RESEARCH ARTICLE

Extending Memory Capacity in Modern Consumer Systems With Emerging Non-Volatile Memory: Experimental Analysis and Characterization Using the Intel Optane SSD

GERALDO F. OLIVEIRA[1], **(Graduate Student Member, IEEE),**
SAUGATA GHOSE[2], **(Member, IEEE), JUAN GÓMEZ-LUNA**[1], **AMIRALI BOROUMAND**[3],
ALEXIS SAVERY[3], **SONNY RAO**[4], **SALMAN QAZI**[3], **GWENDAL GRIGNOU**[3], **RAHUL THAKUR**[3],
ERIC SHIU[4], **AND ONUR MUTLU**[1]

[1]ETH Zürich, 8092 Zürich, Switzerland
[2]Department of Computer Science, University of Illinois Urbana-Champaign, Champaign, IL 61801, USA
[3]Google, Mountain View, CA 94043, USA
[4]Rivos, Mountain View, CA 94043, USA

Corresponding author: Geraldo F. Oliveira (geraldod@safari.ethz.ch)

ABSTRACT DRAM scalability is becoming a limiting factor to the available memory capacity in consumer devices. As a potential solution, manufacturers have introduced emerging non-volatile memories (NVMs) into the market, which can be used to increase the memory capacity of consumer devices by augmenting or replacing DRAM. In this work, we provide the first analysis of the impact of extending the main memory space of consumer devices using off-the-shelf NVMs. We equip real web-based Chromebook computers with the Intel Optane solid-state drive (SSD), which contains state-of-the-art low-latency NVM, and use the NVM as swap space. We analyze the performance and energy consumption of the Optane-equipped Chromebooks, and compare this with (i) a baseline system with double the amount of DRAM than the system with the NVM-based swap space; and (ii) a system where the Intel Optane SSD is naively replaced with a state-of-the-art NAND-flash-based SSD. Our experimental analysis reveals that while Optane-based swap space provides a cost-effective way to alleviate the DRAM capacity bottleneck in consumer devices, naive integration of the Optane SSD leads to several system-level overheads, mostly related to (1) the Linux block I/O layer, which can negatively impact overall performance; and (2) the off-chip traffic to the swap space, which can negatively impact energy consumption. To reduce the Linux block I/O layer overheads, we tailor several system-level mechanisms (i.e., the I/O scheduler and the I/O request completion mechanism) to the currently-running application's access pattern. To reduce the off-chip traffic overhead, we leverage an operating system feature (called Zswap) that allocates some DRAM space to be used as a compressed in-DRAM cache for data swapped between DRAM and the Intel Optane SSD, significantly reducing energy consumption caused by the off-chip traffic to the swap space. We conclude that emerging NVMs are a cost-effective solution to alleviate the DRAM capacity bottleneck in consumer devices, which can be further enhanced by tailoring system-level mechanisms to better leverage the characteristics of our workloads and the NVM.

INDEX TERMS Consumer devices, DRAM, emerging technologies, experimental characterization, I/O systems, memory capacity, memory systems, non-volatile memory, quality of service, solid-state drives, storage systems, system performance, tail latency, user experience, web browsers.

The associate editor coordinating the review of this manuscript and approving it for publication was Mario Donato Marino.

I. INTRODUCTION

The number and diversity of consumer devices (e.g., smartphones, tablets, Chromebooks [1], and wearable devices) are growing rapidly [2], [3], [4], [5], [6]. The number of consumer devices has surpassed the number of desktop computers [3]. For example, web-based computers, such as Chromebooks, account for 58% of all computer shipments to schools in the United States [7]. These devices have different design constraints than traditional computers due to their limited area, power dissipation restrictions, and target market. Therefore, it is essential to guarantee low cost per device while maintaining good performance and high-quality user experience.

One critical component of consumer devices is the main memory (typically consisting of DRAM [8], [9]), which is used not only as the working memory space but also as storage for least-recently-used (i.e., cold) memory blocks (i.e., as swap space [10], [11], [12]). As consumer devices grow in sophistication, many target applications handle increasing amounts of data and require larger main memory capacity to avoid significant performance issues [5], [13], [14], [15], [16], [17], [18]. Unfortunately, it is becoming increasingly challenging to increase DRAM capacity inside consumer devices due to the worsening reliability, cost, and performance issues as manufacturers scale DRAM technology to higher storage capacity levels [19], [20], [21], [22], [23], [24], [25], [26], [27], [28], [29], [30], [31], [32], [33], [34], [35], [36], [37], [38], [39], [40], [41], [42], [43], [44], [45], [46], [47], [48], [49], [50].

As a potential solution to the DRAM scalability challenge, manufacturers have introduced emerging non-volatile memories (NVMs) into the market, which can be used to expand the memory capacity of consumer devices by augmenting or replacing DRAM [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]. However, these NVM-based devices are still slower than DRAM [51], [53], [54], [91], [92], [93], [94], [95], [96]. For example, the state-of-the-art Intel Optane SSD (solid-state drive) [97], which is a low-latency NVM-based SSD device (i.e., an SSD device that uses NVM as its primary persistent storage media), has an access latency that is two orders of magnitude slower than that of DRAM [98], [99] (but it is still one order of magnitude faster than traditional NAND-flash-based SSDs [91], [92], [100], [101], [102], [103], [104], [105], [106]), while providing a better cost-per-byte ($1.50 per GB [107] vs. $5 per GB for DRAM [108]). Previous works propose two different ways to integrate an NVM device into state-of-the-art computers in order to alleviate DRAM scalability issues. The first method uses the device as byte-addressable main memory, which directly replaces DRAM. In this method, the system can access the NVM device directly using load and store instructions [51], [52], [53], [55], [56], [80], [92], [95], [96], [109], [110],

[111], [112], [113], [114], [115], [116], [117]. The second method uses the NVM as a block storage device that replaces a NAND-flash-based SSD [118], [119], [120], [121], [122], [123], [124] or a magnetic hard drive, using the same access protocol and interface [91], [100], [101], [102], [103], [104], [105], [106], [116], [125], [126], [127], [128], [129], [130]. Both methods alleviate DRAM scalability issues by taking advantage of the higher density and lower cost-per-byte that NVMs offer over DRAM. However, *entirely replacing DRAM* with NVM in consumer devices imposes large system integration and design challenges (e.g., due to the high write access latency and limited endurance of NVM [51], [53], [54]).

Integrating emerging NVM-based SSDs in consumer devices can open up new opportunities for memory management. Traditional desktop and enterprise computers employ a swap space [10], [11], [12], [18] to increase the total main memory space available in the system beyond the capacity of available DRAM. In such systems, the system moves cold memory blocks present in DRAM to a swap space, which usually exists in a high-latency and high-capacity storage device (e.g., NAND-flash-based SSD, magnetic hard drive). However, consumer devices usually do *not* employ a swap space [131], [132], [133], [134], [135], [136], [137], [138], [139], [140], [141], [142], [143], since accessing a storage device directly impacts system performance and user experience due to the device's high access latency. Since emerging NVM-based SSDs have an order of magnitude lower access latency than the commonly-used fast storage devices, i.e., NAND-flash-based SSDs, they have the potential to enable the use of a swap space in consumer devices. Recent works [131], [132], [133], [134], [135], [136], [137], [138] propose extending the total main memory space available to applications by using NVM as swap space for DRAM in mobile systems. However, no prior work analyzes the implications of enabling a *real* NVM-based swap space in *real* consumer devices.

In this work, we provide the *first* analysis of the impact of extending the main memory space of consumer devices using off-the-shelf NVMs. We *extensively* examine system performance and energy consumption when the NVM device is used as swap space for DRAM main memory to effectively extend the main memory capacity. Our empirical analyses lead us to several observations and insights that can be useful for the design of future systems and NVMs.

For our experimental evaluation, we equip real web-based Chromebook computers [1] with the Intel Optane SSD [97]. Our target workloads are interactive applications, with a focus on the Google Chrome [144] web browser. We choose such workloads for two reasons. First, in interactive applications, the system needs to respond to user inputs at a target output latency to provide a satisfactory user experience. Second, in Chromebooks, the Chrome browser serves as the main interface to execute services for the user. We compare the performance and energy consumption of interactive

workloads running on our Chromebook with NVM-based swap space, where the Intel Optane SSD capacity is used as swap space to extend main memory capacity, against two state-of-the-art systems: (i) a baseline system with double the amount of DRAM than the system with the NVM-based swap space, which resembles current consumer devices but has high manufacturing cost due to the large DRAM capacity and relatively high cost-per-bit of DRAM; and (ii) a system where the Intel Optane SSD is naively replaced with a state-of-the-art (yet slower) off-the-shelf NAND-flash-based SSD, which we use as a swap space of equivalent size as the NVM-based swap space. The NAND-flash-based SSD provides a cheap alternative to extend the main memory space, but it can penalize system performance due to its high access latency. We use a memory capacity pressure test [145] to measure the impact of the new NVM swap space on user tasks that consist of loading, scrolling, and switching between Chrome browser tabs. We measure how the NVM device increases the 99th-percentile latency (i.e., tail latency) of each task and the total number of Chrome tabs that the user can open without discarding old tabs. We divide our system evaluation, analysis, and optimization into two major parts: (1) *Evaluating NVM for Consumer Devices*, and (2) *System Optimization*.

Evaluating NVM for Consumer Devices: In the first part of our work, we compare the baseline system with the system where we extend the main memory space with the Intel Optane SSD and the system where we extend the main memory space with the NAND-flash-based SSD. We make four major observations. First, we observe that extending the main memory space with the Intel Optane SSD improves the average performance of interactive workloads (measured as the latency of switching across Chrome browser tabs) compared to the baseline system with twice the amount of DRAM. The NVM-based swap space enables the system to leverage a larger aggregate main memory space than the baseline system while also reducing system cost. However, extending the main memory space with the Intel Optane SSD increases the number of violations of the application's target output latency by 2.6× (on average) once the memory traffic between DRAM and the Intel Optane SSD exceeds a threshold, which happens under high system load (e.g., a large number of opened Chrome browser tabs). Since the Intel Optane SSD is integrated to the system via the high-latency off-chip bus, constantly moving data between DRAM and the Intel Optane SSD directly impacts browser performance. Second, we observe that accessing the Intel Optane SSD through the power-hungry off-chip bus *significantly* increases energy consumption. We mitigate this issue by allocating some DRAM space to be used as a compressed in-DRAM cache for data swapped between DRAM and the Intel Optane SSD, which reduces the number of accesses to the Intel Optane SSD by up to 2.11× and, as a result, improves energy efficiency. Third, extending the main memory space even with the

slow NAND-flash-based SSD provides performance benefits compared to the baseline system. However, due to the high access latency of the NAND-flash-based SSD, the number of violations of the application's target output latency increases compared to the baseline system and the system with the Intel Optane SSD's NVM-based swap space. Fourth, we observe that the Linux block I/O layer becomes a major source of performance overhead when the main memory space is extended using NVM, primarily due to (i) I/O scheduling bottlenecks created by the mismatch between our workloads' I/O access patterns and the default I/O scheduling policy; and (ii) overheads related to the asynchronous operation of the I/O request completion mechanism. We mitigate some of these overheads by proposing two system optimizations that can better leverage the characteristics of our workloads and the NVM.

System Optimization: In the second part of our work, we mitigate some of the system-level overheads we identify in the first part of our work by proposing two system optimizations that can better leverage the characteristics of our workloads and the NVM. First, we employ different I/O schedulers that better match our workloads' I/O access patterns, which improves performance. Second, we change the default asynchronous I/O request completion model to a hybrid I/O request completion model that adaptively switches from synchronous to asynchronous operation. The baseline asynchronous I/O request completion model entails non-trivial overheads (e.g., latency overheads of interrupt and context switch). The hybrid I/O request completion model partially avoids these overheads by allowing the process to synchronously wait for the completion of I/O requests for a determined period of time. As a result, in the best case, the process needs to wait for only the low access latency of NVM, instead of incurring the large latency overheads of the asynchronous I/O request completion model.

We make the following key contributions in this work:

- We perform the first experimental analysis of the impact of using off-the-shelf non-volatile memory (NVM) for swap space in a real consumer device. Our studies highlight how a state-of-the-art NVM-based Intel Optane SSD can be used effectively to extend the total main memory space available to interactive applications such as the Google Chrome web browser.
- We demonstrate that using a state-of-the-art off-the-shelf NVM-based SSD as swap space can improve the performance of interactive applications compared to increasing DRAM capacity, but that naively integrating the NVM-based SSD leads to system-level overheads. These overheads primarily arise from the Linux block I/O layer and the off-chip traffic to the SSD.
- We identify two system optimizations that can mitigate some of the system-level overheads that occur when using an NVM-based SSD as swap space. Both of these optimizations adapt system-level mechanisms to

application and runtime behavior, and improve the overall performance of the system.

II. BACKGROUND AND MOTIVATION

We provide the background and motivation required to understand the main components of our experimental setup and evaluated workloads. First, we discuss how the memory system impacts the performance of the Google Chrome web browser [144] (Section II-A). Second, we investigate how users interact with consumer devices and how this interaction contributes to memory capacity pressure (Section II-B). Third, we explain the main characteristics of the Intel Optane SSD device [146] (Section II-C).

A. GOOGLE CHROME WEB BROWSER IN CONSUMER DEVICES

The web browser is one of the main applications of consumer devices. Due to its significance, several web browser applications are present in many mobile benchmark suites [14], [147], [148], [149]. The Google Chrome web browser [144], which has over a billion active users and the largest share of the mobile browsing market [150], [151], is one of the most relevant web browsers available. Therefore, we investigate Google Chrome performance in this work.

Chrome performance can be defined based on three key metrics: (i) the time it takes to load a web page; (ii) the smoothness of scrolling a web page (i.e., whether or not the user can perceive discontinuous movements or jumps when moving up/down inside a web page, measured in frames per second); and (iii) how quickly the browser can switch between web pages in different browser tabs (i.e., its *tab switch latency*). Loading a new web page and scrolling through a web page are highly compute-intensive operations [14], since the main operations that Chrome performs during their execution are rendering [152] and rasterization [153], respectively. In this work, we are primarily interested in evaluating the impact of the swap space on Chrome performance. Therefore, the most relevant metric for our analysis is the tab switch latency, since switching between a recently-opened web page and a previously-opened web page (i.e., a web page that is open in an inactive tab) will likely result in a page fault when the system runs out of memory.

In Chrome, each tab represents a single process associated with the web page displayed in the tab, which improves reliability and security [154], [155]. When the user switches between tabs, the browser executes two main tasks. First, it executes a context-switch between the currently-opened tab and the requested tab. Second, it executes a load operation of the requested web page, which involves loading data frames related to the requested web page from memory and rendering the requested web page. The latency from the time a user clicks a web page to the time the web page is rendered on the screen is crucial since it impacts user satisfaction. This time is mostly dominated by how fast the system can load the data frames related to the requested web page from

memory or disk. However, storing a large number of web pages has become a challenge for consumer devices for two main reasons. First, the total size of a single web page has been growing in recent years due to the increased use of images, JavaScript, and video in web pages [156]. Second, users tend to keep many web pages open concurrently during web browsing, leading to many open tabs. This results in a demand for larger memory space required to keep the web page in physical memory in modern systems.

To understand the impact of the number of open web pages and memory consumption, we evaluate how many web pages it is possible to open in a system with 8 GB of DRAM before the system runs out of physical memory. Figure 1 shows the memory profile of a test that continually opens new Chrome web pages during one hour of execution. We observe that the system runs out of physical memory (i.e., free memory) by opening only 30 web pages (within the course of almost 10 minutes). When this happens, the system enters a *memory capacity pressure* state, leading to increased swap activity (as shown by the increasing red line in Figure 1) and, consequently, performance degradation (not shown in Figure 1). We conclude that there is a clear need to provide more memory capacity to support more concurrently-open web pages and, hence, better user experience in consumer (i.e., mobile) devices.
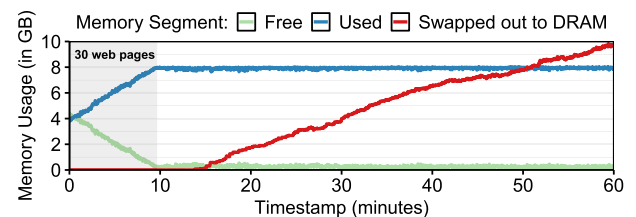

FIGURE 1. System memory usage while loading 30 Chrome web pages.

A traditional way of expanding the memory space in modern systems is to enable page swapping [10], [11], [12]. However, mobile devices usually disable page swapping due to the large performance penalty and user experience degradation imposed on the system by high-latency storage devices [131], [132], [133], [134], [135], [136], [137], [138], [139], [140], [141], [142], [143]. Instead, a kernel module contiguously verifies the memory space and terminates processes to make room for incoming memory requests. This approach is called "*low memory killer*" [18], [157], [158]. Recently, Google enabled a swap alternative for its mobile devices (i.e., Google Pixel smartphones [159] and Google Chromebooks [1]). In these devices, the operating system (OS) enables an in-DRAM compressed swap space, called *ZRAM* [160]. When enabling ZRAM in the system, the OS reserves a fraction of the DRAM space to be used as a swap device. Pages are compressed before being moved from the working region in main memory to ZRAM (i.e., *swapped out*), and decompressed before being moved from ZRAM to the working region in main memory (i.e., *swapped in*).

By using compression in ZRAM, the system can increase the capacity of the swap space by the compression ratio (e.g., by 3:1 [161], [162], [163]).

B. THE IMPACT OF MAIN MEMORY CAPACITY PRESSURE ON CONSUMER DEVICES

We design an experiment to understand the impact of main memory capacity pressure on consumer devices. Our experiment aims to characterize: (1) how users utilize a web browser; (2) how often users suffer from high response latencies from interactive workloads; and (3) how often users push the system into a state of memory capacity pressure. For this purpose, we distributed Chromebook devices[1] to 114 different users at Google, whom we asked to perform their daily activities using the Chromebook devices. We picked the users randomly from a major division at the company that employs thousands of people. We monitored their activity by periodically collecting the following system information over a period of three months:

- *Number of Chrome tabs opened:* We recorded the number of open tabs across all Chrome windows. Data samples were reported every 5 minutes. In total, we collected 19,487 data samples.
- *Tab switch latency:* We collected the tab switch latencies for each tab switch the user performed during their activity. Data samples were reported at each individual tab switch. In total, we collected 62,243 data samples.
- *Memory capacity pressure level:* We periodically (i.e., every five seconds) sampled the current state of the memory to determine which of the following three memory capacity pressure levels the memory was currently experiencing: *no memory capacity pressure, moderate memory capacity pressure*, and *critical memory capacity pressure*. The memory capacity pressure level is defined as follows. First, Chrome calculates the amount of *fill* memory, as $fill = 1 - \frac{(mem\_free + \frac{swap\_free}{RAM\_vs\_swap\_weight})}{(mem\_total + \frac{swap\_total}{RAM\_vs\_swap\_weight})}$; where *mem_free* is the amount of main memory space currently free, *mem_total* is the total amount of main memory space (free and occupied), *swap_free* is the total amount of memory space in the swap device that is currently free, *swap_total* is the total amount of swap space, and *RAM_vs_swap_weight* accounts for the relative ease (considering memory access latency) of allocating RAM directly versus having to swap its contents out first. This parameter has a default value of "4", which the operating system empirically has defined. If *fill* ranges between 60% and 95%, the system is under moderate memory capacity pressure. If *fill* is greater than or equal to 95%, it is under critical memory capacity pressure [164]. In total, we collected 1,571,701 data samples.

Figure 2a shows how many Chrome tabs users opened during our experiments. We observe that users had up to 20 Chrome tabs open in 68% of the samples; 21 to 40 Chrome tabs open in 18% of the samples; 41 to 80 Chrome tabs open in 10% of the samples; and 81 to 160 Chrome tabs open in 4% of the samples (no user had more than 160 tabs open at any time). Even though 4% is a relatively low number of occurrences, it represents 710 sample points where the users kept a large number of Chrome tabs open.
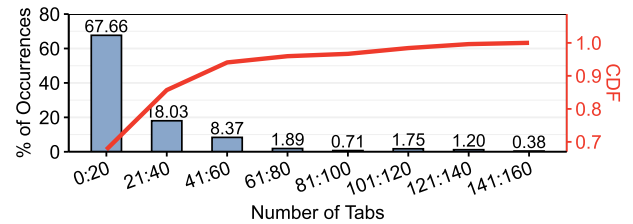
(a) Distribution of the number of open tabs.
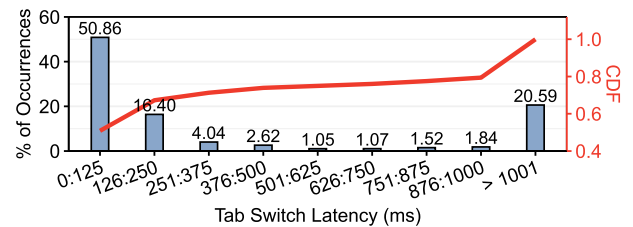
(b) Distribution of the number of tabs switch latencies.

FIGURE 2. Distribution of the number of open tabs and tab switch latencies.

Figure 2b shows the distribution of the tab switch latency the users experienced. We observe that users experienced a tolerable latency (i.e., a tab switch latency less than 250 ms[2]) in 67.3% of our samples. However, the users experienced unacceptable latency from the system (i.e., a tab switch latency greater than or equal to 250 ms) in 32.7% of the samples. The tab switch latency was larger than 1 second for 20.59% of the samples. We periodically collected the memory capacity pressure level of the system to verify that the high tab switch latency was due to high memory capacity pressure. Our experiment shows that 35.8% of our samples, the system experiences moderate to critical memory capacity pressure (563,143 data samples in moderate memory capacity pressure and 42 data samples in critical memory capacity pressure from a total 1,571,701 data samples).

With this experiment, we conclude that real users, for a considerable fraction of their usage time of the Chrome web browser, often push the system to points that induce moderate to critical memory capacity pressure, leading to large and often unacceptable response times for interactive workloads.

C. INTEL OPTANE SSD
In the past several decades, various works [51], [52], [53], [54], [55], [56], [57], [58], [59], [60], [61], [62], [63],

---

[1]The Chromebook devices we use for our experiment consist of off-the-shelf Chromebook devices with 8 GB DRAM capacity, of which 4 GB are reserved to enable an in-DRAM compressed swap space.

[2]We use 250 ms for the tolerable tab switch latency, since a rule of thumb in the web performance community is to provide visual feedback in under 250 ms to keep the user engaged [165], [166].

[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] have investigated how to employ novel data storage technologies (e.g., phase-change memory, PCM [51], [52], [53], [54], [56], [57], [58], [60], [61], [62], [63], [64], [65], [66], [67], [68], [69], [70], [71], [72], [73], [74], [75], [76], [77], [78], [79], [80]; spin-transfer torque magnetic RAM, STT-MRAM [55], [58], [81], [82], [83], [84], [85]; metal-oxide resistive RAM, ReRAM [86], [87], [88], [167], [168], [169], [170], [171]; conductive bridging RAM, CBRAM [89], [172], [173], [174]; ferroelectric RAM, FeRAM [90], [175], [176], [177]) to build fast non-volatile memories. Intel and Micron recently announced the first widely-available commercial NVM device based on the 3D XPoint non-volatile memory technology [178], called Intel Optane [146]. Intel provides two different memory devices based on Optane: (1) the Intel Optane SSD [146], and (2) the Intel Optane DC Persistent DIMM [179]. The key difference between these two devices is their system interface. For the Intel Optane SSD, the device has a system interface similar to current NAND-based flash memory devices, where the system communicates to the device via the PCIe bus [180]. This configuration provides one order of magnitude lower latency than traditional NAND-flash-based SSDs [91], [92], [100], [101], [102], [103], [104], [105], [106]. For the Intel Optane DC Persistent DIMM, the device is integrated into the system with a DIMM-based interface, similar to DRAM devices. The system directly accesses the device using load/store requests at the byte granularity [98], [109], [181], [182], [183], [184], [185], [186], [187], [188], [189]. This configuration provides a much lower access latency, on the order of hundreds of nanoseconds (around 169 ns for sequential reads [98], [187], [188], [189]), but comes at a high cost, 5× the cost of the Intel Optane SSD in dollars-per-bit [107], [190].

Even though the Intel Optane DC Persistent DIMM can provide significant benefits for future systems due to its performance characteristics, there are several challenges to solve before leveraging such devices in future systems, including: (1) the need for system mechanisms for proper data placement between DRAM and the Optane DIMM device [93], [94], [95], [96], [114], [191], (2) difficulties in fabricating printed circuit boards (PCBs) for mobile platforms that can accommodate the Optane DIMM, and (3) accommodating the high cost-per-bit of the Intel Optane DIMM in cost-sensitive mobile systems. Even though prior works [131], [132], [133], [134], [136], [137] propose, using simulation models, to integrate byte-addressable NVM devices as swap space for mobile systems, we choose the Intel Optane SSD for our studies for two main reasons. First, due to the manufacturing difficulties, including high manufacturing cost, and open system-level challenges that need to be solved before integrating the Optane DIMM as swap space in consumer devices. Second, since the goal of our work is to evaluate the performance implications of emerging NVM devices in *real* consumer devices.

III. EXPERIMENTAL SETUP AND METHODOLOGY

In this work, we characterize the performance of interactive workloads running on consumer devices. Our target device is the Google Chromebook web-based computer. We use the Asus Chromebox 3 [192] for our experiments, as it is not physically possible to integrate the Intel Optane SSD module in regular Chromebook due to its limited PCIe lanes. The Asus Chromebox runs the same operating system as the Chromebook device (ChromeOS [193]), and has a similar hardware configuration. The device is equipped with a 7th-generation Intel Core i3-7100U processor [194], 8 GB DDR4 memory [195], and a 32 GB NAND-flash-based SSD [196]. ChromeOS uses up to 50% of the DRAM capacity (i.e., 4 GB) to enable an in-DRAM compressed swap space called ZRAM, capable of holding up to 12 GB of compressed data (assuming a 3:1 compression ratio [161], [162], [163]).[3] We modify the system by (1) removing the in-DRAM compressed swap space and including an Intel Optane SSD module, which the system uses as the swap device for DRAM; and (2) reducing the DRAM size to 4 GB, to hold the non-swap-space DRAM capacity constant. We use the Intel Optane H10 [97] module for our experiments. It contains a 16 GB Intel Optane SSD device and a 256 GB Intel QLC 3D-NAND-flash-based SSD. We modify the Intel Optane H10 firmware to avoid using the NAND-flash-based SSD during our experiments.[4]

We compare the performance and energy consumption of interactive workloads running on our Chromebook using three system configurations, as Table 1 describes:

- *Baseline:* a baseline system with 8 GB of DRAM. 4 GB are used as main memory, which is uncompressed, and the other 4 GB are used as an in-DRAM compressed swap space (ZRAM), which can house up to 12 GB of actual data, assuming a 3:1 compression ratio [161], [162], [163];
- *Optane:* a system with 4 GB of main memory, and 16 GB of Intel Optane SSD swap space;
- *NANDFlash:* a system with 4 GB of main memory, and 16 GB of NAND-flash-based SSD swap space.

One of the main obstacles we faced during our analysis was creating the correct experimental setup. This is challenging for three main reasons, mostly related to the lack of a standard benchmark suite for consumer devices [13] and the lack of automation tools for real-world experiments on mobile devices like Chromebooks:

---

[3]We use a 3:1 ZRAM compression rate as an empirically-evaluated upper bound observed by prior works [161], [162], [163]. In practice, ZRAM compression rate varies with the pages getting swapped out. We observe an average ZRAM compression rate of 1.14:1, with a maximum of 3:1, and a minimum 0.001:1 from our analysis.

[4]We selected the Optane H10 module for our experiments because it was the only Optane device in stock at the time we performed the studies. Based on the technical specifications [97], [197], the H10 module combines the Intel Optane M10 module [197] (i.e., an M.2 module containing only the Intel Optane SSD) with a QLC 3D-NAND-flash-based SSD. In our initial tests, we did *not* observe any performance impact on the raw performance of the H10 module with our modified firmware. Our modified firmware only disables the QLC 3D-NAND-flash-based SSD in the H10 module.

TABLE 1. Evaluated system configurations.

Swap Space Configurations				
Configuration	**DRAM Capacity**	**Swap Space Device**	**Swap Space Size**	**Effective Memory Capacity**
Baseline	8 GB	In-DRAM Compressed Swap Space (ZRAM [160])	12 GB	16 GB
Optane	4 GB	Intel Optane SSD (H10 Module) [97]	16 GB	20 GB
NANDFlash	4 GB	NVMe NAND-flash-based SSD	16 GB	20 GB
Common System Parameters				
Hardware Setup	Asus Chromebox [192]; 7th-generation Intel Core i3-7100U processor [194]; DDR4 main memory [195]; 32 GB NAND-flash-based SSD [195] for storage			
Software Setup	Operating System: ChromeOS [193]; kernel version 4.14; Test Automation Tool: Chromium Project's memory capacity pressure test [145]			

Challenge 1: Executing real-world workloads. Popular interactive workloads for mobile devices are proprietary (e.g., social networks such as Facebook [198] and Instagram [199], messengers such as WhatsApp [200] and Telegram [201], document readers such as Adobe Acrobat Reader [202], games such as Minecraft [203]), and their source code is not openly available. This limits the scope of our analysis, since we can only analyze such applications as a black box, often making it unclear which specific system resources an application uses and why. Prior works [147], [148], [149], [204], [205] put effort into creating benchmark suites for mobile applications. However, they are outdated and often include only a small number of kernels from a small set of applications.

Challenge 2: Automating execution and enabling reproducibility. There is a lack of tools for automating the execution of mobile workloads [13]. This is critical when evaluating an entire system, since experiments need to be executed multiple times to reduce system-level noise (e.g., due to OS tasks, uncontrollable network response times). Without automation, it is difficult to launch and execute applications in an automatic and easily reproducible manner, deal with network traffic, and mimic user interactions with the system, for example.

Challenge 3: Stressing the main memory capacity. As prior works show [18] and as we observe in our analysis, running a single application is usually not enough to stress the main memory capacity and create swap activity, which we aim to study in this work. This happens because many popular interactive applications have a memory footprint of only a few hundreds of megabytes [18], which fit within the main memory of the device. Even though we could mix an increasingly large number of different applications until we place the system under memory capacity pressure, this procedure would be hard to automate, since each application requires different user interactions, and generating random combinations of workloads could dramatically change our analysis.

To overcome these challenges, we rely on the infrastructure that the open-source Chromium project [206] provides to automate the execution of web-based processes. Specifically,

we use the Chromium project's open-source memory capacity pressure test [145] to evaluate the system. The test has three phases. In the first phase (memory pressure), the test opens multiple Chrome tabs in Chrome until the first tab discard occurs (i.e., when Chrome terminates the process associated with an open tab). In the second phase (cold switch), the test opens the least-recently-used tabs (called cold tabs) to induce page faults. In the third phase (heavy load), the test executes tab switches to measure system performance under heavy memory capacity pressure. A tab discard happens when Chrome observes that the system is running out of memory. To calculate the amount of available memory, Chrome computes *available_mem* = *available_RAM* + *num_swap_pages*/*RAM_vs_swap_weight*, where *num_swap_pages* defines the number of available (free and non-defective) page slots in all active swap areas, and *RAM_vs_swap_weight* accounts for the relative ease of allocating RAM directly versus having to swap its contents out first.

Using the memory capacity pressure test in our analysis allows us to overcome all three main challenges we discuss above. We mitigate the first challenge (i.e., executing real-world workloads) by using the open-source and commonly-used Chrome web browser as our primary workload driver. Doing so brings two main advantages for our experimental setup. First, we can fully understand Chrome's internal structure since it is an open-source tool, and can understand the system resources it demands using different profiling tools (e.g., *perf* profiler [207]). Second, we create an environment where the system executes distinct tasks concurrently, since the user utilizes Chrome as the primary interface to execute different services, and Chrome creates a new process for each new Chrome tab. We mimic a multiprocess system that runs different workloads and stresses different segments of the system, by opening different Chrome tabs that execute different services. These services include Google web services (e.g., YouTube [208], Google Maps [209], Google Sheets [210], Google Docs [211]), Facebook [198], and Twitter [212], each of which demands different computational sources. For example, when loading YouTube as one of our Chrome tabs, the newly-created process executes

tasks related to web browsing (e.g., texture tiling, color blitting) and YouTube-related tasks, such as video decoding, locally [213]. We profile our system using the *perf* profiling tool while executing our memory capacity pressure test, to characterize the workloads and tasks executed by our setup that are *unrelated* to web browsing tasks. A non-exhaustive list of workloads executed in our evaluation setup is:

- Video decoding using the FFmpeg [214] and libvpx [215] libraries, which are used to execute the VP8/VP9 [216] video decoder;
- Web Media Player [213] to support HTML5 video playback [217], including audio/video decoders supported by the Mojo System API [218] and the Video Acceleration API [219];
- Audio rendering utilizing Chromium's audio rendering API [213];
- GIF decoding/encoding using the SkGifCodec [220];
- V8 JavaScript engine [221].

We observe that our experimental setup includes two of the workloads also evaluated by prior work on consumer devices [14] (Chrome web browser, VP9 video decoding). In addition, our setup includes several workloads not covered by prior work [14] (e.g., Web Media Player, audio rendering, GIF decoding/encoding, V8 JavaScript engine) that are commonly employed in consumer devices. Our system setup is sufficient for our study, since our goal is not to provide optimizations for a particular workload, but rather to understand the impact of new memory technology in a real system while running real applications.

We mitigate the second challenge (i.e., the lack of tools for automation and reproducibility) by leveraging the memory capacity pressure test's capability to load a new Chrome tab, scroll through a tab, and perform tab switches across open tabs without user's intervention. This is possible since the test utilizes ChromeOS' Tast integration-testing framework [222]. The framework provides APIs that allow the test code to interact with elements of the user interface through the chrome.automation library [223].

We mitigate the third challenge (i.e., stressing the memory system to induce enough swap activity) by launching enough Chrome tabs until the system experiences moderate to critical memory capacity pressure (see Section II-B). We can easily control the amount of memory capacity pressure in the system since we can easily predict the memory footprint of opening a new Chrome tab. As we discuss in Section II-B, we quantitatively analyze a range of memory capacity pressure conditions that are experienced during real user activity in a user study across 114 users.

Metrics. We use two key metrics throughout our analysis in this paper: (i) *tab count*, which is the number of Chrome tabs our memory capacity pressure test can open before a tab discard happens; and (ii) *tab switch latency*, which is the latency of switching across different tabs that are already open. The tab count metric provides an indication of the memory capacity pressure the system can support. The tab switch latency metric is relevant and important for two

main reasons. First, it directly impacts the user experience by affecting the response time to the user. Second, the tab switch latency metric provides a clear indication of the performance impact on our interactive workloads of moving memory blocks back and forth between main memory and the swap space. By switching to a previously-opened tab, whose memory pages have been moved from main memory to the swap space, we force the system to load data from the swap space. Then, the latency associated with the data movement from the swap space to main memory is accounted for in the final tab switch latency.

IV. EVALUATING INTEL OPTANE SSD FOR CONSUMER DEVICES

In this section, we evaluate the performance implications of employing an Intel Optane SSD in consumer devices. First, we evaluate the impact on Chrome web browser performance of reducing DRAM size while using the Intel Optane SSD as swap space for DRAM (Section IV-A). Second, we leverage a compressed in-DRAM cache to reduce tail latency and energy consumption in a system equipped with the Intel Optane SSD (Section IV-B). Third, we evaluate the impact of replacing the Intel Optane SSD with a cheaper state-of-the-art NAND-flash-based SSD (Section IV-C).

Throughout the evaluations we conduct in this section, we push the system to a critical memory capacity pressure state by opening as many Chrome tabs as possible (i.e., until the first tab discard happens). We evaluate such an extreme state since our goal in this section is to fully understand the benefits and drawbacks of employing the Intel Optane SSD device in our system. In addition, we report and analyze the impact of the Intel Optane SSD device as swap space at moderate memory capacity pressure states, where the system has fewer Chrome tabs open.

A. EFFECT OF NVM AS A SWAP SPACE

We evaluate (i) the number of Chrome tabs the memory capacity pressure test can open before a tab discard happens, and (ii) the tab switch latency for the baseline (i.e., the system with 8 GB of DRAM and ZRAM as the swap device) and the Optane (i.e., the system with 4 GB of DRAM and the Intel Optane SSD as the swap device) configurations. During our analysis, we observe that the number of open tabs by the Optane configuration is 24% larger than the number of open tabs by the baseline configuration our memory capacity pressure test (164 vs. 132 open tabs for the Optane configuration and the baseline configuration, respectively). We observe that this increase in the number of open tabs in the Optane configuration is due to the increase in total memory space provided by the Optane configuration. In the Optane configuration, the total memory space available is 4 GB of DRAM plus 16 GB of swap (thus, 20 GB of effective main memory space). In the baseline configuration, this value is 25% lower, since even though the system has 8 GB of DRAM, up to 50% of DRAM space is reserved for the in-DRAM compressed swap space (i.e., ZRAM). With an up to

3:1 compression ratio [161], [162], [163], the total memory space in the baseline configuration becomes 4 GB of DRAM plus 12 GB of swap space (thus, 16 GB of effective main memory space).

Figure 3 shows the tab switch latency distribution for the baseline and Optane configurations. The figure depicts the tab switch ID, i.e., an identifier for a given sorted tab switch latency (x-axis) and the sorted tab switch latency (y-axis, in logarithmic scale). We draw two observations.
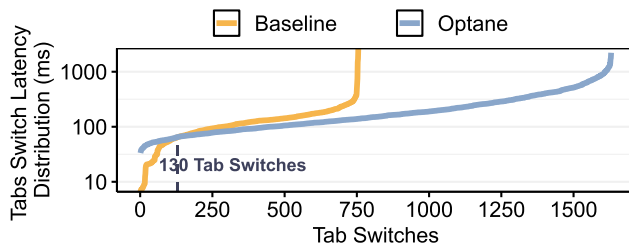

FIGURE 3. Tab switch latency distribution: Baseline (ZRAM) vs. Optane.

First, we observe that the tab switch latency of the baseline configuration is lower than that of the Optane configuration when the number of tab switches is lower than 130. This is because the Optane configuration has half of the DRAM space as the baseline configuration. As a result, the Optane configuration experiences more page faults than the baseline configuration. In our experiments, the baseline configuration experiences 451 page faults until 130 tab switches, which are serviced by the ZRAM space. In contrast, the Optane configuration experiences 10× the page faults of the baseline until it hits 130 tab switches, which significantly reduces Optane performance at low tab switch counts. It is important to highlight that there are fewer page faults for the baseline configuration than for the Optane configuration when the number of tab switches is low, because the physical memory space the OS reserves for ZRAM is *not* statically allocated. Initially, during execution, the baseline configuration enjoys a larger memory space than the Optane configuration since the baseline configuration has 8 GB of physical memory available as working memory. This leads to a lower number of page faults experienced by the baseline configuration than by the Optane configuration when the tab count (and therefore the number of tab switches) is low. However, as swap activity increases with larger number of open tabs, the OS allocates physical memory for the compressed swap space utilized by the baseline. The OS allocates physical memory for the ZRAM swap space until the total swap space size reaches a predefined threshold (4 GB of physical DRAM space in our setup). After the maximum swap space threshold is reached, the baseline configuration can leverage *only* the remaining 4 GB of physical memory as working memory, which leads to a much larger number of page faults.

Second, after 130 tab switches, the Optane configuration provides a lower tab switch latency than the baseline configuration. That is due to how ZRAM operates once the system runs out of memory: Chrome pages need to be constantly swapped into/out of the ZRAM space. This incurs high CPU overhead and data movement between DRAM regions since the processor needs to frequently (1) find cold pages to move from main memory to ZRAM (i.e., swap pages into the ZRAM space), which requires the processor to execute data compression operations; and (2) find and move requested cold pages from ZRAM to main memory in case of a page fault (i.e., swap pages out of the ZRAM space), which requires the processor to execute data decompression operations. In contrast, the Optane configuration greatly alleviates CPU usage caused by compression/decompression activities. When memory pages are swapped into/out of the Optane-based swap space, the processor needs to *only* issue asynchronous read/write requests to the Intel Optane SSD device. We observe that by eliminating the CPU time the system would spend on compression/decompression activities, the page fault latency in the Optane configuration is 35% lower than in the baseline ZRAM configuration, which leads to lower tab switch latencies in the Optane configuration. We conclude that the Optane configuration provides better performance than the ZRAM configuration for high-enough tab counts.

Even though using the Intel Optane SSD as a swap space provides benefits for average tab switch latency (especially at high tab counts), it can also harm tail latency performance. Figure 4 shows the distribution of tab switches with latency larger than 250 ms for both the baseline and Optane configurations. We make two observations. First, the fraction of tab switches with a latency larger than 250 ms for the baseline configuration is 7.1%, on average during the execution of the memory capacity pressure test. In contrast, in the Optane configuration, the fraction of high-latency tab switches is 18.4%, on average. Thus, in the Optane configuration, the fraction of high-latency tab switches is 2.6× those in the baseline configuration. Second, for low tab counts (until 60 tabs), the percentage of high-latency tab switches for the Optane configuration is (i) the *same* as in the baseline configuration for 1–20 tab counts (0% vs. 0% for the Optane and baseline configurations, respectively); and (ii) *slightly* larger than the baseline configuration for 21–40 and 40–61 tab counts (high-latency tab switches make up 5.5% vs. 2% for 21–40 tab counts, and 6.5% vs. 3% for 41–60 tab counts, for the Optane and baseline configurations, respectively). However, after 61 tabs, the number of tab switches with high latency increases significantly for the Optane configuration. To understand the reason of this increase in tab switches with high latency in the Optane configuration, we analyze the page fault latency distribution during the execution of our memory capacity pressure next.

Figure 5 shows the page fault latency distribution for all page faults in the baseline and Optane configurations during the execution of our memory capacity pressure test. We make two observations. First, when analyzing the page fault latency, we observe that at least 98% of the page faults have a response latency of less than 10 µs for both configurations (Figure 5, left). Second, when examining the
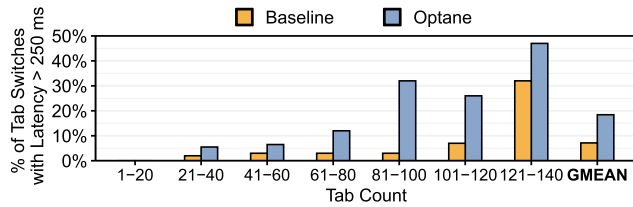
FIGURE 4. High-latency tab switch distribution: Baseline (ZRAM) vs. Optane.

FIGURE 6. Energy and swap traffic. Y-axis values are normalized to the baseline configuration.

tail of the latency distribution (Figure 5, right), the Optane configuration has $4.43\times$ the number of page faults of the baseline (i.e., the number of page faults with a latency larger than $10\,\mu s$ in the Optane configuration is $4.43\times$ that of the baseline configuration). We conclude that the increase in high-latency tab switches for the Optane configuration (in Figure 4) is due to less than 2% of the page faults, with these page faults having a high response latency (more than $10\,\mu s$).
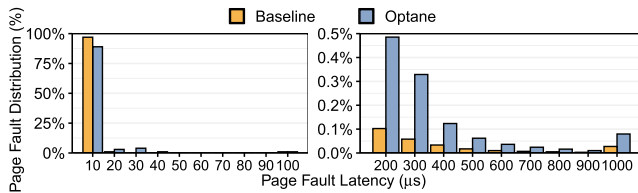
FIGURE 5. Page fault latency distribution.

Figure 6 compares the impact of the baseline and Optane configurations on average memory subsystem energy consumption and swap traffic (i.e., the total number of bytes swapped into/out of the swap space). To model Intel Optane energy consumption, we assume that the device uses PCM-based memory cells, as previous work suggests [224]. Then, we gather the read/write energy consumption of PCM-based memory devices and DRAM, as reported by previous work [225], to model the energy consumption of our system, i.e., 4.4 pJ/2.47 pJ read energy-per-bit and 5.5 pJ/14.03 pJ (set)–19.73 pJ (reset) write energy-per-bit for DRAM/PCM.[5] In the figure, we normalize both metrics to the baseline values (y = 1.0 in the plot). We observe that the Optane configuration increases energy consumption and swap traffic by $69.5\times$ and $37.2\times$, respectively, compared to the baseline configuration. This is due to two main reasons. First, writing one bit to Optane consumes up to $3.6\times$ the energy of writing one bit to DRAM [225]. Even though reading one bit from Optane consumes 56% less energy than reading one bit from DRAM [225], in our analysis, we find that the majority (i.e., 54%) of the accesses to the Intel Optane SSD are write accesses. Second, the Optane configuration generates much more swap activity than the baseline, since the Optane configuration has half the DRAM size of the baseline configuration. We conclude that using

---

[5]We evaluate *only* the energy of the memory subsystem in our analyses. This does *not* include the energy consumed by the processor.
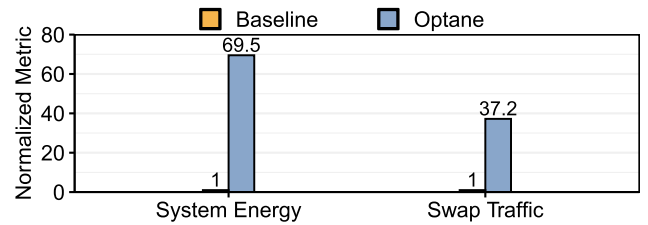
the Intel Optane SSD as a swap space to DRAM severely penalizes system energy consumption.

Summarizing our findings, the Optane configuration provides benefits compared to the baseline system, since it enables a large number of tab switches due to an increase in the main memory space. However, it has significant drawbacks in terms of tail latency, system energy and swap traffic, compared to the baseline with double the amount of DRAM. Most of these downsides come from the large number of accesses, especially write accesses, to the Intel Optane SSD. To solve these problems, we next examine multiple techniques that can (i) reduce the swap traffic in the Optane configuration (Section IV-B) and (ii) improve overall system performance for the Optane configuration (Section V).

B. REDUCING TAIL LATENCY BY ENABLING A COMPRESSED RAM CACHE

The Intel Optane SSD can improve overall performance for consumer devices since it enables an extended memory space. However, as we show in Section IV-A, it can also negatively impact tail latency and system energy consumption due to the need to issue high-latency and power-hungry I/O requests to access the device. In addition, NVM devices such as the Intel Optane SSD suffer from limited endurance [51], [53], [54]. As a result, the large number of write operations caused by page swapping can degrade system reliability.

To overcome these issues introduced by the Intel Optane SSD, we aim to reduce the number of accesses to it. To this end, we augment the Optane configuration with Zswap [226], an in-DRAM swap cache used to store compressed cold-pages. Zswap takes memory pages that are in the process of being swapped out from main memory to the swap device (i.e., the Intel Optane SSD) and attempts to compress them into a dynamically-allocated DRAM-based memory pool. When a page fault happens, the OS checks if the requested memory page is stored in the Zswap cache (i.e., Zswap load hit); if the page is not in the Zswap cache, the OS loads the memory page from the swap space (i.e., Zswap load miss). In case of a Zswap load hit, the OS (1) loads and decompresses the memory page stored in the Zswap cache, and (2) services the page fault request by writing the decompressed memory page into DRAM. The motivation behind Zswap is to trade CPU cycles for a potential reduction

in I/O requests. Zswap can improve performance if the read requests are serviced faster from the in-DRAM compressed swap cache than from the swap device.[6]

To fully leverage Zswap in the Optane configuration, we first need to tailor two important parameters related to Zswap execution. First, as explained in Section II, a tab discard happens when Chrome observes that the system is running out of memory, which is computed based on the `RAM_vs_swap_weight` parameter (i.e., the relative ease of allocating DRAM pages directly versus having to swap its contents out first; see Section III). This parameter is empirically defined for ZRAM as 4. Similarly, we empirically evaluate which value `RAM_vs_swap_weight` enables the memory capacity pressure test to open more tabs before the first tab discard happens. We run tests varying the `RAM_vs_swap_weight` value from 1 to 8, and we observe that `RAM_vs_swap_weight` equals to 1 provides the largest number of open tabs for the Zswap configuration. Second, we need to understand the impact of the maximum Zswap cache size (i.e., `max_pool_size`) on the tab switch latency. To this end, we run the memory capacity pressure test while varying the `max_pool_size` from 0% to 50% of the total DRAM size. As expected, we observe that the tab switch latency increases with the `max_pool_size`. We find that setting `max_pool_size` to 20% of the total DRAM capacity provides a good balance between reduction in I/O traffic and tab switch latency.

Figure 7 compares the total number of open tabs before a tab discard happens in the baseline and Optane configurations, and when we enable Zswap in our Optane configuration (labeled Optane+Zswap). We observe that by enabling Zswap, the number of open tabs reduces by 12% compared to the Optane configuration without Zswap enabled. By enabling Zswap, we effectively reduce the total memory space available, thus causing a discard to happen sooner.
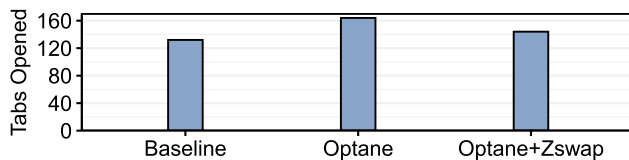

FIGURE 7. Number of open tabs: Baseline vs. Optane vs. Optane with Zswap enabled (Optane+Zswap).

Figure 8a shows the impact of enabling Zswap on the tab switch latency. Enabling Zswap maintains a similar overall tab switch latency as the Optane configuration without

Zswap. Figure 8b shows the impact that Zswap has on high-latency tab switches. We make three observations. First, on average across all tab counts, enabling Zswap leads to a modest increase of 4% on the number of tab switches with latency larger than 250 ms (up to 29% for tab counts of 101–120). Second, the large difference in the fraction of tabs with high latency tab switches between Optane and Optane+Zswap configurations happens when swap activity increases, and the swap cache gets full. When the swap cache gets full, the system needs to (1) free an entry in the swap cache, (2) decompress the selected entry and evict it back to the swap device, and (3) compress the new page and insert it in the swap cache. These operations represent the worst-case latency for a Zswap operation. Third, tab counts of 21–40 and 41–60 have a similar fraction of tab switches whose latencies are unacceptable in both the Optane and Optane+Zswap configurations. Enabling Zswap reduces the number of high-latency tab switches for tab counts of 21–40, resulting in a similar fraction of high-latency tabs as the baseline. For 21–40 tabs, the number of high latency tab switches is 2%, 5.5%, and 3%, for the baseline, Optane, and Optane+Zswap configurations, respectively. We conclude that when a page request hits in the Zswap cache, the system can provide a lower tab switch latency.
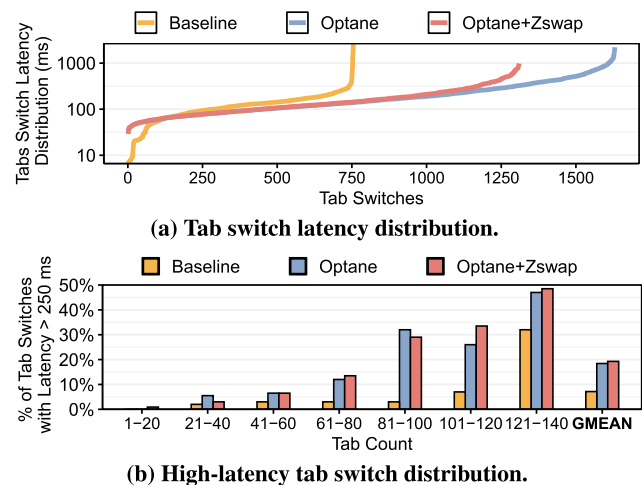

(a) Tab switch latency distribution.

(b) High-latency tab switch distribution.

FIGURE 8. Tab switch latency: Baseline vs. Optane vs. Optane+Zswap.

To fully understand the impact of enabling Zswap in the Optane configuration, we monitor the CPU utilization of the system for both Optane and Optane+Zswap configurations. For this, we execute the memory capacity pressure test for both configurations and monitor the CPU utilization with the `vmstat` tool [227] for one hour. Figure 9 depicts the CPU utilization for both configurations. We make two observations. First, we observe that, when we disable Zswap (Figure 9a), the CPU spends part of the execution time waiting for I/O operations to complete. Towards the end of the execution of the memory capacity pressure test, the I/O waiting time increases dramatically, since the swap activity also greatly increases. Second, we observe that by enabling

---

[6]Even though both ZRAM and Zswap use an in-DRAM compressed memory space to operate, they are fundamentally different mechanisms. While ZRAM is an in-DRAM compressed *swap space*, Zswap is an in-DRAM *swap cache*. The system sees ZRAM as a swap device and Zswap as a cache for memory pages swapped into/out of the swap device. We enable Zswap only in the Optane and NANDFlash configurations, since the goal of enabling Zswap is to reduce I/O traffic to *off-chip swap devices* (i.e., the Intel Optane SSD and the NAND-flash-based SSD in our experiments). We indicate Zswap-enabled configurations explicitly throughout the paper (e.g., the Optane configuration with Zswap is labeled *Optane+Zswap*).

Zswap, the system spends a smaller fraction of its time waiting for I/O operations, as Figure 9b shows. On the other hand, it also spends a larger fraction of its time on kernel activity due to Zswap compression/decompression execution, which reduces the fraction of time spent on user activity and penalizes Chrome browser performance. We conclude that the increase in kernel activity caused by Zswap cache compression/decompression operations is the *primary* cause of the increase in the fraction of high-latency tab switches in the Optane+Zswap configuration.
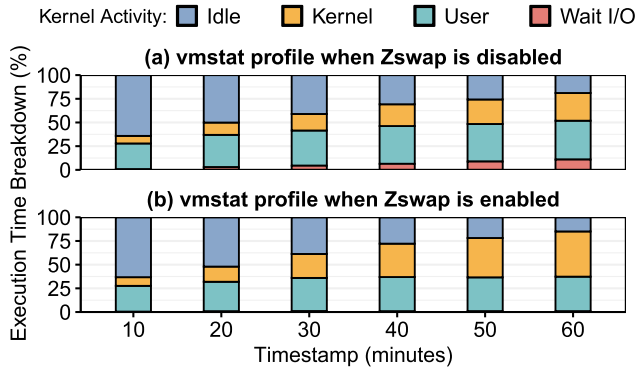

FIGURE 9. System execution time breakdown during a memory capacity pressure test.

To evaluate the effectiveness of the Zswap cache, we analyze the Zswap cache behavior during the execution of the memory capacity pressure test. Figure 10a shows the number of load hits, load misses, and total loads that the Zswap cache services. Figure 10b shows the hit rate of the Zswap cache over the same time. We make the two observations based on Figure 10. First, the Zswap cache provides a high hit rate of 97%, on average during the execution of the memory capacity pressure test. In fact, we see in Figure 10b that the hit rate is close to 100% from 16 minutes to 30 minutes. Second, we observe that the hit rate drops from 100% at 30 minutes to 91% at 60 minutes. We observe that at 30 minutes, the Zswap cache gets full, which requires the system to evict old pages frequently. However, even during high memory capacity pressure, the Zswap cache maintains a significantly high hit rate. We conclude that Zswap cache is an effective cache for the swap space.

Figure 11 shows the distribution of the compression and decompression latencies in the Zswap cache during the execution of the memory capacity pressure test. In the figure, dashed lines represent the average compression/ decompression latency, and solid lines represent the maximum compression/decompression latency. We make two observations. First, the decompression latency, which is on the critical path of Chrome execution when a page fault happens, is 3.9 µs on average during the execution of the memory capacity pressure test (minimum of 1.5 µs, and maximum of 42.6 µs). We observe that 98.7% of the decompression requests have a latency of less than 10 µs. Second, the compression latency is larger, with an average
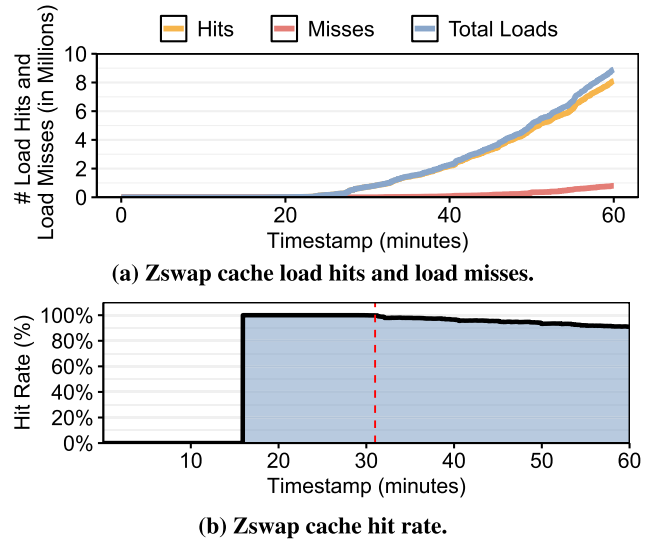

(a) Zswap cache load hits and load misses.

(b) Zswap cache hit rate.

FIGURE 10. Zswap cache performance.

latency of 12.1 µs (minimum of 1.5 µs, and maximum of 138.2 µs). As a comparison, the Intel Optane SSD read latency is 22.4 µs on average (minimum of 9.0 µs, and maximum of 5380 µs). We conclude that Zswap is an effective caching mechanism for the swap space, since it provides significantly lower access latency than directly accessing the swap device.
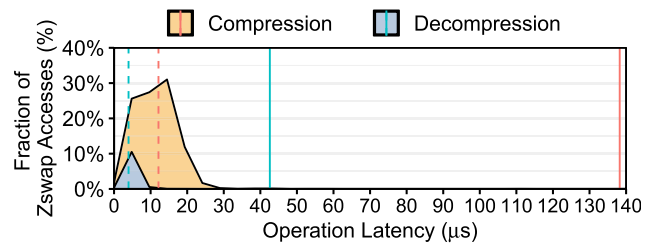

FIGURE 11. Zswap cache compression/decompression latency distribution. Dashed (solid) lines represents average (maximum) compression/decompression latency.

We also study the system energy savings that the Zswap cache provides. Figure 12a shows the energy savings when Zswap is enabled for the Optane configuration. We make three observations from the figure. First, we observe that enabling Zswap reduces overall system energy consumption by 2×. Second, we observe that the majority of the energy is spent on write requests. To understand these energy results, we analyze in Figure 12b the amount of memory swap-in and swap-out activity during the execution of the test. As shown in Figure 12b, with Zswap cache enabled, swap-in and swap-out activity reduces by 2.06× and 2.11×, respectively. This large reduction in swap activity directly translates to a reduction in energy consumption. Third, even with Zswap enabled, the Optane+Zswap configuration consumes 34.75× the energy of the baseline configuration. This large increase in energy consumption is due to (i) an increase in swap activity

since the Optane+Zswap configuration enables significantly more tab switches than the baseline and (ii) the high energy cost of write operations to the Optane device when swapping out pages [225]. We conclude that the Zswap cache greatly reduces energy consumption due to a large reduction of swap traffic. However, the Optane+Zswap configuration still significantly increases the energy consumption compared to the baseline. We expect that such energy can be further reduced by employing techniques to reduce the energy cost of write operations on NVM devices [225], [228], [229], [230].
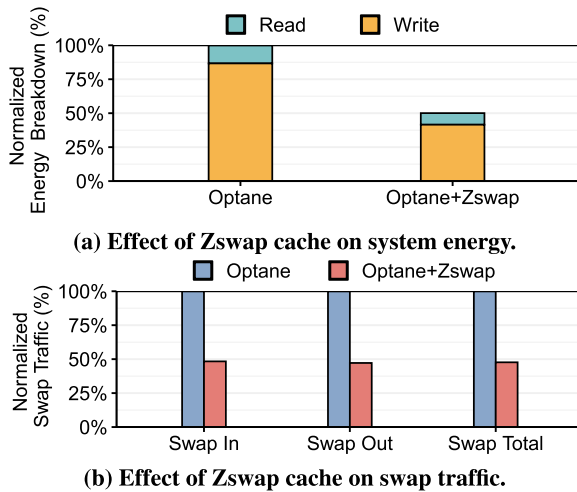

(a) Effect of Zswap cache on system energy.

(b) Effect of Zswap cache on swap traffic.

FIGURE 12. Effect of Zswap cache on system energy and swap traffic.

Lifetime Analysis. One characteristic of NVM devices is their limited write endurance, i.e., a memory cell in the Optane device becomes unreliable beyond a certain number of writes. Therefore, we evaluate how Optane's limited write endurance affects the lifetime of our system when employing the Intel Optane SSD as a swap space. We compare the Intel Optane SSD lifetime (in years) when executing Chrome tab switching and scrolling activities in a system with and without Zswap. To do so, we adopt the lifetime model in [51], which estimates the lifetime of a memory module driven by the access patterns observed in our Chrome workload. We assume a conservative Optane cell endurance of 10^6 writes [231] (i.e., the same cell endurance of PCM-based memory cells [67], [232], [233]) and an *optimistic* wear-leveling mechanism that evenly distributes write requests across all cells of the Intel Optane media (which the Intel Optane SSD is reported to implement [234]). Our model shows that the lifetime of the Optane configuration without (with) Zswap enabled when executing our Chrome workload is 4.5 (8.3) years. Such an expected lifetime can be hard to obtain in practice because it is unlikely that the wear-leveling algorithm will be able to distribute *all* writes across the Optane device *equally*. However, prior works propose more realistic wear-leveling mechanisms that can achieve up to 53% of the lifetime of an optimistic wear-leveling mechanism [235]. Therefore, employing such a wear-leveling mechanism can still guarantee a high lifetime

for our Optane-based system without (with) Zswap enabled of 2.4 (4.4) years.

Summarizing our findings, the Zswap cache is an effective caching mechanism that reduces the swap traffic and system energy consumption when utilizing the Intel Optane SSD as a swap space. These benefits come at the cost of a small increase in the number of high-latency tab switches and a small decrease in open tab count.

C. EFFECT OF USING DIFFERENT NVM DEVICES

In the previous sections, we show that enabling Intel Optane SSD as swap space for consumer devices can provide significant benefits due to the extended main memory space it provides. However, it is essential to understand if we can achieve similar results using cheaper state-of-the-art NAND-flash-based SSDs in place of the Intel Optane SSD. We aim to study whether state-of-the-art NAND-flash-based SSDs that are already widely used (e.g., Micron [236] and Transcend [196] NAND-flash-based SSDs) can provide similar benefits for our workloads as we have observed using the Intel Optane SSD in Sections IV-A and IV-B.

There are three major differences between the Intel Optane SSD and a traditional NAND-flash-based SSD: (1) lower access latency in the Intel Optane SSD, (2) higher endurance in the Intel Optane SSD, and (3) higher cost of the Intel Optane SSD device. First, as previous works [91], [92], [100], [101], [102], [103], [104], [105], [106] show, performing a 4 kB random read using the Intel Optane SSD is approximately 6× faster than using a traditional NAND-flash-based SSD. Second, the Intel Optane SSD can provide 10× the endurance of a traditional NAND-flash-based SSD [237]. Third, a traditional NAND-flash-based SSD is approximately 3× cheaper than the Intel Optane SSD ($0.50/GB [107] vs. $1.5/GB [238], respectively).

We compare the number of open Chrome tabs and the tab switch latency when using the Intel Optane SSD (Optane configuration) versus an NVMe NAND-flash-based SSD as the swap device (NANDFlash configuration). We choose a 16 GB M.2 NVMe NAND-flash-based SSD for our experiments. We also evaluate the effect of enabling Zswap when using the NAND-flash-based SSD (NANDFlash+Zswap).

Figure 13 compares the number of open Chrome tabs under five configurations (baseline, Optane, Optane+Zswap, NANDFlash, NANDFlash+ZSwap). We make two observations. First, the NANDFlash configuration enables 14% more open tabs than the Optane configuration. This is due to the `RAM_vs_swap_weight` parameter. This kernel parameter defines the effort (in terms of swap activity) that the system will demand to allocate more memory. For our Optane configuration, we empirically choose the `RAM_vs_swap_weight` value that gives the best trade-off regarding swap activity, number of tabs open, and tab switch latency. However, since we use the NANDFlash configuration only as a reference, we utilize the default `RAM_vs_swap_weight` value that the kernel suggests.

Thus, even though the kernel can allocate more memory in the NANDFlash configuration than in the Optane configuration, more Chrome tabs result in higher swap activity. Second, when enabling a Zswap cache using 20% of the DRAM capacity in the NANDFlash configuration (the same capacity as in the Optane+Zswap configuration, we evaluate in Section IV-B), the number of open tabs reduces by 12%, due to the decrease in the available DRAM space.
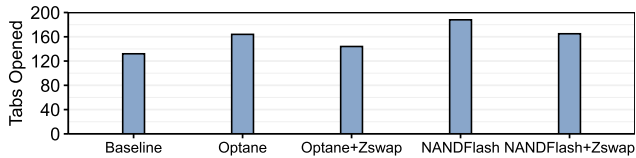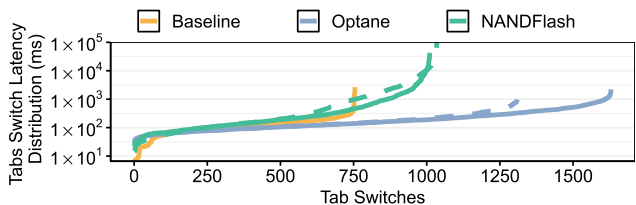

FIGURE 13. Number of open tabs: Baseline vs. Optane vs. NANDFlash.

Figure 14 shows the tab switch latency distribution (Figure 14a) and the distribution of high-latency tab switches (Figure 14b) for all five configurations. We make two observations from the figure. First, we observe (in Figure 14a) that the high access latency of the NAND-flash-based SSD leads to a significant increase in the tab switch latency in the NANDFlash (NANDFlash+Zswap) configuration compared to the Optane (Optane+Zswap) configuration. The average tab switch latency of the NANDFlash (NANDFlash+Zswap) configuration is $3.6\times$ $(10\times)$ that of the Optane (Optane+Zswap) configuration. Second, the number of high-latency tab switches in the NANDFlash configuration (Figure 14b) increases by 35% compared to the baseline configuration, and by 39% compared to the Optane configuration. For high tab counts, the fraction of tabs with high latency is as much as 70% (for 100–120 tabs) in the NANDFlash+Zswap configuration. We conclude that the NANDFlash configuration provides benefits compared to the baseline configuration, but it is unable to approach the performance of the Optane configuration.

(a) Tab switch latency distribution. Dashed lines represent Zswap enabled.
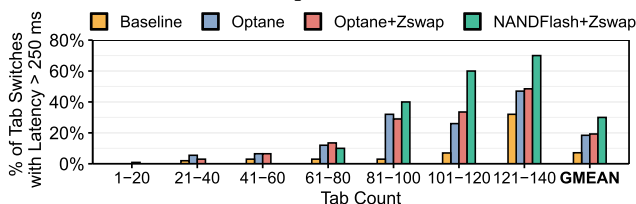
(b) High-latency tab switch distribution.

FIGURE 14. Tab switch latency: Baseline vs. Optane vs. NANDFlash.

Summarizing our findings, using a state-of-the-art NAND-flash-based SSD to extend the main memory capacity of the system improves performance compared to the baseline, considering the number of additional open tabs the NANDFlash configuration provides. However, it cannot achieve similar performance as using the Intel Optane SSD to extend the main memory capacity due to the high access latency of the NAND-flash-based SSD, which leads to a significantly larger number of high-latency tab switches than the Intel Optane SSD. Thus, the Intel Optane SSD can lead to a better user experience than a state-of-the-art NAND-flash-based SSD.

V. SYSTEM OPTIMIZATION

Using the Intel Optane SSD as swap space allows our system to enjoy an extended memory space, which translates to an average latency improvement for our workload at the cost of larger tail latencies. However, longer tail latencies are usually not acceptable. Thus, it is essential to reduce tail latency (i.e., the 99th-percentile latency in our tab switch latency distributions) for interactive workloads since it affects how the user experiences the system.

The goal of this section is to analyze the primary sources of latency overheads that impact the tail latency in the system when we make use of the Intel Optane SSD as swap space. We extensively profile the system when executing the Google Chrome web browser to identify performance bottlenecks caused by the added swap device. We observe that the Linux block I/O layer increases both the average and the 99th-percentile latency for Chrome's page faults significantly, mostly due to I/O scheduling issues and queuing delays, and overheads related to the I/O request completion mechanism. To solve these issues, we tune the system parameters related to the Linux block I/O layer, aiming to improve the 99th-percentile latency for tab switches.

In this section, we limit the number of open tabs in our experiments to 50, as 50 tabs are enough to generate moderate memory capacity pressure in our system and thus examine tail latencies.

A. PROFILING THE CHROME BROWSER

We extensively profile the Google Chrome browser's activity while running the memory capacity pressure test when the Intel Optane SSD is employed as swap space. We use the `perf` profiling tool [239] to collect the execution time breakdown of each Chrome tab (including kernel activity). Figure 15 shows a simplified execution breakdown of one of the representative Chrome tabs that demonstrates long-latency switching times. We observe from the figure that the tab spends more than 96% of its execution time on kernel modules that manage I/O requests (i.e., the *do_page_fault* kernel function, which issues I/O requests in case of a page fault operation; and the *blk_mq_complete_request* kernel function, which receives and completes the processing of I/O requests issued to the swap device). We observe that (i) 50% of the execution time is spent on issuing a block I/O request

due to page faults, and (ii) 46% of the execution time is spent on processing the requested page once the data is received from the swap device. The remaining 4% of the execution time is spent on Chrome's internal processes and other kernel calls. Therefore, we conclude that the block I/O layer is the *primary* source of the tail latency overhead.
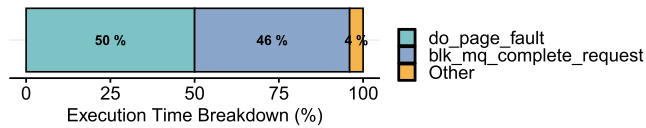

FIGURE 15. `Perf` **results for a Chrome tab.**

B. LINUX BLOCK I/O LAYER

The block I/O layer is the Linux kernel layer responsible for managing block I/O devices (e.g., magnetic hard drives, SSDs) [10], [225]. It is a key system component since accessing block I/O devices involves issuing high-latency and power-hungry operations to the block device. Therefore, the block I/O layer is highly optimized to ensure low latency and high throughput from block devices. Figure 16 illustrates the primary operations the block I/O layer performs when an application or the kernel issues an I/O request (e.g., read from a file, page fault). The block I/O layer works in three main steps.
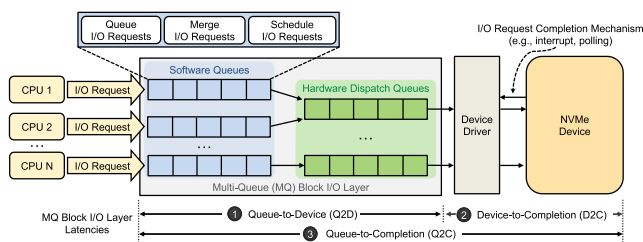

FIGURE 16. **Linux block I/O layer.**

First, when the block I/O layer receives a block I/O request, it queues the request in a request queue (called *software queue*), which is unique per CPU. Second, it attempts to merge and sort requests based on the sector number to avoid costly seek operations in the device. Third, it schedules the requests using an I/O scheduler. The scheduled request is stored in a dispatch queue (called *hardware dispatch queue*). Requests stored in the hardware dispatch queue are issued to the device driver and eventually reach the block device. Once the device completes executing the request, it sends the response back to the block I/O layer, which finalizes the execution of the request by either (1) waking up the requester, in case an interrupt-based (IRQ) I/O request completion mechanism is employed [10]; or (2) forwarding the response to the requester, in case a polling-based I/O request completion mechanism is employed [10]. Recent Linux kernels have adopted a multi-queue (MQ) block I/O layer [240], since modern block devices (e.g., NVMe

devices [241]) can execute many requests concurrently [240], [241], [242]. The MQ block I/O layer employs one request queue per CPU core for each block device.

To monitor and analyze the performance implications of the block I/O layer in the system, we make use of the `blktrace` tool [243]. `blktrace` monitors the activity of the block I/O layer, and provides detailed timing information for each major operation. We analyze three important timings that the tool reports: *(i) queue-to-device (*Q2D*)*, the time from when a block I/O request enters the block I/O layer to the time the request is issued to the block device, including queuing, merging, and scheduling (❶ in Figure 16); *(ii) device-to-completion (*D2C*)*, the time it takes the block device to complete the request (❷ in Figure 16); and *(iii) queue-to-completion (*Q2C*)*, the total end-to-end time for a block I/O request to complete, i.e., Q2C = Q2D + D2C (❸ in Figure 16).

During the execution of the memory capacity pressure test, we profile the block I/O layer using `blktrace`. We analyze the latencies for I/O requests that Chrome processes and the kernel memory management unit (`kswapd`) issue. Figure 17 shows the Q2C, Q2D, and D2C latencies during the execution of the test. The end-to-end block I/O latency (i.e., Q2C latency) is 1.80 ms, on average across both Chrome processes and `kswapd` read and write I/O requests (min. of 0.0085 ms, max. of 414.11 ms). The Q2D latency is 0.03 ms, on average (min. of 0.000 87 ms, max. of 413.80 ms). The D2C latency is 1.78 ms, on average (min. of 0.007 23 ms, max. of 46.10 ms). We make three observations from the reported latencies. First, Chrome processes (`chrome`, `Chrome_IOThread`, and `CompositorTileW`) issue high-latency I/O requests (up to 414.11 ms). Most of these requests are read requests caused by page faults. Second, `kswapd` is responsible for issuing a majority of the write requests to the I/O device.[7] Third, for high-latency I/O requests, most of the I/O latency comes from the block I/O layer rather than from the swap device. While the device latency (i.e., D2C) is at most 46.10 ms, the latency of queuing and scheduling requests in the block I/O layer (i.e., the Q2D latency) dominates the execution time of high-latency I/O requests (as observed by the high maximum Q2D latency of 413.80 ms).

Based on this analysis, we conclude that the block I/O layer operations are the primary bottleneck in high-latency block I/O requests. To alleviate this bottleneck, we investigate how two system optimizations impact block I/O latencies and, consequently, Chrome performance. We investigate the effect of (1) different block I/O schedulers (Section V-C) and (2) different I/O request completion mechanisms (Section V-D) on system's performance.

C. OPTIMIZATION 1: BLOCK I/O SCHEDULERS

The block I/O layer of the Linux kernel provides four different multi-queue I/O schedulers. These schedulers vary in

---

[7]The `kswapd` process issues *only* write I/O requests since its goal is to free memory by reclaiming inactive memory pages. If an inactive memory page is dirty, the `kswapd` process writes this dirty page to swap space.
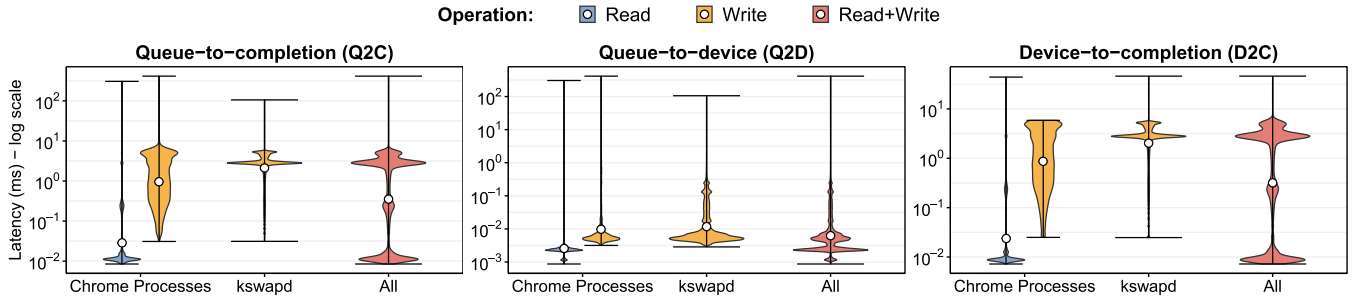
FIGURE 17. `Q2C`, `Q2D`, and `D2C` latency distribution for (i) Chrome processes, (ii) `kswapd`, and (iii) Chrome processes+`kswapd` (*All*) during the execution of the memory capacity pressure test. Error bars depict the minimum and maximum data point values, and a bubble depicts average value of each category.
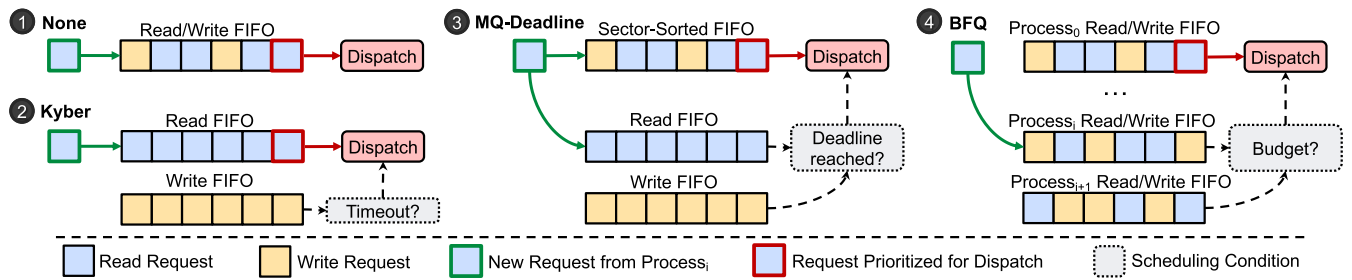

FIGURE 18. Four Linux block I/O layer schedulers.

complexity, and their aim and ability to consider and exploit different properties of different block devices. Therefore, it is essential to tailor the system to make use of the I/O scheduler that matches the requirements and characteristics of the Intel Optane SSD.

We first briefly explain each of the four I/O schedulers. The four I/O schedulers are called None [240], Kyber [244], MQ-Deadline [245], and budget-fair queuing (BFQ) [10]. Figure 18 illustrates the operation of the four I/O schedulers. The None I/O scheduler (❶ in Figure 18) is the simplest I/O scheduler. It employs a simple first-in first-out (FIFO) request queue. Therefore, it does *not* reorder requests. The request queue includes both read and write I/O requests. Due to its simplicity, the None I/O scheduler incurs low overhead, but does not guarantee any quality-of-service.

The Kyber I/O scheduler (❷ in Figure 18) maintains two separate request queues, one for synchronous block I/O read requests, and another for asynchronous block I/O write requests. It prioritizes requests in the read queue over those in the write queue, unless a write request has been outstanding for too long, i.e., it times out by reaching the target write access latency (the default target write access latency for the Kyber I/O scheduler is 10 ms [244]). As such, it is also a simple I/O scheduler that aims to provide better service to read requests.

The MQ-Deadline I/O scheduler (❸ in Figure 18) employs three different request queues: read FIFO, write FIFO, and a sorted FIFO. The sorted FIFO maintains read and write I/O requests that are sorted by the sector number they are to access. The scheduler prioritizes I/O requests from the

sector-sorted FIFO unless any request in either the read or write queue is about to violate its service deadline. The default deadline is 500 ms for read I/O requests, and 5 s for write I/O requests.

The BFQ I/O scheduler (❹ in Figure 18) is the most complex I/O scheduler among the four, which leads to high scheduling overhead. The BFQ I/O scheduler guarantees fairness across processes by distributing the throughput of the block I/O device proportionally to each process via an indirectly assigned weight value. The BFQ I/O scheduler employs an I/O request queue per process and assigns an I/O budget per I/O request queue. It assigns I/O budgets, measured in number of sectors, proportionally to a process's I/O activity. This way, I/O-bound processes with sequential I/O requests are assigned a large I/O budget, while processes with short and sporadic I/O requests are assigned a small I/O budget. The BFQ I/O scheduler uses a variant of the worst-case fair weighted fair queuing+ (WF2Q+) scheduling algorithm [246] to select an I/O request queue to be serviced (typically the I/O request queue with the lowest I/O budget). The selected I/O request queue is prioritized, and its I/O requests are exclusively serviced until its I/O budget finishes. As a result, the BFQ I/O scheduler guarantees a fraction of the device throughput to each process. The Linux kernel uses the BFQ I/O scheduler by default, since it usually provides high system responsiveness and fairness, even though its scheduling decision overhead is higher than the other three I/O schedulers.

Figure 19a shows the tab switch latency for the Optane configuration under the four different I/O schedulers.

We observe that, on average, the four different I/O schedulers provide similar tab switch latencies. The average tab switch latency for the None, Kyber, MQ-Deadline, and BFQ I/O schedulers is 116 ms, 119 ms, 120 ms, and 118 ms, respectively. However, when we examine the fraction of high-latency tab switches in Figure 19b, we observe that the fraction of high-latency tab switches increases significantly with open tab count for the default BFQ I/O scheduler. The BFQ I/O scheduler leads to the largest number of high-latency tab switches when the system has 41–50 tabs open, while the Kyber I/O scheduler provides the lowest number of high-latency tab switches for the same tab-count range, reducing the fraction of high-latency tab switches by 44% compared to the BFQ I/O scheduler. Therefore, to reduce Chrome's tail latency, the Kyber I/O scheduler can potentially be a better I/O scheduler than the default BFQ I/O scheduler.
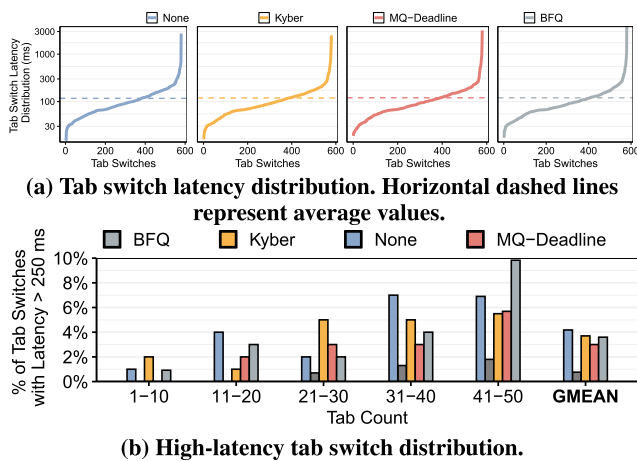

(a) Tab switch latency distribution. Horizontal dashed lines represent average values.

(b) High-latency tab switch distribution.

FIGURE 19. Tab switch latency: Optane with different I/O schedulers.

We further analyze the tab switch latency distribution in Figure 20 for the four I/O schedulers. The figure shows the tab switch latency percentiles (along the x-axis) and the corresponding tab switch latency (y-axis), normalized to the values when the system employs the default BFQ I/O scheduler. We make three observations. First, at the tail (99th-percentile) latency, the None, Kyber, and MQ-Deadline I/O schedulers significantly reduce the tab switch latency compared to the default BFQ I/O scheduler, by 29%, 35%, and 18%, respectively. Second, the BFQ I/O scheduler provides the lowest tab switch latency for latency percentiles outside the tail. The None/Kyber/MQ-Deadline I/O schedulers slightly increase BFQ's tab switch latency by 3%/3%/%/10%, 4%/4%/4%, and 5%/4%/5% at the 50th-, 60th-, and 70th-percentile latencies, respectively. Third, when moving closer to the latency percentiles at the tail (i.e., from the 80th-percentile latency to the 95th-percentile latency), we observe that the None, Kyber, and MQ-Deadline I/O schedulers all provide lower tab switch latencies than the BFQ I/O scheduler. By employing the Kyber/MQ-Deadline I/O schedulers, the tab switch latency reduces by 3%/4% and 7%/6% compared to the BFQ I/O scheduler for the 80th- and 90th-percentile latencies. At the

95th-percentile latency, the None I/O scheduler reduces the tab switch latency by 12% compared to the BFQ I/O scheduler. Therefore, we conclude that even though the default BFQ I/O scheduler provides good performance overall (except at the tail), alternative I/O schedulers can greatly improve tail latency performance, thereby improving user experience.
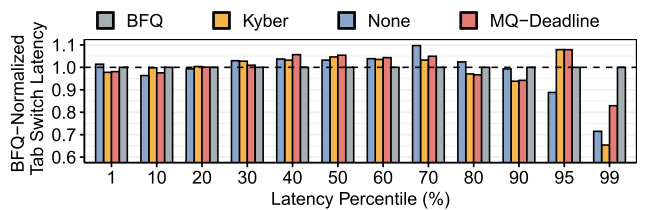

FIGURE 20. Normalized tab switch latency of four schedulers, categorized across different latency percentiles. Y-axis is normalized to the default BFQ I/O scheduler.

To further understand the I/O schedulers' impact on Chrome browser performance, we analyze the Q2C latency (i.e., the end-to-end I/O request latency) for each I/O scheduler. Figure 21 depicts the end-to-end I/O request (i.e., Q2C) latency percentiles (along the x-axis) and the corresponding Q2C latency (y-axis), normalized to the Q2C latency values for the BFQ I/O scheduler. We make three observations. First, the None I/O scheduler reduces Q2C latency in a majority of the latency percentiles. However, such a reduction does *not* directly translate to tab switch latency improvements (as seen in Figure 20). This happens because the None I/O scheduler does not enforce any ordering among read and write I/O requests. Since (i) Chrome mostly issues read I/O requests during the execution of our test (as Figure 17 shows) and (ii) the Intel Optane SSD internally handles read and write requests equally (as characterized by prior work [130]), the None I/O scheduler delays the execution of Chrome's read I/O operations by executing write I/O requests, which hurts Chrome's performance. Second, the Kyber I/O scheduler provides the best Q2C reduction for the 99th-percentile latency, which directly translates to better tab switch latency (Figure 20). The Kyber I/O scheduler improves Chrome's performance since it better fits Chrome's I/O request characteristics by: (i) reducing the average I/O queuing, merging, and scheduling latencies (i.e., Q2D latencies) by up to 3.7× that of the BFQ I/O scheduler (not shown); and (ii) prioritizing read I/O requests over writes. Third, even though Kyber's Q2C latency is larger than BFQ's Q2C latency (Figure 21) at the 90th-percentile latency, Kyber's Q2C latency for read requests is slightly lower (by 15%) than BFQ's Q2C latency for read requests (not shown), which translates to a faster tab switch latency for Kyber at the 90th-percentile latency. However, at the 95th-percentile latency, Kyber's Q2C latencies for both read and write requests are larger than BFQ's Q2C latencies. At the 99th-percentile latency, the system's high memory capacity pressure highlights the high overhead of the

BFQ I/O scheduler, leading to a significant increase in BFQ's `Q2C` latency for both read and write requests. BFQ's `Q2C` latency increases by 99% from the 95th-percentile latency to the 99-percentile latency. In contrast, Kyber's `Q2C` latency increases by only 19% when moving from the 95th-percentile latency to the 99-percentile latency. We conclude that the Kyber I/O scheduler can reduce tail latency for the Chrome browser since it (i) provides low overhead I/O scheduling decisions for already critical I/O requests while (ii) matching the access pattern of our workload by prioritizing read accesses over writes.
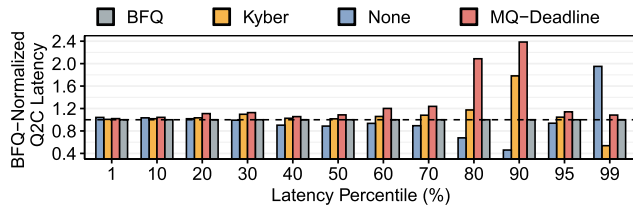

FIGURE 21. `Q2C` **latency of four I/O schedulers. Y-axis is normalized to the default BFQ I/O scheduler.**

Energy Analysis. Figure 22 compares the impact of the different I/O schedulers (x-axis) on the average memory subsystem energy consumption (which includes the energy consumption of main memory and swap space) for the Optane configuration (y-axis; normalized to the baseline BFQ I/O scheduler). We use the same energy model described in Section IV-A for our analysis. We make two key observations from the figure. First, we observe from the figure that all four I/O scheduler mechanisms achieve a *similar* memory subsystem energy consumption during the execution of our test. Second, the Kyber I/O scheduler slightly *increases* energy consumption by 6.9%, while the None I/O scheduler slightly *decreases* energy consumption by 6.3% compared to the baseline BFQ I/O scheduler. This is due to an increase in the number of write I/O requests the Kyber I/O scheduler produces compared to the None I/O scheduler. Recall that the Kyber I/O scheduler uses dedicated queues for read and write quest and dispatch write requests using a pre-defined threshold, while the None I/O scheduler uses a single queue for read and write requests and dispatch both read/write requests using a first-in-first-out approach. This leads to the Kyber I/O scheduler prioritizing more write requests from background kernel processes than the None I/O scheduler and the baseline BFQ mechanism. In contrast, the None I/O scheduler serves the processes that generate an I/O request first (in our case, Chrome processes issuing read I/O requests due to moderate-to-high swap activity). We conclude that since the available I/O schedulers mostly target improving the throughput of block I/O devices, they achieve a similar memory subsystem energy consumption.

We conclude that, we can reduce especially the block I/O tail latency by employing different I/O schedulers (e.g., Kyber) from the default Linux block I/O scheduler. However, the high latency overhead related to managing I/O
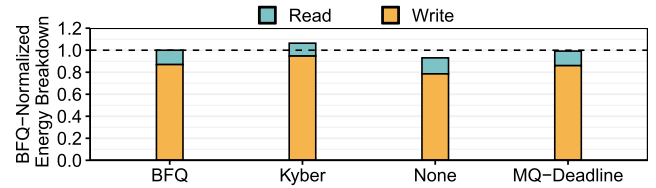

FIGURE 22. **Energy consumption: Optane with different I/O schedulers. Y-axis is normalized to the default BFQ I/O scheduler..**

requests is still large in comparison to the actual device time (as Figure 17 shows). Therefore, we evaluate a second optimization in Section V-D.

D. OPTIMIZATION 2: INTERRUPTS VS. POLLING BASED I/O REQUEST COMPLETION

Another key component of the Linux block I/O layer that directly impacts I/O performance is the I/O request completion mechanism. There are two main I/O request completion mechanisms in current Linux systems: (1) interrupt-based (i.e., IRQ-based) I/O request completion [10] and (2) polling-based I/O request completion [247]. Interrupt-based I/O request completion employs an asynchronous operation model. When a process issues a block I/O request, the OS puts the process to sleep and context switches to another process. When the I/O response arrives, the device driver receives an interrupt and wakes up the sender process. In contrast, polling-based I/O request completion employs a synchronous operation model. When a process issues a block I/O request, the process continuously polls in the CPU waiting for the I/O request to complete (i.e., instead of going to sleep, the process continually executes CPU instructions to check the current status of the I/O request).

While interrupt-based I/O request completion may incur large system overheads due to context switching, polling-based I/O request completion imposes a high CPU load to the system. Previous work [248] proposes a hybrid I/O request completion mechanism, which targets fast NVM devices. In this hybrid I/O request completion mechanism, when a process issues an I/O request, the OS puts the process to sleep, similar to the IRQ mode. However, to remove the context switch latency from the critical path of I/O request completion, the OS wakes up the process after some predefined sleep delay time `t`. Then, the process polls the I/O request queue for completion of its request until the response arrives from the block device. The hybrid I/O request completion mechanism can improve performance compared to polling since it reduces the number of CPU cycles spent on polling.

The hybrid I/O request completion mechanism works in two modes: (1) *fixed latency*, where the user sets the sleep delay time `t` to a specific latency; and (2) *adaptive latency*, where the OS dynamically sets the sleep delay time `t` by attempting to estimate when the I/O request will complete [249]. In the adaptive latency mode, the OS monitors the completion time of the different types of

I/O requests, and then utilizes half of the average of the I/O request completion time for a particular I/O request type as the sleep delay time t for future I/O requests of that type. Based on this estimation, the OS puts the process that issues I/O requests to sleep before entering a polling loop. The adaptive latency mode is enabled by setting t to 0.

We evaluate how different I/O request completion mechanisms impact the Chrome browser performance. The OS employs the interrupt-based I/O request completion mechanism by default. However, Intel recommends enabling the Hybrid technique when using the Intel Optane SSD for enterprise computing [250]. Previous work [248] advocates that fast NVM-based devices can benefit from the polling-based I/O request completion mechanism, since the context switch latency incurred during interrupt-based I/O request completion can be larger than the device access latency. We evaluate all three I/O request completion mechanisms to identify the best-performing one to use for Chrome with an Intel Optane SSD used as swap space. In the Hybrid I/O request competition mechanism, we evaluate the adaptive latency mode (by setting $t = 0$) and fixed latency mode, where we evaluate two values of t (2 µs and 4 µs).

Figure 23 shows the tab switch latency distribution (Figure 23a) and the distribution of high-latency tab switches for the three I/O completion mechanisms (Figure 23b) for the Optane configuration with Zswap enabled, when we employ three different I/O request completion mechanisms: (1) interrupt-based (IRQ); (2) polling-based (Polling); and (3) Hybrid with $t = 0$ (i.e., adaptive latency mode), $t = 2$ µs, and $t = 4$ µs. We make three observations. First, the interrupt-based mechanism provides the lowest average tab switch latency, with an average tab switch latency reduction of 60% compared to the Polling mechanism, which provides the highest average tab switch latency; and 11% compared to the Hybrid ($t = 2$) mechanism, which provides the second-best average tab switch latency (Figure 23a). Second, on average across tab counts, the interrupt-based I/O request completion mechanism leads to the lowest number of high-latency tab switches, with only 2.3% of tab switches being high latency, versus 3.9%/4.8%/7.8%/3.2% from the Polling, Hybrid ($t=0$), Hybrid ($t=2$), and Hybrid ($t=4$) mechanisms, respectively (Figure 23b). Third, we observe that the Polling mechanism eliminates high-latency tab switches when 11–30 tabs are opened. With only a small number of open Chrome tabs (1–10 tabs), Chrome issues few I/O requests due to the system's low swap traffic (as Figure 1 shows). In such case, the Polling mechanism increases the number of high-latency tab switches compared to the interrupt-based I/O completion mechanism since, in the event of an I/O request, the system cannot context switch to Chrome tab, which will likely not issue an I/O request, thus increasing tab switch latency. On the other hand, when the number of open Chrome tabs increases, and consequently the swap traffic (Figure 1), the system can sometimes leverage idle CPU time to wait for the completion of Chrome's I/O requests, reducing I/O request latency. We conclude that, on average, maintaining

the interrupt-based I/O request completion mechanism is still the best approach for consumer devices. However, enabling polling can sometimes be a good alternative.
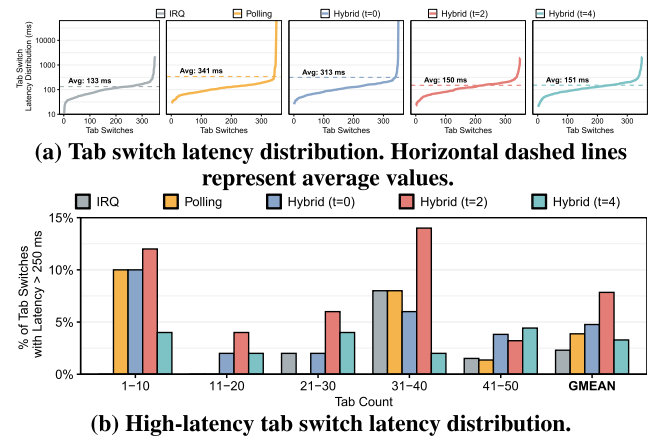

(a) Tab switch latency distribution. Horizontal dashed lines represent average values.

(b) High-latency tab switch latency distribution.

FIGURE 23. Tab switch latency: Optane with different I/O completion mechanisms.

Figure 24 shows the tab switch latency distribution for the three different I/O completion mechanisms. We make two observations. First, the interrupt-based (IRQ) I/O completion mechanism provides the lowest tab switch latency of the evaluated mechanisms up to the 95th-percentile latency. On average, it reduces tab switch latency by 12%, 14%, 20%, and 17% compared to the Polling, Hybrid ($t=0$), Hybrid ($t=2$), and Hybrid ($t=4$) mechanisms, respectively. Second, when examining the 99th-percentile latency, the Hybrid ($t=0$) mechanism reduces the tab switch latency by 7% compared to the IRQ mechanism. This happens because at high memory capacity pressure, most of the processes are waiting for I/O. As a result, the OS has few opportunities (if any) to switch in a process that can make forward progress, and there is therefore little to no performance cost in keeping the waiting processes awake to poll the CPU and eliminating the context switch overhead that they would incur with IRQ. We conclude that the Hybrid I/O request completion mechanism is a good solution to reduce tail latency for Chrome when using an Intel Optane SSD.

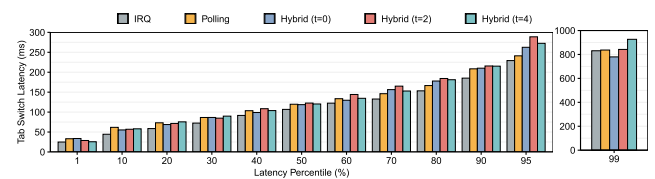**FIGURE 24.** Tab switch latency distribution: different I/O completion mechanisms.

Energy Analysis. Figure 25 compares the impact of the different I/O completion mechanisms (x-axis) on the average memory subsystem energy consumption for the Optane configuration (y-axis; normalized to the baseline IRQ completion mechanism). We use the same energy model described in Section IV-A for our analysis. The figure shows that the

different I/O completion mechanisms have little to no impact on the average memory subsystem energy consumption. This is because such mechanisms do *not* employ any optimization targeting the reduction of I/O traffic or prioritization of I/O requests.
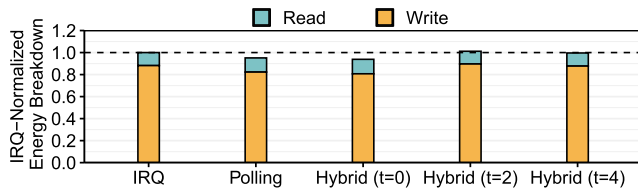

FIGURE 25. Energy consumption: Optane with different I/O completion mechanisms. Y-axis is normalized to the default IRQ I/O completion mechanism.

We conclude that (1) the interrupt-based I/O request completion mechanism provides the best average performance for the Chrome web browser, but (2) at the tail latency (i.e., 99th-percentile latency), the hybrid I/O request completion mechanism can further reduce I/O request latency. We, therefore, believe that there needs to be further research into new I/O request completion mechanisms that provide both the best average and tail performance.

VI. KEY TAKEAWAYS

To summarize, our experimental analysis reveals that extending the main memory space by using the Intel Optane SSD as NVM-based swap space for DRAM provides a cost-effective way to alleviate DRAM scalability issues. However, naively integrating the Intel Optane SSD into the system leads to several system-level overheads that can negatively impact overall performance and energy efficiency. We mitigate such overheads by examining and evaluating system optimizations driven by our analyses.

We provide the following six key takeaways from our empirical analyses:

1) *Effect of Intel Optane SSD as swap space (Section IV-A).* Reducing DRAM size and extending the main memory space with the Intel Optane SSD as swap space provides benefits for the Chrome browser, since it can (a) increase the number of open tabs, and (b) reduce system cost. However, it also leads to an increase in the number of tab switches with high latency compared to the baseline.

2) *Reducing tail latency by enabling Zswap (Section IV-B).* Zswap is a good mechanism to reduce I/O traffic introduced by the Intel Optane SSD, at the cost of a small increase in tab switch latency at large tab counts. The Zswap cache reduces system energy by $2\times$ (compared to the Intel Optane SSD without Zswap enabled), at the cost of increasing the high-latency tab switches by 4% and reducing the number of open tabs by 12%.

3) *Effect of using different NVM devices (Section IV-C).* A state-of-the-art NAND-flash-based SSD provides benefits over both the baseline and the Intel Optane SSD. Importantly, it enables more Chrome tabs to

be open. These benefits come due to the larger effective main memory capacity provided by the state-of-the-art NAND-flash-based SSD over the baseline configuration. Unfortunately, these benefits come at the cost of higher tab switch latencies, compared to both the baseline and Optane configurations, due to the much longer device latencies of NAND flash memory. These large tab switch latencies degrade user experience. Taking both performance and user experience into account, emerging NVM-based SSDs such as the Intel Optane SSD are quite promising to employ in consumer devices, providing performance benefits without the undesirable user experience trade-offs incurred by NAND-flash-based SSDs.

4) *System bottlenecks caused by NVMs (Section V-A).* The Linux block I/O layer is a key system bottleneck when the Intel Optane SSD is used as swap space. We can mitigate some of the overheads caused by the block I/O layer by (a) employing an I/O scheduler that meets the requirements of the application's access pattern and (b) using different I/O request completion mechanisms.

5) *Optimization 1: block I/O schedulers (Section V-C).* We can reduce tab switch latency by changing the default BFQ I/O scheduler in the system that uses the Intel Optane SSD as swap space. We reduce 95th- and 99th-percentile latencies by employing the None and the Kyber I/O schedulers, respectively, as those I/O schedulers reduce I/O scheduling overheads and fit the I/O access pattern of the Chrome web browser.

6) *Optimization 2: interrupt- vs. polling-based I/O request completion (Section V-D).* On average, the interrupt-based I/O request completion mechanism provides the best performance for the system with the Intel Optane SSD device. However, the Hybrid I/O request completion mechanism can help reduce 99th-percentile latency for block I/O requests.

Based on our analysis, we conclude that there is a large optimization space to be explore in order to *efficiently* adopt emerging NVMs in consumer devices. For example, we believe that one of the main issues the system suffers from when executing interactive workloads is that scheduling decisions made by the OS do not consider the response time expected by the workload. Exposing such information to the OS could reduce tail latency and allow the scheduler to take action according to the needs of a particular workload (e.g., by prioritizing the workload with the shorter or more urgent response deadlines). We leave the design, implementation, and evaluation of such ideas for future work.

A. OVERALL LIMITATIONS OF THE TECHNOLOGY

Even though employing the Intel Optane SSD as a swap space can lead to several benefits in terms of cost and performance, it can also impact overall system energy consumption and lifetime. We provide the following two key takeaways from our empirical analyses that highlight the limitations of NVM-based swap space in consumer devices:

1) *Effect of Intel Optane SSD as swap space on energy consumption.* Integrating Intel Optane SSD as a swap space increases average memory subsystem energy consumption to 69.5× that of the baseline ZRAM-based system configuration (Section IV-A; Figure 6). This happens due to the higher swap activity enabled by the Optane-based swap space (Section IV-A). Such an increase in energy consumption can be mitigated by employing a Zswap cache, which reduces the increase caused by the Optane-based swap space to 34.75× that of the baseline (Section IV-B; Figure 12). Unfortunately, tuning the block I/O scheduler (Section V-C) and I/O completion mechanism (Section V-D) do *not* lead to significant energy savings for the Optane configuration, since such optimizations primarily target improving the throughput and latency of I/O operations, rather than reducing energy consumption.

2) *Effect of Intel Optane SSD as swap space on system lifetime.* The Intel Optane SSD, as an NVM-based device, suffers from limited write endurance, which can impact the lifetime of the system. Based on our analysis (Section IV-B), we observe that it would take an Optane-based system (without Zswap) running our Chrome web browser 4.5 years to experience a write-endurance failure. Enabling Zswap increases the lifetime of the Optane-based system to 8.3 years.

Many prior works [51], [56], [60], [79], [228], [229], [232], [233], [235], [251], [252], [253], [254], [255], [256], [257], [258], [259] aim to reduce the impact of emerging NVMs on overall system energy consumption and lifetime. The great majority of such works aim to (i) reduce the number of write operations the system issue to the NVM device using techniques such as caching [51], [79], write-aware data mapping and data allocation algorithms [60], [79], [229], and data compression [228]; and (ii) distribute write operations across NVM cells using diverse wear-leveling techniques [56], [232], [233], [235], [251], [253], [254], [255], [256], [257], [259]. We believe such approaches can be employed to mitigate the limitations of NVMs in consumer devices. We leave such analyses for future work.

VII. RELATED WORK

To our knowledge, this is the first work that (i) comprehensively analyzes the impact of extending the main memory space of consumer devices using *real* off-the-shelf emerging NVM-based SSDs, and (ii) proposes practical system-level optimizations that can mitigate the tail latency of interactive workloads when employing emerging NVM-based SSDs in the system. We discuss the large body of related work on NVM using four broad categories.

1) ENABLING NVM-BASED SWAP SPACE FOR MOBILE DEVICES

Several past works [131], [132], [133], [134], [135], [136], [137] investigate how to efficiently enable swap-based NVMs for mobile devices. Unlike our work, these past works do not utilize *real* NVM devices to evaluate their mechanisms or their system-level implications on a *real* mobile system. Thus, it is not fully clear if their results and insights can be easily translated to a real system employing a real NVM device. We briefly describe the key mechanisms proposed by each of these works.

Two prior works [131], [133] propose to improve swap performance and the lifetime of byte-addressable NVM devices being used as swap space in smartphones. These works emulate swapping behavior by creating a swap area inside DRAM (similar to our ZRAM configuration). CAUSE [132] is a hybrid memory architecture for mobile devices that leverages application access patterns to allocate memory either in DRAM or in NVM within a hybrid DRAM-NVM memory architecture. Similarly, Kim et al. [135] employ an NVM-based swap space for Android devices, which leverages hot/cold data to manage swap activity between DRAM and NVM. Zhong et al. [134] aim to reduce write endurance issues related to NVM devices by identifying and swapping cold pages from DRAM to NVM-based swap space in smartphones. SmartSwap [136] predicts the most-rarely-used applications to be dynamically swapped to a flash-memory-based swap space ahead of time. Kim et al. [137] compare two swap space organizations for mobile devices: (1) a hierarchical swap architecture, where NVM-based swap space is used as a cache for a larger flash-memory-based swap space; and (2) a hybrid swap architecture, where both NVM and flash devices are used as a single-level swap space. As part of this work, the authors propose SPP-CLOCK [137], a mechanism to identify hot/cold data to manage swap activity.

We believe that many of the mechanisms proposed by these prior works can be adapted to be employed in our system to further improve performance and lifetime. We leave such studies to future works.

2) IMPROVING BLOCK I/O LATENCY FOR FAST NVME DEVICES

Previous works [100], [241], [260], [261], [262], [263], [264], [265] propose several techniques to mitigate block I/O latencies for fast NVMe devices. These techniques include software [100], [261], [262], [263], [264], [265] and hardware solutions [241], [260] to provide lower I/O access latency [100], [263], [264], page fault handling [260], and I/O scheduling [241], [261], [265]. Even though these techniques are promising solutions to reduce the high block I/O latencies, they require substantial changes in the hardware and the software stack, which are outside the scope of this work, but can also be used in our proposed system.

Another body of work [59], [266], [267], [268], [269], [270], [271], [272], [273] aims to completely remove the block I/O layer from the system by providing programming models that enable the user to directly access data from fast NVMe devices. Even though this is a promising solution, it involves several challenges such as code refactoring and security, which can be a promising direction for future work.

3) REAL NVM DEVICES IN REAL SYSTEMS

Since the release of the Intel Optane SSD, various works [101], [102], [105], [125], [126], [127], [129], [274], [275] have experimentally shown that the Intel Optane SSD can improve performance, energy, and cost for different work-loads (e.g., databases [101], [105], [275], high-performance computing [274], key-value stores [127], [129], machine learning [102], [125], query processing [126]). Our work differs from these works since we (i) target a different family of workloads (i.e., interactive consumer workloads, and in particular the Google Chrome web browser) and (ii) employ the Intel Optane SSD as an extension of main memory, instead of as a separate storage device.

4) HYBRID DRAM-NVM MEMORY SYSTEMS

A large body of works [51], [52], [53], [54], [55], [56], [80], [93], [94], [95], [96], [114], [276], [277], [278], [279] propose to use NVMs as an alternative technology to DRAM, where the NVM completely replaces DRAM as the main memory device [51], [52], [53], [54], [55], [56], or is incorporated into the memory hierarchy alongside DRAM to create a hybrid DRAM–NVM memory system [80], [93], [94], [95], [96], [114], [277], [279]. Unlike our work, these works either (1) use NVMs as part of main memory, and not as swap space, which increases the complexity of the memory architecture; or (2) mainly leverage simulation infrastructures to evaluate their proposals, and thus, do not examine *real* NVM devices and their implications on real systems with real measurement data.

VIII. CONCLUSION

In this paper, we comprehensively evaluate the performance implications of leveraging real emerging NVMs as an extension of main memory space in real consumer devices, while targeting interactive workloads. We employ a state-of-the-art NVM-based SSD device (i.e., the Intel Optane SSD) as swap space for DRAM, which increases the effective main memory capacity in our system. We observe that using the Intel Optane SSD can improve the average and tail latency performance of the Chrome web browser, compared to a baseline system with double the amount of DRAM, and to a system where a state-of-the-art NAND-flash-based SSD is used for the swap space. We identify that the Linux block I/O layer becomes a major source of performance overhead when the main memory space is extended using NVM, primarily due to (i) I/O scheduling bottlenecks; and (ii) overheads related to the asynchronous operation of the I/O request completion mechanism. We mitigate some of these overheads by proposing two system optimizations that can better leverage the characteristics of our workloads and the NVM. We also evaluate the limitations of real emerging NVMs in consumer devices and conclude that real systems need to employ solutions to mitigate the issues associated with energy increase and lifetime degradation NVM devices introduce. We conclude that emerging NVMs are a cost-effective solution to alleviate the DRAM capacity bottleneck in consumer devices. We hope that the results of our study can inspire and drive novel hardware and software optimizations in future NVM-based computing systems.

REFERENCES

[1] Google LLC. *Chromebook*. [Online]. Available: https://www.google.com/chromebook/

[2] *Slowing Growth Ahead for Worldwide Internet Audience*, eMarketer, New York, NY, USA, 2016.

[3] V. J. Reddi, H. Yoon, and A. Knies, "Two billion devices and counting," *IEEE Micro*, vol. 38, no. 1, pp. 6–21, Jan. 2018.

[4] Arm Holdings plc and Qualcomm Incorporated, "Enabling the next mobile computing revolution with highly integrated ARMv8-A based SoCs," ARM Qualcomm, White Paper, 2014.

[5] M. Halpern, Y. Zhu, and V. J. Reddi, "Mobile CPU's rise to power: Quantifying the impact of generational mobile CPU design trends on performance, energy, and user satisfaction," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Mar. 2016.

[6] Canalys. (2021). *Chromebooks Lead PC Revival in Q1 2021 With 275% Growth*. [Online]. Available: https://rb.gy/jm7xu

[7] B. Heater, "As chromebook sales soar in schools, apple and Microsoft fight back," TechCrunch, 2017. [Online]. Available: https://techcrunch.com/2017/04/27/as-chromebook-sales-soar-in-schools-apple-and-microsoft-fight-back/

[8] R. H. Dennard, "Field-effect transistor memory," U.S. Patent 3 387 286, Jun. 4, 1968.

[9] Y. Kim and O. Mutlu, "Memory systems," in *Computing Handbook: Computer Science and Software Engineering*, 3rd ed. Abingdon, U.K.: Taylor & Francis, 2014.

[10] D. P. Bovet and M. Cesati, *Understanding the Linux Kernel: From I/O Ports to Process Management*, 3rd ed. Sebastopol, CA, USA: O'Reilly Media, 2005.

[11] A. S. Tanenbaum and A. S. Woodhull, *Operating Systems: Design and Implementation*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1997.

[12] O. Mutlu. (2020). *Lecture Notes for Digital Design and Computer Architecture—Lecture 23b: Virtual Memory*. [Online]. Available: https://rb.gy/qzj7r

[13] M. Badr, C. Delconte, I. Edo, R. Jagtap, M. Andreozzi, and N. E. Jerger, "Mocktails: Capturing the memory behaviour of proprietary mobile architectures," in *Proc. ACM/IEEE 47th Annu. Int. Symp. Comput. Archit. (ISCA)*, May 2020.

[14] A. Boroumand, S. Ghose, Y. Kim, R. Ausavarungnirun, E. Shiu, R. Thakur, D. Kim, A. Kuusela, A. Knies, P. Ranganathan, and O. Mutlu, "Google workloads for consumer devices: Mitigating data movement bottlenecks," in *Proc. ASPLOS*, 2018.

[15] J. Mohan, D. Purohith, M. Halpern, V. Chidambaram, and V. J. Reddi, "Storage on your smartphone uses more energy than you think," in *Proc. HotStorage*, 2017.

[16] A. Boroumand, "Practical mechanisms for reducing processor-memory data movement in modern workloads," Ph.D. dissertation, Dept. Elect. Comput. Eng., Carnegie Mellon Univ., Pittsburgh, PA, USA, 2020.

[17] R. Nelson. (2017). *The Size of iPhone's Top Apps Has Increased by 1,000% in Four Years*. [Online]. Available: https://sensortower.com/blog/ios-app-size-growth

[18] N. Lebeck, A. Krishnamurthy, H. M. Levy, and I. Zhang, "End the senseless killing: Improving memory management for mobile operating systems," in *Proc. USENIX ATC*, 2020.

[19] O. Mutlu, "Memory scaling: A systems architecture perspective," in *Proc. 5th IEEE Int. Memory Workshop*, May 2013.

[20] O. Mutlu and L. Subramanian, "Research problems and opportunities in memory systems," *Supercomput. Frontiers Innov.*, vol. 1, no. 3, pp. 19–55, 2015.

[21] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors," in *Proc. ACM/IEEE 41st Int. Symp. Comput. Archit. (ISCA)*, Jun. 2014.

[22] O. Mutlu and J. S. Kim, "RowHammer: A retrospective," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 8, pp. 1555–1571, Aug. 2020.

[23] J. S. Kim, M. Patel, A. G. Yaglikci, H. Hassan, R. Azizi, L. Orosa, and O. Mutlu, "Revisiting RowHammer: An experimental analysis of modern DRAM devices and mitigation techniques," in *Proc. ACM/IEEE 47th Annu. Int. Symp. Comput. Archit. (ISCA)*, May 2020.

[24] O. Mutlu, "Main memory scaling: Challenges and solution directions," in *More Than Moore Technologies for Next Generation Computer Design*. New York, NY, USA: Springer, 2015.

[25] U. Kang, H.-S. Yu, C. Park, H. Zheng, J. Halbert, K. Bains, S. Jang, and J. S. Choi, "Co-architecting controllers and DRAM to enhance DRAM process scaling," Memory Forum, 2014.

[26] S. Hong, "Memory technology trend and future challenges," in *IEDM Tech. Dig.*, Dec. 2010.

[27] S. Kanev, J. P. Darago, K. Hazelwood, P. Ranganathan, T. Moseley, G.-Y. Wei, and D. Brooks, "Profiling a warehouse-scale computer," in *Proc. 42nd Annu. Int. Symp. Comput. Archit.*, Jun. 2015.

[28] O. Mutlu, "The RowHammer problem and other issues we may face as memory becomes denser," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017.

[29] S. Ghose, A. G. Yaglikci, R. Gupta, D. Lee, K. Kudrolli, W. X. Liu, H. Hassan, K. K. Chang, N. Chatterjee, A. Agrawal, and M. O'Connor, "What your DRAM power models are not telling you: Lessons from a detailed experimental study," in *Proc. SIGMETRICS*, 2018.

[30] J. Liu, B. Jaiyen, Y. Kim, C. Wilkerson, and O. Mutlu, "An experimental study of data retention behavior in modern DRAM devices: Implications for retention time profiling mechanisms," in *Proc. 40th Annu. Int. Symp. Comput. Archit.*, Jun. 2013.

[31] P. Frigo, E. Vannacc, H. Hassan, V. V. der Veen, O. Mutlu, C. Giuffrida, H. Bos, and K. Razavi, "TRRespass: Exploiting the many sides of target row refresh," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2020.

[32] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, "RAIDR: Retention-aware intelligent DRAM refresh," in *Proc. 39th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2012.

[33] M. Patel, J. S. Kim, and O. Mutlu, "The reach profiler (REAPER): Enabling the mitigation of DRAM retention failures via profiling at aggressive conditions," in *Proc. 44th Annu. Int. Symp. Comput. Archit.*, Jun. 2017.

[34] M. K. Qureshi, D.-H. Kim, S. Khan, P. J. Nair, and O. Mutlu, "AVATAR: A variable-retention-time (VRT) aware refresh for DRAM systems," in *Proc. 45th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, Jun. 2015.

[35] J. A. Mandelman, R. H. Dennard, G. B. Bronner, J. K. DeBrosse, R. Divakaruni, Y. Li, and C. J. Radens, "Challenges and future directions for the scaling of dynamic random-access memory (DRAM)," *IBM J. Res. Develop.*, vol. 46, no. 2.3, pp. 187–212, Mar. 2002.

[36] S. Khan, D. Lee, Y. Kim, A. R. Alameldeen, C. Wilkerson, and O. Mutlu, "The efficacy of error mitigation techniques for DRAM retention failures: A comparative experimental study," in *Proc. SIGMETRICS*, Jun. 2014.

[37] S. Khan, D. Lee, and O. Mutlu, "PARBOR: An efficient system-level technique to detect data-dependent failures in DRAM," in *Proc. 46th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2016.

[38] S. Khan, C. Wilkerson, Z. Wang, A. R. Alameldeen, D. Lee, and O. Mutlu, "Detecting and mitigating data-dependent DRAM failures by exploiting current memory content," in *Proc. 50th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2017.

[39] D. Lee, Y. Kim, G. Pekhimenko, S. Khan, V. Seshadri, K. Chang, and O. Mutlu, "Adaptive-latency DRAM: Optimizing DRAM timing for the common-case," in *Proc. IEEE 21st Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2015.

[40] D. Lee, S. Khan, L. Subramanian, S. Ghose, R. Ausavarungnirun, G. Pekhimenko, V. Seshadri, and O. Mutlu, "Design-induced latency variation in modern DRAM chips: Characterization, analysis, and latency reduction mechanisms," in *Proc. ACM SIGMETRICS/Int. Conf. Meas. Model. Comput. Syst.*, Jun. 2017.

[41] K. K. Chang, "Understanding and improving the latency of DRAM-based memory systems," Ph.D. dissertation, Dept. Elect. Comput. Eng., Carnegie Mellon Univ., Pittsburgh, PA, USA, 2017.

[42] K. K. Chang, A. G. Yaglikci, S. Ghose, A. Agrawal, N. Chatterjee, A. Kashyap, D. Lee, M. O'Connor, H. Hassan, and O. Mutlu, "Understanding reduced-voltage operation in modern DRAM devices: Experimental characterization, analysis, and mechanisms," in *Proc. SIGMETRICS*, 2017.

[43] K. K. Chang, A. Kashyap, H. Hassan, S. Ghose, K. Hsieh, D. Lee, T. Li, G. Pekhimenko, S. Khan, and O. Mutlu, "Understanding latency variation in modern DRAM chips: Experimental characterization, analysis, and optimization," in *Proc. SIGMETRICS*, 2016.

[44] K. K.-W. Chang, D. Lee, Z. Chishti, A. R. Alameldeen, C. Wilkerson, Y. Kim, and O. Mutlu, "Improving DRAM performance by parallelizing refreshes with accesses," in *Proc. IEEE 20th Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2014.

[45] J. Meza, Q. Wu, S. Kumar, and O. Mutlu, "Revisiting memory errors in large-scale production data centers: Analysis and modeling of new trends from the field," in *Proc. 45th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, Jun. 2015.

[46] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu, "Memory power management via dynamic voltage/frequency scaling," in *Proc. 8th ACM Int. Conf. Autonomic Comput.*, Jun. 2011.

[47] Q. Deng, D. Meisner, L. Ramos, T. F. Wenisch, and R. Bianchini, "MemScale: Active low-power modes for main memory," in *Proc. ASPLOS*, 2011.

[48] A. G. Yağlıkçı, H. Luo, G. F. De Oliveira, A. Olgun, M. Patel, J. Park, H. Hassan, J. S. Kim, L. Orosa, and O. Mutlu, "Understanding RowHammer under reduced wordline voltage: An experimental study using real DRAM devices," in *Proc. DSN*, 2022.

[49] L. Orosa, A. G. Yaglikci, H. Luo, A. Olgun, J. Park, H. Hassan, M. Patel, J. S. Kim, and O. Mutlu, "A deeper look into RowHammer's sensitivities: Experimental analysis of real DRAM chips and implications on future attacks and defenses," in *Proc. 54th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2021.

[50] H. Hassan, Y. C. Tugrul, J. S. Kim, V. Van der Veen, K. Razavi, and O. Mutlu, "Uncovering in-DRAM RowHammer protection mechanisms: A new methodology, custom RowHammer patterns, and implications," in *Proc. MICRO*, 2021.

[51] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable dram alternative," in *Proc. 36th Annu. Int. Symp. Comput. Archit.*, Jun. 2009.

[52] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," in *Proc. 36th Annu. Int. Symp. Comput. Archit.*, Jun. 2009.

[53] B. C. Lee, P. Zhou, J. Yang, Y. Zhang, B. Zhao, E. Ipek, O. Mutlu, and D. Burger, "Phase-change technology and the future of main memory," *IEEE Micro*, vol. 30, no. 1, p. 143, Jan. 2010.

[54] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Phase change memory architecture and the quest for scalability," *Commun. ACM*, vol. 53, no. 7, pp. 99–106, Jul. 2010.

[55] E. Kultursay, M. Kandemir, A. Sivasubramaniam, and O. Mutlu, "Evaluating STT-RAM as an energy-efficient main memory alternative," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Apr. 2013.

[56] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," in *Proc. 36th Annu. Int. Symp. Comput. Archit.*, Jun. 2009.

[57] H.-S. P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson, "Phase change memory," *Proc. IEEE*, vol. 98, no. 12, pp. 2201–2227, Oct. 2010.

[58] J. Meza, J. Li, and O. Mutlu, "A case for small row buffers in non-volatile main memories," in *Proc. IEEE 30th Int. Conf. Comput. Design (ICCD)*, Sep. 2012.

[59] 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. WEED*, 2013.

[60] S. Song, A. Das, O. Mutlu, and N. Kandasamy, "Improving phase change memory performance with data content aware access," in *Proc. ACM SIGPLAN Int. Symp. Memory Manage.*, Jun. 2020.

[61] S. Song, A. Das, O. Mutlu, and N. Kandasamy, "Aging-aware request scheduling for non-volatile main memory," in *Proc. 26th Asia South Pacific Design Autom. Conf.*, Jan. 2021.

[62] S. Song, A. Das, O. Mutlu, and N. Kandasamy, "Enabling and exploiting partition-level parallelism (PALP) in phase change memories," *ACM Trans. Embedded Comput. Syst.*, vol. 18, no. 5s, pp. 1–25, Oct. 2019.

[63] G. Atwood, "PCM applications and an outlook to the future," in *Phase Change Memory: Device Physics, Reliability and Applications*. New York, NY, USA: Springer, 2017.

[64] S. Bock, B. Childers, R. Melhem, D. Mosse, and Y. Zhang, "Analyzing the impact of useless write-backs on the endurance and energy consumption of PCM main memory," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (IEEE ISPASS)*, Apr. 2011.

[65] G. W. Burr, B. N. Kurdi, J. C. Scott, C. H. Lam, K. Gopalakrishnan, and R. S. Shenoy, "Overview of candidate device technologies for storage-class memory," *IBM J. Res. Develop.*, vol. 52, no. 4.5, pp. 449–464, Jul. 2008.

[66] Y. Du, M. Zhou, B. R. Childers, D. Mossé, and R. Melhem, "Bit mapping for balanced PCM cell programming," in *Proc. 40th Annu. Int. Symp. Comput. Archit.*, Jun. 2013.

[67] A. P. Ferreira, M. Zhou, S. Bock, B. Childers, R. Melhem, and D. Mosse, "Increasing PCM main memory lifetime," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2010.

[68] L. Jiang, Y. Zhang, B. R. Childers, and J. Yang, "FPB: Fine-grained power budgeting to improve write throughput of multi-level cell phase change memory," in *Proc. 45th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2012.

[69] L. Jiang, Y. Du, B. Zhao, Y. Zhang, B. R. Childers, and J. Yang, "Hardware-assisted cooperative integration of wear-leveling and sal-vaging for phase change memory," *ACM Trans. Archit. Code Optim.*, vol. 10, no. 2, pp. 1–25, May 2013.

[70] S. Kannan, M. Qureshi, A. Gavrilovska, and K. Schwan, "Energy aware persistence: Reducing energy overheads of memory-based persistence in NVMs," in *Proc. Int. Conf. Parallel Architectures Compilation*, Sep. 2016.

[71] M. K. Qureshi, "Pay-as-you-go: Low-overhead hard-error correction for phase change memories," in *Proc. 44th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2011.

[72] M. K. Qureshi, M. M. Franceschini, and L. A. Lastras-Montano, "Improving read performance of phase change memories via write cancellation and write pausing," in *Proc. 16th Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Jan. 2010.

[73] M. K. Qureshi, M. M. Franceschini, L. A. Lastras-Montaño, and J. P. Karidis, "Morphable memory system: A robust architecture for exploiting multi-level phase change memories," in *Proc. 37th Annu. Int. Symp. Comput. Archit.*, Jun. 2010.

[74] A. Sebastian, T. Tuma, N. Papandreou, M. Le Gallo, T. Kull, T. Parnell, and E. Eleftheriou, "Temporal correlation detection using computational phase-change memory," *Nature Commun.*, vol. 8, no. 1, Oct. 2017.

[75] R. Wang, L. Jiang, Y. Zhang, L. Wang, and J. Yang, "Exploit imbalanced cell writes to mitigate write disturbance in dense phase change memory," in *Proc. 52nd Annu. Design Autom. Conf.*, Jun. 2015.

[76] J. Yue and Y. Zhu, "Accelerating write by exploiting PCM asymmetries," in *Proc. IEEE 19th Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2013.

[77] M. Zhou, Y. Du, B. Childers, R. Melhem, and D. Mossé, "Writeback-aware partitioning and replacement for last-level caches in phase change main memory systems," *ACM Trans. Archit. Code Optim.*, vol. 8, no. 4, pp. 1–21, Jan. 2012.

[78] M. Zhou, Y. Du, B. R. Childers, R. Melhem, and D. Mosse, "Writeback-aware bandwidth partitioning for multi-core systems with PCM," in *Proc. 22nd Int. Conf. Parallel Architectures Compilation Techn.*, Sep. 2013.

[79] H. Yoon, N. Muralimanohar, J. Meza, O. Mutlu, and N. P. Jouppi, "Techniques for data mapping and buffering to exploit asymmetry in multi-level cell (phase change) memory," SAFARI Res. Group, Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. TR-SAFARI-2013-002, 2013.

[80] G. Dhiman, R. Ayoub, and T. Rosing, "PDRAM: A hybrid PRAM and DRAM main memory system," in *Proc. 46th Annu. Design Autom. Conf.*, Jul. 2009.

[81] K. L. Wang, J. G. Alzate, and P. Khalili Amiri, "Low-power non-volatile spintronic memory: STT-RAM and beyond," *J. Phys. D, Appl. Phys.*, vol. 46, no. 7, Feb. 2013, Art. no. 074003.

[82] E. Chen, D. Apalkov, Z. Diao, A. Driskill-Smith, D. Druist, D. Lottis, V. Nikitin, X. Tang, S. Watts, S. Wang, and S. A. Wolf, "Advances and future prospects of spin-transfer torque random access memory," *IEEE Trans. Magn.*, vol. 46, no. 6, pp. 1873–1878, Jun. 2010.

[83] Z. Diao, Z. Li, S. Wang, Y. Ding, A. Panchula, E. Chen, L.-C. Wang, and Y. Huai, "Spin-transfer torque switching in magnetic tunnel junctions and spin-transfer torque random access memory," *J. Phys., Condens. Matter*, vol. 19, no. 16, Apr. 2007, Art. no. 165209.

[84] M. Hosomi, H. Yamagishi, T. Yamamoto, K. Bessho, Y. Higo, K. Yamane, H. Yamada, M. Shoji, H. Hachino, C. Fukumoto, H. Nagao, and H. Kano, "A novel nonvolatile memory with spin torque transfer magnetization switching: Spin-ram," in *IEDM Tech. Dig.*, 2005.

[85] A. Raychowdhury, D. Somasekhar, T. Karnik, and V. De, "Design space and scalability exploration of 1T-1STT MTJ memory arrays in the presence of variability and disturbances," in *IEDM Tech. Dig.*, Dec. 2009.

[86] H. Akinaga and H. Shima, "Resistive random access memory (ReRAM) based on metal oxides," *Proc. IEEE*, vol. 98, no. 12, pp. 2237–2251, Dec. 2010.

[87] H.-S. P. Wong, H.-Y. Lee, S. Yu, Y.-S. Chen, Y. Wu, P.-S. Chen, B. Lee, F. T. Chen, and M.-J. Tsai, "Metal–oxide RRAM," *Proc. IEEE*, vol. 100, no. 6, pp. 1951–1970, Jun. 2012.

[88] J. J. Yang, D. B. Strukov, and D. R. Stewart, "Memristive devices for computing," *Nature Nanotechnol.*, vol. 8, pp. 13–24, Jan. 2013.

[89] M. Kund, G. Beitel, C.-U. Pinnow, T. Rohr, J. Schumann, R. Symanczyk, K. Ufert, and G. Müller, "Conductive bridging RAM (CBRAM): An emerging non-volatile memory technology scalable to sub 20 nm," in *IEDM Tech. Dig.*, 2005.

[90] D. Bondurant, "Ferroelectric ram memory family for critical data storage," *Ferroelectrics*, vol. 112, no. 1, pp. 273–282, Dec. 1990.

[91] B. Harris and N. Altiparmak, "Ultra-low latency SSDs' impact on overall energy efficiency," in *Proc. HotStorage*, 2020.

[92] K. Wu, Z. Guo, G. Hu, K. Tu, R. Alagappan, R. Sen, K. Park, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "The storage hierarchy is not a hierarchy: Optimizing caching on modern storage devices with orthus," in *Proc. FAST*, 2021.

[93] C. Wang, H. Cui, T. Cao, J. Zigman, H. Volos, O. Mutlu, F. Lv, X. Feng, and G. H. Xu, "Panthera: Holistic memory management for big data processing over hybrid memories," in *Proc. 40th ACM SIGPLAN Conf. Program. Lang. Design Implement.*, Jun. 2019.

[94] R. Salkhordeh, O. Mutlu, and H. Asadi, "An analytical model for performance and lifetime estimation of hybrid DRAM-NVM main memories," *IEEE Trans. Comput.*, vol. 68, no. 8, pp. 1114–1130, Mar. 2019.

[95] H. Yoon, J. Meza, R. Ausavarungnirun, R. A. Harding, and O. Mutlu, "Row buffer locality aware caching policies for hybrid memories," in *Proc. IEEE 30th Int. Conf. Comput. Design (ICCD)*, Sep. 2012.

[96] J. Meza, J. Chang, H. Yoon, O. Mutlu, and P. Ranganathan, "Enabling efficient and scalable hybrid memories using fine-granularity DRAM cache management," *IEEE Comput. Archit. Lett.*, vol. 11, no. 2, pp. 61–64, Jul./Dec. 2012.

[97] Intel Corporation. *Intel Optane Memory H10 With Solid State Storage*. [Online]. Available: https://rb.gy/f682j

[98] J. Izraelevitz, J. Yang, L. Zhang, J. Kim, X. Liu, A. Memaripour, Y. J. Soh, Z. Wang, Y. Xu, S. R. Dulloor, J. Zhao, and S. Swanson, "Basic performance measurements of the Intel optane DC persistent memory module," 2019, *arXiv:1903.05714*.

[99] G. Psaropoulos, I. Oukid, T. Legler, N. May, and A. Ailamaki, "Bridging the latency gap between NVM and DRAM for latency-bound operations," in *Proc. 15th Int. Workshop Data Manage. New Hardw.*, Jul. 2019.

[100] 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 *Proc. USENIX ATC*, 2019.

[101] J. Zhang, P. Li, B. Liu, T. G. Marbach, X. Liu, and G. Wang, "Performance analysis of 3D XPoint SSDs in virtualized and non-virtualized environments," in *Proc. IEEE 24th Int. Conf. Parallel Distrib. Syst. (ICPADS)*, Dec. 2018.

[102] S. W. D. Chien, S. Markidis, C. P. Sishtla, L. Santos, P. Herman, S. Narasimhamurthy, and E. Laure, "Characterizing deep-learning I/O workloads in TensorFlow," in *Proc. IEEE/ACM 3rd Int. Workshop Parallel Data Storage Data Intensive Scalable Comput. Syst. (PDSW-DISCS)*, Nov. 2018.

[103] J. Yang, B. Li, and D. J. Lilja, "Exploring performance characteristics of the optane 3D xpoint storage technology," *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 5, no. 1, pp. 1–28, Mar. 2020.

[104] F. T. Hady, A. Foong, B. Veal, and D. Williams, "Platform storage performance with 3D XPoint technology," *Proc. IEEE*, vol. 105, no. 9, pp. 1822–1833, Sep. 2017.

[105] K. Wu, A. Arpaci-Dusseau, R. Arpaci-Dusseau, R. Sen, and K. Park, "Exploiting Intel optane SSD for Microsoft SQL server," in *Proc. 15th Int. Workshop Data Manage. New Hardw.*, Jul. 2019.

[106] S. Imamura and E. Yoshida, "Reducing CPU power consumption for low-latency SSDs," in *Proc. IEEE 7th Non-Volatile Memory Syst. Appl. Symp. (NVMSA)*, Aug. 2018.

[107] Amazon.com, Inc. *Intel Optane Memory Module 16 GB M.2 80 mm PCIe 3.0 20 nm 3D XPoint MEMPEK1W016GA*. [Online]. Available: https://amzn.to/33Z6bws

[108] DRAMeXchange. *World Leading DRAM and NAND Flash Market Research Firm, With More Than a Decade of Most Authoritative Database*. [Online]. Available: https://www.dramexchange.com/

[109] I. B. Peng, M. B. Gokhale, and E. W. Green, "System evaluation of the Intel optane byte-addressable NVM," in *Proc. MemSys*, 2019.

[110] B. Metzler and A. Trivedi, "Prototyping byte-addressable NVM access," in *Proc. OpenFabrics Developers Workshop*, 2015.

[111] A. Hassan, H. Vandierendonck, and D. S. Nikolopoulos, "Energy-efficient hybrid DRAM/NVM main memory," in *Proc. Int. Conf. Parallel Archit. Compilation (PACT)*, Oct. 2015.

[112] H. Chauhan, I. Calciu, V. Chidambaram, E. Schkufza, O. Mutlu, and P. Subrahmanyam, "NVMOVE: Helping programmers move to byte-based persistence," in *Proc. INFLOW*, 2016.

[113] H. Yoon, J. Meza, N. Muralimanohar, N. P. Jouppi, and O. Mutlu, "Efficient data mapping and buffering techniques for multilevel cell phase-change memories," *ACM Trans. Archit. Code Optim.*, vol. 11, no. 4, pp. 1–25, Jan. 2015.

[114] Y. Li, S. Ghose, J. Choi, J. Sun, H. Wang, and O. Mutlu, "Utility-based hybrid memory management," in *Proc. IEEE Int. Conf. Cluster Comput. (CLUSTER)*, Sep. 2017.

[115] W. Zhang, X. Zhao, S. Jiang, and H. Jiang, "ChameleonDB: A key-value store for optane persistent memory," in *Proc. 16th Eur. Conf. Comput. Syst.*, Apr. 2021.

[116] D.-H. Bae, I. Jo, Y. A. Choi, J.-Y. Hwang, S. Cho, D.-G. Lee, and J. Jeong, "2B-SSD: The case for dual, byte- and block-addressable solid-state drives," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2018.

[117] S. Kim and J.-S. Yang, "Optimized I/O determinism for emerging NVM-based NVMe SSD in an enterprise system," in *Proc. 55th ACM/ESDA/IEEE Design Autom. Conf. (DAC)*, Jun. 2018.

[118] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, "Error characterization, mitigation, and recovery in flash-memory-based solid-state drives," *Proc. IEEE*, vol. 105, no. 9, pp. 1666–1704, Sep. 2017.

[119] Y. Luo, S. Ghose, Y. Cai, E. F. Haratsch, and O. Mutlu, "Improving 3D NAND flash memory lifetime by tolerating early retention loss and process variation," in *Proc. Abstr. ACM Int. Conf. Meas. Model. Comput. Syst.*, Jun. 2018.

[120] Y. Luo, S. Ghose, Y. Cai, E. F. Haratsch, and O. Mutlu, "HeatWatch: Improving 3D NAND flash memory device reliability by exploiting self-recovery and temperature awareness," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2018.

[121] Y. Cai, S. Ghose, Y. Luo, K. Mai, O. Mutlu, and E. F. Haratsch, "Vul-nerabilities in MLC NAND flash memory programming: Experimental analysis, exploits, and mitigation techniques," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2017.

[122] Y. Luo, S. Ghose, Y. Cai, E. F. Haratsch, and O. Mutlu, "Enabling accurate and practical online flash channel modeling for modern MLC NAND flash memory," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 9, pp. 2294–2311, Sep. 2016.

[123] Y. Cai, Y. Luo, S. Ghose, and O. Mutlu, "Read disturb errors in MLC NAND flash memory: Characterization, mitigation, and recovery," in *Proc. 45th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, Jun. 2015.

[124] O. Mutlu. (2020). *Lecture Notes for Computer Architecture—Lecture 26: Flash Memory and Solid-State Drives*. [Online]. Available: https://rb.gy/xqis8

[125] Z.-L. Ke, H.-Y. Cheng, and C.-L. Yang, "LIRS: Enabling efficient machine learning on NVM-based storage via a lightweight implemen-tation of random shuffling," 2018, *arXiv:1810.04509*.

[126] X. Liu, Y. Pan, Y. Li, G. Wang, and X. Liu, "An NVM SSD-optimized query processing framework," in *Proc. 29th ACM Int. Conf. Inf. Knowl. Manage.*, Oct. 2020.

[127] S. Han, D. Jiang, and J. Xiong, "SplitKV: Splitting IO paths for different sized key-value items with advanced storage devices," in *Proc. HotStorage*, 2020.

[128] A. Papagiannis, G. Xanthakis, G. Saloustros, M. Marazakis, and A. Bilas, "Optimizing memory-mapped I/O for fast storage devices," in *Proc. USENIX ATC*, 2020.

[129] Y. Jia and F. Chen, "From flash to 3D XPoint: Performance bottlenecks and potentials in RocksDB with storage evolution," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Aug. 2020.

[130] K. Wu, A. Arpaci-Dusseau, and R. Arpaci-Dusseau, "Towards an unwritten contract of Intel optane SSD," in *Proc. HotStorage*, 2019.

[131] K. Zhong, T. Wang, X. Zhu, L. Long, D. Liu, W. Liu, Z. Shao, and E. H.-M. Sha, "Building high-performance smartphones via non-volatile memory: The swap approach," in *Proc. 14th Int. Conf. Embedded Softw.*, Oct. 2014.

[132] Y. Kim, M. Imani, S. Patil, and T. S. Rosing, "CAUSE: Critical application usage-aware memory system using non-volatile memory for mobile devices," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2015.

[133] D. Liu, K. Zhong, X. Zhu, Y. Li, L. Long, and Z. Shao, "Non-volatile memory based page swapping for building high-performance mobile devices," *IEEE Trans. Comput.*, vol. 66, no. 11, pp. 1918–1931, Nov. 2017.

[134] K. Zhong, D. Liu, L. Long, J. Ren, Y. Li, and E. H.-M. Sha, "Building NVRAM-aware swapping through code migration in mobile devices," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 11, pp. 3089–3099, Nov. 2017.

[135] J. Kim and H. Bahn, "Analysis of smartphone I/O characteristics—Toward efficient swap in a smartphone," *IEEE Access*, vol. 7, pp. 129930–129941, 2019.

[136] X. Zhu, D. Liu, K. Zhong, J. Ren, and T. Li, "SmartSwap: High-performance and user experience friendly swapping in mobile systems," in *Proc. 54th Annu. Design Autom. Conf.*, Jun. 2017.

[137] J. Kim and H. Bahn, "Comparison of hybrid and hierarchical swap architectures in Android by using NVM," *J. Semicond. Technol. Sci.*, vol. 18, no. 6, pp. 651–657, Dec. 2018.

[138] K. Zhong, X. Zhu, T. Wang, D. Zhang, X. Luo, D. Liu, W. Liu, and E. H.-M. Sha, "DR. Swap: Energy-efficient paging for smartphones," in *Proc. Int. Symp. Low Power Electron. Des.*, Aug. 2014.

[139] S.-H. Kim, J. Jeong, and J.-S. Kim, "Application-aware swapping for mobile systems," *ACM Trans. Embedded Comput. Syst.*, vol. 16, no. 5s, pp. 1–19, Oct. 2017.

[140] J. Kim, C. Kim, and E. Seo, "*ezswap*: Enhanced compressed swap scheme for mobile devices," *IEEE Access*, vol. 7, pp. 139678–139691, 2019.

[141] J. Kim and H. Bahn, "Maintaining application context of smartphones by selectively supporting swap and kill," *IEEE Access*, vol. 8, pp. 85140–85153, 2020.

[142] W. Guo, K. Chen, H. Feng, Y. Wu, R. Zhang, and W. Zheng, "MARS: Mobile application relaunching speed-up through flash-aware page swapping," *IEEE Trans. Comput.*, vol. 65, no. 3, pp. 916–928, Mar. 2016.

[143] Y. Liang, J. Li, R. Ausavarungnirun, R. Pan, L. Shi, T.-W. Kuo, and C. J. Xue, "Acclaim: Adaptive memory reclaim to improve user experience in Android systems," in *Proc. USENIX ATC*, 2020.

[144] Google LLC. *Chrome Browser*. [Online]. Available: https://www.google.com/chrome/

[145] Chromium Project. *MemoryPressure Tast Test*. [Online]. Available: https://rb.gy/j1ft7

[146] *Intel Optane SSD 900P Series*, Intel Corp., Santa Clara, CA, USA, 2018.

[147] A. Gutierrez, R. G. Dreslinski, T. F. Wenisch, T. Mudge, A. Saidi, C. Emmons, and N. Paver, "Full-system analysis and characterization of interactive smartphone applications," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, Nov. 2011.

[148] Y. Huang, Z. Zha, M. Chen, and L. Zhang, "Moby: A mobile benchmark suite for architectural simulators," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Mar. 2014.

[149] D. Pandiyan, S.-Y. Lee, and C.-J. Wu, "Performance, energy characterizations and architectural implications of an emerging mobile platform benchmark suite—MobileBench," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, Sep. 2013.

[150] B. Popper. (2017). *Google Announces Over 2 Billion Monthly Active Devices on Android*. [Online]. Available: https://rb.gy/yyk1b

[151] Net Applications. *Market Share Statistics for Internet Technologies*. [Online]. Available: https://www.netmarketshare.com/

[152] Chromium Project. *Blink Rendering Engine*. [Online]. Available: https://rb.gy/j32v9

[153] Google LLC. *Skia Graphics Library*. [Online]. Available: https://skia.org/

[154] C. Reis and S. D. Gribble, "Isolating web programs in modern browser architectures," in *Proc. 4th ACM Eur. Conf. Comput. Syst.*, Apr. 2009.

[155] A. Barth, C. Jackson, and C. Reis, "The security architecture of the chromium browser," Google Chrome Team, Stanford Univ., Stanford, CA, USA, Tech. Rep., 2008.

[156] HTTP Archive. [Online]. Available: http://httparchive.org/

[157] D. Rientjes, "OOM killer rewrite; when the kernel runs out of memory," LinuxCon, Boston, MA, USA, Tech. Rep., 2010.

[158] C. Collins, M. D. Galpin, and M. Kaeppler, *Android in Practice*. Shelter Island, NY, USA: Manning Publications, 2011.

[159] Google LLC. *Pixel Smartphones*. [Online]. Available: https://www.google.com/pixel/

[160] S. Jennings, "Transparent memory compression in Linux," LinuxCon, 2013.

[161] E. Shiu and S. Prakash, "System challenges and hardware requirements for future consumer devices: From wearable to ChromeBooks and devices in-between," in *Proc. Symp. VLSI Technol. (VLSI Technol.)*, Jun. 2015.

[162] E. Shiu and S. Lim, "Driving innovation in memory architecture of consumer hardware with digital photography and machine intelligence use cases," in *Proc. IEEE Int. Memory Workshop (IMW)*, May 2017.

[163] G. Pekhimenko, V. Seshadri, Y. Kim, H. Xin, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Linearly compressed pages: A low-complexity, low-latency main memory compression framework," in *Proc. MICRO*, 2013.

[164] Chromium Project. (2016). *Memory Coordinator*. [Online]. Available: https://bit.ly/3lOW7w7

[165] I. Grigorik, "High performance networking in chrome," Perform. Open Source Appl., Speed, Precis., Bit Serendipity, 2013.

[166] S. Lohr. (2012). *For Impatient Web Users, an Eye Blink Is Just Too Long to Wait*. [Online]. Available: https://rb.gy/50zon

[167] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016.

[168] L. Song, Y. Zhuo, X. Qian, H. Li, and Y. Chen, "GraphR: Accelerating graph processing using ReRAM," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2018.

[169] L. Song, X. Qian, H. Li, and Y. Chen, "PipeLayer: A pipelined ReRAM-based accelerator for deep learning," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2017.

[170] P. Yao, H. Wu, B. Gao, S. B. Eryilmaz, X. Huang, W. Zhang, Q. Zhang, N. Deng, L. Shi, H.-S.-P. Wong, and H. Qian, "Face classification using electronic synapses," *Nature Commun.*, vol. 8, no. 1, May 2017.

[171] M. Hu, J. P. Strachan, Z. Li, E. M. Grafals, N. Davila, C. Graves, S. Lam, N. Ge, J. J. Yang, and R. S. Williams, "Dot-product engine for neuromorphic computing: Programming 1T1M crossbar to accelerate matrix-vector multiplication," in *Proc. 53rd Annu. Design Autom. Conf.*, Jun. 2016.

[172] C. Gopalan, Y. Ma, T. Gallo, J. Wang, E. Runnion, J. Saenz, F. Koushan, P. Blanchard, and S. Hollmer, "Demonstration of conductive bridging random access memory (CBRAM) in logic CMOS process," *Solid-State Electron.*, vol. 58, no. 1, pp. 54–61, Apr. 2011.

[173] D. Jana, S. Roy, R. Panja, M. Dutta, S. Z. Rahaman, R. Mahapatra, and S. Maikap, "Conductive-bridging random access memory: Challenges and opportunity for 3D architecture," *Nanosc. Res. Lett.* vol. 10, no. 188, pp. 1–23, 2015.

[174] J.-H. Cha, S. Y. Yang, J. Oh, S. Choi, S. Park, B. C. Jang, W. Ahn, and S.-Y. Choi, "Conductive-bridging random-access memories for emerging neuromorphic computing," *Nanoscale*, vol. 12, no. 27, pp. 14339–14368, 2020.

[175] J. F. Scott and C. A. P. De Araujo, "Ferroelectric memories," *Science*, vol. 246, no. 4936, pp. 1400–1405, 1989.

[176] J. F. Scott, "Applications of modern ferroelectrics," *Science*, vol. 315, no. 5814, pp. 954–959, Feb. 2007.

[177] T. Mikolajick, C. Dehm, W. Hartner, I. Kasko, M. J. Kastner, N. Nagel, M. Moert, and C. Mazure, "FeRAM technology for high density applications," *Microelectron. Rel.*, vol. 41, no. 7, pp. 947–950, Jul. 2001.

[178] M. Webb, "3D XPoint status and forecast," in *Proc. Flash Memory Summit*, 2016.

[179] I. Cutress and B. Tallis, "Intel launches optane DIMMs up to 512 GB: Apache pass is here!" AnandTech, 2016. [Online]. Available: https://www.anandtech.com/show/12828/intel-launches-optane-dimms-up-to-512gb-apache-pass-is-here

[180] *PCI Express Base Specification Revision 5.0, Version 1.0*, PCI-SIG, Beaverton, OR, USA, 2019.

[181] O. Patil, L. Ionkov, J. Lee, F. Mueller, and M. Lang, "Performance characterization of a DRAM-NVM hybrid memory architecture for HPC applications using Intel optane DC persistent memory modules," in *Proc. Int. Symp. Memory Syst.*, Sep. 2019.

[182] G. Gill, R. Dathathri, L. Hoang, R. Peri, and K. Pingali, "Single machine graph analytics on massive datasets using Intel optane DC persistent memory," 2019, *arXiv:1904.07162*.

[183] Y. Wu, K. Park, R. Sen, B. Kroth, and J. Do, "Lessons learned from the early performance evaluation of Intel optane DC persistent memory in DBMS," in *Proc. 16th Int. Workshop Data Manage. New Hardw.*, Jun. 2020.

[184] M. Weiland, H. Brunst, T. Quintino, N. Johnson, O. Iffrig, S. Smart, C. Herold, A. Bonanni, A. Jackson, and M. Parsons, "An early evaluation of Intel's optane DC persistent memory module and its impact on high-performance scientific applications," in *Proc. SC*, 2019.

[185] A. Shanbhag, N. Tatbul, D. Cohen, and S. Madden, "Large-scale in-memory analytics on Intel optane DC persistent memory," in *Proc. 16th Int. Workshop Data Manage. New Hardw.*, Jun. 2020.

[186] V. Mironov, I. Chernykh, I. Kulikov, A. Moskovsky, E. Epifanovsky, and A. Kudryavtsev, "Performance evaluation of the Intel optane DC memory with scientific benchmarks," in *Proc. IEEE/ACM Workshop Memory Centric High Perform. Comput. (MCHPC)*, Nov. 2019.

[187] J. Yang, J. Kim, M. Hoseinzadeh, J. Izraelevitz, and S. Swanson, "An empirical guide to the behavior and use of scalable persistent memory," in *Proc. FAST*, 2020.

[188] L. Benson, L. Papke, and T. Rabl, "PerMA-bench: Benchmarking persistent memory access," *Proc. VLDB Endowment*, vol. 15, no. 11, pp. 2463–2476, Jul. 2022.

[189] L. Xiang, X. Zhao, J. Rao, S. Jiang, and H. Jiang, "Characterizing the performance of Intel optane persistent memory: A close look at its on-DIMM buffering," in *Proc. 17th Eur. Conf. Comput. Syst.*, Mar. 2022.

[190] Tom's Hardware. (2019). *Intel Optane DIMM Pricing*. [Online]. Available: https://rb.gy/873zd

[191] D. Bittman, P. Alvaro, P. Mehra, D. D. E. Long, and E. L. Miller, "Twizzler: A data-centric OS for non-volatile memory," in *Proc. USENIX ATC*, 2020.

[192] Asus, Inc. *ASUS Chromebox 3*. [Online]. Available: https://rb.gy/e9cnq

[193] A. Wright, "Ready for a web OS?" *Commun. ACM*, vol. 52, no. 12, pp. 16–17, 2009.

[194] Intel Corporation. (2016). *Intel Core i3-7100U Processor*. [Online]. Available: https://rb.gy/2ifwc

[195] *SK Hynix 4GB DDR4 HMA851S6AFR6N-UH*, SK Hynix Inc., San Jose, CA, USA, Rev. 1.4, Sep. 2017.

[196] *SATA III M.2 Solid State Drive M.2 SSD 400S*, Transcend Inf. Inc., Taipei, Taiwan, 2020.

[197] Intel Corporation. *Intel Optane Memory M10 Series*. [Online]. Available: https://rb.gy/atuol

[198] Facebook, Inc. *Facebook*. [Online]. Available: https://www.facebook.com/

[199] Facebook, Inc. *Instagram*. [Online]. Available: https://about.instagram.com/about-us/

[200] Facebook, Inc. *WhatsApp Messenger*. [Online]. Available: https://www.whatsapp.com/

[201] Telegram FZ-LLC. *Telegram Messenger*. [Online]. Available: https://telegram.org/

[202] Adobe Inc. *Adobe Acrobat Reader*. [Online]. Available: https://get.adobe.com/reader/

[203] Mojang Studios. *Minecraft*. [Online]. Available: https://www.minecraft.net/

[204] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *Proc. 4th Annu. IEEE Int. Workshop Workload Characterization (WWC)*, 2001.

[205] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, "MediaBench: A tool for evaluating and synthesizing multimedia and communications systems," in *Proc. 30th Annu. Int. Symp. Microarchitecture*, 1997.

[206] Chromium Project. *The Chromium Project*. [Online]. Available: https://rb.gy/osxys

[207] Linux Kernel Organization, Inc. *perf: Linux Profiling With Performance Counters*. [Online]. Available: https://perf.wiki.kernel.org/index.php/Main_Page

[208] Google LLC. *YouTube*. [Online]. Available: https://www.youtube.com

[209] Google LLC. *Google Maps*. [Online]. Available: http://maps.google.com/

[210] Google LLC. *Google Sheets*. [Online]. Available: https://www.google.com/sheets/about/

[211] Google LLC. *Google Docs*. [Online]. Available: https://www.google.com/docs/about/

[212] Twitter, Inc. *Twitter*. [Online]. Available: https://www.twitter.com/

[213] Chromium Project. *Chromium Media*. [Online]. Available: https://rb.gy/n4bnw

[214] FFmpeg Team. *FFmpeg Documentation*. [Online]. Available: https://rb.gy/tr393

[215] WebM Project. *WebM*. [Online]. Available: https://www.webmproject.org/code/

[216] A. Grange, P. de Rivaz, and J. Hunt. *VP9 Bitstream & Decoding Process Specification*. [Online]. Available: https://rb.gy/1cfjh

[217] Web Hypertext Application Technology Working Group. (2021). *HTML Living Standard*. [Online]. Available: https://html.spec.whatwg.org/multipage/

[218] Chromium Project. *Mojo*. [Online]. Available: https://rb.gy/ynbj8

[219] Chromium Project. *VaAPI*. [Online]. Available: https://rb.gy/5ri9b

[220] Chromium Project. *SkGifCodec*. [Online]. Available: https://rb.gy/gzenm

[221] Chromium Project. *V8 JavaScript Engine*. [Online]. Available: https://v8.dev/

[222] Chromium Project. *Tast*. [Online]. Available: https://rb.gy/073gv

[223] Google LLC. *chrome.automation*. [Online]. Available: https://rb.gy/t6kxp

[224] J. Choe, "Intel 3D XPoint memory die removed from Intel optane PCM (phase change memory)," TechInsights, 2017. [Online]. Available: https://www.techinsights.com/blog/intel-3d-xpoint-memory-die-removed-intel-optanetm-pcm-phase-change-memory

[225] J. Chen, R. C. Chiang, H. H. Huang, and G. Venkataramani, "Energy-aware writes to non-volatile main memory," *ACM SIGOPS Operating Syst. Rev.*, vol. 45, no. 3, pp. 48–52, Jan. 2012.

[226] B. Zolnierkiewicz, "Efficient memory management on mobile devices," LinuxCon, 2013. [Online]. Available: http://events17.linuxfoundation.org/sites/events/files/slides/Efficient_Memory_Management_on_Mobile_Devices_0.pdf

[227] B. K. Tanaka, "Monitoring virtual memory with vmstat," *Linux J.*, Oct. 2005.

[228] Y. Guo, Y. Hua, and P. Zuo, "A latency-optimized and energy-efficient write scheme in NVM-based main memory," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 1, pp. 62–74, Jan. 2020.

[229] J.-H. Choi and G.-H. Park, "NVM way allocation scheme to reduce NVM writes for hybrid cache architecture in chip-multiprocessors," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 10, pp. 2896–2910, Oct. 2017.

[230] S. Swami, J. Rakshit, and K. Mohanram, "SECRET: Smartly EnCRypted energy efficient non-volatile memories," in *Proc. 53rd Annu. Design Autom. Conf.*, Jun. 2016.

[231] Intel Corporation. *Intel Optane Memory*. [Online]. Available: https://rb.gy/v31hy

[232] Y.-M. Chang, P.-C. Hsiu, Y.-H. Chang, C.-H. Chen, T.-W. Kuo, and C.-Y.-M. Wang, "Improving PCM endurance with a constant-cost wear leveling design," *ACM Trans. Design Autom. Electron. Syst.*, vol. 22, no. 1, pp. 1–27, Jan. 2017.

[233] H. Aghaei Khouzani, Y. Xue, C. Yang, and A. Pandurangi, "Prolonging PCM lifetime through energy-efficient, segment-aware, and wear-resistant page allocation," in *Proc. Int. Symp. Low power Electron. design*, Aug. 2014.

[234] F. T. Hady. *Intel Optane Technology Delivers New Levels of Endurance*. Accessed: Jun. 21, 2023. [Online]. Available: https://rb.gy/c83ee

[235] M. K. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abali, "Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling," in *Proc. 42nd Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2009.

[236] Micron Technology, Inc. *SLC NAND*. [Online]. Available: https://rb.gy/855sx

[237] J. Handy. *Examining 3D XPoint's 1,000 Times Endurance Benefit—The Memory Guy*. [Online]. Available: https://rb.gy/mvd5a

[238] diskprices.com. *Disk Prices (U.S.)*. Accessed: Apr. 2, 2020. [Online]. Available: https://bit.ly/2STO9We

[239] A. C. de Melo, "The new Linux 'perf' tools," in *Proc. Linux Kongress*, 2010.

[240] M. Bjørling, J. Axboe, D. Nellans, and P. Bonnet, "Linux block IO: Introducing multi-queue SSD access on multi-core systems," in *Proc. SYSTOR*, 2013.

[241] A. Tavakkol, M. Sadrosadati, S. Ghose, J. Kim, Y. Luo, Y. Wang, N. M. Ghiasi, L. Orosa, J. Gomez-Luna, and O. Mutlu, "FLIN: Enabling fairness and enhancing performance in modern NVMe solid state drives," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2018.

[242] A. Tavakkol, J. Gómez-Luna, M. Sadrosadati, S. Ghose, and O. Mutlu, "MQSim: A framework for enabling realistic studies of modern multi-queue SSD devices," in *Proc. FAST*, 2018.

[243] A. D. Brunelle, "Blktrace user guide," 2007. [Online]. Available: https://manualzz.com/doc/4196014/blktrace-user-guide

[244] O. Sandoval. (2017). *Kyber MQ I/O Scheduler*. [Online]. Available: https://rb.gy/6azue

[245] J. Axboe. (2016). *MQ Deadline I/O Scheduler*. [Online]. Available: https://rb.gy/xxdro

[246] J. C. R. Bennett and H. Zhang, "Hierarchical packet fair queueing algorithms," *IEEE/ACM Trans. Netw.*, vol. 5, no. 5, pp. 675–689, Oct. 1997.

[247] J. Yang, D. B. Minturn, and F. Hady, "When poll is better than interrupt," in *Proc. FAST*, 2012.

[248] D. Le Moal, "I/O latency optimization with polling," in *Proc. Vault*, 2017.

[249] Linux Kernel Organization, Inc. (2009). *Linux Kernel Documentation: Queue sysfs Files*. [Online]. Available: https://rb.gy/9thuf

[250] Intel Corporation. *Tuning the Performance of Intel Optane SSDs on Linux Operating Systems*. [Online]. Available: https://rb.gy/uunhr

[251] L. Yavits, L. Orosa, S. Mahar, J. D. Ferreira, M. Erez, R. Ginosar, and O. Mutlu, "WoLFRaM: Enhancing wear-leveling and fault tolerance in resistive memories using programmable address decoders," in *Proc. ICCD*, 2020.

[252] C.-H. Chen, P.-C. Hsiu, T.-W. Kuo, C.-L. Yang, and C.-Y.-M. Wang, "Age-based PCM wear leveling with nearly zero search cost," in *Proc. 49th Annu. Design Autom. Conf.*, Jun. 2012.

[253] S.-W. Cheng, Y.-H. Chang, T.-Y. Chen, Y.-F. Chang, H.-W. Wei, and W.-K. Shih, "Efficient warranty-aware wear leveling for embedded systems with PCM main memory," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 7, pp. 2535–2547, Jul. 2016.

[254] J. Fan, S. Jiang, J. Shu, L. Sun, and Q. Hu, "WL-reviver: A framework for reviving any wear-leveling techniques in the face of failures on phase change memory," in *Proc. 44th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, Jun. 2014.

[255] Y. Han, J. Dong, K. Weng, Y. Wang, and X. Li, "Enhanced wear-rate leveling for PRAM lifetime improvement considering process variation," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 1, pp. 92–102, Jan. 2016.

[256] 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.

[257] Y. Joo, D. Niu, X. Dong, G. Sun, N. Chang, and Y. Xie, "Energy- and endurance-aware design of phase change memory caches," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2010.

[258] D. Liu, T. Wang, Y. Wang, Z. Shao, Q. Zhuge, and E. H.-M. Sha, "Application-specific wear leveling for extending lifetime of phase change memory in embedded systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 33, no. 10, pp. 1450–1462, Oct. 2014.

[259] M. K. Qureshi, A. Seznec, L. A. Lastras, and M. M. Franceschini, "Practical and secure PCM systems by online detection of malicious write streams," in *Proc. IEEE 17th Int. Symp. High Perform. Comput. Archit.*, Feb. 2011.

[260] G. Lee, W. Jin, W. Song, J. Gong, J. Bae, T. J. Ham, J. W. Lee, and J. Jeong, "A case for hardware-based demand paging," in *Proc. ACM/IEEE 47th Annu. Int. Symp. Comput. Archit. (ISCA)*, May 2020.

[261] K. Oh, J. Park, and Y. I. Eom, "H-BFQ: Supporting multi-level hierarchical cgroup in BFQ scheduler," in *Proc. IEEE Int. Conf. Big Data Smart Comput. (BigComp)*, Feb. 2020.

[262] W. Shin, Q. Chen, M. Oh, H. Eom, and H. Y. Yeom, "OS I/O path optimizations for flash solid-state drives," in *Proc. USENIX ATC*, 2014.

[263] D. Vučinić, Q. Wang, C. Guyot, R. Mateescu, F. Blagojevic, L. Franca-Neto, D. Le Moal, T. Bunker, J. Xu, S. Swanson, and Z. Bandic, "DC express: Shortest latency protocol for reading phase change memory over PCI express," in *Proc. FAST*, 2014.

[264] J. Zhang, M. Kwon, D. Gouk, S. Koh, C. Lee, M. Alian, M. Chun, M. T. Kandemir, N. S. Kim, J. Kim, and M. Jung, "FlashShare: Punching through server storage stack from kernel to firmware for ultra-low latency SSDs," in *Proc. OSDI*, 2018.

[265] M. Liu, H. Liu, C. Ye, X. Liao, H. Jin, Y. Zhang, R. Zheng, and L. Hu, "Towards low-latency I/O services for mixed workloads using ultra-low latency SSDs," in *Proc. 36th ACM Int. Conf. Supercomput.*, Jun. 2022.

[266] 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," in *Proc. 17th Int. Conf. Archit. Support Program. Lang. Operating Syst.*, Mar. 2012.

[267] 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 *Proc. HotStorage*, 2016.

[268] S. Scargall, "Introducing the persistent memory development kit," in *Programming Persistent Memory*. New York, NY, USA: Springer, 2020.

[269] Z. Yang, J. R. Harris, B. Walker, D. Verkamp, C. Liu, C. Chang, G. Cao, J. Stern, V. Verma, and L. E. Paul, "SPDK: A development kit to build high performance storage applications," in *Proc. IEEE Int. Conf. Cloud Comput. Technol. Sci. (CloudCom)*, Dec. 2017.

[270] Samsung Electronics Co., Ltd. *Open Memory Platform Development Kit: User Level NVMe Driver*. [Online]. Available: https://github.com/OpenMPDK/uNVMe

[271] 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," *ACM Trans. Comput. Syst.*, vol. 33, no. 4, pp. 1–30, Jan. 2016.

[272] H.-J. Kim and J.-S. Kim, "A user-space storage I/O framework for NVMe SSDs in mobile smart devices," *IEEE Trans. Consum. Electron.*, vol. 63, no. 1, pp. 28–35, Feb. 2017.

[273] Y. Kwon, H. Fingler, T. Hunt, S. Peter, E. Witchel, and T. Anderson, "Strata: A cross media file system," in *Proc. 26th Symp. Operating Syst. Princ.*, Oct. 2017.

[274] K. Wu, F. Ober, S. Hamlin, and D. Li, "Early evaluation of Intel optane non-volatile memory with HPC I/O workloads," 2017, *arXiv:1708.02199*.

[275] Z. Lu and Q. Cao, "A case study of migrating RocksDB on Intel optane persistent memory," in *Proc. IEEE Int. Conf. Netw., Archit. Storage (NAS)*, Oct. 2021.

[276] G. Singh, R. Nadig, J. Park, R. Bera, N. Hajinazar, D. Novo, J. Gómez-Luna, S. Stuijk, H. Corporaal, and O. Mutlu, "Sibyl: Adaptive and extensible data placement in hybrid storage systems using online reinforcement learning," in *Proc. 49th Annu. Int. Symp. Comput. Archit.*, Jun. 2022.

[277] G. Oh, S. Kim, S.-W. Lee, and B. Moon, "SQLite optimization with phase change memory for mobile applications," *Proc. VLDB Endowment*, vol. 8, no. 12, pp. 1454–1465, Aug. 2015.

[278] Y. Li, L. Zeng, G. Chen, C. Gu, F. Luo, W. Ding, Z. Shi, and J. Fuentes, "A multi-hashing index for hybrid DRAM-NVM memory systems," *J. Syst. Archit.*, vol. 128, Jul. 2022, Art. no. 102547.

[279] A. Raybuck, T. Stamler, W. Zhang, M. Erez, and S. Peter, "HeMem: Scalable tiered memory management for big data applications and real NVM," in *Proc. ACM SIGOPS 28th Symp. Operating Syst. Princ.*, Oct. 2021.

SAUGATA GHOSE (Member, IEEE) received the dual B.S. degree in computer science and in computer engineering from Binghamton University and The State University of New York, and the M.S. and Ph.D. degrees in electrical and computer engineering from Cornell University. He is currently an Assistant Professor with the Department of Computer Science, University of Illinois Urbana-Champaign. Prior to joining Illinois, he was a Postdoctoral Researcher and later a Systems Scientist with Carnegie Mellon University. He received the Best Paper Award from DFRWS-EU, in 2017, for work on solid-state drive forensics. He was a 2019 Wimmer Faculty Fellow at CMU. His current research interests include data-oriented computer architectures and systems, new interfaces between systems software and architectures, low-power memory and storage systems, and architectures for emerging platforms and domains. For more information visit the link (https://ghose.cs.illinois.edu/).

JUAN GÓMEZ-LUNA received the B.S. and M.S. degrees in telecommunication engineering from the University of Seville, Spain, in 2001, and the Ph.D. degree in computer science from the University of Córdoba, Spain, in 2012. Between 2005 and 2017, he was a Faculty Member of the University of Córdoba. He is currently a Senior Researcher and a Lecturer with the SAFARI Research Group, ETH Zürich. His research interests include processing-in-memory, memory systems, heterogeneous computing, and the hardware and software acceleration of medical imaging and bioinformatics. He is the lead author of PrIM (https://github.com/CMU-SAFARI/prim-benchmarks), the first publicly-available benchmark suite for a real-world processing-in-memory architecture, and Chai (https://github.com/chai-benchmarks/chai), a benchmark suite for heterogeneous systems with CPU/GPU/FPGA.

AMIRALI BOROUMAND received the B.S. degree in computer hardware engineering from the Sharif University of Technology, Tehran, Iran, in 2014, and the Ph.D. degree in computer architecture from Carnegie Mellon University, Pittsburgh, PA, USA, in 2020. He is currently with Google, Mountain View, CA, USA.

ALEXIS SAVERY is currently with Google, Mountain View, CA, USA.

SONNY RAO received the B.S. degree in computer science from the Georgia Institute of Technology, Atlanta, GA, USA, in 2003. He is currently with Rivos Inc., Mountain View, CA, USA.

GERALDO F. OLIVEIRA (Graduate Student Member, IEEE) received the B.S. degree in computer science from the Federal University of Viosa, Viosa, Brazil, in 2015, and the M.S. degree in computer science from the Federal University of Rio Grande do Sul, Porto Alegre, Brazil, in 2017. He is currently pursuing the Ph.D. degree with Onur Mutlu with ETH Zürich, Zürich, Switzerland. His current research interests include system support for processing-in-memory and processing-using-memory architectures, data-centric accelerators for emerging applications, approximate computing, and emerging memory systems for consumer devices. He has several publications on these topics.

SALMAN QAZI is currently with Google, Mountain View, CA, USA.

GWENDAL GRIGNOU is currently with Google, Mountain View, CA, USA.

RAHUL THAKUR received the B.S. degree in engineering, technology, science, semiconductors, embedded, VLSI from the University of Mumbai, Mumbai, India, in 2011, and the M.S. degree in computer engineering from The University of Texas at Austin, Austin, TX, USA, in 2013. He is currently with Google, Mountain View, CA, USA, as a Senior Hardware Engineer—a SoC Architect.
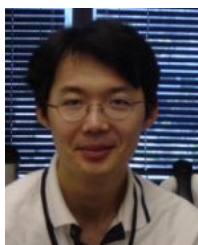
ERIC SHIU received the B.S. degree in electrical engineering from National Chiao Tung University, Hsinchu, Taiwan, in 1994, and the M.S. degree in electrical engineering from Stanford University, Stanford, CA, USA, in 1998. He is currently with Rivos Inc., Mountain View, CA, USA, as a Hardware Engineer.

ONUR MUTLU received the B.S. degree in computer engineering and psychology from the University of Michigan, Ann Arbor, and the M.S. and Ph.D. degrees in ECE from The University of Texas at Austin. He is currently a Professor in computer science with ETH Zürich. He is also a Faculty Member with Carnegie Mellon University, where he previously held the Strecker Early Career Professorship. His current research interests include computer architecture, systems, hardware security, and bioinformatics. A variety of techniques, along with his group and collaborators, he has invented over the years have influenced industry and he have been employed in commercial microprocessors and memory/storage systems. He started the Computer Architecture Group, Microsoft Research (2006–2009), and he held various product and research positions with Intel Corporation, Advanced Micro Devices, VMware, and Google. He received the Google Open Source Peer Bonus Award, Huawei OlympusMons Award for Storage Systems Research, Google Security and Privacy Research Award, Persistent Impact Prize of the Non-Volatile Memory Systems Workshop, the Intel Outstanding Researcher Award, the IEEE High Performance Computer Architecture Test of Time Award, the IEEE Computer Society Edward J. McCluskey Technical Achievement Award, ACM SIGARCH Maurice Wilkes Award, the inaugural IEEE Computer Society Young Computer Architect Award, the inaugural Intel Early Career Faculty Award, the U.S. National Science Foundation CAREER Award, Carnegie Mellon University Ladd Research Award, faculty partnership awards from various companies, and a healthy number of best paper, "Top Pick" paper, and best artifact recognitions at various computer systems, architecture, and security venues. He is an ACM Fellow and an elected member of the Academy of Europe (Academia Europaea). His computer architecture and digital logic design course lectures and materials are freely available on YouTube (https://www.youtube.com/OnurMutluLectures), and his research group makes a wide variety of software and hardware artifacts freely available online (https://safari.ethz.ch/ and https://github.com/CMU-SAFARI). For more information visit the link (https://people.inf.ethz.ch/omutlu/).

● ● ●