Towards an Adaptable Systems Architecture for Memory Tiering at Warehouse-Scale

Padmapriya Duraisamy
Google, USA

Wei Xu
Google, USA

Scott Hare
Google, USA

Ravi Rajwar
Google, USA

David Culler
Google, USA
University of California at Berkeley
Berkeley, USA

Zhiyi Xu
Google, USA

Jianing Fan
Google, USA

Christopher Kennelly
Google, USA

Bill McCloskey
Google, USA

Danijela Mijailovic
Google, USA

Brian Morris
Google, USA

Chiranjit Mukherjee
Google, USA

Jingliang Ren
Google, USA

Greg Thelen
Google, USA

Paul Turner
Google, USA

Carlos Villavieja
Google, USA

Parthasarathy Ranganathan
Google, USA

Amin Vahdat
Google, USA

ABSTRACT

Fast DRAM increasingly dominates infrastructure spend in large scale computing environments and this trend will likely worsen without an architectural shift. The cost of deployed memory can be reduced by replacing part of the conventional DRAM with lower cost albeit slower memory media, thus creating a tiered memory system where both tiers are directly addressable and cached. But, this poses numerous challenges in a highly multi-tenant warehouse-scale computing setting. The diversity and scale of its applications motivates an application-transparent solution in the general case, adaptable to specific workload demands.

This paper presents TMTS (Transparent Memory Tiering System), an application-transparent memory tiering management system that implements an adaptive, hardware-guided architecture to dynamically optimize access to the various directly-addressed memory tiers without faults. TMTS has been deployed at scale for two years serving thousands of production services, successfully meeting service level objectives (SLOs) across diverse application classes in the fleet. The solution is developed in terms of system level metrics it seeks to optimize, and evaluated across the diverse workload mix to guide advanced policies embodied in a user-level agent. It sustains less than 5% overall performance degradation while replacing 25% of DRAM with a much slower medium.

CCS CONCEPTS

• **Software and its engineering** → **Memory management**; • **Computer systems organization** → **Distributed architectures**.

KEYWORDS

Memory Tiering, Memory Management, Warehouse-Scale Computing

ACM Reference Format:
Padmapriya Duraisamy, Wei Xu, Scott Hare, Ravi Rajwar, David Culler, Zhiyi Xu, Jianing Fan, Christopher Kennelly, Bill McCloskey, Danijela Mijailovic, Brian Morris, Chiranjit Mukherjee, Jingliang Ren, Greg Thelen, Paul Turner, Carlos Villavieja, Parthasarathy Ranganathan, and Amin Vahdat. 2023. Towards an Adaptable Systems Architecture for Memory Tiering at Warehouse-Scale. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3 (ASPLOS '23), March 25–29, 2023, Vancouver, BC, Canada.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3582016.3582031

1 INTRODUCTION

DRAM is a large and growing portion of infrastructure spend in large scale computing environments, growing faster than other system components including CPU and storage. By some estimates, DRAM will be what limits cost-effective compute capacity [5]. Several possible approaches may reduce the relative spend on DRAM, ranging from application-level optimizations to improved software memory management to new memory hierarchy and hardware.

In this paper, we explore reducing the cost of deployed memory by replacing a fraction of the conventional DRAM primary tier (*tier1*) with a lower cost albeit slower secondary tier memory media (*tier2*), thus creating a tiered memory system where both tiers are directly addressable and cached. This system design point is quite distinct from virtual memory [13], where a very large swap device or remote resource [14, 23, 35, 39] provides a backing store for much smaller physical memory, and zswap [32], where the backing store is a region of DRAM holding compressed pages, as both tiers are directly accessed, without page faults. Together, the tiers form the physical memory beyond L3, unlike 'memory mode' of Intel® Optane™ Persistent Memory [2], where DRAM functions as an L4 cache in front of a much larger slow memory tier. Further, tier usage is application transparent, unlike hybrid models of persistent memory [12, 40, 54], where persistent data structures explicitly use semantically distinct tiers. The system actively manages page placement and the associated virtual-to-physical mapping to keep most frequently accessed (hot) pages in tier1 and least accessed (cold) ones in tier2, even though both are directly addressed and accessible without incurring a fault.

Memory capacity tends to be provisioned more conservatively because memory is inelastic relative to CPU and the implications of running out of memory (OOM) on a system is an OOM exception which is extremely undesirable, even for lower tier applications. This presents a unique opportunity to offload memory that is rarely needed to lower cost media, delivering the same aggregate memory capacity to applications with little or no performance impact, leveraging wide variability in required access latency and bandwidth. This approach inherently involves trade-offs between cost and performance, as cost savings are intrinsically related to differences in bandwidth and latency of the two media. It poses numerous challenges to deployment in a highly multi-tenant warehouse-scale computing (WSC [9]) environment with diverse workloads [48]. Machines run a range of heterogeneous workloads at high utilization while meeting specific application performance and reliability requirements. The diversity and scale of these applications motivate an application-transparent solution in the general case.

TMTS is a memory tiering management system that implements an adaptive, hardware-guided architecture to transparently manage accesses to the various memory tiers. The current system implementation provisions 25% of total memory in a slower, lower-cost tier2 using a variant of Intel Optane Persistent Memory [2]. This design point maximizes cost reduction within performance degradation constraints, further constrained by discrete DIMM capacity offerings. Even with 25%, given the scale of deployment, the upside is substantial.

TMTS has been deployed in a multi-tenant WSC environment successfully serving production services for two years. It uses specific metrics to deliver robust performance and meet various service level objectives (SLOs) across a highly diverse workload mix. TMTS implements mechanisms where software and hardware layers below the application work together to transparently identify and migrate application pages between different tiers.

The paper makes the following contributions:

- Presents the design and implementation of an unconventional tiered memory system and its management that achieves, in production, a target <5% performance impact with 25% replacement of DRAM with a much slower tier in a diverse, highly dynamic, multi-tenant warehouse-scale setting.
- Defines machine-level optimization metrics for memory tiering used to adaptively balance complex high-level fleet-wide application performance and utilization goals (Section 2).
- Introduces a robust A/B testing methodology for live complex system evaluation at scale (Section 4).
- Presents the first comprehensive analysis of a directly accessible tiered memory system in a production warehouse-scale environment successfully serving diverse application classes (Sections 5 and 7).
- Evaluates a range of policies and demonstrates the effectiveness of hardware-assisted event profiling to meet performance requirements (Section 6), identifying the importance of proactive demotion and rapid detection of promotion candidates.
- Reveals how address translation overhead, interference effects, and page-size issues become critical challenges once placement is well optimized, calling for new malloc-level techniques to reduce "access fragmentation", especially with hugepages.
- Utilizes the live A/B testing regime in evidence-driven development of adaptive tiering-aware cluster scheduling to reduce performance impact tails while maximizing utilization, and application-guided, tiering-aware hugepage management to reduce access fragmentation and improve tier2 utilization (Section 8).

Our experience highlights the complexity of managing memory tiers at scale where workload behavior may change daily or be stable for weeks and then change suddenly. We believe this system design and the methodology used to capture the impact of complex interactions of live diverse workloads at scale will open new, increasingly important avenues of research.

2 CONSIDERATIONS FOR MEMORY TIERING AT SCALE

Our WSC environment consists of a global fleet of geographically distributed clusters, each hosting many cluster-wide application services. A cluster manager (the Borg scheduler [50]) schedules tasks to maximize utilization and meet various SLOs [48]. In a tiered memory architecture, tier2 (e.g., low cost Optane) has lower bandwidth and higher latency than tier1 (e.g., DRAM). However, applications can access pages resident in tier2 directly without taking a fault. Deploying such an architecture in our environment motivates special considerations to deal with performance differences.

Application services fall into two classes relevant for memory tiering - *high importance latency sensitive* (HILS) and the rest (non-HILS). HILS includes user-facing applications with tight response time requirements, caching applications in a critical path of other HILS applications, and data processing tasks (Production Tier in [48]). Non-HILS includes throughput-oriented applications, batch, ML training pipelines, and others with weaker SLOs. The classes are often co-located on the same machine. The Borg scheduler actively manages jobs across the cluster based on observed performance, which memory tiering also impacts.

HILS SLOs often require sustaining a certain throughput per machine within a bounded tail latency. To deal with SLO diversity, cluster-level scheduling and machine-level resource management can collectively leverage multi-tenancy to deploy policies where applications with weaker SLOs can be evicted to meet requirements for latency sensitive ones. Such multi-tenant computing environments must also optimize for aggregate demands and not just individual application needs.

While aspects of these issues have analogs in longstanding techniques for virtual memory management, the workload mix, cluster-wide execution, sensitivity to tail latency, and availability requirements give the problem of memory tiering at scale unique characteristics. Managing a non-faulting 2-tier memory system is similar to page migration across NUMA nodes [18]. The system continuously monitors and updates virtual-to-physical address mappings, moving less frequently accessed *cold* pages to tier2 and *hot* pages to tier1, where both tiers are directly addressed and cacheable. Unlike conventional virtual memory or swap, no page faults occur. Latency, bandwidth and interference characteristics of tier2 media are sufficiently different as to require new management mechanisms.

The above challenges motivate the following goal for a tiered memory design and deployment: *Maximize fleet-wide memory cost reduction (the product of per-machine tiered replacement ratio and the fleet-wide 2-tier machine fraction) while minimizing impact on overall machine throughput and meeting various individual application SLOs.* The focus on fleet-wide rather than an individual machine provides multiple optimization vectors to explore. Two key considerations emerge as a result of the performance differences between tiers.

(1) *Impact to utilization*: If we cannot identify sufficient application memory to move to tier2 while meeting our objectives, we will have machines with lower effective total memory capacity. This in turn impacts throughput (i.e., the number of tasks and the size of tasks run on the machine), resulting in under-utilized compute and hence lower CPU utilization.
(2) *Impact to individual task runtime*: A slower, cheaper tier has longer latency and lower bandwidth. Latency sensitive tasks accessing it will experience slowdown unless carefully managed. This may impact effective utilization of tier2 capacity. Contention for this bandwidth-constrained resource may also impact performance of other tasks.

These observations lead to two machine-level metrics for memory tiering stack at warehouse-scale - in general terms, aggregate task capacity and application performance. Meeting the fleet-wide goal requires them to be considered simultaneously. Unfortunately, multi-tenancy, multiple resource dimensions (e.g., CPU, memory), and variations in workload mix and behavior over time combined with cluster-level loads make these metrics insufficient to isolate system effectiveness.

To better analyze the tiering stack, we define two proxy metrics directly connected to the tiering architecture itself:

- **Secondary Tier Residency Ratio (STRR)** is the fraction of allocated memory residing in tier2. It provides a normalized perspective on tier usage. STRR serves as a proxy for measuring impact to utilization.
- **Secondary Tier Access Ratio (STAR)** is the fraction of all memory accesses of an application directed towards pages resident in tier2. A lower STAR means a lower performance impact. STAR serves as a proxy for application performance degradation.

As each machine has a certain fraction of its memory in tier2 (25% in this study), the system aims to bring STRR close to that value in steady state. Keeping STRR high maximizes memory capacity, but there is no benefit in under utilizing tier1 DRAM. Increasing STAR reflects degraded performance. The more tier2 is utilized, the greater the risk, but the risk is mitigated by keeping STAR low. These proxies sharpen the system objective: to minimize STAR while tracking STRR close to the machine ratio.

At low memory utilization, STAR and STRR can be 0 by keeping all allocations in tier1. To meaningfully assess these metrics, we consider machines at total memory utilization of >75% at median and >90% at 95th percentile.

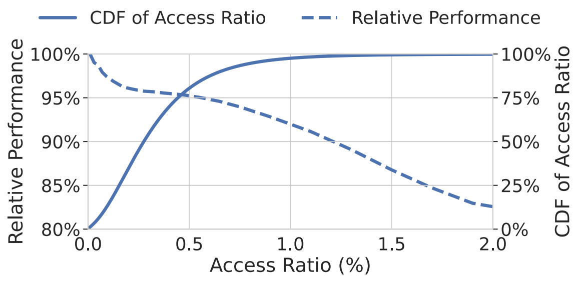


Figure 1: Performance and tier2 access ratio correlation.

Figure 1 summarizes our findings, developed fully below, showing the observed range of STAR and the corresponding impact to application performance across multiple clusters on thousands of machines over several months. The cumulative distribution function (CDF) of STAR shows that the tier2 access ratio is maintained below 1% at the socket level in more than 99% of instances. We achieve our operational goal of limiting performance degradation to less than 5% in aggregate specifically when STAR is below 0.5%.

This relationship allows the system objective for the current deployment to be stated more precisely; it maintains median STAR below 0.5% and p95 STAR within 1.5% while STRR approaches 25% when memory utilization exceeds 75%. The specific thresholds will vary depending on media and workload characteristics, but any tiering stack must adapt with diverse time-varying demand to meet an objective of this form.

To meet SLOs in a diverse environment, a tiering stack must support flexible application-specific tier management policies. We achieve this by driving policy decisions from userspace and using kernel mechanisms to actuate policy decisions.

3 BASE ARCHITECTURE FOR TMTS

TMTS dynamically manages tiered memory placement at page granularity. Its system architecture comprises four layers, illustrated in Figure 2. The bottom layer abstracts the hardware, which presents a physical address space segmented across two or more types of memory devices. The second layer from the top separates the page management policy from the mechanisms for candidate detection and page migration, which are performed by the lower layer kernel components. A key lesson from past production experience is that effective use of kernel mechanisms requires many design iterations addressing subtle behaviors only witnessed at production scale, as well as opportunities to optimize for specific aspects of the workload. A user space policy layer provides this flexibility and velocity. The top cluster-level layer consists of the Borg scheduler that works with

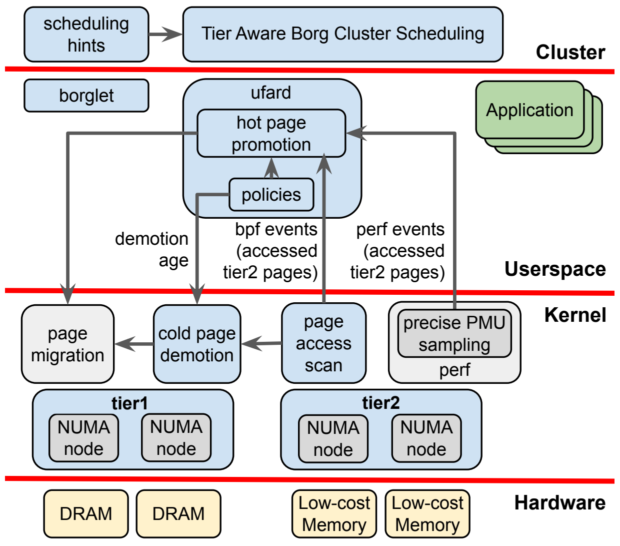


Figure 2: Base architecture of TMTS.

the node agent, Borglet, and manages a continuous stream of multi-machine job requests, observing load and performance metrics on each machine and dispatching tasks to individual servers [48]. In TMTS, it employs tier-aware scheduling policies for better job placement, described in Section 8.2.

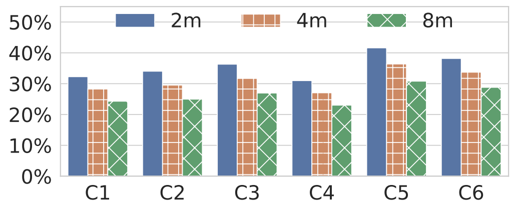
Linux enumerates and initializes tier2 memory devices as NUMA memory nodes. TMTS defines a new tiered hierarchy over memory nodes. This results in two groupings for the current deployment, but is expandable to handle more than two tiers or alternative memory hierarchies, such as CXL attached devices.

We use measured access patterns to migrate pages between tier1 and tier2, *demoting* infrequently accessed "cold" pages to lower tiers and *promoting* frequently accessed "hot" pages to upper tiers. Both involve page migration using existing kernel mechanisms [18]. A kernel daemon [32] periodically scans page access bits to determine page idle age and identify cold and hot pages. Cold page demotion largely follows the pattern used for conventional virtual memory [21]. Hot page promotion is more novel: no page fault event serves to trigger it. Memory that is mapped to tier2 can be accessed directly, but potentially substantial application slowdown incurs if frequently accessed. Hot usage is determined through system observation of access frequency. This aspect has much in common with NUMA node migration, but the penalties for hot page mischaracterization are much greater. Cold page detection inherently needs only infrequent scans, while rapid detection is critical for newly hot pages. TMTS uses precise hardware sampling of tier2 memory access events through the hardware performance monitoring unit (PMU) to enable timely detection.

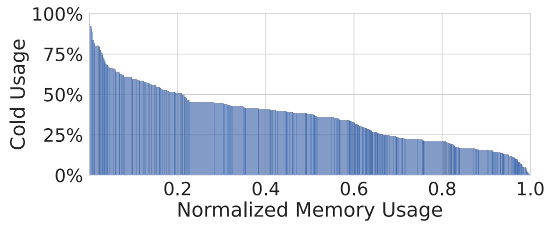
A key design aspect is the division of responsibilities and information exchanged between the kernel and the userspace daemons, Borglet and ufard, responsible for promotion and demotion policies. Later sections investigate our evidence-driven approach to determine effective policies using the rapid exploration this separation enables. Those explorations set the stage for more sophisticated adaptive policies, described later.

3.1 Cold Page Detection and Demotion

Page demotion requires identifying and moving most infrequently accessed pages from tier1 to tier2. We classify a page as *cold with threshold t* when it has not been accessed in the prior *t* seconds [32]. Figure 3 shows pages classified as cold for various thresholds and aggregate workloads across several large clusters used in this study. This data is inline with cold memory patterns reported by other large WSCs [32, 52] and drives the design point of 25% tier2 capacity. We provide more detail about the workloads, their scale and mix in Section 4. Figure 3(b) gives a visual sense of how cold memory distributes throughout the workloads. Jobs are ordered by cold memory ratio at the 2-minute threshold and the area of the bar is the portion of total memory usage due to the job. Most of the workload has ample cold usage for the 25% tier2 design point, but jobs vary significantly. The demotion policy choice determines the value of *t* and its granularity by application class, per application, or time varying. A demotion mechanism detects candidates according to such policy and migrates selected ones.



(a) Cold memory ratio across 6 clusters at 2m, 4m, 8m ages



(b) Cold memory ratio by task memory usage at 2m age

Figure 3: Distribution of cold memory normalized to total memory usage.

The daemon, ufard, conveys demotion policy to the kernel by specifying the cold age threshold in each task's cgroup. The kernel uses these per-application values to select pages to demote during scan. To enable proactive policies, the kernel provides the userspace daemon a *cold age histogram* [32] - the frequency distribution of inter-access interval duration. It answers questions such as how many pages were not accessed for at least 2 minutes. The policy engine uses this to identify application access patterns and adjust parameter values.

Since HILS and non-HILS co-locate on a machine and demotion consumes constrained tier2 bandwidth, TMTS avoids bursty demotions that may interfere with HILS applications. TMTS prefers to penalize non-HILS, including potentially evicting them from the machine to ensure minimal impact to HILS application tail latencies.

Linux treats file-backed pages and swap-backed pages differently and other work [52] shows considerable complexity in swapping file-backed pages. Applications in our environment make extensive use of distributed storage and, hence, rely on swap-backed memory (anonymous and `tmpfs`) for more than 98% of their pages. Thus, only swap-backed pages are treated as demotion candidates.

Tiering gives rise to memory management concerns roughly analogous to false sharing where hot and cold objects are mixed in a page, exacerbated by hugepages. Techniques to address this issue are examined in the empirically-driven policy development of later sections.

3.2 Promotion of Hot Pages

Page promotion requires identifying and moving frequently accessed pages from tier2 to tier1. While the page faults required in swap-based approaches [32, 52] can serve to identify candidates on first access, microsecond penalties on the critical path is of deep concern. Also, for rarely accessed pages, promotion on first access may be counterproductive. Multiple accesses over a short interval provide a stronger indication of change in usage of a previously cold page. Given the latency-critical nature of our HILS applications, promotions must be as non-disruptive as possible. Therefore TMTS focuses on proactive fault-less methods to rapidly identify promotion candidates, and uses *memory access event sampling* through the hardware PMU coupled with *proactive and periodic scanning* of the page accessed bits (A-bits).

Sampling: Event based sampling enables timely identification. Modern CPUs support precise event based sampling, such as PEBS on Intel platforms [16] and IBS on AMD platforms [19]. We use precise event based sampling on Intel to profile recently accessed addresses in tier2 memory by sampling last level cache (LLC) miss events. Since tier2 is cachable, only LLC miss events need be considered. Sampling all LLC miss events is impractical and not useful. The vast majority of LLC miss traffic targets tier1, which is not pertinent to promotion. The hardware deployed with TMTS supports precise events filtered to loads sourced from tier2. Unfortunately, the hardware does not support such filtering for memory stores, so stores are not sampled. We configure sampling to collect 1% of memory loads from tier2 and promote all the pages identified by this sampling. We examine the effectiveness of this pragmatic, albeit imperfect detector in Section 6.2.

Scanning: To detect hot pages that may be missed by sampling, we also perform proactive, periodic scan-based promotion. We define the page *hot age* as number of scan periods during which the page was recently accessed. We extend our in-kernel page A-bit scanner to track page hot age which allows it to differentiate between actively and lightly accessed pages. In the current deployment, the hot scan period is configured to 30 seconds to strike a balance between scan overhead and timeliness of identification (see Section 6.2 for more detail). The base policy promotes pages touched in two or more consecutive hot scan periods. For efficiency, we do not do TLB invalidations in A-bit scans. Even though this optimization may sacrifice some page age accuracy, we have not seen it impacting demotion and promotion effectiveness in practice.

The deployed demotion policy ensures that a page resident in tier1 has not been accessed for an extended period (O(min)) prior to a subsequent demotion. This allows the system to serve short lived allocations from tier1 and avoids thrashing, where pages repeatedly move between tier1 and tier2 consuming scarce tier2 bandwidth.

Table 1: tier2 DIMM Bandwidth

Access Pattern	Bandwidth (GB/s)
Sequential Read	6.9
Sequential Write	2.2
Random Read	1.8
Random Write	0.5

3.3 Policy Management

The policy daemon in TMTS, ufard, uses the mechanisms provided by the Linux kernel to actuate page migration policies. It uses the standard `perf` interface in Linux (e.g., `perf_event_open()`) to set up sampling and process access events. It also installs a small BPF [3] program into the kernel to optimize the collection of tier2 hot page ages and their page addresses from the in-kernel page A-bit scanner into per NUMA node BPF ring buffers. ufard selects tier2 resident pages that meet the hot age threshold for promotion. The system currently manages page events with physical page addresses and promotes selected physical pages through a custom system call.

3.4 Policy Constraints due to Hardware Restrictions

Our deployment uses a variant of Intel Optane Persistent Memory as tier2. Such tier2 DIMMs have highly constrained bandwidth, supporting < 1/10 the memory bandwidth of the DDR4 channels typically on Intel® Xeon® Scalable Processors. In the current hardware implementation, tier2 DIMM bandwidth saturation also impacts regular DRAM latency as the tier2 DIMMs share a memory channel with DRAM DIMMs. This limits promotion and demotion aggressiveness. Table 1 lists the measured bandwidth of tier2 DIMMs in our configuration under different access patterns. The measured idle read access latency is about 325ns, cf, [28, 55] for details. Based on cold page profiles (Figure 3), available DIMM capacity, and hardware bandwidth constraints, we target 25% of system memory capacity for tier2. These hardware limitations result in the following policy decision constraints for our current deployment.

No direct allocations into tier2: This constraint avoids scenarios where newly allocated pages (with no access history) in tier2 end up hot and accessed before promotion. The system allocates task memory only into tier1 and relies on demotions to populate tier2. Allocating only into tier1 can increase OOM conditions since all allocated pages occupy tier1 until pages are identified as cold and demoted.

No demotions to remote socket tier2: Hardware Quality of Service (QoS) features limit the impact to DRAM latency under high tier2 bandwidth demand. QoS depends on feedback from the memory controller to the requesting CPU core. QoS is less effective at throttling accesses to tier2 DIMMs attached to the remote socket because the feedback signal must travel between sockets. The drop in effectiveness (shown in Section 6.3) is large enough that we only demote the memory of tasks to tier2 DIMMs on the socket where the task is running.

These choices are not fundamental to TMTS, but an artifact of the underlying hardware implementation. Future hardware implementations may not suffer from these restrictions thus relaxing the constraints discussed above. Current constraints have the side-benefit of demonstrating the resilience of the system design.

4 EVALUATION METHODOLOGY

We have deployed TMTS in a portion of our fleet across multiple clusters serving production services for nearly two years. We use 2-socket Intel Xeon servers with DDR4 DIMMs as tier1 and a variant of Optane DIMMs as tier2.

We performed a series of A/B experiments using 2200 experimental servers and 1000 control servers deployed across 6 clusters spread geographically and serving hundreds of thousands of production services daily over this period utilizing a subset of the machines for various experiments. The experimental machines (2-tier) have 75% of the memory capacity served by a DDR4 tier1 and the remaining 25% of memory capacity served by tier2 DIMMS. The control machines (1-tier) use the same server configuration and total memory capacity as the experimental machines, except have all-DDR4 memory. Each cluster has a mix of control and experimental machines.

The Borg scheduler schedules production jobs across all application classes (HILS and non-HILS) onto both control and experimental machines assigning tasks from various production services to sets of machines using the same algorithm as in the rest of the fleet, (i.e., without any experiment-specific scheduling policies including the ones described in Section 8.2). Less than 1% of workloads in these clusters avoid tier2 for business reasons. Jobs routinely run on both experimental and control machines. The dynamic nature of scheduling constraints may lead to an imbalanced scheduling of jobs across these machines, which is inherent to such a complex computing environment. We ensure causality of inferences by comparing task replicas from jobs running on experimental machines for each service with those running on control machines from the same service.

The total study population used in evaluation consists of over 100K distinct jobs, roughly a quarter HILS, for which we have over 1M distinct HILS application instances and 330M non-HILS. Of these, 70% of HILS and 90% of non-HILS have sufficient coverage of experimental and control machines to match for use in comparative analysis (cf, Figure 6). We normalize the performance impact to the job's compute footprint to evaluate how well the stack works for the applications that are most critical to target metrics. Experiments are run for an extended period, typically about 4 weeks, to gain adequate statistical power given the high diversity of the workload. The mean requested CPU limit for HILS is twice that of non-HILS. It has a narrower stddev (1.24x mean, vs 1.7) but much larger P99 (5.3x, vs 2.0). CPU usage shows a wider dispersion. Memory limits show a similar picture, except non-HILS P99 being ten times the mean. The sub-population used for comparative analysis shows a similar distribution to the full population. The workload characteristics are consistent with that detailed in [48], which calls out the extreme diversity that makes population-based analysis at scale essential and is not captured in stand-alone benchmarks.

We measure the effectiveness of the deployed system using various metrics (including STAR and STRR). We compute a confidence interval and control the false positive rate at 5%.

5 EMPIRICAL RESULTS

The effectiveness of the 2-tier design is reflected in the product of total task capacity and per-task performance. Figure 1, introduced above, provides an overall multi-cluster, total workload assessment relative to our primary system goal - identifying 25% of active address space for demoting to the secondary tier while maintaining the overall performance impact to under 5%. It shows that the performance goal is met at a STAR of 0.5% in the current deployment. This section establishes the empirical basis underlying these results and shows their variation across the workload by building from the constituent proxy measures to the resulting overall behavior for a single policy and then investigates policy effects. The data presented in this section is collected from a slice of 300 servers across 6 clusters during a 5-week period.

5.1 Memory Utilization / Task Capacity

As applications have widely different, time-varying memory and computing demand, memory utilization serves as a proxy for obtained computing capacity. If tier2 utilization causes performance degradation so as to threaten SLOs, the Borg scheduler will schedule less load onto the machine, thereby reducing memory demand and resulting utilization. Figure 4 shows memory utilization for 2-tier (experimental) and 1-tier (control) machines over all clusters for the experiment duration. The distribution of memory utilization is similar in both groups: averaging 75%, 77% at the median (P50), 80% at the mode, and 91% at the tail (P95). This strongly corroborates that the same task capacity is maintained in both groups.

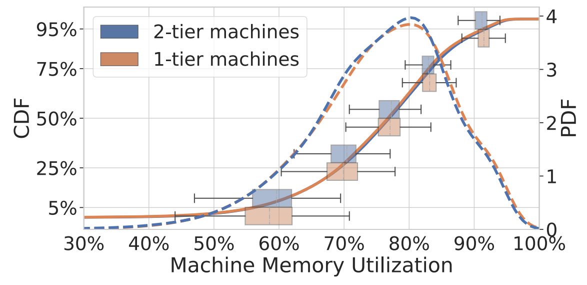


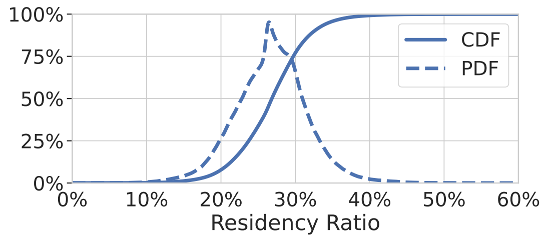
Figure 4: Distribution (PDF, CDF, variation) of memory utilization on 2-tier and 1-tier machines.

Memory utilization varies substantially across machines and over time. To demonstrate that the deployment maintains capacity across operational variation, Figure 4 overlays the CDF of memory utilization of the total experiment duration with box plots of the percentiles (P5, P25, P50, P75 and P95) across the series of time-windows in the observation period. For example, the memory utilization at P50 is 77%, but at different points of time, the P50 varies between 70% and 83% for 1-tier machines.

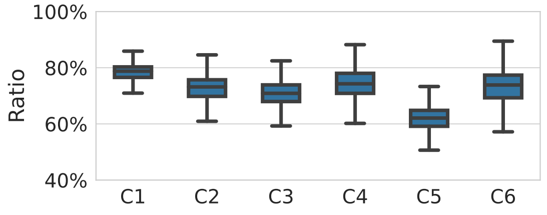
The similarity of the memory utilization trends indicates the tiering stack maintains task capacity stably across the complicated and dynamic machine utilization reality in a WSC. At lower utilization strata, we see slightly higher utilization and lower variation in the 2-tier machines than in the 1-tier. At upper strata (P90) 2-tier machines have slightly lower memory utilization than that of 1-tier though well above 80%. We measured CPU utilization and total task counts on the 2-tier and 1-tier machines and found they aligned with the memory utilization trends.

5.2 Residency Ratios

Given the similarity of overall memory utilization, we focus on the 2-tier machines and how the pages resident there relate to available cold pages.



(a) Tier2 residency ratio



(b) Cold memory coverage

Figure 5: Tier2 residency ratio and cold memory coverage.

Figure 5(a) shows the overall STRR. The mode, mean, and median are slightly above the 25% of deployed hardware capacity ratio, showing that the promotion/demotion strategy closely tracks this target. Considerable variation is present, but P10-P90 operate within 20-35% utilization.

An important reference point is the relative amount of cold pages available in an address space, observed to vary between 28% and 42% in the experiment in Figure 3(a). We define *cold memory coverage* as the fraction of 2 minute cold pages stored in tier2. Figure 5(b) shows that this metric is relatively high especially compared to swap based solutions [32], consistently > 50% and averaging 75% across all 6 clusters. This demonstrates that TMTS is successful at targeting a substantial portion of total addressable cold memory. Note that TMTS can adapt the promotion/demotion strategy to achieve any deployed tier2 hardware capacity ratio within bounds of the hardware performance characteristics.

5.3 Access Ratios / Bandwidth

Having established that overall task capacity is maintained and tier2 capacity is well-utilized, we turn to access frequency of pages resident in tier2.

Figure 6(a) shows STAR of 2-tier machines. The median access ratio is 0.3% and the P95 tail less than 1%. For reference, the congestion limit for the tier2 hardware occurs at STAR approximately 1.5%, discussed further below. Note that we do see a tail extending to 2%. The performance impact of this long tail STAR is also visible in Figure 6(b) and Figure 6(c) which capture the application performance impact. In Section 8, we discuss several strategies to mitigate such tails.

These results indicate that the promotion/demotion process is highly effective in keeping only cold pages resident in tier2. However, delay in detecting the increased reference rate can increase the amount of tier2 references, but the relationship is subtle because blocks of those pages are cached and the eventual promotion is more likely to operate from cache.

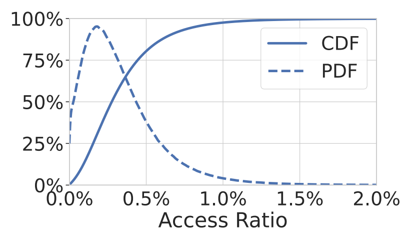
To better understand this behavior, Figure 7 breaks down the tier2 bandwidth usage to application traffic and promotion/demotion traffic relative to total tier2 bandwidth usage across the full operating range. For all but very low utilization where demotion dominates, about 80% of tier2 bandwidth is due to applications accessing pages resident in that tier, promotion being about 1/3 of the remaining and demotion 2/3. This suggests the system is effective in selecting pages for demotion while avoiding thrashing/ping-pong effects.

5.4 Overall Performance Impact

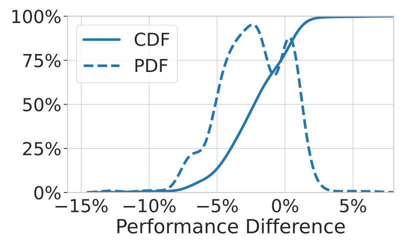
Putting the above together, the performance impact illustrated in Figure 1 can be broken down across the diverse production workloads. The principal figure of merit is the difference in instructions per cycle (IPC) obtained by a job on 1-tier control machines vs. that on 2-tier experimental machines, normalized by the IPC of the control group. This metric is assessed over the population of jobs in the production environment. Figure 6(b) shows the distribution of application IPC shift. The median IPC impact is 2.3%. At P5, the impact is about 7%. The impact is bi-modal with significant spread over -5% to 2%. The upper end 2% performance benefit has wide variance with 0 in the confidence interval (cf. Figure 6(c)) which is indicative of noise rather than a consistent pattern of IPC improvement.

The performance variability is within typical inter-platform performance variations seen in a WSC. Over various such studies, we see even the median impact ranges considerably, reflecting the inherent challenge of design evaluation on dynamic production workloads in a diverse fleet.

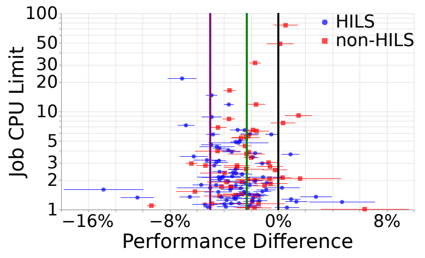
Figure 6(c) shows a per-application IPC difference for the largest applications comprising half of the total usage differentiated by application class. The jobs in control and experiment groups are matched and the % performance difference is calculated in aggregate for each. The horizontal line for each job shows the 95% confidence interval. The purple line at -5% shows the performance target. The green line at -2.3% shows aggregate IPC impact weighted by compute footprint. Note the log vertical scale reflecting the vast diversity in job size even after filtering for the largest jobs, and the substantial variation even for individual jobs. We monitored application-level performance metrics and observed the trends to be inline with the IPC distribution. While individual jobs may be treated as benchmarks to indicate the potential impacts of tiered



(a) CDF and PDF of tier2 access ratio



(b) CDF and PDF of weighted application IPC difference relative to 1-tier machines



(c) Weighted application IPC difference relative to 1-tier machines by application classes

Figure 6: Overall performance impact of 2-tier memory.

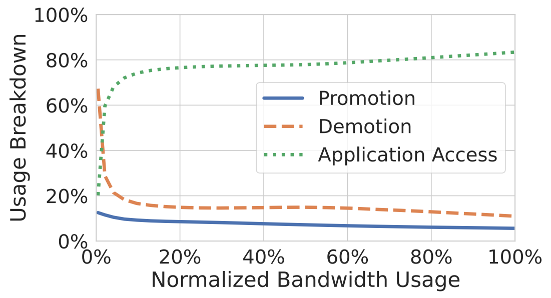


Figure 7: Bandwidth consumption of promotions, demotions and application accesses to tier2.

memory, the large variation means that population studies are necessary to determine the actual impact. Further, tail performance impact variability is critical for HILS applications and even more subtle to evaluate. We later discuss strategies to mitigate the tail.

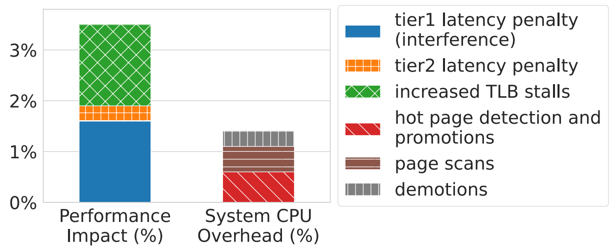


Figure 8: Tier2 performance impact and overheads.

Figure 8 breaks down the aggregate performance impact and TMTS overheads in a larger study on 1000 2-tier machines over a different 4 weeks, showing an aggregate performance impact of 3.5%. This degradation is larger than that in the smaller study above, but within target and expected variation. Some impact due to increase in latency is expected, given the 3.5x higher miss penalty on an L3 cache miss to tier2, but TMTS migrations keep it quite low. However, the increase in data translation lookaside buffer (TLB) misses and the tier1 DRAM latency increase deserves attention.

5.4.1 TLB Misses and Huge Pages. Huge pages (e.g. 2MB) mitigate the virtual-to-physical address translation latencies and TLB coverage problem posed by 4KB pages. But these sizes introduce a new challenge to page-migration based tiered memory systems. Migration results in TLB invalidations due to changes in address mappings. Additionally, a small hot region in a hugepage causes the entire page to appear hot.

In the current implementation, objects are initially allocated in tier1. If TMTS identifies a hugepage as a demotion candidate, the page is first split into 4KB pages and then demoted. This allows future accesses to promote individual 4KB pages and reduces migration cost and unnecessary occupancy of tier1. Not all demoted 4KB pages of an original hugepage may get promoted thus preventing them from successfully recombining into a hugepage.

However, "mostly cold" hugepages may never demote, reducing tier2 utilization and diluting tier1. This is exacerbated by optimizations to encourage high hugepage coverage (e.g. [27]), which improve performance but also increase the challenge of cold memory identification and migration.

Since the system has lower DRAM capacity, we see an increase in DRAM pressure, which fragments tier1. This reduces available 2MB blocks and translates to a lower hugepage coverage for the working set on 2-tier machines relative to the 1-tier ones. The resulting hugepage coverage is approximately 30% on 2-tier machines, versus 45% coverage on 1-tier machines. Note that the portion of the hugepage coverage gap related to cold pages being backed by 4KB pages is inconsequential to performance. However, the impact in hugepage coverage loss to application working sets can be seen in an increase of 7% in TLB misses on 2-tier machines compared to 1-tier machines, corroborating the causal explanation of the 1.5% impact.

To address the lower hugepage coverage and increase in TLB misses, we ran two experiments, both of which increased TLB hit rates with minimal side effects. First, we tried migrating hugepages intact, without breaking them apart into 4KB pages on demotion. This reduced TLB misses by 4.7% on average, and improved average performance by 0.5%. Promotion bandwidth was increased as expected, but by <1%, and without an increase in bandwidth congestion events. Average STRR was decreased by <1%. Second, we increased the aggressiveness of memory compaction, with the goal of increasing the rate of recombination of 4KB pages in tier1. We found that hugepage coverage was increased by 25% with an average performance improvement of 0.5%. The additional CPU cost was trivial at < 0.1%.

We explore more potential solutions to some of these issues in Section 8.3.

5.4.2 DRAM Latency Increase and Bandwidth Interference. Increase in DRAM latency results in 1.5% performance impact. When the Cascade Lake Optane DIMM controller's buffer becomes full, the current hardware implementation causes it to stall the memory channel shared with other DRAM devices [57]. This in turn interferes with other channels on the socket. More recent Optane Series 200 provide greater bandwidth per module and partial mitigation. We expect the introduction of CXL-based memory [1] to mitigate this impact.

5.4.3 System Overheads. The TMTS stack performs three key operations that add about 1% in system overhead - demotions, page scans and hot page detection+promotion (ufard) - and contribute to machine utilization. These overheads are typically absorbed as WSC servers operate below 100% utilization to improve predictability and meet SLOs. The increased utilization can potentially result in the Borg scheduler steering work elsewhere. This effect is reflected in the observed overall task capacity of the cluster, even though due to scheduling dynamics, it cannot be isolated on a per-machine basis.

6 EVALUATING POLICIES

The analysis of STAR, STRR, and cluster-wide performance impact above establishes how our demotion, promotion, and remote socket policies meet the joint objective of less than 5% degradation with 25% DRAM replacement. This section discusses the development of the baseline policy, i.e., the results above reflect the behavior of the final policies described here. Unless noted separately, we applied each policy to a slice of 300 machines spread across 6 clusters for a 2-week duration.

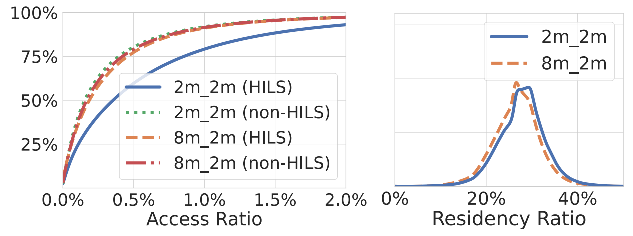
6.1 Demotion Policies

Increasing the cold age threshold for demotion decreases available cold page demotion opportunities, but also decreases scan overheads and TLB impact. The demotion policy goal therefore is to determine the coldest point that ensures a sufficient STRR. We evaluated several static policies to establish a baseline for the system and set the stage for future work in application-specific and dynamically-adaptive policies.

We evaluate the following static policies. 2m represents an aggressive threshold whereas 8m a conservative one.

- 2m_2m: A single static demotion age of 2 minutes for all classes.
- 8m_2m: A static demotion age of 8 minutes for HILS and 2 minutes for non-HILS.

The CDF of tier2 access ratio in Figure 9(a) shows that with a 2m_2m policy, HILS applications enter the congestion threshold of 1.5% STAR even at P80, while few non-HILS experience such a high STAR. The 8m_2m policy brings HILS STAR on par with non-HILS and within target range: median and P95 STAR reduce by 46% (from 0.35% to 0.19%) and 39% (from 2.35% to 1.43%) respectively.



(a) CDF of tier2 (b) PDF of tier2

Figure 9: Distribution of tier2 access ratio and tier2 residency ratio under 2m_2m and 8m_2m demotion policies.

Figure 9(b) shows that we maintain residency with the stricter cold criteria for HILS, although with a slight decrease, about 1.1% at the median. Applying a simple application class specific demotion policy meets our performance targets while allowing sufficient cold pages to be demoted.

6.2 Promotion Policies

Promotion policies involve subtle trade-offs scarcely considered in prior literature. Fault-based approaches like zswap[32, 52] promote immediately on first access. Non-faulting approaches enabled by a directly addressable tier2 allows the application to continue execution while accessing (and caching) no-longer-cold tier2 pages. These accesses may experience high miss penalties and induce interference.

We lack a hardware mechanism to directly detect such tier2 bursts. Promotion dynamics differ greatly from demotion. Since cold pages have been cold for a long time and are likely to continue to be cold, these pages have little sensitivity to demotion timing. However, a newly referenced page is likely to be referenced again soon and frequently.

While basic policies leverage the scan process used for demotion, memory access event sampling ensures timeliness of promotions (Section 3.2). We examine three policies and show the importance of fast detection.

- 60s: Promote pages with memory accesses detected in 2 subsequent 30-second page scans.
- 30s: Promote pages with a memory access detected in a single 30-second page scan.
- 60s+PMU: Promote pages with any access detected using PMU-based sampling OR with page accesses detected in 2 subsequent 30-second page scans.

Promotion latency is the time between a tier2 access and subsequent promotions to tier1. Figure 10 shows the CDF of the promotion latencies for the three promotion policies. The 30s and 60s policies exhibit approximately linear sections expected under evenly distributed wait times between memory accesses. PMU-based monitoring triggers promotions much sooner, especially at median, with 50% of promotions latencies <1 second, compared to about 13 seconds (30s policy) and about 25 seconds (60s policy) respectively.

Figure 11 shows the promotion policy impact to STRR and STAR metrics. All policies maintain the target residency. The aggressive

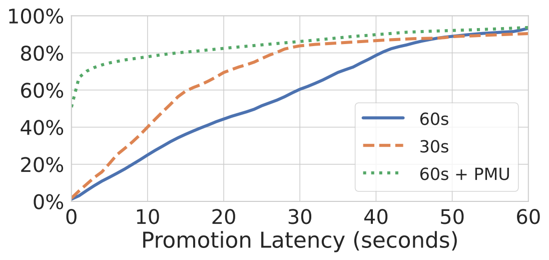
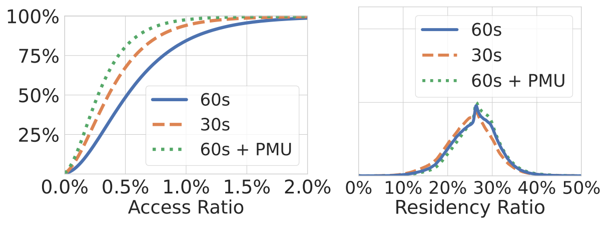


Figure 10: CDF of promotion latencies under different policies.



(a) CDF of tier2 (b) PDF of tier2

Figure 11: Distribution of tier2 access ratio and tier2 residency ratio under different promotion policies.

30s policy reduces the amount of memory eligible to be demoted by 10% compared to the 60s policy. The 60s+PMU policy preferentially detects and promotes hotter pages quicker than the conservative 60s. PMU-based promotions decreases STAR by 45% at the median and 44% at P95, compared to the 60s policy alone and is superior to the 30s policy.

6.3 Remote Socket Policies

Multi-socket servers add additional complexity to policy consideration. While we disallow remote socket demotions (Section 3.4, referred to as NUMA jailing hereafter), we verify this decision by observing behavior when we disable NUMA jailing on 300 experimental machines in a cluster. Figure 12 shows the PDF of memory access latency to tier1 DRAM under the two NUMA policies. While median DRAM access latency is roughly similar across the two machine sets, tail latency is significantly elevated (by orders of magnitude) when remote demotion is permitted. The effects were so pronounced that we observed a significant elevation in kernel soft lockups due to excessive slowdown and had to rollback this configuration within a few days to avoid impact to production services.

7 TWO APPLICATION CASE STUDIES

To see beyond aggregate workload behaviors, this section picks out two specific applications to demonstrate the unique challenges posed by the diversity in a WSC. The first is Spanner[15], a distributed relational database service with in-memory caches that serves millions of operations per second and is in the critical path of many latency sensitive production services. The second is a throughput-oriented machine learning training pipeline built using the TensorFlow framework [6], with high memory bandwidth usage and unpredictable access patterns.

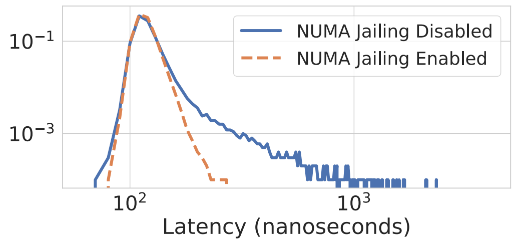


Figure 12: PDF of DRAM access latency in nanoseconds under NUMA jailing enabled and NUMA jailing disabled policies.

7.1 Database Application

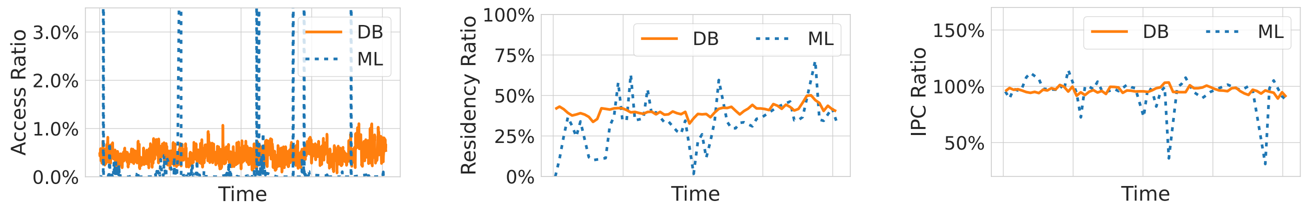
Spanner is a distributed database application that runs at the HILS application class. Much of Spanner's memory footprint is used for caching. Most cache entries are accessed relatively rarely, so this data is a good candidate for tier2 memory. The remaining fraction of Spanner's memory is part of its "control plane", used for metadata that is accessed for nearly every request. This data must be kept in tier1 for latency reasons. We expect Spanner's access patterns are similar to open source databases like CockroachDB [46] or Cassandra [33], which use similar storage engines to Spanner.

Figure 13 shows a time series of our key metrics from task replicas of this application running on experimental and control machines in one cluster over a period of one week. As expected, it demonstrates a consistently low STAR of 0.5% (within our target) while enabling identification of significant cold memory pages suitable for demotion. It can support a higher than expected STRR (>25%) which increases tier2 utilization. It sees an average IPC impact of 2.5% with tail latency impact at 4.5%, well below the aggregate impact observed across all HILS applications. The stable tier2 residency ratio makes estimating tier2 demand easy.

7.2 Machine Learning Application

The machine learning application is a throughput-oriented training pipeline that makes frequent updates to the ML model in memory during the training phase. The in-memory data stored by this application can be dense or sparse depending on how much of it is required to update the ML model that is currently being trained. These ML model updates are memory bandwidth intensive and their memory access patterns are unpredictable because the application is optimized to efficiently read the training data from disk into memory as opposed to optimizing for locality of the in-memory accesses. Any slowdowns to a single instance of the training worker could slow down the entire training phase of the application leading to wasted CPU and accelerator cycles.

This application poses a challenge to our system because the tier2 bandwidth is a poorly-isolated and constrained resource. The unpredictable access patterns prevent the tiering stack from staying ahead of bandwidth congestion events driven by burst of tier2



(a) tier2 access ratio (b) tier2 residency ratio (c) Application IPC impact

Figure 13: Distribution of tier2 access ratio, tier2 residency ratio and IPC impact over time for task replicas of a database application and a machine learning application.

accesses. Figure 13 shows a time series of our key metrics collected from replicas of this application running across experimental and control machines over a period of one week. It has a noticeably elevated STAR and significant variance in access rate, which is reflected in the large IPC impact, sometimes even 50% degradation. It also demonstrates a highly variable tier2 residency ratio ranging all the way from 0% to >50%. This degree of variability makes this application unfriendly for using tier2. Additionally, this application contributed to >80% of severe bandwidth saturation events on all experimental machines making it a noisy neighbor on multi-tenant systems.

8 ADAPTIVE POLICIES TO TACKLE WSC SCALE

While the deployed TMTS meets performance targets for most applications, several outliers experience significant performance degradation, shown in Figure 6(c). The case studies above demonstrate the diverse mix of cold memory access patterns typical to our computing environment. To scale tiering in such an environment, we must enable vertical integration across the node, cluster and application layers and develop policies that collectively enable higher cold memory identification with minimal performance degradation. This section discusses three adaptive policies that enabled scaling TMTS.

8.1 Secondary Tier Access Directed Demotions

The analyses above highlight the need to limit STAR bursts in order to reduce the impact to tail performance. Policies evaluated in Section 6 use relatively simple criteria to select pages for demotion, but do not adapt to potentially detrimental conditions. This section outlines adaptive policies to maintain STAR within a target range in an online manner.

Lagar-Cavilla et al [32] describe a dynamic, per-application cold age threshold policy that seeks to maintain a target promotion rate. We use a similar approach, though the differences are deeply rooted in the underlying mechanism. For swap based secondary tiers, the application impact occurs entirely on first access to a page after being demoted. This causes a fault that suspends the instruction for the duration of the page load or decompression. Subsequent accesses are to faster DRAM. With a direct-access secondary tier, the first access is a longer-latency cache miss rather than a fault, and further accesses may pay a cost until the page is promoted. Moreover, promotion detection relies on completely asynchronous page access monitoring.

To build policies, we augment the *cold age histogram* passed to ufard with a *promotion histogram* based on the periodically monitored PMU counters to estimate the potential STAR for each application at various cold ages. A simple control loop periodically determines the smallest demotion age that permits STAR to remain within target. To obtain a stable threshold and avoid page thrashing, the history of demotion ages is tracked to choose the K-th percentile age for each cycle. Early results from deploying this policy to 200 experimental machines in one cluster with a target STAR of 0.5% demonstrated high effectiveness in reducing the number of bandwidth saturation events (by >50%).

Future work would identify additional environmental signals to incorporate (such as tier2 bandwidth saturation). As some applications exhibit time varying access patterns, the target range itself can be specific to the application and adjustable. Driving demotion decisions from userspace enables such application-specific optimizations while also allowing for rapid iterations over the parameter space.

8.2 Secondary Tier Aware Cluster Scheduling

The diverse nature of applications in a WSC requires solutions specifically to tackle variability. The scheduler assigns tasks to machines based on resource availability. The Borg scheduler runs two phases - *feasibility* checking, to find the set of machines where the task is able to fit and run and *scoring*, which picks the best among feasible machines. If the cluster scheduling policy is oblivious to tier2, it may produce suboptimal task placements, such that cold memory across tasks co-located on a given machine is insufficient to occupy the deployed tier2 capacity so the resource is stranded. However, the scheduler operates over time scales that are significantly larger than variations of STAR. Moreover, the scheduler needs to maintain tight SLOs on scheduling latency while managing hundreds of tasks per machine [48], each with unique access patterns and tens of thousands of machines per cluster with heterogeneous tier2. TMTS employs an end to end solution to maintain overall application SLOs by distributing responsibilities between node and scheduler layers.

The node agent, Borglet, can observe and quickly respond to application performance impact. Borglet is also aware of the specific performance capabilities of the tier2 hardware. Thus it maintains the responsibility to protect the performance SLOs of the applications that are collocated on a given machine under the constraints of the tier2 hardware provisioned on that machine. By employing the demotion policy described in 8.1, Borglet in conjunction with ufard maintains STAR within expected range for each application. If a machine starts to run out of sufficient tier1 capacity to fit the hot workingset of the scheduled tasks, Borglet temporarily signals to the scheduler to reduce load. Similarly, if the combined bandwidth usage across different tasks on a machine exceeds the tier2 bandwidth capacity (even if each task's STAR is within permissible thresholds), it can result in tail performance impact due to bandwidth congestion. If there is a persistent congestion situation detected on a machine, Borglet notifies the scheduler to evict tasks in an SLO-safe manner.

For reliability purposes, Borglets do not communicate with one another directly. Thus, the scheduler is the only entity that has a centralized view of all machines and available workload in a cluster. It has the responsibility of optimal workload collocation. We introduce *scheduling hints*, a workload segmentation mechanism to guide the Borg scheduler to preferentially schedule some jobs (and conversely, avert some jobs) on machines with tier2 memory. An offline pipeline uses historical analysis of cold memory access patterns and performance by jobs to determine jobs suitable to use tiered memory systems versus unsuitable ones, (e.g., access patterns that make them noisy neighbors or extreme sensitivity to latency.) We use tier2 usage ratio and tier2 bandwidth usage as criteria to categorize jobs as *friendly*, *unfriendly* and neutral. Friendly jobs are preferred to schedule on 2-tier machines while unfriendly jobs are preferred to schedule on 1-tier machines. Neutral jobs are applied the same scheduling policy as in Section 5. Deploying these heuristics, we observed on average about 17% of jobs in a cluster to be *unfriendly* and about 20% of jobs to be *friendly*. The fraction of *friendly* and *unfriendly* jobs varies across different clusters, as shown in Figure 14. Note that these hints are applied as a best effort policy and sometimes the scheduler can't follow them due to stronger constraints such as CPU and RAM capacity. This design choice was made to prevent risk of violating job scheduling SLOs which are higher priority than potential tail performance concerns.

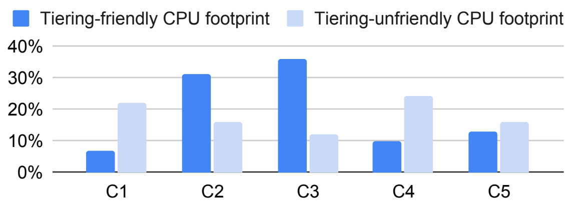
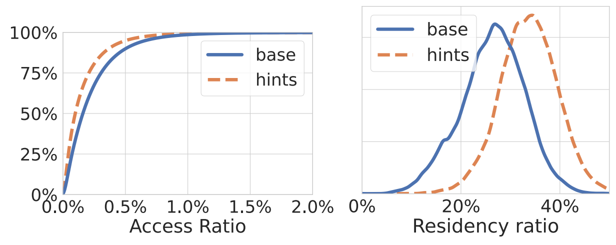


Figure 14: Distribution of tiering-friendly and unfriendly job footprints by cluster.

During the *scoring* phase of the scheduler, a tag applied by the offline pipeline is considered in addition to existing scheduling criteria to determine the best workload mix for each machine. The offline pipeline runs continuously to update the tags in response to changes in workload behavior over time and any tag changes are pushed to the scheduler via existing automation at a predetermined cadence. Additionally, the scheduler considers the tier1 utilization of machines to spread the hot workingset more evenly across machines to reduce the likelihood of node memory pressure.

Early experiments with the above scheduling policies applied to 600 machines in 1 cluster demonstrated (Figure 15) a 30% reduction in STAR resulting in a 4% improvement in performance for latency sensitive workloads relative to machines with tier2 that did not have these policies applied.

STRR being maintained within expected range is critical at the cluster level while making deployment decisions due to implications on resource stranding. While no scheduling or demotion policy optimizes for a specific STRR in real time, provisioning decisions are made by simulating the combined set of policies to determine the optimal mix of tier1 and tier2 capacities to deploy to each cluster to maximize TCO savings considering stranding implications.



(a) CDF of tier2 (b) PDF of tier2

Figure 15: Distribution of tier2 access ratio and tier2 residency ratio with and without scheduling hints.

8.3 Application-Hinted Cold Memory Allocation

Section 5.4 discusses the interplay of hugepages and memory tiering, especially for mostly cold hugepages. Initially, the Spanner database exhibited 10% cold memory due to the collocation of hot and cold objects within a hugepage, preventing it from being identified as cold. To avoid collocating hot and cold objects, we extended the new operator to accept a hint parameter indicating how frequently accessed the allocated object is expected to be. The open-source TCMalloc implementation [4] for the C++ memory allocator uses this parameter to separate "cold" and "hot" allocations in the virtual address space. Since infrequently accessed objects were clustered onto pages together, the kernel could be advised to not map cold regions with transparent hugepages. These strategies lead to a separation of frequently accessed and infrequently accessed objects and distinct policies for each.

We annotated 2 allocation sites with the cold hint parameter in Spanner, which resulted in an increase of cold memory identification from 10% to 42%. More densely packing frequently accessed allocations reduced the number of TLB entries required to span the working set, simultaneously reducing query latency by 2% by reducing TLB misses.

We believe similar approaches that use software hints to help shape memory allocation and object placements within pages will be increasingly important. For example, in addition to developers annotating allocations within applications as likely hot or cold, the compiler may use automated techniques such as profile-guided optimizations to provide similar hints.

9 RELATED WORK

Tiered memory systems. Intel Optane memory mode [2] implements OS-transparent tiered memory systems in hardware, on which software cannot customize tiering policies for various applications. [38, 42, 51] are hardware-assisted tiered memory systems, which are evaluated in simulation. [7, 24, 26, 29, 31, 37, 56, 58] manage tiered memory systems transparently to applications at the OS level, similar to our system. Nimble [56] focuses on optimizing migration of hugepages. HeteroOS [29] provides heterogeneous memory to guest OS and coordinates guest OS and VMM for page placement. Thermostat [7] uses page sampling to classify pages as hot or cold. [24, 26, 31, 37] extends page reclaim to demote cold pages in lieu of swapping, whereas our system applies demotion age policies specific to each application class. [24, 26, 31, 37, 58] also extends Linux NUMA balancing page migration to promote hot pages on minor hinting page faults. Such page fault based techniques can detect access to tier2 pages more quickly than A-bit scanning and memory access sampling, though at the cost of increased page faults and TLB invalidations, particularly for repeated page faults that these techniques incur when they don't always promote tier2 pages on the first access. AutoTiering [31] uses Optane as tier2 memory, while [7, 29, 37, 56, 58] are evaluated with emulated tier2 memory, which does not accurately reflect the performance impact of slower memory tiers. None of these systems are evaluated with production workloads in warehouse-scale data centers.

There are tiered memory systems managed at application or library levels, such as X-Mem [20], Unimem [53], AIFM [44], and pVM [30], which rely on custom memory APIs and software modifications. Among them, HeMem [43] is similar to our system in using Optane as tier2 memory and PEBS to track hot pages on Optane for promotion. Many techniques in these approaches, such as profiling [20, 51, 53], custom allocations and prefetching [10, 36], can be leveraged to optimize application performance in our system, similar to allocation hints in Section 8.3.

Swap-based far memory. Using swap to extend memory is well known. The swap target can be in-memory compression [32, 52], local disks [52], or remote devices via RDMA [8], Infiniband [23], or other interconnects. Cold page identification techniques [17, 41, 59] for page reclaim and swap are applicable to memory tiering as well. Some feedback techniques to control swap aggressiveness, such as PSI [52], may also be extended to be applicable to direct-accessed tiered memory systems. The order of magnitude difference in access latency between swap devices and slow tier memory demands different page placement policies as well.

Disaggregated memory. Many recent software runtime efforts for disaggregated memory [11, 45] are designed for RDMA over network, which has an order of magnitude higher latency than tiered memory attached directly or via CXL.mem. DirectCXL [22] and Pond [34] extend memory in a machine with rack-scale disaggregated memory via CXL.mem. Such hardware systems can be configured to be either a non-tiered memory system for capacity and bandwidth expansion [22] or a tiered memory system which our tiering software can be applied to. Moreover, Pond studies 158 applications in isolation, whereas we evaluate our system with over 100K applications in a dynamic production setup.

Cluster scheduling policies. Large scale cluster managers such as Mesos [25], Twine [47], YARN [49] and Borg [50] can schedule jobs across a large number of heterogeneous machines to optimize the efficiency of data centers. These cluster managers can optimize their scheduling policies to take into account the differences between memory tiers as illustrated in Section 8.2.

10 CONCLUDING REMARKS

This paper presents the first comprehensive analysis of a non-faulting tiered memory system deployed at scale in a warehouse-scale environment successfully serving production services across different application classes. We demonstrated the effectiveness of adaptive policies and hardware-assisted event profiling. Our experience highlighted the complexity of successfully managing a multi-tenant tiered memory deployment at scale in the presence of a diverse set of application services with varying SLO requirements. We expect the adaptive nature of techniques proposed in this paper built on solid foundational metrics to apply to future systems independent of how their latency and bandwidth profiles evolve and the interface they support (e.g., CXL). Future system parameters may indeed change STAR/STRR targets allowing for more aggressive demotions and higher tier-2 replacement ratios.

We have presented a data point in a spectrum of different solutions possible. The ever changing application space, usage models, and hardware technologies - both in media and interfaces will require further innovations in scalable memory tiering management. We expect a rich set of alternatives for cheaper tiers with different bandwidth and latency profiles, even with the discontinuation of Optane. New hardware mechanisms may enable rapid, accurate identification of promotion candidates. New approaches to object allocation, i.e., new, may dramatically reduce access fragmentation, thereby changing the trade-offs in the use of large/huge pages and even the hardware design points.

We hope our work provides a strong foundation for further innovation around future cluster-level tiering-aware scheduling policies.

ACKNOWLEDGEMENTS

The results presented in this paper build on the work of a great larger team at Google and we are grateful to all our colleagues involved in the design and operation of the systems that we discuss in this paper. In particular, we would like to thank Aleksandr Savchenkov, Biswa Panda, Brad Strand, Christian Warloe, Jennifer Mansur, Jesus Francisco Anaya Gonzalez, Kristine Jassmann, Lai Nguyen, Laura Xu, Linlin Li, Md Kamruzzaman, Peng Gu, Ravi Kavuri, Srividhya Balaji, Todd Lipcon, Yili Zheng and Zach O'Keefe for their invaluable contributions. We would also like to thank Eric Brewer, Hank Levy, Danner Stodolsky, the anonymous reviewers, as well as our shepherd for their feedback on the paper.

REFERENCES

[1] [n. d.]. Compute Express Link (CXL). https://www.computeexpresslink.org/.
[2] [n. d.]. Intel Optane Persistent Memory. https://www.intel.com/content/www/us/en/products/docs/memory-storage/optane-persistent-memory/overview.html.
[3] [n. d.]. Linux Kernel BPF Documentation. https://docs.kernel.org/bpf/index.html.
[4] [n. d.]. TCMalloc new Extension. https://github.com/google/tcmalloc/blob/master/tcmalloc/new_extension.h.
[5] 2020. CXL And Gen-Z Iron Out A Coherent Interconnect Strategy. https://www.nextplatform.com/2020/04/03/cxl-and-gen-ziron-out-a-coherent-interconnect-strategy/.

[6] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 265–283. https://doi.org/10.5555/3026877.3026899

[7] Neha Agarwal and Thomas F. Wenisch. 2017. Thermostat: Application-Transparent Page Management for Two-Tiered Main Memory. In *Proceedings of the 22nd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. https://doi.org/10.1145/3037697.3037706

[8] Emmanuel Amaro, Christopher Branner-Augmon, Zhihong Luo, Amy Ousterhout, Marcos K. Aguilera, Aurojit Panda, Sylvia Ratnasamy, and Scott Shenker. 2020. Can far memory improve job throughput?. In *Proceedings of the 15th European Conference on Computer Systems (EuroSys)*. https://doi.org/10.1145/3342195.3387522

[9] Luiz André Barroso, Jimmy Clidaras, and Urs Hölzle. 2013. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Second Edition*. http://dx.doi.org/10.2200/S00516ED2V01Y201306CAC024

[10] Christopher Branner-Augmon, Narek Galstyan, Sam Kumar, Emmanuel Amaro, Amy Ousterhout, Aurojit Panda, Sylvia Ratnasamy, and Scott Shenker. 2022. 3PO: Programmed Far-Memory Prefetching for Oblivious Applications. https://arxiv.org/abs/2207.07688.

[11] Irina Calciu, M. Talha Imran, Ivan Puddu, Sanidhya Kashyap, Hasan Al Maruf, Onur Mutlu, and Aasheesh Kolli. 2021. Rethinking software runtimes for disaggregated memory. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. https://doi.org/10.1145/3445814.3446713

[12] Youmin Chen, Youyou Lu, Fan Yang, Qing Wang, Yang Wang, and Jiwu Shu. 2020. FlatStore: An Efficient Log-Structured Key-Value Storage Engine for Persistent Memory. In *Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. https://doi.org/10.1145/3373376.3378515

[13] Edward G. Coffman and Peter J. Denning. 1973. *Operating Systems Theory*. Prentice Hall Professional Technical Reference.

[14] Douglas Comer and Jim Griffioen. 1990. A New Design for Distributed Systems: The Remote Memory Model. In *USENIX Summer*.

[15] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson C. Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. 2013. Spanner: Google's Globally Distributed Database. *ACM Transactions on Computer Systems* 31 (2013), 8. https://doi.org/10.1145/2491245

[16] Intel Corporation. 2023. Intel 64 and IA-32 Architectures Software Developer's Manual. https://software.intel.com/articles/intel-sdm.

[17] Vladimir Davydov. 2015. Idle memory tracking. https://lwn.net/Articles/643578/.

[18] The Linux Kernel Documentation. [n. d.]. Linux Memory Management Documentation - Page Migration. https://www.kernel.org/doc/html/v5.15/vm/page_migration.html.

[19] Paul J. Drongowski. 2007. Instruction-Based Sampling: A New Performance Analysis Technique for AMD Family 10h Processors. https://developer.amd.com/wordpress/media/2012/10/AMD_IBS_paper_EN.pdf.

[20] Subramanya R. Dulloor, Amitabha Roy, Zheguang Zhao, Narayanan Sundaram, Nadathur Satish, Rajesh Sankaran, Jeff Jackson, and Karsten Schwan. 2016. Data tiering in heterogeneous memory systems. In *Proceedings of the 11th European Conference on Computer Systems (EuroSys)*. https://doi.org/10.1145/2901318.2901344

[21] Mel Gorman. [n. d.]. Understanding the Linux Virtual Memory Manager - Page Frame Reclamation. https://www.kernel.org/doc/gorman/html/understand/understand013.html.

[22] Donghyun Gouk, Sangwon Lee, Miryeong Kwon, and Myoungsoo Jung. 2022. Direct Access, High-Performance Memory Disaggregation with DirectCXL. In *Proceedings of the USENIX Annual Technical Conference (USENIX ATC)*. https://www.usenix.org/conference/atc22/presentation/gouk

[23] Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang G. Shin. 2017. Efficient Memory Disaggregation with Infiniswap. In *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. https://doi.org/10.5555/3154630.3154683

[24] Dave Hansen. 2020. Migrate Pages in lieu of discard. https://lwn.net/Articles/824830/.

[25] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, and Ion Stoica. 2011. Mesos: A Platform for Fine-grained Resource Sharing in the Data Center. In *Symposium on Networked Systems Design and Implementation (NSDI)*. https://doi.org/10.5555/1972457.1972488

[26] Ying Huang. 2019. autonuma: Optimize memory placement in memory tiering system. https://lwn.net/Articles/803663/.

[27] A.H. Hunter, Jane Street Capital, Chris Kennelly, Paul Turner, Darryl Gove, Tipp Moseley, and Parthasarathy Ranganathan. 2021. Beyond malloc efficiency to fleet efficiency: a hugepage-aware memory allocator. In *Proceedings of the 15th USENIX Conference on Operating Systems Design and Implementation (OSDI)*.

[28] Joseph Izraelevitz, Jian Yang, Lu Zhang, Juno Kim, Xiao Liu, Amirsaman Memaripour, Yun Joon Soh, Zixuan Wang, Yi Xu, Subramanya R Dulloor, et al. 2019. Basic Performance Measurements of the Intel Optane DC Persistent Memory Module. CoRR abs/1903.05714 (2019). *arXiv preprint arXiv:1903.05714* (2019).

[29] Sudarsun Kannan, Ada Gavrilovska, Vishal Gupta, and Karsten Schwan. 2017. HeteroOS: OS Design for Heterogeneous Memory Management in Datacenter. In *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA)*. https://doi.org/10.1145/3079856.3080245

[30] Sudarsun Kannan, Ada Gavrilovska, and Karsten Schwan. 2016. pVM: persistent virtual memory for efficient capacity scaling and object storage. In *Proceedings of the 11th European Conference on Computer Systems (EuroSys)*. https://doi.org/10.1145/2901318.2901325

[31] Jonghyeon Kim, Wonkyo Choe, and Jeongseob Ahn. 2021. Exploring the Design Space of Page Management for Multi-Tiered Memory Systems. In *USENIX Annual Technical Conference (USENIX ATC)*.

[32] Andres Lagar-Cavilla, Junwhan Ahn, Suleiman Souhlal, Neha Agarwal, Radoslaw Burny, Shakeel Butt, Jichuan Chang, Ashwin Chaugule, Nan Deng, Junaid Shahid, Greg Thelen, Kamil Adam Yurtsever, Yu Zhao, and Parthasarathy Ranganathan. 2019. Software-Defined Far Memory in Warehouse-Scale Computers. In *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. https://doi.org/10.1145/3297858.3304053

[33] Avinash Lakshman and Prashant Malik. 2010. Cassandra: A Decentralized Structured Storage System. *SIGOPS Operating Systems Review* 44, 2 (apr 2010), 35–40. https://doi.org/10.1145/1773912.1773922

[34] Huaicheng Li, Daniel S. Berger, Stanko Novakovic, Lisa Hsu, Dan Ernst, Pantea Zardoshti, Monish Shah, Samir Rajadnya, Scott Lee, Ishwar Agarwal, Mark D. Hill, Marcus Fontoura, and Ricardo Bianchini. 2023. Pond: CXL-Based Memory Pooling Systems for Cloud Platforms. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. https://doi.org/10.48550/ARXIV.2203.00241

[35] Kevin Lim, Jichuan Chang, Trevor Mudge, Parthasarathy Ranganathan, Steven K. Reinhardt, and Thomas F. Wenisch. 2009. Disaggregated memory for expansion and sharing in blade servers. In *Proceedings of the 36th annual international symposium on Computer architecture (ISCA)*. https://doi.org/10.1145/1555754.1555789

[36] Hasan Al Maruf and Mosharaf Chowdhury. 2020. Effectively Prefetching Remote Memory with Leap. In *Proceedings of the USENIX Annual Technical Conference (USENIX ATC)*. https://doi.org/10.5555/3489146.3489204

[37] Hasan Al Maruf, Hao Wang, Abhishek Dhanotia, Johannes Weiner, Niket Agarwal, Pallab Bhattacharya, Chris Petersen, Mosharaf Chowdhury, Shobhit Kanaujia, and Prakash Chauhan. 2023. Aqua: Transparent Page Placement for CXL-Enabled Tiered Memory. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. https://doi.org/10.48550/ARXIV.2206.02878

[38] Mitesh R. Meswani, Sergey Blagodurov, David Roberts, John Slice, Mike Ignatowski, and Gabriel H. Loh. 2015. Heterogeneous memory architectures: A HW/SW approach for mixing die-stacked and off-package memories. In *IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. https://doi.org/10.1109/HPCA.2015.7056027

[39] Feeley Michael J, Wdliam E. Morgan, Frederic H. Pighin, Anna R. Karlin, and Henry M. Levy. 1995. Implementing Global Memory Management in a Workstation Cluster. In *ACM SIGOPS Operating Systems Review*. https://doi.org/10.1145/224057.224072

[40] Ismail Oukid, Johan Lasperas, Anisoara Nica, Thomas Willhalm, and Wolfgang Lehner. 2016. FPTree: A Hybrid SCM-DRAM Persistent and Concurrent B-Tree for Storage Class Memory. In *Proceedings of the International Conference on Management of Data*. https://doi.org/10.1145/2882903.2915251

[41] SeongJae Park. 2020. Introduce Data Access MONitor (DAMON). https://lwn.net/Articles/834721/.

[42] Luiz E. Ramos, Eugene Gorbatov, and Ricardo Bianchini. 2011. Page placement in hybrid memory systems. In *Proceedings of the International Conference on Supercomputing (ICS)*.

[43] Amanda Raybuck, Tim Stamler, Wei Zhang, Mattan Erez, and Simon Peter. 2021. HeMem: Scalable Tiered Memory Management for Big Data Applications and Real NVM. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP)*. https://doi.org/10.1145/3477132.3483550

[44] Zhenyuan Ruan, Malte Schwarzkopf, Marcos K. Aguilera, and Adam Belay. 2020. AIFM: High-Performance, Application-Integrated Far Memory. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. https://doi.org/10.5555/3488766.3488784

[45] Yizhou Shan, Yutong Huang, Yilun Chen, and Yiying Zhang. 2018. LegoOS: A Disseminated, Distributed OS for Hardware Resource Disaggregation. In *13th

USENIX Symposium on Operating Systems Design and Implementation (OSDI). https://doi.org/10.5555/3291168.3291175

[46] Rebecca Taft, Irfan Sharif, Andrei Matei, Nathan VanBenschoten, Jordan Lewis, Tobias Grieger, Kai Niemi, Andy Woods, Anne Birzin, Raphael Poss, Paul Bardea, Amruta Ranade, Ben Darnell, Bram Gruneir, Justin Jaffray, Lucy Zhang, and Peter Mattis. 2020. CockroachDB: The Resilient Geo-Distributed SQL Database. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 1493–1509. https://doi.org/10.1145/3318464.3386134

[47] Chunqiang Tang, Kenny Yu, Kaushik Veeraraghavan, Jonathan Kaldor, Scott Michelson, Thawan Kooburat, Aravind Anbudurai, Matthew Clark, Kabir Gogia, Long Cheng, Ben Christensen, Alex Gartrell, Maxim Khutornenko, Sachin Kulkarni, Marcin Pawlowski, Tuomas Pelkonen, Andre Rodrigues, Rounak Tibrewal, Vaishnavi Venkatesan, and Peter Zhang. 2020. Twine: a unified cluster management system for shared infrastructure. In *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation (OSDI).* https://doi.org/10.5555/3488766.3488811

[48] Muhammad Tirmazi, Adam Barker, Nan Deng, Md Ehtesam Haque, Zhijing Gene Qin, Steven Hand, Mor Harchol-Balter, and John Wilkes. 2020. Borg: the Next Generation. In *EuroSys'20.* https://doi.org/10.1145/3342195.3387517

[49] Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. 2013. Apache Hadoop YARN: Yet Another Resource Negotiator. In *ACM Symposium on Cloud Computing (SOCC).* https://doi.org/10.1145/2523616.2523633

[50] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. 2015. Large-scale Cluster Management at Google with Borg. In *Proceedings of European Conference on Computer Systems (EuroSys).* https://doi.org/10.1145/2741948.2741964

[51] Wei Wei, Dejun Jiang, Sally A. McKee, Jin Xiong, and Mingyu Chen. 2015. Exploiting Program Semantics to Place Data in Hybrid Memory. In *International Conference on Parallel Architecture and Compilation (PACT).* https://doi.org/10.1109/PACT.2015.10

[52] Johannes Weiner, Niket Agarwal, Dan Schatzberg, Leon Yang, Hao Wang, Blaise Sanouillet, Bikash Sharma, Tejun Heo, Mayank Jain, Chunqiang Tang, and Dimitrios Skarlatos. 2022. TMO: transparent memory offloading in datacenters. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS).* https://doi.org/10.1145/3503222.3507731

[53] Kai Wu, Yingchao Huang, and Dong Li. 2017. Unimem: Runtime data managementon non-volatile memory-based heterogeneous main memory. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis.* https://doi.org/10.1145/3126908.3126923

[54] Fei Xia, Dejun Jiang, Jin Xiong, and Ninghui Sun. 2017. HiKV: A Hybrid Index Key-Value Store for DRAM-NVM Memory Systems. In *Proceedings of the USENIX Annual Technical Conference (USENIX ATC).* https://doi.org/10.5555/3154690.3154724

[55] Lingfeng Xiang, Xingsheng Zhao, Jia Rao, Song Jiang, and Hong Jiang. 2022. Characterizing the Performance of Intel Optane Persistent Memory: A Close Look at Its on-DIMM Buffering. In *Proceedings of the Seventeenth European Conference on Computer Systems* (Rennes, France) *(EuroSys '22)*. Association for Computing Machinery, New York, NY, USA, 488–505. https://doi.org/10.1145/3492321.3519556

[56] Zi Yan, Daniel Lustig, David Nellans, and Abhishek Bhattacharjee. 2019. Nimble Page Management for Tiered Memory Systems. In *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS).* https://doi.org/10.1145/3297858.3304024

[57] Jian Yang, Juno Kim, Morteza Hoseinzadeh, Joseph Izraelevitz, and Steven Swanson. 2020. An Empirical Guide to the Behavior and Use of Scalable Persistent Memory. In *Proceedings of the 18th USENIX Conference on File and Storage Technologies (FAST).* https://doi.org/10.5555/3386691.3386708

[58] Zhuohui Duan; Haikun Liu; Xiaofei Liao; Hai Jin; Wenbin Jiang; Yu Zhang. 2019. HiNUMA: NUMA-Aware Data Placement and Migration in Hybrid Memory Systems. In *IEEE 37th International Conference on Computer Design (ICCD).* https://doi.org/10.1109/ICCD46524.2019.00058

[59] Yu Zhao. 2022. Multi-Gen LRU Framework. https://lwn.net/Articles/904697/.

Received 2022-10-20; accepted 2023-01-19