduling solution to implement their write strategy.

3.5 Flash I/O Scheduling

Wu and He [94] note that in Flash memory, once a program/erase operation is issued to the Flash chip, the subsequent read operations have to wait till the slow program/erase operation is completed. They propose different strategies to suspend the on-going program and erase operations to service pending reads and resume the suspended operation later on. During a program operation, the page buffer contains the data to be written, which may be lost when a read request arrives. To address this, they provision a shadow buffer where the contents of page buffer are stored during suspension. After completion of read operation, it reloads the page buffer with the original data.

In Flash based disk cache, out-of-place writes increase the overhead of garbage collection and also degrade the performance of Flash. To address this, Kgil et al. [91] propose splitting the Flash based disk cache into separate read and write regions (e.g. 90 percent read region and 10 percent write region). Read critical Flash blocks are located in the read region that may only evict Flash blocks and pages on read misses. The write region captures all the writes to the Flash and performs out-of-place writes. Wear-leveling is applied globally to all the regions. Partitioning of Flash into read/write region reduces the number of blocks that need to be considered for garbage collection, which significantly reduces the number of read, write and erase operations.

Lee et al. [58] propose a semi-preemptive garbage collection scheme to improve the performance of garbage collection in Flash memory. The moving operation of a valid page involves page read, data transfer, page write and meta data update. If the origin and destination blocks are in the same plane, the data-transfer operation can be replaced by copy-back operations. Based on this, their scheme decides possible preemption points and allows preemption only on those points to minimize the preemption overhead, hence the name semi-preemptive. Preemption allows servicing pending I/O requests in the queue which improves response time. Also, if the incoming request accesses the same page in which the GC process is attending, it can be merged and if it accesses different page but is of the same type as current request (e.g. read after read), it can be pipelined. They have shown that their garbage collection scheme is especially useful for heavily bursty and write-dominant workloads.

Wang et al. [57] propose an I/O scheduler for Flash-based SSDs which improves performance by leveraging the parallelism inherent in SSDs. The scheduler speculatively divides the whole SSD space into many subregions and associates each subregion with a dedicated dispatching subqueue. Incoming requests are placed into a subqueue corresponding to their accessing addresses. This facilitates simultaneous execution of the requests leading to enhanced parallelism. Their scheduler also sorts the pending requests in the same subqueue to create sequentiality and reduce the harmful effect of random writes on performance and lifetime.

3.6 Programming Models and APIs for Persistent Memory System Design

The non-volatility property of NVMs enables design of persistent memory systems which can survive program and system failures [139]. Several techniques have been proposed to implement and optimize it. We now discuss a few of them.

Volos et al. [13] present Mnemosyne, an interface for programming with SCMs. Mnemosyne enables applications to declare static variables with values that persist across system restarts. It also allows the application to allocate memory from the heap, which is backed by persistent memory. The ordering of writes to persistent memory is controlled by software using a combination of non-cached write modes, cache line flush instructions, and memory barriers. Further, it provides primitives for directly modifying persistent variables and supports consistent updates through a lightweight transaction mechanism.

Coburn et al. [110] present NV-heaps, a persistent object system for providing transactional semantics and a persistence model. For ensuring application-consistency, NV-heaps supports the application in separating volatile and nonvolatile data. As an example, applications can ensure that no pointers exist from persistent memory to DRAM space, and thus, the consistency of persisted state is ensured after a reboot. Also, applications can name heaps to make programming with persistent media easier. Both Mnemosyne and NV-Heaps help applications differentiate among DRAM, persistent memory (SCM), and Flash when allocating new pages for their heap or stack.

Narayanan and Hodson [62] present a persistence approach for systems where all the system main memory is non-volatile. Their approach uses 'flush-on-fail' which implies that the transient state held in CPU registers and cache lines is flushed to the NVM only on a failure (and not during program execution), using a small residual energy window provided by the system power supply. On the restart, the OS and application states are restored like a transparent checkpoint, thus all the state is recovered after a failure. Thus, their approach provides suspend and resume functionality and also eliminates the runtime overhead of flushing cache lines on each update.

Zhao et al. [133] present a persistent memory design that employs a non-volatile last level cache and non-volatile main memory to construct a persistent memory hierarchy. In their approach, when an NV cache line is updated, it represents the newly updated version while the clean data stored in NV memory represents the old version. When the dirty NV cache line is evicted, the old version in NV memory will be automatically updated. With this multiversioned persistent memory hierarchy, their approach enables persistent in-place updates without logging or copy-on-write. This brings the performance of a system with persistent support close to that of one without persistent support. They also develop software interface and architecture extensions to provide atomicity and consistency support for their persistent memory design.

Ransford et al. [96] present an approach for supporting execution of long-running tasks on transiently powered computers, such as RFID-scale (radio-frequency identification) devices which work under very tight resource constraints. Their technique transforms a program into interruptible computations so that they can be spread across multiple power lifecycles. Their technique continually monitors the voltage of the energy source and when it drops below a threshold, the volatile stage of the program is written to NVM, where it survives a power loss. At next-boot, a checkpoint-restoration routine copies the state back to volatile memory for resuming execution.

Dong et al. [3] propose use of PCM to improve checkpointing performance. Conventionally, HDD is used for making checkpoints, however, its low bandwidth leads to large overheads in checkpointing. Also, while DRAM is relatively fast, its high leakage and volatile nature makes it unsuitable for checkpointing and while Flash is non-volatile, its low write-endurance limits the checkpoint frequency. They use a 3D PCM architecture for checkpointing which provides high overall I/O bandwidth. Also, for a massively parallel processing system, they propose a hybrid local/global checkpointing mechanism, where in addition to the global checkpoint, local checkpoints are also made that periodically backup the state of each node in their own private memory. The frequency at which local/global checkpoints are made can be tuned to achieve a balance between performance and resiliency, for example, systems with high transient failures can make frequent local checkpoints and only few global checkpoints. Also, if on a failure, local checkpoint itself is lost, the state of the system can be restored by the global checkpoint.

Condit et al. [103] present a transactional file system that leverages byte addressability of SCM to reduce the amount of metadata written during an update and achieves consistency through shadow updates. Their file system guarantees that file system writes will become durable on persistent storage media in the time it takes to flush the cache and that each file system operation is performed atomically and in program order. For ordering updates, they use epoch-barriers. A cache line is tagged with an epoch number and the cache hardware is modified to ensure that memory updates due to write backs always happen in epoch order.

3.7 System-Level Redesign of NVM Storage and Memory

Accounting for the properties of NVMs at system-level helps in achieving several optimizations in the memory system/hierarchy, as shown by the following techniques.

Jung and Cho [106] propose a system architecture, named Memorage that utilizes the persistent memory resources of a system (viz. persistent main memory and persistent storage device) in an integrated manner. It is well-known that due to over-provisioning of storage resources, its utilization remains small. To address this, Memorage collapses the traditional static boundary between main memory and storage resources and allows main memory to borrow directly accessible memory resources from the storage device to cope with memory shortages. This avoids the need of swapping the pages. Further, since the lifetime of storage device is much higher than that of main memory, the system lifetime is limited by that of main memory. To address this, Memorage allows the storage device to donate its free capacity to the main memory and thus, offers "virtual" over-provisioning without incurring the cost for physical over-provisioning. This helps in extending the lifetime of main memory, at the cost of reduction in lifetime of the storage.

TABLE 3
Classification of Techniques Utilizing or Comparing Multiple Memory Technologies

Classification	References
Use of multiple memory technologies	
HDD + SSD	[8], [39], [44], [59], [61], [65], [66], [83], [86], [87]
PCM+Flash+HDD	[68]
PCM+Flash	[31], [75]
DRAM+ PCM or NOR Flash	[111]
DRAM+PCM	[3], [99]
DRAM+SCM	[103], [112]
DRAM+Flash	[69]
Comparison of multiple memory technologies	
SSD v/s HDD	[18], [20], [25], [39], [67], [97]
NVMs v/s DRAM and HDD	[27]
PCM v/s HDD	[3], [68]
STT-RAM v/s HDD	[134]

Balakrishnan et al. [60] present a technique to improve the reliability of an array of SSDs. Their technique works on the observation that balancing the number of writes in SSD arrays can lead to correlated failures since they may use-up their erasure cycles at similar rates. Thus, the array can be in a state where multiple devices have reached the end of their erasure limits, and thus all of them need to be replaced by newer devices. Their technique distributes parity blocks unevenly across the array. Since the parity blocks are updated more often than data blocks due to random access patterns, the devices holding more parity receive more writes and consequently age faster. This creates an age differential on drives and reduces the probability of correlated failures. When an oldest device is replaced by a new one, their technique reshuffles the parity distribution.

4 Research Projects on Combining or Comparing Multiple Memory Technologies

Since each memory technology has its own advantages and disadvantages, several researchers propose combining multiple technologies to bring together the best of them, while others present a comparison study to provide insights into the tradeoffs involved. Table 3 summarizes these techniques. We now discuss a few of them.

4.1 Flash+ HDD

Chen et al. [8] present a technique to leverage low cost of HDD and high speed of SSD to bring the best of both together. Their technique detects performance-critical blocks based on workload access patterns and moves only the most performance-critical blocks in the SSD. Also, semantically-critical blocks (e.g. file system metadata) are given priority to stay in SSD for improving performance. Further, for improving the performance of write-intensive workloads, incoming writes are buffered into the low-latency SSD.

Yang and Ren [44] present a storage design consisting of both HDD and SSD which aims to leverage the best features of both. In their design, SSD stores read-intensive reference blocks and HDD stores the deltas between currently accessed blocks and the corresponding reference blocks. They also use an algorithm that computes deltas upon I/O writes and combines deltas with reference blocks upon I/O reads to interface the OS. Thus, their approach aims to utilize fast read performance of SSD and fast sequential write performance of HDD, while avoiding slow SSD writes which improves its performance and lifetime.

Soundararajan et al. [59] propose a method to increase the lifetime of SSDs by using hard disk drive as a persistent write cache for an MLC-based SSD. Their technique appends all the writes to a log stored on the HDD and eventually migrates them to SSD, preferably before subsequent reads. They observe that HDDs can match the sequential write bandwidth of mid-range SSDs. Also, typical workloads contain a significant fraction of block overwrites and hence, by maintaining a log-structured HDD, the HDD can be operated at its fast sequential write mode. At the same time, the write-coalescing performed by the HDD increases the sequentiality of the workload as observed by the SSD and also reduces writes to it, which improves its lifetime.

For virtual memory management, Liu et al. [65] propose integrating HDD with SSD to overcome the limited endurance issue of Flash, while also bounding performance loss incurred due to swapping to fulfill QoS requirements. Their technique sequentially swaps a set of pages of virtual memory to the HDD if they are expected to be read together. Further, based on the page-access history, their technique creates an out-of-memory virtual memory page layout which spans both HDD and SSD. Using this, the random reads can be served by SSD and the sequential reads are asynchronously served by the HDD, thus the total bandwidth of the two devices can be used to accelerate page swapping, while also avoiding their individual limitations.

Wu and Reddy [83] present a technique to leverage both SSD and HDD to improve performance in a hybrid storage system. Their technique utilizes both initial block allocation and migration to reach an equilibrium state where the response times of different devices equalize. Their technique keeps track of the request response times of different devices (viz. SSD or HDD) and performs allocation to prefer the faster device while also achieving workload-balancing on the devices. Also, their technique detects whether a block is cold/hot and cold data are migrated only in the background.

Kim et al. [39] present a hybrid HDD-SSD design that exploits the complementary properties of these two media to provide high performance and service differentiation under a given cost budget. Their technique uses statistical models for performance of SSD and HDD to make dynamic request partitioning decisions. Since random writes cause fragmentation on SSD and increase the garbage collection overhead and latency of SSD, their technique periodically migrates some pages from SSD to HDD so that portions of writes can be redirected to the HDD and the randomness can be reduced. They also develop a model to find the most economical HDD/SSD configuration for given workloads using Mixed Integer Linear Programming (ILP).

4.2 Flash+PCM

Sun et al. [31] propose using PCM as the log region of the NAND Flash memory storage system. PCM log region supports in-place updating and avoids the need of out-of-date log records. Their approach reduces both read and erase operations to Flash, which improve the lifetime, performance and energy efficiency of Flash storage. Due to the byte-addressable nature of PCM, the performance of read-operations is also improved. They also propose techniques to ensure that the PCM log region does not not wear out before the Flash memory.

4.3 PCM+Flash+HDD

Kim et al. [68] evaluate the potential of PCM in storage hierarchy, considering its cost and performance. They observe that although based on the *material-level* characteristics, write to Flash is slower than that to PCM, based on *system-level* characteristics, writes to a Flash-based SSD can be faster than that of a PCM-based SSD, due to reasons such as power constraints etc. Based on this insight, they study two storage use-cases: tiering and caching. For tiering, they model a storage system consisting of Flash, HDD, and PCM to identify the combinations of device types that offer the best performance within cost constraints. They observe that PCM can improve the performance of a tiered storage system and certain combinations (e.g. 30% PCM + 67% Flash + 3% HDD combination) can provide higher performance per dollar than a combination without PCM. For caching, they compare aggregate I/O time and read latency of PCM with that of Flash. They observe that a combination of Flash and PCM SSDs can provide better read latency and aggregate I/O time than a Flash only configuration.

4.4 DRAM+PCM

Qureshi et al. [99] present a hybrid main memory system where DRAM is used as a "page cache" for the PCM memory to combine latency advantage of DRAM with capacity advantage of PCM. They propose several policies to mitigate write-overhead to PCM memory and increase its lifetime. On a page fault, the page fetched from hard disk is written to DRAM only. This page is written to PCM only when it is evicted from DRAM and it is marked as dirty. Also, the writes to a page are tracked at the granularity of a cache line and only the lines modified in a page are written back to reduce the effective number of writes to main memory. Further, to achieve wear-leveling, the lines in each page are stored in the PCM in a rotated manner.

4.5 DRAM+SCM

Bailey et al. [112] present a persistent, versioned, key-value store with a standard get/put interface for applications. Their approach maintains the permanent data in SCM and uses DRAM as a thin layer on top of SCM to address its performance and wear-out limitations. The threads maintain and manipulate local, volatile data in DRAM and only committed, persistent state is written to SCM, thus avoiding the slow-write bottleneck of SCMs. The byte-addressable nature of SCM is used for fine-grained transactions in non-volatile memory, and snapshot isolation is used for supporting concurrency, consistency, recoverability and versioning, for example, on a power-loss failure, the data previously committed to SCM can be accessed on resumption and will be in a consistent state.

Payer et al. [87] present a technique to leverage both low-cost, high-capacity HDD and high-cost low-capacity SSD at same level in memory hierarchy. Their technique allows using either a low-performance SSD or a high-performance SSD. For the case of a low-performance SSD, executable files and program libraries are moved to the SSD and the remaining files are moved to HDD; also randomly accessed files are moved to SSD and the files with mixed- or contiguous-access pattern are moved to HDD. When a high-performance SSD is used which has throughput nearly equal to that of HDD, the most frequently used files are moved are SSD and the remaining files are moved to HDD.

4.6 SSD v/s HDD

Narayanan et al. [25] compare the performance, energy consumption and cost of hard-disks with that of Flash-based SSDs in year 2009. For a range of data-center workloads, they analyze both complete replacement of disks by SSDs, and use of SSDs as an intermediate tier between disks and DRAM. They observe that due to low capacity per dollar of SSDs, replacing disks by SSDs is not an optimal solution for their workloads. They find that the capacity/dollar of SSDs need to be improved by a factor of 3 to 3,000 to make them comparable with disks.

Albrecht et al. [97] performed similar study in year 2013 and found that due to declining price of Flash along with relatively steady performance of disk, the break-even point has been shifting and Flash is now becoming economical for a larger range of workloads.

4.7 NVMs v/s DRAM and HDD

Caulfield et al. [27] compare several memory technologies, such as DRAM, NVMs and HDD. They measure the impact of these technologies on I/O-intensive, database, and memory-intensive applications which have varying latencies, bandwidth requirements and access patterns. They also study the effect of different options for connecting memory technologies to the host system. They observe that NVMs provide large gains in both application-level and raw I/O performance. For some applications, PCM and STT-RAM can provide a magnitude-order improvement in performance over HDDs.

5 Future Challenges and Conclusion

In an era of data explosion, the storage and processing demands on memory systems are increasing tremendously. While it is clear that conventional memory technologies fall far short of the demands of future computing systems, NVM technologies, as they stand today, are also unable to meet the performance, energy efficiency and reliability targets for future systems. We believe that meeting these challenges will require effective management of NVMs at multiple layers of abstraction, ranging from device level to system level.

At device level, for example, 3D design can lead to denser form factor, smaller footprint and lower latencies [21]. Also, fabrication of higher capacity SCM prototypes and their mass production will fuel further research into them. Although MLC NVM provides higher density and lower price than SLC NVM, its endurance is generally two-three orders of magnitude lower than that of SLC NVM, for example, the endurance of a 70 nm SLC Flash is around 100 K cycles, that of a 2-bit 2x nm MLC Flash is around 3 K cycles while for 3-bit MCL Flash, this value is only a few hundred cycles [100]. Since the increasing demand of memory capacity may necessitate the use of MLC NVM, effective software-schemes are required for mitigating the NVM write overhead. Further, NVMs are generally considered immune to radiation-induced soft errors, however, soft errors may appear in NVMs from stochastic bit-inversions due to thermal noise [138] or change of value stored in MLC PCM cell over time due to resistance drift [120]. In near future, along with performance and energy, researchers need to also look into issues such as soft-error resilience of NVMs.

At architecture level, synergistic integration of techniques for write-minimization, wear-leveling and fault-tolerance will help in achieving magnitude order improvement in lifetime of NVM memory systems. At system-level, unification of memory/storage management in a single address space via hardware/software cooperation can further minimize the overheads of the two-level storage model. Since computing systems of all sizes and shapes, ranging from milli-watt handheld systems to megawatt data centers and supercomputers depend on efficient memory systems and present different constraints and optimization objectives, accounting for their unique features will be extremely important to design techniques optimized for different platforms and usage scenarios. Finally, these technologies also need to be highly cost-competitive to justify their use in commodity market.

In this paper, we presented a survey of techniques which utilize NVMs for main memory and storage systems. We also discussed techniques which combine or compare multiple memory technologies to study their relative merits and bring the best of them together. We classified these techniques on several key parameters to highlight their similarities and differences and identify major research trends. It is hoped that this survey will inspire novel ideas for fully leveraging the potential of NVMs in future computing systems.

Acknowledgments

Support for this work was provided by U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research.

References

[1] (2015). [Online]. Available: www.youtube.com/yt/press/statistics.html
[2] (2013). [Online]. Available: http://goo.gl/qwyFHe
[3] X. Dong, N. Muralimanohar, N. Jouppi, R. Kaufmann, and Y. Xie, "Leveraging 3D PCRAM technologies to reduce checkpoint overhead for future exascale systems," in *Proc. High Perform. Comput. Netw., Storage Anal.*, 2009, pp. 57:1–57:12.
[4] B. Giridhar, M. Cieslak, D. Duggal, R. Dreslinski, H. M. Chen, R. Patti, B. Hold, C. Chakrabarti, T. Mudge, and D. Blaauw, "Exploring DRAM organizations for energy-efficient and resilient exascale memories," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2013, pp. 23:1–23:12.
[5] S. Mittal and J. Vetter, "A survey of methods for analyzing and improving GPU energy efficiency," *ACM Comput. Surveys*, vol. 47, no. 2, pp. 19:1–19:23, 2015.
[6] J. S. Vetter and S. Mittal, "Opportunities for nonvolatile memory systems in extreme-scale high performance computing," *Comput. Sci. Eng.*, vol. 17, no. 2, pp. 73–82, Mar.-Apr. 2015.

[7] S. Mittal, J. S. Vetter, and D. Li, "A survey of architectural approaches for managing embedded DRAM and non-volatile on-chip caches," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 6, pp. 1524–1537, Jun. 2014.

[8] F. Chen, D. A. Koufaty, and X. Zhang, "Hystor: Making the best use of solid state drives in high performance storage systems," in *Proc. Int. Conf. Supercomput.*, 2011, pp. 22–32.

[9] (2012). Flash drives replace disks at Amazon, Facebook, Dropbox. [Online]. Available: www.wired.com/2012/06/flash-data-centers/

[10] (2011). Meet Gordon, the world's first flash supercomputer. [Online]. Available: www.wired.com/2011/12/gordon-super-computer/

[11] F. Schürmann, F. Delalondre, P. S. Kumbhar, J. Biddiscombe, M. Gila, D. Tacchella, A. Curioni, B. Metzler, P. Morjan, J. Fenkes, M. M. Franceschini, R. S. Germain, L. Schneidenbach, T. J. C. Ward, and B. G. Fitch, "Rebasing I/O for scientific computing: Leveraging storage class memory in an IBM BlueGene/Q supercomputer," in *Proc. Int. Conf. Supercomput.*, 2014, pp. 331–347.

[12] (2014). Oak Ridge to acquire next generation supercomputer. [Online]. Available: http://goo.gl/d315UD

[13] H. Volos, A. J. Tack, and M. M. Swift, "Mnemosyne: Lightweight persistent memory," *ACM SIGARCH Comput. Archit. News*, vol. 39, no. 1, 2011, pp. 91–104.

[14] S. Chen, P. B. Gibbons, and S. Nath, "Rethinking database algorithms for phase change memory," in *Proc. 5th Biennial Conf. Innovative Data Syst. Res.*, 2011, pp. 21–31.

[15] P. Pavan, R. Bez, P. Olivo, and E. Zanoni, "Flash memory cells-an overview," *Proc. IEEE*, vol. 85, no. 8, pp. 1248–1271, Aug. 1997.

[16] S. Mittal, "A survey of power management techniques for phase change memory," *Int. J. Comput. Aided Eng. Technol.*, 2014.

[17] M. Kryder and C. Kim, "After hard drives-what comes next?" *IEEE Trans. Magn.*, vol. 45, no. 10, pp. 3406–3413, Oct. 2009.

[18] F. Chen, D. A. Koufaty, and X. Zhang, "Understanding intrinsic characteristics and system implications of flash memory based solid state drives," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 37, no. 1, pp. 181–192, 2009.

[19] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. S. Manasse, and R. Panigrahy, "Design tradeoffs for SSD performance," in *Proc. USENIX Annu. Tech. Conf.*, 2008, pp. 57–70.

[20] M. A. Sanvido, F. R. Chu, A. Kulkarni, and R. Selinger, "NAND flash memory and its role in storage architectures," *Proc. IEEE*, vol. 96, no. 11, pp. 1864–1874, Nov. 2008.

[21] M. Poremba, S. Mittal, D. Li, J. S. Vetter, and Y. Xie, "DESTINY: A tool for modeling emerging 3D NVM and eDRAM caches," in *Proc. Des. Autom. Test Eur.*, 2015, pp. 1543–1546.

[22] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings," in *Proc. ACM Int. Conf. Archit. Support Programm. Lang. Operating Syst.*, 2009, pp. 229–240.

[23] L.-P. Chang and L.-C. Huang, "A low-cost wear-leveling algorithm for block-mapping solid-state disks," *SIGPLAN Notices*, vol. 46, no. 5, pp. 31–40, 2011.

[24] S.-W. Lee and B. Moon, "Design of flash-based DBMS: an in-page logging approach," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2007, pp. 55–66.

[25] D. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, and A. Rowstron, "Migrating server storage to SSDs: Analysis of tradeoffs," in *Proc. Eur. Conf. Comput. Syst.*, 2009, pp. 145–158.

[26] J. Meza, Y. Luo, S. Khan, J. Zhao, Y. Xie, and O. Mutlu, "A case for efficient hardware/software cooperative management of storage and memory," in *Proc. Workshop Energy Efficient Des.*, 2013.

[27] A. M. Caulfield, J. Coburn, T. Mollov, A. De, A. Akel, J. He, A. Jagatheesan, R. K. Gupta, A. Snavely, and S. Swanson, "Understanding the impact of emerging non-volatile memories on high-performance, IO-intensive computing," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2010, pp. 1–11.

[28] R. Fackenthal, M. Kitagawa, W. Otsuka, K. Prall, D. Mills, K. Tsutsui, J. Javanifard, K. Tedrow, T. Tsushima, Y. Shibahara, and G. Hush, "A 16 Gb ReRAM with 200 MB/s write and 1 GB/s read in 27 nm technology," in *Proc. IEEE Int. Solid-State Circuits Conf.*, 2014, pp. 338–339.

[29] M. Wu and W. Zwaenepoel, "eNVy: A non-volatile, main memory storage system," *ACM SigPlan Notices*, vol. 29, no. 11, pp. 86–97, 1994.

[30] L.-P. Chang, "On efficient wear leveling for large-scale flash-memory storage systems," in *Proc. ACM Symp. Applied Comput.*, 2007, pp. 1126–1130.

[31] G. Sun, Y. Joo, Y. Chen, D. Niu, Y. Xie, Y. Chen, and H. Li, "A hybrid solid-state storage architecture for the performance, energy consumption, and lifetime improvement," in *Proc. Int. Symp. High Perform. Comput. Archit.*, 2010, pp. 1–12.

[32] S. Lee, J. Kim, and A. Mithal, "Refactored design of I/O architecture for flash storage," *Comput. Archit. Lett.*, 2014, Doi: 10.1109/LCA.2014.2329423.

[33] Y. Cai, G. Yalcin, O. Mutlu, E. F. Haratsch, O. Unsal, A. Cristal, and K. Mai, "Neighbor-cell assisted error correction for MLC NAND flash memories," in *Proc. Int. Conf. Meas. Modeling Comput. Syst.*, 2014, pp. 491–504.

[34] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai, "Error patterns in MLC NAND flash memory: Measurement, characterization, and analysis," in *Proc. Des., Autom. Test Eur.*, 2012, pp. 521–526.

[35] Y. Cai, O. Mutlu, E. F. Haratsch, and K. Mai, "Program interference in MLC NAND flash memory: Characterization, modeling, and mitigation," in *Proc. Int. Conf. Comput. Des.*, 2013, pp. 123–130.

[36] F. Chen, T. Luo, and X. Zhang, "CAFTL: A content-aware flash translation layer enhancing the lifespan of flash memory based solid state drives," in *Proc. USENIX Conf. File Storage Technol.*, vol. 11, 2011.

[37] F. Chen, R. Lee, and X. Zhang, "Essential roles of exploiting internal parallelism of flash memory based solid state drives in high-speed data processing," in *Proc. Int. Symp. High Perform. Comput. Archit.*, 2011, pp. 266–277.

[38] B. Cully, J. Wires, D. Meyer, K. Jamieson, K. Fraser, T. Deegan, D. Stodden, G. Lefebvre, D. Ferstay, and A. Warfield, "Strata: Scalable high-performance storage on virtualized non-volatile memory," in *Proc. USENIX Conf. File Storage Technol.*, 2014, pp. 17–31.

[39] Y. Kim, A. Gupta, B. Urgaonkar, P. Berman, and A. Sivasubramaniam, "HybridStore: A cost-efficient, high-performance storage system combining SSDs and HDDs," in *Proc. Int. Symp. Modeling, Anal. Simul. Comput. Telecommun. Syst.*, 2011, pp. 227–236.

[40] S. Park, Y. Kim, B. Urgaonkar, J. Lee, and E. Seo, "A comprehensive study of energy efficiency and performance of flash-based SSD," *J. Syst. Archit.*, vol. 57, no. 4, pp. 354–365, 2011.

[41] M. Jung, W. Choi, J. Shalf, and M. T. Kandemir, "Triple-A: A Non-SSD based autonomic all-flash array for high performance storage systems," in *Proc. 19th Int. Conf. Archit. Support Programm. Lang. Operating Syst.*, 2014, pp. 441–454.

[42] S. Cho, C. Park, H. Oh, S. Kim, Y. Yi, and G. R. Ganger, "Active disk meets flash: A case for intelligent SSDs," in *Proc. Int. Conf. Supercomput.*, 2013, pp. 91–102.

[43] P. Huang, G. Wan, K. Zhou, M. Huang, C. Li, and H. Wang, "Improve effective capacity and lifetime of solid state drives," in *Proc. Int. Conf. Netw., Archit. Storage*, 2013, pp. 50–59.

[44] Q. Yang and J. Ren, "I-CASH: Intelligently coupled array of SSD and HDD," in *Proc. Int. Symp. High Perform. Comput. Archit.*, 2011, pp. 278–289.

[45] Y. Wang, L. A. D. Bathen, N. D. Dutt, and Z. Shao, "Meta-Cure: A reliability enhancement strategy for metadata in NAND flash memory storage systems," in *Proc. Des. Autom. Conf.*, 2012, pp. 214–219.

[46] C. Wang and W.-F. Wong, "SAW: System-assisted wear leveling on the write endurance of NAND flash devices," in *Proc. Des. Autom. Conf.*, 2013, pp. 1–9.

[47] Y.-H. Chang, J.-W. Hsieh, and T.-W. Kuo, "Endurance enhancement of flash-memory storage systems: An efficient static wear leveling design," in *Proc. Des. Autom. Conf.*, 2007, pp. 212–217.

[48] C. Wang and W.-F. Wong, "Extending the lifetime of nand flash memory by salvaging bad blocks," in *Proc. Conf. Des., Autom. Test Eur.*, 2012, pp. 260–263.

[49] C. Wang and W.-F. Wong, "Observational wear leveling: an efficient algorithm for flash memory management," in *Proc. Des. Autom. Conf.*, 2012, pp. 235–242.

[50] J. Liao, F. Zheng, L. Li, and G. Xiao, "Adaptive wear-leveling in flash-based memory," *Comput. Archit. Lett.*, 2014, Doi: 10.1109/LCA.2014.2329871.

[51] R. Maddah, S. Cho, and R. Melhem, "Data dependent sparing to manage better-than-bad blocks," *Comput. Archit. Lett.*, vol. 12, no. 2, pp. 43–46, 2013.

[52] L. M. Grupp, A. M. Caulfield, J. Coburn, S. Swanson, E. Yaakobi, P. H. Siegel, and J. K. Wolf, "Characterizing flash memory: Anomalies, observations, and applications," in *Proc. Int. Symp. Microarchit.*, 2009, pp. 24–33.

[53] S. Boboila and P. Desnoyers, "Write endurance in flash drives: Measurements and analysis," in *Proc. USENIX Conf. File Storage Technol.*, vol. 10, 2010.

[54] L. M. Grupp, J. D. Davis, and S. Swanson, "The bleak future of NAND flash memory," in *Proc. USENIX Conf. File Storage Technol.*, 2012.

[55] Y. Zhang, L. P. Arulraj, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "De-indirection for flash-based SSDs with nameless writes," in *Proc. USENIX Conf. File Storage Technol.*, 2012.

[56] R.-S. Liu, C.-L. Yang, and W. Wu, "Optimizing NAND flash-based SSDs via retention relaxation," in *Proc. USENIX Conf. File Storage Technol.*, 2012.

[57] H. Wang, P. Huang, S. He, K. Zhou, C. Li, and X. He, "A novel I/O scheduler for SSD with improved performance and lifetime," in *Proc. Symp. Mass Storage Syst. Technol.*, 2013, pp. 1–5.

[58] J. Lee, Y. Kim, G. M. Shipman, S. Oral, F. Wang, and J. Kim, "A semi-preemptive garbage collector for solid state drives," in *Proc. Int. Symp. Perform. Anal. Syst. Softw.*, 2011, pp. 12–21.

[59] G. Soundararajan, V. Prabhakaran, M. Balakrishnan, and T. Wobber, "Extending SSD Lifetimes with disk-based write caches," in *Proc. USENIX Conf. File Storage Technol.*, 2010, vol. 10, pp. 101–114.

[60] M. Balakrishnan, A. Kadav, V. Prabhakaran, and D. Malkhi, "Differential RAID: Rethinking RAID for SSD reliability," *ACM Trans. Storage*, vol. 6, no. 2, p. 4, 2010.

[61] J. Matthews, S. Trika, D. Hensgen, R. Coulson, and K. Grimsrud, "Intel® turbo memory: Nonvolatile disk caches in the storage hierarchy of mainstream computer systems," *ACM Trans. Storage*, vol. 4, no. 2, pp. 4:1–4:24, 2008.

[62] D. Narayanan and O. Hodson, "Whole-system persistence," in *Proc. ACM SIGARCH Comput. Archit. News*, vol. 40, no. 1, pp. 401–410, 2012.

[63] C. Dirik and B. Jacob, "The performance of PC solid-state disks (SSDs) as a function of bandwidth, concurrency, device architecture, and system organization," *ACM SIGARCH Comput. Archit. News*, vol. 37, no. 3, pp. 279–289, 2009.

[64] H. Kim and S. Ahn, "BPLRU: A buffer management scheme for improving random writes in flash storage," in *Proc. USENIX Conf. File Storage Technol.*, 2008, vol. 8, pp. 1–14.

[65] K. Liu, X. Zhang, K. Davis, and S. Jiang, "Synergistic coupling of SSD and hard disk for QoS-aware virtual memory," in *Proc. Int. Symp. Perform. Anal. Syst. Softw.*, 2013, pp. 24–33.

[66] I. Koltsidas and S. D. Viglas, "Flashing up the storage layer," *VLDB Endowment*, vol. 1, no. 1, pp. 514–525, 2008.

[67] S. Park and K. Shen, "A performance evaluation of scientific I/O workloads on flash-based SSDs," in *Proc. IEEE Int. Conf. Cluster Comput. Workshops*, 2009, pp. 1–5.

[68] H. Kim, S. Seshadri, C. L. Dickey, and L. Chiu, "Evaluating phase change memory for enterprise storage systems: A study of caching and tiering approaches," in *Proc. USENIX Conf. File Storage Technol.*, 2014, pp. 33–45.

[69] A. Badam and V. S. Pai, "SSDAlloc: Hybrid SSD/RAM memory management made easy," in *Proc. USENIX Conf. Netw. Syst. Des. Implementation*, 2011, pp. 16–16.

[70] Y. Oh, J. Choi, D. Lee, and S. H. Noh, "Caching less for better performance: balancing cache size and update cost of flash memory cache in hybrid storage systems," in *Proc. USENIX Conf. File Storage Technol.*, vol. 12, 2012.

[71] T. Pritchett and M. Thottethodi, "SieveStore: A highly-selective, ensemble-level disk cache for cost-performance," in *Proc. Int. Symp. Comput. Archit.*, 2010, pp. 163–174.

[72] L. Shi, K. Qiu, M. Zhao, and C. J. Xue, "Error model guided joint performance and endurance optimization for flash memory," *IEEE Trans. Comput.-Aided Des. Integrated Circuits Syst.*, vol. 33, no. 3, pp. 343–355, Mar. 2014.

[73] J. Yang, N. Plasson, G. Gillis, N. Talagala, S. Sundararaman, and R. Wood, "HEC: Improving endurance of high performance flash-based cache devices," in *Proc. Int. Syst. Storage Conf.*, 2013, p. 10.

[74] K. Zhao, K. Venkataraman, X. Zhang, J. Li, N. Zheng, and T. Zhang, "Over-clocked SSD: Safely running beyond flash memory chip I/O clock specs," in *Proc. Int. Symp. High Perform. Comput. Archit.*, 2014, pp. 536–545.

[75] L. Shi, J. Li, C. Jason Xue, and X. Zhou, "Hybrid nonvolatile disk cache for energy-efficient and high-performance systems," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 18, no. 1, pp. 8:1–8:23, 2013.

[76] M. Saxena, M. M. Swift, and Y. Zhang, "FlashTier: A lightweight, consistent and durable storage cache," in *Proc. Eur. Conf. Comput. Syst.*, 2012, pp. 267–280.

[77] R. Koller, L. Marmol, R. Rangaswami, S. Sundararaman, N. Talagala, and M. Zhao, "Write policies for host-side flash caches," in *Proc. USENIX Conf. File Storage Technol.*, 2013, pp. 45–58.

[78] R.-S. Liu, C.-L. Yang, C.-H. Li, and G.-Y. Chen, "DuraCache: A durable SSD cache using MLC NAND flash," in *Proc. Des. Autom. Conf.*, 2013, p. 166.

[79] P. Huang, P. Subedi, X. He, S. He, and K. Zhou, "FlexECC: Partially relaxing ECC of MLC SSD for better cache performance," in *Proc. USENIX Annu. Tech. Conf.*, 2014, pp. 489–500.

[80] Y. Pan, G. Dong, Q. Wu, and T. Zhang, "Quasi-nonvolatile SSD: Trading flash memory nonvolatility to improve storage system performance for enterprise applications," in *Proc. Int. Symp. High Perform. Comput. Archit.*, 2012, pp. 1–10.

[81] Y. Cai, G. Yalcin, O. Mutlu, E. F. Haratsch, A. Cristal, O. S. Unsal, and K. Mai, "Flash correct-and-refresh: Retention-aware error management for increased flash memory lifetime," in *Proc. Int. Conf. Comput. Des.*, 2012, pp. 94–101.

[82] L. Shi, C. Xue, J. Hu, W.-C. Tseng, X. Zhou, and E. Sha, "Write activity reduction on flash main memory via smart victim cache," in *Proc. Great Lakes Symp.*, 2010, pp. 91–94.

[83] X. Wu and A. N. Reddy, "Exploiting concurrency to improve latency and throughput in a hybrid storage system," in *Proc. Int. Symp. Modeling, Anal. Simul. Comput. Telecommun. Syst.*, 2010, pp. 14–23.

[84] V. Papirla and C. Chakrabarti, "Energy-aware error control coding for flash memories," in *Proc. Des. Autom. Conf.*, 2009, pp. 658–663.

[85] A. Sampson, J. Nelson, K. Strauss, and L. Ceze, "Approximate storage in solid-state memories," in *Proc. Int. Symp. Microarchit.*, 2013, pp. 25–36.

[86] H. Shim, J. Kim, and S. Maeng, "BEST: Best-effort energy saving techniques for NAND flash-based hybrid storage," *IEEE Trans. Consum. Electron.*, vol. 58, no. 3, pp. 841–848, Aug. 2012.

[87] H. Payer, M. A. Sanvido, Z. Z. Bandic, and C. M. Kirsch, "Combo drive: Optimizing cost and performance in a heterogeneous storage device," in *Proc. Workshop Integrating Solid-State Memory into Storage Hierarchy*, 2009, vol. 1, pp. 1–8.

[88] S. Jung, Y. Lee, and Y. Song, "A process-aware hot/cold identification scheme for flash memory storage systems," *IEEE Trans. Consum. Electron.*, vol. 56, no. 2, pp. 339–347, May 2010.

[89] C. Wang, S. S. Vazhkudai, X. Ma, F. Meng, Y. Kim, and C. Engelmann, "NVMalloc: Exposing an aggregate SSD store as a memory partition in extreme-scale machines," in *Proc. Int. Parallel Distrib. Process. Symp.*, 2012, pp. 957–968.

[90] M. Gamell, I. Rodero, M. Parashar, and S. Poole, "Exploring energy and performance behaviors of data-intensive scientific workflows on systems with deep memory hierarchies," in *Proc. Int. Conf. High Perform. Comput.*, 2013, pp. 226–235.

[91] T. Kgil, D. Roberts, and T. Mudge, "Improving NAND flash based disk caches," in *Proc. Int. Symp. Comput. Archit.*, 2008, pp. 327–338.

[92] Q. Wu and T. Zhang, "OFWAR: Reducing SSD response time using on-demand fast-write-and-rewrite," *IEEE Trans. Comput.*, vol. 63, no. 10, pp. 2500–2512, Oct. 2014.

[93] J. Jeong, S. S. Hahn, S. Lee, J. Kim, J. Jeong, S. S. Hahn, S. Lee, and J. Kim, "Lifetime improvement of NAND flash-based storage systems using dynamic program and erase scaling," in *Proc. 12th USENIX Conf. File Storage Technol.*, 2014, pp. 61–74.

[94] G. Wu and X. He, "Reducing SSD read latency via NAND flash program and erase suspension," in *Proc. FAST*, 2012.

[95] B. Van Essen, H. Hsieh, S. Ames, R. Pearce, and M. Gokhale, "DI-MMAP-a scalable memory-map runtime for out-of-core data-intensive applications," *Cluster Comput.*, vol. 18, no. 1, pp. 15–28, 2013.

[96] B. Ransford, J. Sorber, and K. Fu, "Mementos: System support for long-running computation on RFID-scale devices," in *Proc. Archit. Support Programm. Lang. Operating Syst.*, 2011, pp. 159–170.

[97] C. Albrecht, A. Merchant, M. Stokely, M. Waliji, F. Labelle, N. Coehlo, X. Shi, and E. Schrock, "Janus: Optimal flash provisioning for cloud storage workloads," in *Proc. USENIX Annu. Tech. Conf.*, 2013, pp. 91–102.

[98] N. Liu, J. Cope, P. Carns, C. Carothers, R. Ross, G. Grider, A. Crume, and C. Maltzahn, "On the role of burst buffers in leadership-class storage systems," in *Proc. Symp. Mass Storage Syst. Technol.*, 2012, pp. 1–11.

[99] M. Qureshi, V. Srinivasan, and J. Rivers, "Scalable high performance main memory system using phase-change memory technology," in *Proc. Int. Symp. Comput. Architect.*, 2009, pp. 24–33.

[100] S. Lee, T. Kim, K. Kim, and J. Kim, "Lifetime management of flash-based SSDs using recovery-aware dynamic throttling," in *Proc. FAST*, 2012.

[101] D. Li, J. S. Vetter, G. Marin, C. McCurdy, C. Cira, Z. Liu, and W. Yu, "Identifying opportunities for byte-addressable non-volatile memory in extreme-scale scientific applications," in *Proc. Int. Parallel Distrib. Process. Symp.*, 2012, pp. 945–956.

[102] R.-S. Liu, D.-Y. Shen, C.-L. Yang, S.-C. Yu, and C.-Y. M. Wang, "NVM duet: Unified working memory and persistent store architecture," in *Proc. Archit. Support Programm. Lang. Operating Syst.*, 2014, pp. 455–470.

[103] J. Condit, E. B. Nightingale, C. Frost, E. Ipek, B. Lee, D. Burger, and D. Coetzee, "Better I/O through byte-addressable, persistent memory," in *Proc. ACM SIGOPS 22nd Symp. Operating Syst. Principles*, 2009, pp. 133–146.

[104] E. Giles, K. Doshi, and P. Varman, "Bridging the programming gap between persistent and volatile memory using WrAP," in *Proc. Int. Conf. Comput. Frontiers*, 2013, p. 30.

[105] E. Lee, S. Yoo, J.-E. Jang, and H. Bahn, "Shortcut-JFS: A write efficient journaling file system for phase change memory," in *Proc. Symp. Mass Storage Syst. Technol.*, 2012, pp. 1–6.

[106] J.-Y. Jung and S. Cho, "Memorage: Emerging persistent ram based malleable main memory and storage architecture," in *Proc. Int. Conf. Supercomput.*, 2013, pp. 115–126.

[107] A. Akel, A. M. Caulfield, T. I. Mollov, R. K. Gupta, and S. Swanson, "Onyx: A protoype phase change memory storage array," in *Proc. 3rd USENIX Conf. Hot Topics Storage File Syst.*, 2011, p. 2.

[108] Z. Liu, B. Wang, P. Carpenter, D. Li, J. S. Vetter, and W. Yu, "PCM-Based durable write cache for fast disk I/O," in *Proc. Int. Symp. Modeling, Anal. Simul. Comput. Telecommun. Syst.*, 2012, pp. 451–458.

[109] A. M. Caulfield, A. De, J. Coburn, T. I. Mollow, R. K. Gupta, and S. Swanson, "Moneta: A high-performance storage array architecture for next-generation, non-volatile memories," in *Proc. Int. Symp. Microarchit.*, 2010, pp. 385–395.

[110] J. Coburn, A. M. Caulfield, A. Akel, L. M. Grupp, R. K. Gupta, R. Jhala, and S. Swanson, "NV-Heaps: Making persistent objects fast and safe with next-generation, non-volatile memories," in *Proc. ACM SIGARCH Comput. Archit. News*, vol. 39, no. 1, 2011, pp. 105–118.

[111] J. C. Mogul, E. Argollo, M. A. Shah, and P. Faraboschi, "Operating system support for NVM+ DRAM hybrid main memory," in *Proc. 12th Conf. Hot Topics Operating Syst.*, 2009, p. 14.

[112] K. A. Bailey, P. Hornyack, L. Ceze, S. D. Gribble, and H. M. Levy, "Exploring storage class memory with key value stores," in *Proc. Workshop Interactions NVM/FLASH Operating Syst. Workloads*, 2013, p. 4.

[113] S. Pelley, P. M. Chen, and T. F. Wenisch, "Memory persistency," in *Proc. Int. Symp. Comput. Archit.*, 2014, pp. 265–276.

[114] S. Kannan, A. Gavrilovska, and K. Schwan, "Reducing the cost of persistence for nonvolatile heaps in end user devices," in *Proc. Int. Symp. High Perform. Comput. Archit.*, 2014, pp. 512–523.

[115] S. Kannan, A. Gavrilovska, K. Schwan, and D. Milojicic, "Optimizing checkpoints using NVM as virtual memory," in *Proc. Int. Symp. Parallel Distrib. Process.*, 2013, pp. 29–40.

[116] E. Ipek, J. Condit, E. B. Nightingale, D. Burger, and T. Moscibroda, "Dynamically replicated memory: Building reliable systems from nanoscale resistive memories," in *Proc. Archit. Support Programm. Lang. Operating Syst.*, 2010, pp. 3–14.

[117] S. Schechter, G. H. Loh, K. Straus, and D. Burger, "Use ECP, not ECC, for hard failures in resistive memories," in *Proc. Int. Symp. Comput. Archit.*, 2010, pp. 141–152.

[118] N. H. Seong, D. H. Woo, V. Srinivasan, J. A. Rivers, and H.-H. S. Lee, "SAFER: Stuck-at-fault error recovery for memories," in *Proc. Int. Symp. Microarchit.*, 2010, pp. 115–124.

[119] D. H. Yoon, N. Muralimanohar, J. Chang, P. Ranganathan, N. P. Jouppi, and M. Erez, "FREE-p: Protecting non-volatile memory against both hard and soft errors," in *Proc. Int. Symp. High Perform. Comput. Archit.*, 2011, pp. 466–477.

[120] M. Awasthi, M. Shevgoor, K. Sudan, B. Rajendran, R. Balasubramonian, and V. Srinivasan, "Efficient scrub mechanisms for error-prone emerging memories," in *Proc. Int. Symp. High Perform. Comput. Archit.*, 2012, pp. 1–12.

[121] J. Chen, G. Venkataramani, and H. H. Huang, "RePRAM: Recycling PRAM faulty blocks for extended lifetime," in *Proc. Int. Conf. Dependable Syst. Netw.*, 2012, pp. 1–12.

[122] T. Gao, K. Strauss, S. M. Blackburn, K. S. McKinley, D. Burger, and J. Larus, "Using managed runtime systems to tolerate holes in wearable memories," in *Proc. Conf. Programm. Lang. Des. Implementation*, 2013, pp. 297–308.

[123] Y. Fang, H. Li, and X. Li, "SoftPCM: Enhancing energy efficiency and lifetime of phase change memory in video applications via approximate write," in *Proc. IEEE Asian Test Symp.*, 2012, pp. 131–136.

[124] S. Im and D. Shin, "Differentiated space allocation for wear leveling on phase-change memory-based storage device," *IEEE Trans. Consum. Electron.*, vol. 60, no. 1, pp. 45–51, Feb. 2014.

[125] I. Moraru, D. G. Andersen, M. Kaminsky, N. Tolia, P. Ranganathan, and N. Binkert, "Consistent, durable, and safe memory management for byte-addressable non volatile main memory," in *Proc. 1st ACM SIGOPS Conf. Timely Results Operating Syst.*, 2013, pp. 1:1–1:17.

[126] S. Venkataraman, N. Tolia, P. Ranganathan, and R. H. Campbell, "Consistent and durable data structures for non-volatile byte-addressable memory," in *Proc. USENIX Conf. File Storage Technol.*, 2011, pp. 61–75.

[127] X. Wu and A. Reddy, "SCMFS: A file system for storage class memory," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2011, p. 39.

[128] S. Sardashti and D. Wood, "UniFI: Leveraging non-volatile memories for a unified fault tolerance and idle power management technique," in *Proc. Int. Conf. Supercomput.*, 2012, pp. 59–68.

[129] F. A. Aouda, K. Marquet, and G. Salagnac, "Incremental checkpointing of program state to NVRAM for transiently-powered systems," in *Proc. Int. Symp. Reconfigurable Commun.-Centric Syst.-on-Chip*, 2014, pp. 1–4.

[130] S. R. Dulloor, S. Kumar, A. Keshavamurthy, P. Lantz, D. Reddy, R. Sankaran, and J. Jackson, "System software for persistent memory," in *Proc. Eur. Conf. Comput. Syst.*, 2014, pp. 15:1–15:15.

[131] M. K. Qureshi, M. M. Franceschini, A. Jagmohan, and L. A. Lastras, "PreSET: Improving performance of phase change memories by exploiting asymmetry in write times," in *Proc. Int. Symp. Comput. Archit.*, 2012, pp. 380–391.

[132] D. R. Chakrabarti, H.-J. Boehm, and K. Bhandari, "Atlas: Leveraging locks for non-volatile memory consistency," in *Proc. Int. Conf. Object Oriented Programm. Syst. Lang. Appl.*, 2014, pp. 433–452.

[133] J. Zhao, S. Li, D. H. Yoon, Y. Xie, and N. P. Jouppi, "Kiln: Closing the performance gap between systems with and without persistence support," in *Proc. Int. Symp. Microarchit.*, 2013, pp. 421–432.

[134] H. Kim, J. Ahn, S. Ryu, J. Choi, and H. Han, "In-memory file system for non-volatile memory," in *Proc. Res. Adaptive Convergent Syst.*, 2013, pp. 479–484.

[135] S. Kannan, A. Gavrilovska, K. Schwan, D. Milojicic, and V. Talwar, "Using active NVRAM for I/O staging," in *Proc. Petascal Data Analytics: Challenges Opportunities*, 2011, pp. 15–22.

[136] Z. Wang, H. Yi, R. Liu, M. Dong, and H. Chen, "Persistent transactional memory," *Comput. Archit. Lett.*, 2014, Doi: 10.1109/LCA.2014.2329832.

[137] A. Pande, V. Ahuja, R. Sivaraj, E. Baik, and P. Mohapatra, "Video delivery challenges and opportunities in 4G networks," *IEEE MultiMedia*, vol. 20, no. 3, pp. 88–94, Jul.-Aug. 2013.

[138] S. Mittal and J. Vetter, "A survey of techniques for modeling and improving reliability of computing systems," *IEEE Trans. Parallel Distrib. Syst.*, 2015, Doi: 10.1109/TPDS.2015.2426179.

[139] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "OceanStore: An architecture for global-scale persistent storage," *ACM Sigplan Notices*, vol. 35, no. 11, pp. 190–201, 2000.

Sparsh Mittal received the BTech degree in electronics and communications engineering from IIT, Roorkee, India and the PhD degree in computer engineering from Iowa State University. He is currently working as a post-doctoral research associate at ORNL. His research interests include non-volatile memory, memory system power efficiency, cache, and GPU architectures. He is a member of the IEEE.

Jeffrey S. Vetter received the PhD degree from the Georgia Institute of Technology (GT). He holds a joint appointment between ORNL and GT. At ORNL, he is a distinguished R&D staff member, and the founding group leader of the Future Technologies Group. At GT, he is a joint professor in the Computational Science and Engineering School, the project director for the US National Science Foundation (NSF) Track 2D experimental computing facility for large scale heterogeneous computing using graphics processors, and the director of the NVIDIA CUDA Center of Excellence. His research interests include massively multithreaded processors, non-volatile memory, and heterogeneous multicore processors. He is a senior member of the IEEE.