The Impact of Non-Volatile Memory on Modern Computer Systems

Giulia Frascaria
Vrije Universiteit, Amsterdam
The Netherlands
g.frascaria@student.vu.nl

Abstract—**Storage was considered a necessary evil for many decades, with Hard Disk Drive (HDD) performance being the main bottleneck. However, recent advancements in Non-Volatile Memory technologies introduced NAND Flash and Intel Optane memory, that can deliver orders-of-magnitude improved performance. The physical characteristics of the new storage media being significantly different from the HDD ones, researchers were motivated to update the software layers involved in I/O. In addition to adapting existing layers of the storage stack, the improved performance of NVM fostered new research fields that put storage in a central role in computer systems rather than the outcast. This paper explores the impact that new NVM technologies are having in storage stack research, from low-level optimization to new research trends like heterogeneous memory systems and storage programmability**

1. Introduction

Computer systems have undergone a massive evolution in the recent years, and performance has been the utmost concern for improvements. However, for many decades the slow HDD technology was the only viable option for persistent storage, for both storage size and cost metrics. This relegated the storage to an outcast role in computer systems, given the heavy performance penalty that was involved in I/O operations. However, over the past decade a lot of new persistent memory technologies have been introduced (e.g. Flash Storage, Intel Optane Memory [1], Phase-Change Memory) and are currently in the process of being included in commercial devices and datacenters. The new storage media deliver orders-of-magnitude improved performance (Figure 1), with PCM and Optane memory offering latency and bandwidth that are comparable to DRAM. Although the performance of the media is significantly improved, updates in the software layers are needed to adapt the storage stack to the new medium and its physical characteristics.

An upgrade in storage technologies and software is particularly important in the current booming of Big Data and data-intensive applications like Machine Learning and Data Mining. The need to perform computation on the data, as well as the cost for retrieving and moving data across the (distributed) system, advocates for the need to improve performance throughout the whole storage stack.

The new NVM media promise to deliver an access latency that is orders of magnitude lower than that of HDDs, higher bandwidth and increased parallelism at a commercially feasible cost. This can clearly have a massive impact on the performance of the overall computer systems, since the I/O performance has been one of the main performance bottlenecks, leading to the integration of multiple memory layers with reduced access latencies as they get closer to the CPU. The assumption that access to persistent storage is now efficient clearly stimulates to reconsider all design decisions that are in place to this point, and that guided the design of HDD-based file systems and storage stacks in general. However, despite the technological advances, research is still active in the field to find optimal solutions that allow to exploit their full potential. The different hardware characteristics force to adapt the old interfaces and software stacks to the new underlying hardware. The increased variety of storage media highlights an increasingly important issue, that is the need for an efficient support for heterogeneous hardware.

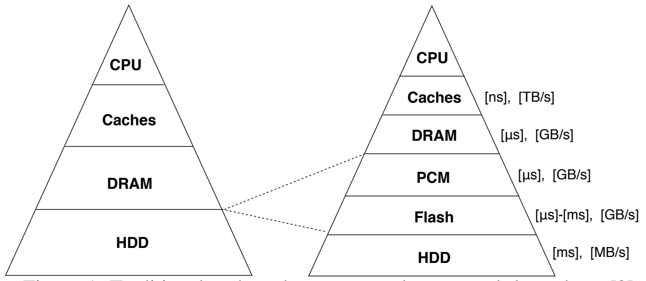

Figure 1. Traditional and modern memory latency and throughput [2]

The data-heavy workloads that are increasingly common nowadays also shed light on the importance of exploiting the computational power of embedded chips in storage, in order to offload part of the computation on the storage devices. In this way, data transfers can be reduced and and the overall response time of systems can be reduced. This opens new opportunities for the redesign of the storage stack, and a lot of research is happening in the field, especially in the area of programmability. Computational offloading of tasks can be implemented to allow users to push their use-case functionality to the storage, thus moving away from general-purpose storage stacks to specialized ones.

In this paper, we will provide an overview of the characteristics and different requirements of modern storage technologies (Section 2), as well as a survey of the state-of-the-art research in the field of storage technologies (Section 3). In the end, we will explore the latest trends and open challenges for storage research (Section 4, Section 5)

2. Non-Volatile Storage Technologies

New NVM technologies offer many improvements over the HDD, in terms of access latency, bandwidth and parallelism. Starting from the now well-established, NAND Flash Solid-State Drives to the new Intel Optane Memory, the latest persistent memory technologies offer undeniable gains in terms of performance, and can be placed in between DRAM and HDD in the memory hierarchy (Figure 1). However, all the new technologies come with different physical characteristics. A lot of research prototypes and experiments are being published to move away from the optimizations that are specific to HDD devices.

While optimizing access to the new media is paramount, the addition of multiple storage layers to the hierarchy also highlights the problem of support for heterogeneity, since the new storage technologies do not have uniform characteristics. Each of them comes with a different architecture, performance and requirements that match with specific access patterns, workloads and software stacks. In this section, we will summarize the characteristics of new technologies and we will identify the key requirements that form the ”unwritten contract” [3] [4] of the new storage technologies.

2.1. First Generation: Flash Memory

The first technology that started to revolutionize the landscape of enterprise and commodity storage systems is the NAND-flash memory, that is used in the now widespread Solid State Disks (SSD). When compared to the previous generation of storage devices like magnetic Hard Drives, the SSD shows performance improvements in access latency and bandwidth, due to the absence of moving parts (i.e. rotating disk, mechanical arms) in favour of circuits . The shift from mechanical parts to circuits allowed the Flash SSD to perform well with random access reads, which are instead detrimental for HDDs due to the seek time and rotation latency that is needed to read in random sectors of the disk. However, due to the internal architecture, random writes still have a heavy performance penalty.

The SSD is composed of one or more chips or dies, each consisting of planes, that are organized in blocks. Blocks are divided into pages, that are typically 4KB in size, as shown in Figure 2. The operations that can be performed on data are read, write/program and erase. While read and write operations happen at page granularity, erase operations are executed at block granularity, that usually have a size that goes from 256 KB to 4 MB.

These internal mechanisms of the NAND Flash have crucial impact on the characterizes performance of SSD drives. In particular, the asymmetry between erase and write block sizes is a crucial aspect impacting the performance of random writes. Since a write operation can only happen on a pre-erased block, even a single-byte write can trigger an erase operation of a 4MB block, and a subsequent write of the content that was previously in the block, with the additional update. This phenomenon is called **write amplification** [5] and is a crucial bottleneck of Flask memory performance.

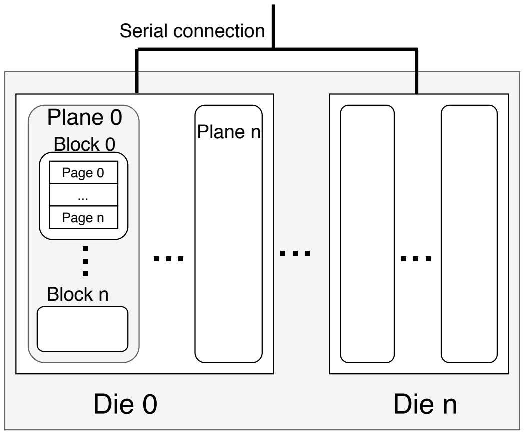

Figure 2. Flash memory architecture, as outlined in [6]

The read and write operations have a degree of internal parallelism, since operations on different planes can happen in parallel [6]. While parallelism can help to increase the throughput, the bottleneck and corner-piece for SSD performance is the erase operation and garbage collection. Not only the erase operation is slower than the other ones (milliseconds instead of microseconds), but also a block can only be programmed for a finite amount of times before wearing out. For this reason, the FTL keeps an index of pre-erased blocks, balance the erase operations on all pages to ensure uniform degradation, and carefully select pages to erase to minimize the amount of valid data that needs to be preventively moved to another block. All these operations are performed by a garbage collector, whose quality is one of the crucial aspects in SSD performance and lifetime [3].

An educated management of writes is necessary to ensure a good performance of the SSD, especially for the case of small random writes. For example, write amplification is an effect that should be mitigated by an efficient FTL implementation. Since writes happen at page level, a small write will cause the whole block to be erased, and then a whole page to be written. In this way, internally to the SSD there is no difference between writing a byte of data and a full page.

Implications of this design and performance characteristics lead to a series of tradeoffs that SSD designers need to take into account in order to improve performance [6]. Key aspects in the design of SSDs are data placement, parallelism, write ordering and workload management.

The internal architecture of the SSD led researchers to evaluate the key characteristics that should be taken into account when using such storage devices. The rules form an unwritten contract for SSD usage, and highlight the need to adapt parts of the storage software stack to fit the internals of the SSD. He et al. [3] identify five rules, **request scale**, **locality**, **aligned sequentiality**, **grouping by death time** and **uniform data lifetime**. The five rules suggest access and write patterns that should improve the performance. Requests to the SSD should be large in scale or number to exploit internal parallelism, and accesses should be done with write locality. Sequential writes that are aligned to the beginning of the block are advised. Optimizations to garbage collection and wear-levelling suggest to group together objects that have similar lifetime and death time, so that blocks can be reclaimed and erased by the garbage collector without costly data movement.

2.2. Second Generation: Phase-Change Memory and Intel Optane

The SSD technology is now widespread and embedded in almost all new devices, but lately a new family of Non-Volatile storage devices is emerging as a viable commercial alternative. The next generation of NVM relies on the technology of Phase-Change Memory (PCM). The PCM devices store information by changing the phase of a material, that alternates between crystalline and amorphous states to store bit information. The main commercial product of this generation of storage is Intel Optane Memory, but it is still debated whether the underlying physical layer (that Intel calls 3D XPoint) is pure PCM [7]. This technology is still used on a small scale due to its higher cost. However, Optane memory performs significantly better that the NAND-Flash SSDs in terms of access latency, so much so that researchers are considering its use not only as a new persistent memory technology, but also as a substitute or addition to DRAM [8].

Optane memory has different characteristics than both HDD and SSD, and this led researchers to define an updated unwritten contract [4] for this new technology, in order to steer software developers towards an efficient, optimal use of the new medium. While the internals of the Intel Optane memory are not known to the public, experiments showed that the 3D XPoint memory that is used in Optane has 1000x lower latency than NAND Flash. Moreover, Optane memory is byte-addressable and does not incur in the performance penalty of garbage collection, that was shown to be one of the main bottlenecks in SSD implementations.

However, Wu et al. [4] showed that the performance of the drives depends on the access patterns and workloads. In particular, unlike the SSD, Optane memory performs better with small requests because the internal parallelism is less than that offered by SSDs. Instead, random access for both read and write operations does not incur in the performance penalty of HDD and SSD, due to the possibility to perform in-place updates that eliminate the write amplification phenomenon in 3D XPoint memory. Overall, the use of Optane memory performs well with unstructured accesses instead of sequential ones.

2.3. NVM Express

With the introduction of NVM technologies, the communication protocol with the main CPU also needs to be upgraded. The high throughput of the NVM devices cannot be fully exploited in presence of a bottleneck in the data transfer phase. All the legacy storage protocols (SCSI, SATA) are tailored to the throughput of a Hard Disk, and are coordinated by a Host Bus Adapter (HBA) either on-chip or external. Both these characteristics do not fit the lower access latency of the SSD and Optane Memory, becoming one of the major components of the access overhead. A faster interconnect option is the Peripheral Component Interconnect express (PCIe). In order to harmonize the different vendor implementations for PCIe, the NVM Express (NVMe) standard was presented.

The development of the NVMe standard was specifically made for Non Volatile Memories, but is not bound to a single technology so that it can accommodate future developments. Its goal is to enable lower access latencies, and parallelism in order to increase performance. This was achieved providing a better hardware interface with multiple I/O queues to improve parallelism, shorter data path and a simplified software stack, as shown in Figure 3. A detailed analysis of the performance gains of NVMe drives can be found in [9], in which Xu et al. highlight that a redesign of the software stack in Linux can deliver 4x improvements in software overhead, using the NVMe standard.

Overall, the result of the performance analysis highlights a solid improvement over the previous interconnects, and one of the key factors is the reduction of software overhead. This highlights the need to re-engineer the preexisting I/O software stack of the operating system, as well as data-intensive applications like DBMSs.

2.4. Performance Studies

With the introduction of new storage technologies, it is vital to understand their performance characteristics and variation, in order to produce optimized software stacks. This is particularly true for storage devices, since they are increasingly important as the shift towards data-intensive workloads continues to happen, and storage performance becomes increasingly important.

Initial work on the impact of NVM technologies performance is [10]. Caulfield et al. perform tests on multiple drive/interconnect configurations and workloads. Overall, they showed that the NVM drives can improve performance as expected. However, performance gains were sometimes as low as 10%. The main performance bottleneck in their experiment are the interconnect technology, since NVMe was not yet specified in 2010, and the software stack, since the authors focused on benchmarking pre-existing file systems and applications.

Studies on NVM performance were then repeated in 2015 [11] by Zhang et al.. In particular, the focus is on the application performance when NVM devices are directly connected to the main memory bus, eliminating the interconnect bottleneck. The main finding of the paper is that the performance of the system is not significantly affected if the NVM memory is used instead of DRAM in certain storage applications, even though DRAM has lower access latencies. However, the authors also identify data persistence and flushing as a key factor that influences performance degradation over time.

Overall, performance studies on NVM triggered research to alleviate the software-induced performance overhead as well as to find the optimal system configurations for the system architecture. In the next sections we will explore the landscape of solutions that were presented so far.

3. The Evolution of Storage Stacks

The advances in technologies for storage triggered research in the field of storage software stacks. The research is very diverse and ranges from abstract, conceptual papers that try to revise the computer organization to fit the new storage requirements, to new file system implementations and applications (e.g. key-value, databases) optimized for NVM storage.

Traditional file systems and storage stacks within the kernel are organized around an architecture that featured HDDs as persistent memory layer, and optimized operations to mitigate the impact of I/O access to the disk. With NVM, a series of new characteristics are introduced. First of all, the penalty latency for persistent memory I/O is decreased by orders of magnitude. Moreover, read operations in sequential and random patterns no longer perform differently, and this extends to write operations as well for Optane memory. On the other hand a new problem needs to be addressed, since the writes to disk need to be optimized to ensure an adequate, uniform wear level of the device. The need to adapt to the new memory technologies produced a lot of research in new file systems. Researchers tried to optimise a variety of aspects, ranging from Flash memory access, to synchronous I/O, to log-structured and user-space file systems. Moreover, a lot of attention was dedicated to application-specific storage in order to optimize some workloads, especially in the area of Machine Learning, Artificial Intelligence and Big Data.

3.1. Optimizations

One of the optimization efforts in the storage stack is at bare-metal level, with solutions that try to optimize internal functions of the storage device or that of user-defined operations. One of the key aspects in SSD performance for example is the quality of the FTL implementation.

An example of low-level SSD software optimizations is proposed in [13]. In this work, the effort is to optimize the performance of the SSD without redesigning the complete software stack. Kim et al. [13] propose the AutoSSD architecture, a redesign of SSD internal software that improves on standard FTLs. In particular, the work focuses on guaranteeing predictable performance to tasks that need it, by implementing a priority mechanism and redesigning internal management capabilities and scheduling of tasks.

In addition to optimizations of Flash-based SSD software, researchers also presented implementations of full-fledged architectures and file systems that are specific to

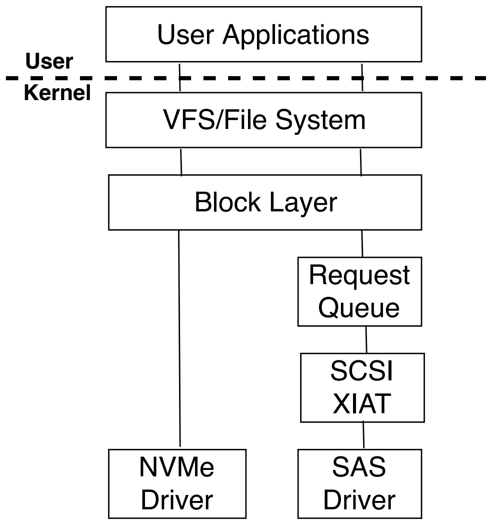

Figure 3. NVMe and traditional storage stack [12]

the underlying memory technology. Many directions can be taken in file system design. Overall, we can identify different trends and design choices that evolved over time. Initially, the main tendency was to provide new OS-level file systems. However, especially after the introduction of Optane Memory, the software overhead provided by the kernel became significant in the overall I/O latency. For this reason, new solutions were proposed to bypass kernel mechanisms in favour of user-level I/O management.

Gordon [14] is a supercomputer that was built with the goal of optimizing data-intensive applications. It provides a redesign of the system architecture by combining processing power with Flash memory. The system is optimized for highly-parallel operations on data, and can deliver up to 1.5x performance improvement at a lower power consumption.

Research has been carried on file system optimizations at the OS level. The Direct File System (DFS) [15] for example provides a simplification of the file system software in the kernel. It simplifies the memory access by exposing the device with DMA capabilities, using a virtualized flash storage layer that implements a large address space, and at the same time embeds FTL functionalities instead of leaving them in the SSD.

Another common approach to the redesign of file systems is that of removing layers of software to minimize the software overhead, that becomes a non-negligible part of the I/O latency with NVM devices. An example for this is Moneta [16]. Caulfield et al. implement a high-performance storage array of 4 indepentent PCM memory units with controllers. The array is controlled and coordinated by an optimized scheduler that allows to gain up to 10x in performance when compared to Flash memory, and are able to gain performance by optimizing the scheduling of I/O operations and exploiting Direct Memory Access (DMA) for writes. Moneta also has a follow-up research, Moneta-D [17], that further optimizes the performance of I/O operations. The authors highlight that even with the optimizations that Moneta provides, kernel operations account for 30% if the latency in small requests (4KB). The optimize this bypassing kernel file permission checks and offloading the operation to the Moneta-D hardware. Yang et al. [18] present a prototype of a system that aims at further reducing the software overhead, compared to Moneta. They argue that the widespread interrupt-driven I/O is not optimal if fast persistent memory is used, because context switches and interrupt handling are time consuming and constitute a heavy performance penalty. Instead, they propose to substitute interrupts with polling, implementing a busy wait on the device. They verify that a synchronous I/O model can outperform the legacy asynchronous model, if SSDs are used instead of HDDs.

The idea of removing the kernel from the data path altogether is explored by Peter et al. [19]. Their prototype allows user applications to bypass the kernel in read and write operations, while the operating system manages protection, mapping and translation mechanisms.

Flashshare [20] is another work that aims at minimizing latency when accessing an Ultra-Low-Latency SSD. In this work, Zhang et al. highlight that the sole use of high-performance hardware is not a guarantee for latency and Service Level Agreements requirements. For this reason, the authors propose a redesign of many components across the whole software stack, from kernel to firmware level, in order to include abstractions for high-priority workloads. Lee et al. [21] identify another performance problem in the I/O process. While the requests to the HDD are generally performed asynchronously, a lot of preparation (page allocation, DMA mapping) are performed in a synchronous fashion. If the device is fast, this queue of operations has a significant impact on the overall access time. For this reason, the authors present an asynchronous implementation of the software stack.

In addition to the improved I/O latency, new NVM storage technologies also offer improved bandwidth. In traditional file systems, the assumption of the underlying HDD technology does not allow to fully exploit parallelism. For this reason Kang et al. [22] implement SpanFS, a file system based on Ext4 that is specifically targeted to multi-core systems, in order to better exploit the Flash memory parallelism.

One of the new capabilities of NVM memory is the possibility to perform Direct Memory Access. With DAX, read and write accesses to memory can bypass traditional file system mechanisms like page caches in the Virtual File System, as shown in Figure 4. The Memory Management Unit (MMU) can instead provide a direct memory map, allowing direct byte access to persistent memory [23]. The possibility to perform DAX in NVM storage devices however also highlights the need to re-implement fault-tolerance mechanisms that can replace file system protection. An example for this is Pangolin [24], that implements detection and recovery from data corruption with a minimal memory and latency overhead.

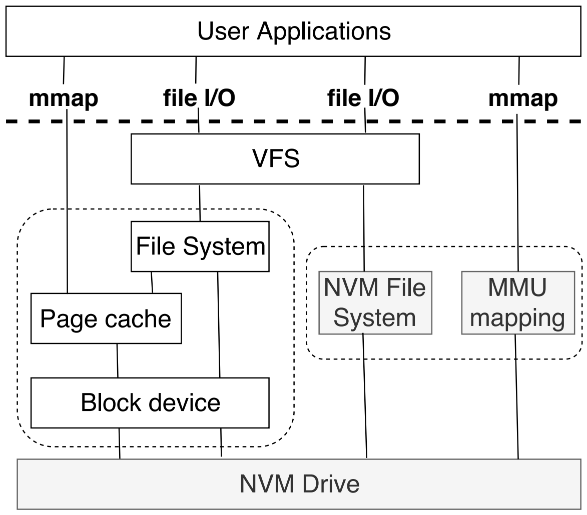

Figure 4. Traditional and DAX access mechanisms for NVM devices [23]

3.2. Beyond the block-device interface

One of the main disadvantages of the old interface to storage media is the block-device interface. This interface is based on the HDD internals. Bjorling et al. presented research on the reasons and ways to move away from the block-device interface, in order to improve SSD performance [25]. They argue that, although the block-device interface is compatible with SSDs and Phase Change Memory, the change in internals of the storage should be acknowledged to provide improved performance. In particular, the OS should be aware of the increased capability for parallelism. They support the claim with software prototypes. A possible solution is presented in [26], with a redesign of the Linux block layer that allows for increased parallelism. The implementation is based on a multi-queue access mechanism that can exploit the fast NVM-Express interface and SSD parallelism. Follow-up work is presented with the implementation of LightNVM [27], that moves FTL functionalities to the host and implements a Linux subsystem to manage Open-Channel SSDs. Open-Channel SSDs expose their internals to the host instead of exposing a block I/O interface. Dullor et al. [28] argue that the ability to address bytes in NVM drives should suggest a complete overcoming of the block-device interface. For this reason they propose PMFS, a POSIX-compliant file system that is redesigned to provide fast, direct access to byte-addressable persistent memory eliminating the block-device interface.

3.3. Log-based file systems

While the solutions that were presented so far do not affect file system data structures and abstractions, and rather focus on redesigning some mechanisms, there are many file systems that target NVM memory and adopt a log structure. The idea of a log-structured file system is not new and there is work that can be traced back to 1991 [29]. Rosenblum et al. designed a prototype of a log-structured file system, in which all changes to disk are logged and then written sequentially to memory in an append-only manner.

With the advent of NVM devices, log-structured file systems are increasingly gaining popularity. One of the reasons for this is the convenience of the log structure, that allows to transform random small writes in a sequential bigger write operation to the medium. This is convenient especially in NAND Flash SSDs, since small random writes do not have a good performance profile and should be avoided, in favour of big sequential writes.

A Flash-optimized implementation of a Linux file system is presented in [30], adopting a log-structured approach. The F2FS file system completely redesigns the software abstractions, and uses data structures that fit the underlying hardware instead of using legacy HDD-optimized data structures. For example, 90% of small (4KB) random writes are converted in sequential writes when comparing Ext4 to F2FS, and this leads to a 3x performance improvement and increased device lifetime. In general, F2FS was tested on both mobile and server scenarios, and performed either comparable to other file systems or better.

Research on log-structured file systems also focused on improving parallelism, when compared to traditional file systems. ParaFS [31] is a log-structured file system developed by Zhang et al., and it improves the degree of parallelism by exposing the internal structure of the Flash memory through a redesigned FTL layer. By doing this, the authors report achieving an improvement of up to 3.1x over F2FS.

While log-structured file systems are popular for Flash and Optane memory because of their performance characteristics, there are pitfalls with the use of logs. Yang et al. [32] study the performance of F2FS and highlight that, since the FTL usually has an inner log-like mapping, aggregating small writes in a single block write, creating a log-based file system would create a log-on-log scenario. This comes with performance issues, since log stacks exhibit the behavior of increasing write pressure on the drive, this accelerating the wear-out of the SSD. This happens because, while a single log has the goal of obtaining a sequential write out of many random writes, this effect can be scattered and randomized when multiple logs are independently operating and committing writes.

3.4. Development Tools

Many of the user-defined implementations of the storage stack aim at bypassing kernel mechanisms, in favour of specialized user-defined functionalities. In order to achieve this, the user has to implement low-level, potentially complex functionalities that are usually provided by the operating system.

A lot of development tools and libraries are available to help the developers to interface directly with NVM SSDs. One of them is the Intel SPDK [33] for storage devices. This development kit comprises a series of user-space drivers and a reference architecture for Intel-supported storage stacks. For memory-like NVM, the pmem [34] tool is available for Linux and Windows distributions. pmem facilitates the implementation of DAX access to byte-addressable persistent memory, using the underlying OS support for the functionality.

A third tool that is available for use is NVMeDirect [35], and aims at overcoming the limitations of the Intel SPDK. A crucial limitation of SPDK is that it works moves all I/O drivers in user space, and can only be used by a single user and application. NVMeDirect is also a user-level framework that leverages the NVMe interface, but it can coexist with the kernel stack by providing access to a different partition of the disk. With NVMeDirect, different applications can coexist and specify individual I/O scheduling policies.

3.5. File Systems in User Space

While file systems have traditionally been designed for kernel-space execution, mediating the access to persistent memory through the kernel introduces a performance penalty due to context switches. As the complexity of the file system grows and the I/O latency is reduced, the design of lighweight, specialized file systems becomes an attractive idea. For this reason, File Systems in User Space are being explored by researchers. The debate on the actual production feasibility of user-level file systems is still debated due to potential performance issues [36], and even criticised by Linus Torvalds himself within the Linux environment [37]. However, research produced interesting results in the field of NVM-focused file systems in the user space. This was facilitated by the simulation tools like SPDK, as well as the introduction of a library and in-kernel module to develop File Systems in User-Space in Linux (FUSE) using an API to the kernel Virtual File System (VFS) [38].

Experiments with user-level file systems for fast NVM memory presented interesting results in terms of latency reduction and increased support for user-defined operations. An example for such a file system is Aerie [39]. Volos et al. implement a decentralized file systemm that allows user applications to have direct access to fast storage, bypassing the kernel mediation. The file system is divided into an untrusted user-mode library that enables access to storage, and a trusted file system service that coordinates updates, while the kernel has only the role of resource multiplexing. An alternative implementation is presented by Yoshimura et al.. EvFS [40]. Once again, the goal of the file system is to bypass the kernel VFS, and substitute it with a redesigned storage stack (implemented through SPDK). Using the NVM drivers in SPDK allows to reduce the number of context switches. I/O requests are handled in an event-driven fashion, by threads that are active in the EvFS file system.

An alternative to the monolithic system approach is presented by Kannan et al. [41] with DevFS, a user-level file system that entirely resides on a fast, NVMe-attached storage device. This file system provides direct storage access without involving the kernel in the path, since it is offloaded to the storage and applications can communicate with it directly, except for OS coordination to share process-level credentials and permissions. In addition to this, direct access to storage allows to have awareness of the underlying hardware.

Overall, user-level file systems allow for a good degree of flexibility in the implementation of functionalities, and this property is desirable for prototyping support for Non-Volatile Memory.

3.6. Application-Specific Storage

File systems tend to provide a standard, uniform API to access the storage at the kernel level, and customization of functionality is usually limited due to security reasons. However, data-intensive applications can have use-case specific requirements that should be implemented for I/O operations. For this reason, we witnessed a tendency to shift file system functionality from kernel to user space in order to provide custom functionalities.

The concept of active disks [42], where significant computational power is present in the storage, was originally proposed in 1998 by Acharya et al., but went through a new wave of interest in the with the introduction of Flash memory and later Optane memory. A recurring trend in the area of application-specific storage is that of Near Data Processing (NDP) [43], with computation that is pushed close to the persistent memory to improve the overall performance. NDP is not a novel concept, and can be traced back to the 1990s. However, technological limitations made it not feasible to effectively offload a significant portion of processing to the storage. With technological improvements in storage performance, data movement to the CPU is becoming a bottleneck, especially in Big Data scenarios. For this reason, the notion of data movement wall started to recur.

NDP challenges the idea of the standard Von Neumann architecture for computers, in which main memory and processing capabilities are separate and connected by a bus, but all computation happens in the CPU after transferring the data. There are two different approaches to NDP: Processing In Memory (PIM) and In Storage Computation (ISC), that locate computing resources with main memory and persistent memory respectively.

Implementations of ISC on active disks usually try to offload part of the computation on data, and for this reason are particularly interesting in Big Data scenarios (e.g. [44] [45]). For example, Cho et al. [46] make the case for an implementation of active disk with SSD, in order to speed up operations on data. In particular, the authors highlight that programming models like the widespread MapReduce [] first apply a filter on massive amounts of data (TB), and then process the result. Offloading the filter operation to the active disk can provide speedup of up to 4x, as well as up to 27x more energy efficiency.

Database system optimizations can also be implemented with an ISC approach. Jo et al. [47] for example present YourSQL, an implementation of an SQL-based database, that offloads filtering of data to the SSD using FPGAs and obtaining a performance improvement of up to 3.6x.

Wang et al. explore the potential of in-storage computing on SSDs to improve search engine systems [48]. They perform a study on the most common operations on data (intersection, ranking, difference etc.) and modify the open source Apache Lucene to offload some of them to the SSD.

Biscuit [49] is another system that allows users to push user-defined functionality to the storage. The SSDs in the system are equipped with ICS hardware that can be programmed by the user in C++.

Another typical use case is that of machine learning and deep learning. The Cognitive SSD presented by Liang et al. [50] is an example. This system is tailored towards deep learning, and offloads computation to the SSD for data queries and computation of deep learning workloads. The system optimizes data retrieval of unstructured data like videos, since this is a very common use case for modern big data workloads. The acceleration is delivered by an additional hardware module that works alongside the SSD and can access data directly from NAND flash.

3.7. Operating System Support

While for solid 60 years the computer organization could assume that the persistent memory performance was orders of magnitude worse than that of main memory and caches, this is increasingly no longer the case for systems that use NVM storage. Since this assumption shaped the storage stack of operating systems, it is fair to investigate whether the existing computer organization is still optimal.

New research questions on the effects of NVM in operating system architectures are raised by Bailey et al. [51] in a paper that challenges the very basic concepts underlying operating system design. The authors argue that since the assumption that the latency to access persistent storage is orders of magnitude larger than that of DRAM doesn't hold anymore, the whole memory hierarchy organization can be revised. 3 different options are presented to integrate NVM in the system architecture. The first one is simply to replace HDD with NVM. This is not revolutionary but should lead to the adaptation of the access patterns, since already highlighted the difference in the unwritten contract and the implications on the software stack. An alternative is to use NVM and DRAM in parallel, depending on whether persistence is needed. The third approach is to completely remove DRAM in favour of NVRAM. Any change in the architecture should also correspond to adaptations in software and data structures, and the authors investigate on the possible adaptations of Virtual Address spaces, paging and page granularity. Overall, a few experiments and research operating systems are exploring the support for heterogeneous NVM memory technologies.

As we highlighted in the previous section, it is now clear that the kernel software overhead in I/O operations is too high, when fast NVM storage is involved. While a lot of research focused on the optimization of the file system or the implementation of user-level mechanisms to bypass the kernel, Peter et al. also explored the possibility to rewrite a whole kernel [52]. Arrakis is an operating system that allows applications to directly manage the I/O operations to their reserved Virtual Storage Area, while the kernel is in the control plane and only implements basic operations of naming and allocation, in order to connect the application with its storage.

A solution for memory management in datacenter was proposed in [53]. The HeteroOS operating system that explicitly targets the management of heterogeneous memory in datacenters. This work is not tailored at a single storage technology, and rather aims at providing a platform where multiple storage technologies can be efficiently exploited, by providing the guest OS with awareness of the underlying main memory technologies (DRAM and NVM).

An alternative approach to this is to drop the idea of a monolithic OS and to instead embrace hardware diversity with the creation of ”modular” operating systems like Barrelfish or LegoOS [54] [55]. LegoOS is an operating system that can manage disaggregated resources. In contrast to monolithic kernels or servers, where each compute node has a complete set of hardware resources (processor, main memory, disk, accelerators etc.), a disaggregated kernel deploys these resources in an independent way. This improves the modularity of the system and makes it easier to implement efficient solution for every storage technology independently of the others.

Disaggregated OS solutions are one of the directions that OS research can follow in the direction of NVM-specific optimizations and In Storage Computing. The operating system should evolve to accommodate heterogeneous devices and to offload computation to storage resources systematically. The approach of ISC has seen an increase in interest, in the light of the emerging data-intensive computations like machine learning and data mining. This represents a change in how the OS should perform computation, having more than one CPU available, with many NUMA nodes that should perform computation on local data. A more in-depth discussion on the topic can be found in [56].

3.8. Distributed Storage

writeintro

In the era of Big Data, traditional file systems do not always represent the optimal data storage and access solution. One of the most widespread solutions that are being increasingly used in datacenters and Big Data applications in general is the Key-Value storage. todo [57] [58] todo [59]

4. Latest Trends

Despite the fervent work on NVM storage and software stacks, a lot of directions still need to be investigated in order to improve current results. In particular, we identified three trends and keywords that recurrently appear in the latest research on the topic. First of all, the diversification in the hardware landscape is not only affecting storage devices, but computer systems as a whole. A core focus of current research is the optimization of systems for a variety of hardware configurations rather than a standard one. A second keyword in the storage research is that of programmability. Over time, data manipulation is shifting from a standard interface to a multitude of specialized processing operations. For this reason, researchers are investigating the programmability of storage in order to allow users to implement their own business logic efficiently. Last but not least, security of data is of utmost importance, especially in the scenario of In Storage Computation.

4.1. Heterogeneity

With more and more storage technologies being available on the market, there is no one-size-fits-them-all solution for memory and storage management. The heterogeneity of the hardware is a challenge for efficient resource access and management, since every technology has unique characteristics that imply different optimal usage patterns. While the majority of file system research so far focused on the implementation and optimization of a uniform storage technology layer, be it Flash or Optane memory, some research is also being carried out to support a heterogeneous memory layout, where different technologies coexist.

The need for heterogeneous memory systems is supported by the study by Dulloor et al. [60], where the authors present an extensive study of representative applications (key-value storage, in-memory database, graph analytics) on a heterogeneous memory system. The results highlight that the optimal usage of memory depends on the application that is using storage and memory, due to varying access patterns and read/write sizes.

The performance gains of NVM technologies not only justify the redesign of software for existing configurations, and rather also challenge to create new architectures. Xu et al. for example propose Nova, a file system that optimizes hybrid DRAM-NVM storage architecture [61]. The reduced access latency of NVM makes previous file systems inefficient because they were tailored to HDD of SSD access times. NOVA adopts a log-structured file system that can run on hybrid DRAM-NVM configurations, delivering strong consistency guarantees that old-fashioned file systems delegated to the Hard Drive. An extension of this file system is provided in [62] by Xu et al., with Nova-fortis adding consistent snapshots as an additional consistency and reliability mechanism.

The NOVA file system is further developed into Ziggurat [63] by Zheng et al.. The authors explore the challenges of heterogeneous memory management and propose a solution that combines DRAM, Optane and SSD technologies, and schedules writes to disk based on optimization parameters like write size and access pattern. The authors show that even a small-sized Optane disk can outperform SSD-based Ext4 file systems up to 38.9x.

Researchers have tried different approaches when designing hybrid memory systems. One of the results is Strata [64], a multi-layer file system. It consists of two main components, user-level and kernel-level. The user-lever module performs log writes on NVM, and the kernel in the background manages propagation to the other layers of the storage hierarchy (i.e. and SSD and a HDD) in an efficient way. The log data is digested into tree data structures in order to be propagated. In this way, small writes are efficient on the user side, but read is optimized in the tree data structure in the other layers.

File systems are not the only layer of the storage stack that can implement solutions for heterogeneous storage. An example for this is Pocket [65], that implements heuristics to place serverless workloads in diverse storage technologies (i.e. HDD, DRAM and Flash memory). The choice is made according to the job characteristics and latency requirements. Apache Crail [66] is another architecture that can harvest diverse remote storage resources (i.e. GPU, DRAM and Flash memory) through high performance networks, in order to perform data processing in a distributed system.

4.2. Programmability

The increased need for specialized functionality and higher performance, in particular in the field of Big Data processing, led to the need to deliver solutions to push user-defined behavior to the file system or close to the storage. However, the solutions that we presented so far for Near Data Processing were usually tailored to a single use case, and more importantly a single tenant. As it happens with compute resources however, storage is usually also shared in datacenters. In order to implement NDP in this scenario, different users should be able to push their functionality to the storage.

With the introduction of high-performance persistent storage in systems as well as the increased size of data, the standard 2-layer memory architecture for computation is facing a challenge. Moving massive amounts of data to the CPU for computation is not always feasible, nor is it the optimal solution. Since many memory units have embedded CPUs and local memory, storage systems can now start to offload computation on data directly to the storage instead of transferring data closer to the CPU. This allows to exploit the full internal bandwidth of storage, that is often higher than that of the interconnection to main memory. For this reason, there is a lot of interest towards In Storage Computation (ISC). The user should be able to program the storage functionality in the same way as CPU operations on data can be programmed.

The adoption of programmable storage in datacenters is also a crucial step in the evolution of cloud computing [67]. The ability for users to program storage functionality can help to bridge the gap between the flexibility and speed of updates in the rest of the infrastructure (Application and OS updates), and the relatively long-lived storage media.

4.2.1. FPGA. One of the options to implement programmable logic close to the storage is using hardware support, in particular FPGAs.

Willow [68] is an early attempt at providing programmable functionalities in the SSD. It exposes an interface with a set of basic, general-purpose operations with which the users compose specialized functionalities, and program the SSD for in-storage computation and data transfers. The interface is programmed on FPGAs that manage the SSDs in the Willow architecture.

INSIDER [69] is a full storage stack implementation for a system consisting of a compute node that is attached to storage nodes with PCIe interfaces. In this system, FPGAs are used as reconfigurable fabric in the storage nodes. The FPGAs execute user-defined code in space-multiplexing, since different slots of the FPGA are programmed and assigned to different tasks to be executed in parallel. As we saw in Section ..., programming the whole storage stack is not the only option. In-storage acceleration of user-defined behaviors can also target single applications. Caribou [70] implements a distributed key-value storage that can offload tasks to storage nodes using FPGA for programmability. The operations that are implemented in Caribou are explicitly related to the key-value store, and the FPGA logic allows to offload part of the computation to the storage nodes.

FPGA-based acceleration can also be provided for specialized tasks, like REGISTOR [71]. In this system, the FPGAs are used to accelerate user-defined regex matching directly in the NVMe SSD.

While INSIDER is not explicitly operating in a distributed scenario, Caribou can manage disaggregated storage because the communication between nodes happens through the network.

4.2.2. Language-based Extensions. It is not always possible to introduce specialized hardware like FPGA accelerators inside clusters. For this reason, an appealing approach to programmable storage is the use of the embedded controller to execute user-defined code [42]. As always, a key requirement is to ensure isolation between different users, to avoid data leaks and corruption of both the data and the system. One option for this is to use Rust [72]. Rust is a type-safe and memory-safe programming language, and its safety guarantees allow to push user defined behavior to the storage, sharing in-storage resources among tenants. Splinter [73] uses Rust to allow users to program remote storage with bare-metal extensions. Usually, protection and isolation between tenants is done with virtualization, executing user code in sandboxed environments that are administered by the infrastructure provider. This is a powerful solution, but comes at a cost because it increases the complexity and depth of the software stack. Using bare-metal extensions like Splinter has a great impact on performance, removing the need for virtualization, while still keeping tenants isolated. Splinter is not a full software stack, rather it is a distributed key-value storage with the option of user programmability.

While Splinter is a key-value storage, researchers have also investigated the possibility to push programmability down to the file system. This is challenging because file system features are usually privileged and operate in the kernel level. User-defined code extensions in the file system can then be implemented in two ways. The first one is to keep user logic in the user's privilege level. This has the downside that every I/O operation must trigger kernel functionalities under the hood, thus resulting in context switches. Potentially, this can cause a significant overhead. In order to avoid this, a mechanism to safely execute user-defined functionalities in the kernel level is needed. A possibility that researchers are exploring nowadays is to use eBPF (enhanced Berkley Packet Filter) to push user functionality in the kernel and execute it in a safe sandboxed environment. eBFP is mainly used to implement monitoring tools and packet filters, but Bijlani et al. are exploring the possibility to use it as a language to write user extensions for file systems in user space [74]. eBPF can ensure safety of the execution because it comes with a formal verifier of the code, and among other things it rejects backward loops like for cycles, and protects accesses to memory. eBPF programs are run in a kernel-level sandboxed environment, thus ensuring high performance and safety.

5. Open Challenges

Storage technologies and stacks have been going through a radical transformation, from hardware to user applications. The evolution of technologies shaped the redesign of traditional components of the operating system like the file system, as well as common user applications like key-value storage and distributed databases. However, so far the revolution only covered a part of the ever-changing landscape in computer systems.

All the work that was covered in this survey focused on traditional computer systems and applications, and the emerging trends that we could identify in this areas. However, researchers will need to address a lot of challenges for the most recent evolutions of computer systems, in particular in the field of AI, IoT and edge computing. A broad overview of the most important open research problems in the field of storage can be found in [75].

Amvrodiasis et al. highlight emerging issues, like the increasing need for support of heterogeneous devices that will stem from IoT, as well as the need to implement and deploy a continuous layer of persistent storage from the cloud to the edge, close to the end users. In Artificial Intelligence, storage is also likely to gain importance. The need to store massive datasets and perform computations on them suggests the need to provide specialized, active storage for AI, that can for example proactively cache intermediate results and have awareness of data pipelines.

6. Conclusion

In this survey we covered the main advancements in modern storage stacks, and highlighted the latest trends and open challenges for the future. We followed a bottom-up approach, starting from the main technical details of SSD and Optane memory, and progressively zoomed out to cover file system implementations for NVM memory, OS adaptations and evolutions, up to distributed systems and application like key-value storage. We identified the latest trends and keywords of storage research being programmability, heterogeneity and security, and highlighted the far-reaching research goals in storage for IoT, edge computing and AI.

7. Methodology

This survey aims at covering the advancements of the storage stack software, under the light of the introduction of new NVM memory technology. In order to reduce the scope of the survey and offer a consistent and non-dispersive view on the topic, the survey mainly focused on single-system changes. Distributed system changes are introduced when relevant, but they are not covered systematically.

The paper selection was made recursively, starting from the last 5 years of storage-related publications in tier 1 conferences like Usenix ATC, Usenix FAST, Systor, OSDI, SOSP. Older references were included when relevant for concept definitions, and in case of frequently cited and background work.

References

[1] "Intel® Optane^{TM} Technology," https://www.intel.com/content/www/us/en/architecture-and-technology/intel-optane-technology.html, accessed: 2019-06-25.

[2] "Intel," https://software.intel.com/en-us/articles/memory-performance-in-a-nutshell, accessed: 2019-06-25.

[3] J. He, S. Kannan, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "The unwritten contract of solid state drives," in *Proceedings of the twelfth European conference on computer systems*. ACM, 2017, pp. 127–144.

[4] K. Wu, A. Arpaci-Dusseau, and R. Arpaci-Dusseau, "Towards an unwritten contract of intel optane ssd," in *11th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 19). USENIX Association, Renton, WA*, 2019.

[5] X.-Y. Hu, E. Eleftheriou, R. Haas, I. Iliadis, and R. Pletka, "Write amplification analysis in flash-based solid state drives," in *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*. ACM, 2009, p. 10.

[6] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. S. Manasse, and R. Panigrahy, "Design tradeoffs for ssd performance." in *USENIX Annual Technical Conference*, vol. 57, 2008.

[7] "Intel 3D," https://www.theregister.co.uk/2016/01/29/xpoint_examination/, accessed: 2019-06-25.

[8] "optaneDimm," https://www.techspot.com/news/79483-intel-announces-optane-dc-persistent-memory-dimms.html, accessed: 2019-06-25.

[9] Q. Xu, H. Siyamwala, M. Ghosh, T. Suri, M. Awasthi, Z. Guz, A. Shayesteh, and V. Balakrishnan, "Performance analysis of nvme ssds and their implication on real world databases," in *Proceedings of the 8th ACM International Systems and Storage Conference*. ACM, 2015, p. 6.

[10] 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 *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society, 2010, pp. 1–11.

[11] Y. Zhang and S. Swanson, "A study of application performance with non-volatile main memory," in *2015 31st Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 2015, pp. 1–10.

[12] "NVMe," https://nvmexpress.org/about/why-nvm-express/, accessed: 2019-06-25.

[13] B. S. Kim, H. S. Yang, and S. L. Min, "Autossd: an autonomic {SSD} architecture," in *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, 2018, pp. 677–690.

[14] A. M. Caulfield, L. M. Grupp, and S. Swanson, "Gordon: using flash memory to build fast, power-efficient clusters for data-intensive applications," *ACM Sigplan Notices*, vol. 44, no. 3, pp. 217–228, 2009.

[15] W. K. Josephson, L. A. Bongo, K. Li, and D. Flynn, "Dfs: A file system for virtualized flash storage," *ACM Transactions on Storage (TOS)*, vol. 6, no. 3, p. 14, 2010.

[16] A. M. Caulfield, A. De, J. Coburn, T. I. Mollow, R. K. Gupta, and S. Swanson, "Moneta: A high-performance storage array architecture for next-generation, non-volatile memories," in *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2010, pp. 385–395.

[17] A. M. Caulfield, T. I. Mollov, L. A. Eisner, A. De, J. Coburn, and S. Swanson, "Providing safe, user space access to fast, solid state disks," *ACM SIGARCH Computer Architecture News*, vol. 40, no. 1, pp. 387–400, 2012.

[18] J. Yang, D. B. Minturn, and F. Hady, "When poll is better than interrupt." in *FAST*, vol. 12, 2012, pp. 3–3.

[19] S. Peter, J. Li, I. Zhang, D. R. Ports, T. Anderson, A. Krishnamurthy, M. Zbikowski, and D. Woos, "Towards high-performance application-level storage management," in *6th {USENIX} Workshop on Hot Topics in Storage and File Systems (HotStorage 14)*, 2014.

[20] J. Zhang, M. Kwon, D. Gouk, S. Koh, C. Lee, M. Alian, M. Chun, M. T. Kandemir, N. S. Kim, J. Kim *et al.*, "Flashshare: punching through server storage stack from kernel to firmware for ultra-low latency ssds," in *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, 2018, pp. 477–492.

[21] G. Lee, S. Shin, W. Song, T. J. Ham, J. W. Lee, and J. Jeong, "Asynchronous i/o stack: A low-latency kernel i/o stack for ultra-low latency ssds," in *2019 USENIX Annual Technical Conference ({USENIX}{ATC} 19)*, 2019.

[22] J. Kang, B. Zhang, T. Wo, W. Yu, L. Du, S. Ma, and J. Huai, "Spanfs: a scalable file system on fast storage devices," in *2015 {USENIX} Annual Technical Conference ({USENIX}{ATC} 15)*, 2015, pp. 249–261.

[23] "New Types of Memory, their support in Linux and how to use them with RDMA," https://www.openfabrics.org/images/2018workshop/presentations/112_CLameter_NewTypesofMemory.pdf, accessed: 2019-06-25.

[24] L. Zhang and S. Swanson, "Pangolin: A fault-tolerant persistent memory programming library," *arXiv preprint arXiv:1904.10083*, 2019.

[25] M. Bjørling, P. Bonnet, L. Bouganim, and N. Dayan, "The necessary death of the block device interface." in *CIDR*, 2013.

[26] M. Bjørling, J. Axboe, D. Nellans, and P. Bonnet, "Linux block io: introducing multi-queue ssd access on multi-core systems," in *Proceedings of the 6th international systems and storage conference*. ACM, 2013, p. 22.

[27] M. Bjørling, J. González, and P. Bonnet, "Lightnvm: The linux open-channel {SSD} subsystem," in *15th {USENIX} Conference on File and Storage Technologies ({FAST} 17)*, 2017, pp. 359–374.

[28] S. R. Dulloor, S. Kumar, A. Keshavamurthy, P. Lantz, D. Reddy, R. Sankaran, and J. Jackson, "System software for persistent memory," in *Proceedings of the Ninth European Conference on Computer Systems*. ACM, 2014, p. 15.

[29] M. Rosenblum and J. K. Ousterhout, "The design and implementation of a log-structured file system," in *ACM SIGOPS Operating Systems Review*, vol. 25, no. 5. ACM, 1991, pp. 1–15.

[30] C. Lee, D. Sim, J. Hwang, and S. Cho, "F2fs: A new file system for flash storage," in *13th USENIX Conference on File and Storage Technologies ({FAST} 15)*, 2015, pp. 273–286.

[31] J. Zhang, J. Shu, and Y. Lu, "Parafs: A log-structured file system to exploit the internal parallelism of flash devices," in *2016 {USENIX} Annual Technical Conference ({USENIX}{ATC} 16)*, 2016, pp. 87–100.

[32] J. Yang, N. Plasson, G. Gillis, N. Talagala, and S. Sundararaman, "Don't stack your log on my log," in *2nd Workshop on Interactions of NVM/Flash with Operating Systems and Workloads ({INFLOW} 14)*, 2014.

[33] "Intel SPDK," https://software.intel.com/en-us/articles/introduction-to-the-storage-performance-development-kit-spdk, accessed: 2019-06-25.

[34] "pmem," https://pmem.io/, accessed: 2019-06-25.

[35] H.-J. Kim, Y.-S. Lee, and J.-S. Kim, "Nvmedirect: A user-space i/o framework for application-specific optimization on nvme ssds," in *8th {USENIX} Workshop on Hot Topics in Storage and File Systems (HotStorage 16)*, 2016.

[36] B. K. R. Vangoor, V. Tarasov, and E. Zadok, "To {FUSE} or not to {FUSE}: Performance of user-space file systems," in *15th {USENIX} Conference on File and Storage Technologies ({FAST} 17)*, 2017, pp. 59–72.

[37] "Linus Torvads on File Systems in Userspace," https://www.phoronix.com/scan.php?page=news_item&px=OTYwMA, accessed: 2019-06-25.

[38] "FUSE file systems in userspace," https://github.com/libfuse/libfuse, accessed: 2019-06-28.

[39] H. Volos, S. Nalli, S. Panneerselvam, V. Varadarajan, P. Saxena, and M. M. Swift, "Aerie: Flexible file-system interfaces to storage-class memory," in *Proceedings of the Ninth European Conference on Computer Systems*. ACM, 2014, p. 14.

[40] T. Yoshimura, T. Chiba, and H. Horii, "Evfs: user-level, event-driven file system for non-volatile memory," in *11th {USENIX} Workshop on Hot Topics in Storage and File Systems (HotStorage 19)*, 2019.

[41] S. Kannan, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, Y. Wang, J. Xu, and G. Palani, "Designing a true direct-access file system with devfs," in *16th {USENIX} Conference on File and Storage Technologies ({FAST} 18)*, 2018, pp. 241–256.

[42] A. Acharya, M. Uysal, and J. Saltz, "Active disks: Programming model, algorithms and evaluation," *ACM SIGPLAN Notices*, vol. 33, no. 11, pp. 81–91, 1998.

[43] R. Balasubramonian, J. Chang, T. Manning, J. H. Moreno, R. Murphy, R. Nair, and S. Swanson, "Near-data processing: Insights from a micro-46 workshop," *IEEE Micro*, vol. 34, no. 4, pp. 36–42, 2014.

[44] S. Boboila, Y. Kim, S. S. Vazhkudai, P. Desnoyers, and G. M. Shipman, "Active flash: Out-of-core data analytics on flash storage," in *012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 2012, pp. 1–12.

[45] D.-H. Bae, J.-H. Kim, S.-W. Kim, H. Oh, and C. Park, "Intelligent ssd: a turbo for big data mining," in *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*. ACM, 2013, pp. 1573–1576.

[46] S. Cho, C. Park, H. Oh, S. Kim, Y. Yi, and G. R. Ganger, "Active disk meets flash: A case for intelligent ssds," in *Proceedings of the 27th international ACM conference on International conference on supercomputing*. ACM, 2013, pp. 91–102.

[47] I. Jo, D.-H. Bae, A. S. Yoon, J.-U. Kang, S. Cho, D. D. Lee, and J. Jeong, "Yoursql: a high-performance database system leveraging in-storage computing," *Proceedings of the VLDB Endowment*, vol. 9, no. 12, pp. 924–935, 2016.

[48] J. Wang, D. Park, Y. Papakonstantinou, and S. Swanson, "Ssd in-storage computing for search engines," *IEEE Transactions on Computers*, 2016.

[49] B. Gu, A. S. Yoon, D.-H. Bae, I. Jo, J. Lee, J. Yoon, J.-U. Kang, M. Kwon, C. Yoon, S. Cho *et al.*, "Biscuit: A framework for near-data processing of big data workloads," in *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3. IEEE Press, 2016, pp. 153–165.

[50] S. Liang, Y. Wang, Y. Lu, Z. Yang, H. Li, and X. Li, "Cognitive {SSD}: A deep learning engine for in-storage data retrieval," in *2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19)*, 2019.

[51] K. Bailey, L. Ceze, S. D. Gribble, and H. M. Levy, "Operating system implications of fast, cheap, non-volatile memory." in *HotOS*, vol. 13, 2011, pp. 2–2.

[52] S. Peter, J. Li, I. Zhang, D. R. K. Ports, D. Woos, A. Krishnamurthy, T. Anderson, and T. Roscoe, "Arrakis: The operating system is the control plane," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. Broomfield, CO: USENIX Association, 2014, pp. 1–16. [Online]. Available: https://www.usenix.org/conference/osdi14/technical-sessions/presentation/peter

[53] S. Kannan, A. Gavrilovska, V. Gupta, and K. Schwan, "Heteroos—os design for heterogeneous memory management in datacenter," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2017, pp. 521–534.

[54] A. Schüpbach, S. Peter, A. Baumann, T. Roscoe, P. Barham, T. Harris, and R. Isaacs, "Embracing diversity in the barrelfish manycore operating system," in *Proceedings of the Workshop on Managed Many-Core Systems*, vol. 27, 2008.

[55] Y. Shan, Y. Huang, Y. Chen, and Y. Zhang, "Legoos: A disseminated, distributed {OS} for hardware resource disaggregation," in *2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19)*, 2019, pp. 69–87.

[56] A. Barbalace, A. Iliopoulos, H. Rauchfuss, and G. Brasche, "It's time to think about an operating system for near data processing architectures," in *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*. ACM, 2017, pp. 56–61.

[57] S. Kannan, N. Bhat, A. Gavrilovska, A. Arpaci-Dusseau, and R. Arpaci-Dusseau, "Redesigning lsms for nonvolatile memory with novelsm," in *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, 2018, pp. 993–1005.

[58] K. Kourtis, N. Ioannou, and I. Koltsidas, "Reaping the performance of fast {NVM} storage with udepot," in *17th {USENIX} Conference on File and Storage Technologies ({FAST} 19)*, 2019, pp. 1–15.

[59] M. Balakrishnan, D. Malkhi, V. Prabhakaran, T. Wobbler, M. Wei, and J. D. Davis, "{CORFU}: A shared log design for flash clusters," in *Presented as part of the 9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12)*, 2012, pp. 1–14.

[60] S. R. Dulloor, A. Roy, Z. Zhao, N. Sundaram, N. Satish, R. Sankaran, J. Jackson, and K. Schwan, "Data tiering in heterogeneous memory systems," in *Proceedings of the Eleventh European Conference on Computer Systems*. ACM, 2016, p. 15.

[61] J. Xu and S. Swanson, "{NOVA}: A log-structured file system for hybrid volatile/non-volatile main memories," in *14th {USENIX} Conference on File and Storage Technologies ({FAST} 16)*, 2016, pp. 323–338.

[62] J. Xu, L. Zhang, A. Memaripour, A. Gangadharaiah, A. Borase, T. B. Da Silva, S. Swanson, and A. Rudoff, "Nova-fortis: A fault-tolerant non-volatile main memory file system," in *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 2017, pp. 478–496.

[63] S. Zheng, M. Hoseinzadeh, and S. Swanson, "Ziggurat: a tiered file system for non-volatile main memories and disks," in *17th {USENIX} Conference on File and Storage Technologies ({FAST} 19)*, 2019, pp. 207–219.

[64] Y. Kwon, H. Fingler, T. Hunt, S. Peter, E. Witchel, and T. Anderson, "Strata: A cross media file system," in *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 2017, pp. 460–477.

[65] A. Klimovic, Y. Wang, P. Stuedi, A. Trivedi, J. Pfefferle, and C. Kozyrakis, "Pocket: Elastic ephemeral storage for serverless analytics," in *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, 2018, pp. 427–444.

[66] P. Stuedi, A. Trivedi, J. Pfefferle, R. Stoica, B. Metzler, N. Ioannou, and I. Koltsidas, "Crail: A high-performance i/o architecture for distributed data processing." *IEEE Data Eng. Bull.*, vol. 40, no. 1, pp. 38–49, 2017.

[67] J. Do, S. Sengupta, and S. Swanson, "Programmable solid-state storage in future cloud datacenters," *Communications of the ACM*, vol. 62, no. 6, pp. 54–62, 2019.

[68] S. Seshadri, M. Gahagan, S. Bhaskaran, T. Bunker, A. De, Y. Jin, Y. Liu, and S. Swanson, "Willow: A user-programmable {SSD}," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, 2014, pp. 67–80.

[69] Z. Ruan, T. He, and J. Cong, "Insider: Designing in-storage computing system for emerging high-performance drive," in *2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19)*, 2019.

[70] Z. István, D. Sidler, and G. Alonso, "Caribou: intelligent distributed storage," *Proceedings of the VLDB Endowment*, vol. 10, no. 11, pp. 1202–1213, 2017.

[71] S. Pei, J. Yang, and Q. Yang, "Registor: A platform for unstructured data processing inside ssd storage," *ACM Transactions on Storage (TOS)*, vol. 15, no. 1, p. 7, 2019.

[72] "Rust," https://www.rust-lang.org/, accessed: 2019-06-25.

[73] C. Kulkarni, S. Moore, M. Naqvi, T. Zhang, R. Ricci, and R. Stutsman, "Splinter: bare-metal extensions for multi-tenant low-latency storage," in *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, 2018, pp. 627–643.

[74] A. Bijlani and U. Ramachandran, "Extension framework for file systems in user space," in *2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19)*, 2019.

[75] G. Amvrosiadis, A. R. Butt, V. Tarasov, E. Zadok, M. Zhao, I. Ahmad, R. H. Arpaci-Dusseau, F. Chen, Y. Chen, Y. Chen *et al.*, "Data storage research vision 2025: Report on nsf visioning workshop held may 30–june 1, 2018," 2018.