Telescope: Telemetry for Gargantuan Memory Footprint Applications

Alan Nair* *Intel Labs*
Sandeep Kumar *Intel Labs*
Aravinda Prasad *Intel Labs*
Ying Huang *Intel Corporation*
Andy Rudoff† *Intel Labs*
Sreenivas Subramoney *Intel Labs*

Abstract

Data-hungry applications that require terabytes of memory have become widespread in recent years. To meet the memory needs of these applications, data centers are embracing tiered memory architectures with near and far memory tiers. Precise, efficient, and timely identification of hot and cold data and their placement in appropriate tiers is critical for performance in such systems. Unfortunately, the existing state-of-the-art telemetry techniques for hot and cold data detection are ineffective at terabyte scale.

We propose Telescope, a novel technique that profiles different levels of the application's page table tree for fast and efficient identification of hot and cold data. Telescope is based on the observation that for a memory- and TLB-intensive workload, higher levels of a page table tree are also frequently accessed during a hardware page table walk. Hence, the hotness of the higher levels of the page table tree essentially captures the hotness of its subtrees or address space sub-regions at a coarser granularity. We exploit this insight to quickly converge on even a few megabytes of hot data and efficiently identify several gigabytes of cold data in terabyte-scale applications. Importantly, such a technique can seamlessly scale to petabyte-scale applications.

Telescope's telemetry achieves 90%+ precision and recall at just 0.9% single CPU utilization for microbenchmarks with 5 TB memory footprint. Memory tiering based on Telescope results in 5.6% to 34% throughput improvement for real-world benchmarks with 1–2 TB memory footprint compared to other state-of-the-art telemetry techniques.

1 Introduction

The rise of big data applications has resulted in an exponential increase in data volume being generated and processed. The memory footprints of applications in fields such as analytics, machine learning, databases, and high-performance computing exceed petabytes in size [12, 21, 38]. For example, Meta's database solutions mine information from geographically distributed databases spanning petabytes in size [12], and genome-sequencing workloads operate on in-memory data sets that span terabytes [58].

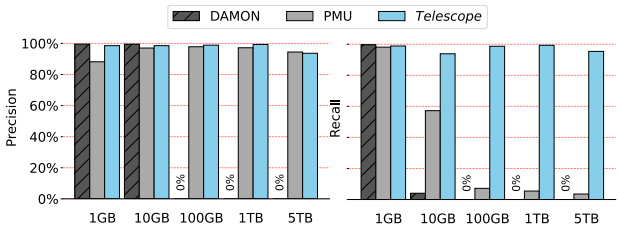


Figure 1: Telemetry efficiency based on precision and recall (§6.2). For state-of-the-art techniques, efficiency degrades quickly as the application footprint escalates.

Increasing the DRAM memory capacity of data center servers to accommodate the needs of big data applications is not a viable solution for two reasons. First, memory cost has already surpassed compute cost and now accounts for up to 50% of server cost in data centers [3, 54]. Increasing the DRAM capacity further can escalate the total cost of ownership (TCO) and directly impact cloud or data center economics. Second, prior studies in cloud data centers have shown that more than half of the data is not frequently accessed and hence are cold data [14, 40, 47, 50]. Using costly DRAM memory to store infrequently accessed cold data is imprudent. Storing the cold data in disk or SSD-based swap space solves the capacity issue but imposes an unacceptable latency overhead when the cold data is accessed.

Tiered memory architectures offer an attractive solution to solve memory inefficiencies with *near* and *far* memory tiers [41]. Near memory tiers are typically DRAM-based and are low-latency, and low-capacity memory tiers. In contrast, far memory tiers are typically high-latency (higher than DRAM, but lesser than disk or Flash latency), and high-capacity memory tiers. Examples of far memory tiers include

*Work done during internship. Now at The University of Edinburgh
†Work done while at Intel

CXL-attached memory pools [10, 11, 16, 18, 20], non-volatile memories (NVM) [4, 39, 57], compressed memory pools [34], and disaggregated remote memory [13, 22]. The key idea is to store hot data in the near memory tier and cold data in the far memory tier.

However, ***memory tiering is only as good as the telemetry (hot and cold data detection)***. Effective use of tiered memory requires precise and timely identification of hot and cold data sets and then proactively placing them in appropriate tiers. Incorrect or delayed telemetry may place a hot data set in the far memory tier and a cold data set in the near memory tier, resulting in significant performance degradation, which can offset the TCO savings achieved through memory tiering. *Importantly, AI/ML models that either autotune page placement across memory tiers or predict hot and cold data sets completely depend on precise telemetry data for offline analysis and training [34].*

Unfortunately, existing state-of-the-art telemetry techniques for hot and cold data detection, even though effective at gigabyte scale, are either ineffective or completely fail at terabyte scale. For example, techniques that linearly scan the virtual address space [25, 36] of applications to find hot and cold data cannot provide timely telemetry due to the large number of pages that need to be scanned. Hardware and software-based approaches [9, 41, 45, 48, 51] that sample accesses to data pages for telemetry do not scale when the application's working set grows beyond a few gigabytes as shown in Figure 1.

In this paper, we propose *Telescope*, a novel telemetry technique for fast and efficient identification of hot and cold data regions. We observe that different levels of a multi-level page table tree, from the leaf entry to the root entry, have access bits that are updated during a hardware page table walk. This implies that a hot data region will also have hot entries at all levels of a page table tree corresponding to the path of the page table walk. Similarly, if the access bit at a particular level of a page table tree is not set, then none of the data pages under its subtree have recently been accessed and, hence, are cold. We leverage this insight to quickly converge from the root of the page table tree to the actual hot data region and thus efficiently identify gigabytes of cold data regions at terabyte scale.

Since we exploit the natural layout of the page table structure to identify hot and cold data regions, our technique can seamlessly scale beyond terabyte-scale to even petabyte-scale applications on a five-level page table without any significant performance overheads.

We implement Telescope in Linux kernel and `x86_64` architecture, but *Telescope is portable across hardware architectures* that support radix page tables [7, 44, 53]. For a 5 TB microbenchmark, Telescope achieves 90%+ precision and recall compared to 0% by Linux kernel's DAMON and less than 10% by hardware counters. For 1–2 TB memory footprint real-world in-memory database benchmarks, memory tiering based on Telescope results in 5.6% to 34% throughput improvement, while the throughput improves marginally for hardware counters and drops for DAMON. In addition, Telescope applied to MGLRU improves scanning time by 12.38% for real-world benchmarks.

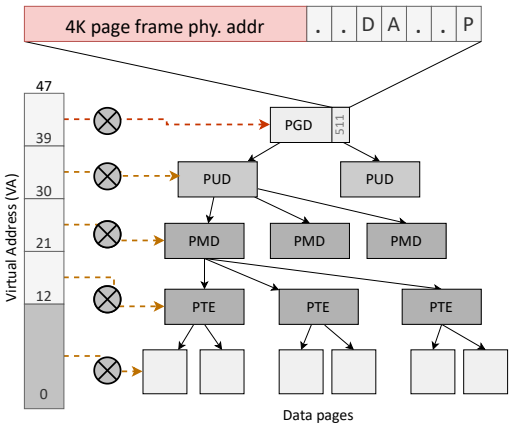


Figure 2: A 4-level page table structure.

The primary contributions of this paper are as follows:

- Compare and contrast the efficiency of the state-of-the-art telemetry techniques for terabyte-scale workloads.
- Propose Telescope, a fast, efficient, and scalable telemetry that can seamlessly scale beyond terabyte scale.
- To the best of our knowledge, Telescope is the first technique to profile page table tree for efficient telemetry.

2 Background

Before describing the contributions of this paper, we provide the necessary background on modern page table layouts.

2.1 Page Table

The page table of a process is a hardware-defined radix tree-based structure maintained by the operating system (OS) to manage the virtual address (VA) to physical address (PA) mappings. A radix page table tree is supported in many hardware architectures, such as x86_64 [27], ARM [53], RISC-V [44], POWER [7].

Modern architectures such as `x86_64` support both 4-level and 5-level page table structures. We use the 4-level page table layout that supports virtual address sizes up to 256 TB for our discussion. The page global directory (PGD) is the root of the page table tree, which has 512 entries of size 8 bytes. Each PGD entry points to the base physical address of a page upper directory (PUD) and also maintains a set of flags (e.g., page is `PRESENT`, page is `READ_ONLY`, page has been

Table 1: Table summarizing the pros and cons of different telemetry techniques. Techniques marked with a "#" are pure telemetry techniques, while others are memory tiering solutions that are designed based on the corresponding telemetry category.

Category	Pros and Cons	Techniques/Solutions
Linear scanning	+ No false positives or false negatives as all the data pages are scanned + Efficient for small footprint workloads – Does not scale as it scans individual pages – High scanning overheads at terabyte scale	kstaled# [36], Idle page tracking# [25], MGLRU# [59], TPP [41], HeteroVisor [24], AutoTiering [30]
Region-based Sampling	+ Low profiling overheads due to sampling – False positives and false negatives due to sampling – Failure to quickly converge to hot regions at terabyte scale	DAMON# [45], Thermostat [9], MTM [51]
Hardware counters	+ No false positives – Requires hardware support – High false negatives at terabyte scale (misses hot pages due to sampling)	PEBS# [28], HeMem [48], TPP-Chameleon [41], MEMTIS [35]
Page table profiling	+ Seamlessly scales for large memory footprint applications	***Telescope***#

recently ACCESSED, or page is DIRTY [27]). A similar layout is followed for PUD entries, page middle directory (PMD) entries, and page table entries (PTE) but at different levels (Figure 2).

Upon a TLB miss, the hardware page table walker walks the page table to find the VA to PA mapping. During the page table walk, the first 9 bits of the 48-bit VA are used as an index into the PGD to extract the physical address of the PUD. The next 9 bits in the VA index into the PUD to extract the physical address of the PMD, and similarly, the next 9 bits index into the PMD to extract the physical address of the PTE. Finally, the last 9 bits in the VA is used to index into the PTE to extract the physical address of the data page. The remaining 12 bits in the VA are used as an offset in the data page of size 4 KB.

During a page table walk, the hardware sets the ACCESSED bit at all levels (from PGD to PTE) of the page table tree.

3 Related Work & Motivation

In this section, we discuss the related works and highlight their design limitations. We then motivate the need for novel telemetry techniques that are precise and efficient for terabyte-scale applications and beyond.

Several memory management systems have been proposed for tiered memory systems in recent years [9,17,29–31,34,37,55]. Some of the prior works are pure telemetry techniques that provide information on hot and cold data pages [25,28,36,45,59], while others propose policies and optimizations for proactive data migration and placement in near and far memory tiers based on the telemetry data [9,31,35,37,41,48,51]. We classify the pure telemetry techniques into three broad groups: ❶ *linear scanning*, ❷ *region-based sampling*, and ❸ *hardware counters* (Table 1).

3.1 Linear scanning

Techniques in this group linearly scan the entire virtual address space of the application to identify hot and cold data by leveraging the ACCESSED bit in PTE. It requires two full scans of the virtual address space to identify the accessed data pages. The first scan resets the ACCESSED bit in the PTE entry for every data page while the second scan checks the entire virtual address space to find the data pages with the ACCESSED bit set. A set bit indicates that the page was accessed at least once since the last reset [36]. It periodically scans the virtual address space and checks for data pages that were accessed during a time window. Using this access information, it classifies the data pages into hot and cold sets.

Limitations: Linear scanning does not scale for workloads with terabytes of memory. The time and compute overheads associated with scanning the application's virtual address space increase with the application's memory footprint. Such inefficiencies are pointed out by HeMem [48] where it is shown that the single scanning time at terabyte scale can be orders of seconds.

It is important to note that multiple scans are required to build a precise profile of hot data pages. Furthermore, as an application's hot and cold data sets can dynamically change over time, scanning cannot be paused. Always-on aggressive scanning results in prohibitively high CPU overheads, impacting system and application performance. However, occasionally scanning the address space can reduce CPU overheads but can fail to quickly recognize changing data access patterns in applications. This can cause hot data regions to reside in far memory for an extended duration of time. *Thus, optimizing address space scanning time and the associated CPU overheads is critical for terabyte-scale applications.*

3.2 Region-based sampling

To reduce the overheads of telemetry, region-based sampling [45] limits the number of data pages that need to be tracked. It divides the application's virtual address space into fixed-size regions to reduce the number of pages it has to track. It then randomly samples one or more data pages in that region and tracks accesses to them using the ACCESSED bit in PTE. The total number of accesses detected based on the

sampled pages is assumed to represent the hotness or coldness of the entire region.

A hot memory region thus identified is gradually split into smaller regions to monitor memory accesses at a finer granularity. Adjacent cold regions are merged to form a larger region to reduce the monitoring overheads. DAMON (Data Access Monitor) [45] is one such technique that has been incorporated into the mainline Linux kernel.

Limitations: Although effective for workloads with gigabytes of memory footprint, this method does not scale for terabyte-scale applications. As the memory footprint increases, the probability of sampling an address belonging to the hot data region reduces.

In this technique, the convergence to the correct hot data set completely depends on the pages picked by random sampling. If in case only cold data pages are sampled, instead of hot data pages, then this technique fails to converge or identify hot region as the whole region is incorrectly marked as cold. However, increasing the sampling rate increases the probability of finding a hot data region, but with increased CPU overheads.

Figure 1 clearly shows that region-based sampling is not suitable for terabyte-scale applications, as the efficiency of hot data detected by DAMON deteriorates with the increase in memory footprint.

3.3 Hardware counters

Techniques in this group leverage the hardware counters or performance monitoring units (PMUs) to identify an application's hot and cold data pages. PMU events such as retired load/store instructions, TLB misses or L3 cache misses are typically monitored for hotness tracking. Once a PMU event is enabled for monitoring, the hardware increments a counter at each occurrence of the event, and when the counter overflows, the hardware generates an exception. OS handles the exception and saves the event state to in-memory buffers. The virtual addresses that caused the PMU events are also saved in the buffers, which are then used to identify the hot data set.

Limitations: Hardware counters are also based on sampling, where PMUs sample the hardware events. Similar to region-based sampling, the efficiency of hot data detected by PMU deteriorates with the increase in memory footprint as shown in Figure 1 with Intel's PEBS [28]. Increasing the sampling rate improves the probability of hot and cold page detection but can negatively impact the application performance because higher sampling rates result in frequent PMU interrupts to the OS.

In addition, the overheads of this technique are proportional to the size of the hot region. For instance, a 10 TB footprint application with a total of 1 TB hot region monitored using TLB miss event requires generating *at least* 268 million events (one event per 4 KB page) to precisely identify the entire hot region. This can generate thousands of PMU interrupts to the OS impacting system and application performance. Furthermore, operating systems such as Linux monitor the rate at which PMU interrupts are triggered and automatically lower the sampling frequency if the percentage of time spent in interrupt processing exceeds a certain threshold [8]. This automatically reduces the number of samples generated which in turn reduces the precision at which hot data regions are identified. Hence, hardware counters do not scale for terabyte-scale applications.

3.4 Telemetry with huge pages

Linear scanning: The use of 2 MB huge pages for linear scanning reduces the overheads of linear scanning by an order of magnitude as a single huge page covers 512 base pages of size 4 KB. However, as memory footprint scales to several terabytes, linear scanning at huge page granularity still requires scanning several million huge pages and hence results in high scanning time as observed in HeMem [48].

Region-based sampling: The use of 2 MB huge pages for region-based sampling does not improve telemetry efficiency because the probability of sampling a hot huge page in a large region can still be low for applications with several terabytes of memory footprint. For example, consider a 100 MB region with Y hot 4 KB pages. The probability of selecting a hot page for sampling in this region is $Y/25,600$ (as 100 MB contains 25,600 4 KB pages). Applications with a footprint in terabytes can have large regions. Hence, for a 50 GB region (an increase of $\approx 512\times$ from 100 MB) with Y hot huge pages, the probability of selecting a hot huge page remains the same as selecting 4 KB in a 100 MB region.

Hardware counters: For hardware counters, using huge pages does not improve profiling efficiency, for example, when retired load/store instructions or L3 cache misses are monitored to identify hot data sets as these events are independent of the page size used by the application.

Hence, the use of huge pages does not fundamentally solve the telemetry inefficiencies of the state-of-the-art profiling techniques at a terabyte scale.

3.5 Summary

It can be concluded that linear scanning, region-based sampling, and hardware counters do not enable fast and efficient identification of hot and cold data sets. Further, their effectiveness degrades quickly as application footprints escalate (Figure 1). As memory tiering is only as good as the telemetry, we strongly argue for the need for novel telemetry techniques that are precise, timely, and efficient for *gargantuan* memory footprint applications.

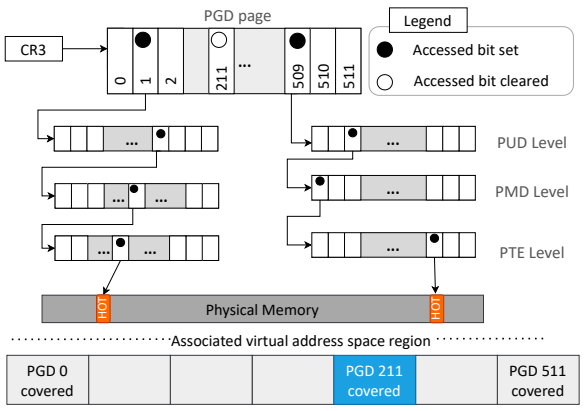


Figure 3: Diagram depicting the high-level overview of Telescope

4 Design Principles

In this section, we explain the principles that guide the design of Telescope.

Existing state-of-the-art telemetry techniques [23, 36, 45] rely on checking ACCESSED bits only at the leaf level of the page table tree. However, for the past several decades, hardware architectures have supported ACCESSED bits at all the levels of a page table tree by updating them during the page table walk. To the best of our knowledge, Telescope is the first technique that exploits this hardware feature by dynamically profiling different levels of a multi-level page table tree to precisely and efficiently identify hot and cold data regions at terabyte scale.

Telescope leverages the following key insight: as ACCESSED bits at all levels of the page table tree are updated during a hardware page table walk, a hot data page should also have a hot PMD, PUD, and PGD entry for a memory and TLB-intensive application. Similarly, if the access bit in a PGD entry (or a PUD/PMD entry) is not set, then none of the memory regions represented by the PGD entry (or PUD/PMD entry) subtree are accessed and hence can be considered as cold.

Telescope profiles ACCESSED bits at the higher levels of the page table to initially identify hot regions at coarser granularity as they cover larger virtual address mappings. Upon detecting accesses, Telescope dynamically profiles lower levels of the page table tree to converge to hot regions.

We explain Telescope with an example. Consider a terabyte-scale application with hot data pages as shown in Figure 3. To identify a hot data page, Telescope starts profiling at the PGD level by resetting the access bits and periodically checks if they are set by the hardware page walker. As a single PGD entry covers 512 GB of virtual to physical address mapping, if the access bit for a PGD entry is set then one or more data pages in the 512 GB PGD subtree is hot.

As PGD entries 1 and 509 have the ACCESSED bit set (Figure 3), Telescope dynamically traverses down the page table tree corresponding to these PGD entries to profile at PUD level. Telescope resets the ACCESSED bit at the PUD level and periodically checks if they are set by the hardware page walker. If the access bit for a PUD entry is set then one or more data pages in the 1 GB PUD subtree is hot (each PUD entry covers 1 GB mapping). Similarly Telescope traverses down the page table tree by dynamically profiling at PMD and PTE levels if the ACCESSED bits are set to find the actual hot data page. As Telescope traverses down from PGD to PTE it converges from a large 512GB region to the actual 4K hot data page.

Now consider a scenario where the ACCESSED bit for a PGD entry that was cleared during profiling is still not set (PGD entry 211 in Figure 3). In such a case the entire 512 GB virtual address space subtree corresponding to PGD entry 211 is cold; it is not required to traverse down the page table tree any further. This way several gigabytes of cold regions can be quickly identified without enumerating individual data pages.

Summary. Telescope at every iteration converges to the hot data set by traversing down the tree (similar to search technique in a tree data structure) to a set of subtrees that contain hot data pages (i.e., for entries with ACCESSED bit set at that page table level) while it stops further traversing down the subtree if the ACCESSED bits are not set to identify the cold data pages.

5 Telescope Design

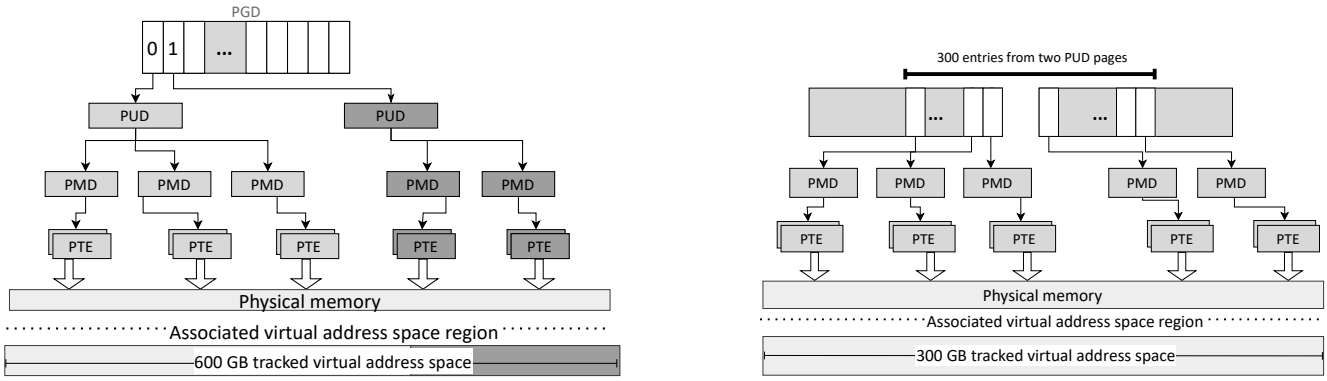
Telescope introduces a novel concept to identify hot and cold data pages in a workload's memory footprint using page table profiling. In this section, we explain the design of Telescope applied to region-based and scanning-based techniques.

5.1 Region-based telemetry

Telescope design for region-based telemetry has two main components (i) region management and (ii) region profiling as explained below.

5.1.1 Region management

Telescope's region management is inspired by the design employed in the Linux kernel for DAMON [45] as we find it efficient. Telescope decomposes the workload's virtual address space into a set of equally sized regions to begin with. A profiling window is used during which data accesses to each region are monitored. Based on the number of accesses seen, each region is assigned a score that reflects the hotness or coldness of the entire region. At the end of each profiling window, the following actions are performed: (i) each region is split into random-sized small subregions. We find random splitting of regions employed in the Linux kernel effective



(a) (b)

Figure 4: Bounded variant used by Telescope to select page table level at which to track the accessed bits

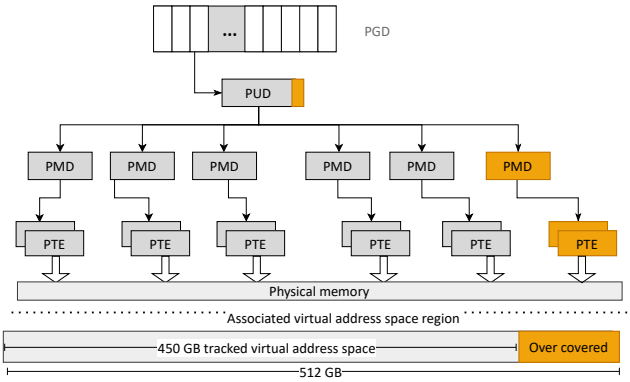


Figure 5: Flex variant used by Telescope to select page table level at which to track the accessed bits

under the dynamically changing memory access patterns of the workloads, (ii) adjacent regions with similar hotness or coldness score are merged into a bigger region and (iii) information is provided to user space regarding the number of regions, the virtual address range of the regions and the associated hotness or coldness score. This information can be used to take suitable actions such as migrating data pages to appropriate tiers or can be fed to AI/ML models for offline training.

Splitting ensures that the regions that contain hot data pages are narrowed down to precision with time, while merging ensures that cold regions are tracked at a coarser granularity.

5.1.2 Region profiling

Region profiling is the core and critical component of Telescope that precisely identifies hot and cold data regions.

Each region identified by the region management subsystem is profiled independently in every profiling window. For each region, in each profiling window, multiple profiling samples are recorded at regular intervals. For each sampling interval, a page table entry at one of the levels from PTE to PGD is identified for profiling. The `ACCESSED` bit for the identified page table entry is reset at the beginning of the sampling interval and Telescope checks whether the reset `ACCESSED` bit is set at the end of the sampling interval. If set, then one or more data pages covered by the identified page table entry were accessed during the sampled interval. Hence the access count for the region is incremented. At the end of the profiling window, Telescope performs the region management actions such as splitting and merging as discussed before for all the regions.

5.1.3 Profiling variants

We implement two different profiling variants (i) *bounded* and (ii) *flex*, which identifies a page table level from PTE to PGD for profiling. The variants offer two different trade-offs between convergence aggression and profiling accuracy.

Bounded variant. This variant ensures that for each region, the identified page table entry is at the highest page table level whose address range is within the region bounds. That is, the page table level identified should not span multiple regions. Tracking the accessed bit at the highest possible level of the page table increases the likelihood of convergence, as just a single access bit can track data accesses for a large virtual address range.

Consider the example in Figure 4a with a region size of 600 GB. In this example, the highest possible page table level that Telescope can pick is PGD. The virtual address space covered by PGD entry 0 is within the region bounds and does not span multiple regions. By profiling PGD entry 0, Telescope can quickly identify even a single page access in the 512 GB virtual address space.

However, PGD entry 0 does not cover the entire 600 GB region. Hence, for the subsequent sampling intervals, Telescope should pick a page table entry that includes the rest of the 88 GB to ensure coverage of the entire region. But for the

remaining 88 GB, PGD entry 1 cannot be used as it covers virtual address space from 512 GB to 1024 GB, which is beyond the address space bounds of the region being profiled. Therefore, the highest possible page table level that Telescope can pick for profiling the 88 GB portion is PUD.

As Telescope progresses, the hot regions are split, resulting in smaller regions. Now, the virtual address ranges of the higher levels of the page table do not fit within the region's bounds, forcing Telescope to pick lower levels. Nevertheless, this facilitates the precise convergence of regions to actual hot pages within the application's memory footprint. For example, consider a region with size 300 GB. Telescope cannot pick a PGD entry as it covers virtual address space beyond this region. In such scenarios, PUD entry is the highest page table level whose address range is within the region bounds. This is also the case when the region is mapped by two PUD pages as shown in Figure 4b. Hence, one of the 300 PUD entries is randomly picked for profiling during every sampling interval.

Similarly, the highest page table level that can be picked for profiling can be a PMD or PTE entry depending on the size of the region being profiled.

Flex variant. This variant requires that the identified page table entry is at the highest page table level but the address range of the picked entry need not be always within the region bounds. Telescope can be flexible and go beyond the region's bounds but with a certain error threshold. This ensures better region coverage but at the cost of accuracy.

Consider the example in Figure 5 with a region size of 450 GB where the error threshold is 15%. The virtual address space covered by PGD entry 3 exceeds the region bounds by 72 GB, but is well within the error threshold. Hence, Telescope is allowed to pick PGD entry 3 for profiling. Consider another example where the region size is 300 GB. Telescope cannot pick a PGD entry as the virtual address space covered by a PGD entry is beyond the error threshold. In such scenarios, Telescope falls back to the bounded variant where one of the 300 PUD entries is randomly picked for profiling during every sampling interval.

5.2 Scanning-based telemetry

In this section, we explain the Telescope design for multi-generation LRU (MGLRU) [59] which is a scanning-based telemetry technique.

MGLRU, part of the Linux kernel, maintains multiple generation LRU lists instead of the typical "active" and "inactive" lists used in the classical OS implementation of LRU. Each MGLRU scan starts by creating a new generation LRU list. In every scan, frequently accessed or hot pages are moved to newer generations, while cold pages are "aged" to "old generation" LRU lists. When it is required to reclaim memory, pages in the older generation LRU lists are considered for placement in "far memory".

To identify pages to move to a newer generation, MGLRU linearly scans the page table tree by checking the `ACCESSED` bits at the leaf level. If the `ACCESSED` bit is set then the page is moved to the new generation and the `ACCESSED` bit is cleared for that page. If the `ACCESSED` bit is not set then the page was not accessed and it remains in the same LRU list. Therefore frequently accessed pages are always moved to a newer generation list, while cold pages remain in the older generation list. As MGLRU requires scanning the entire linear address space of the application, it is prohibitively expensive for large memory footprint applications. To optimize the scanning overheads, the current MGLRU implementation in the Linux kernel takes advantage of `ACCESSED` bits at the PMD level, but it does *not* take advantage of non-leaf `ACCESSED` bits at the higher levels of the page table tree.

Telescope applied to MGLRU (henceforth referred to as MGLRU-TS) checks the `ACCESSED` bit at every page table tree level starting with PGD. If the `ACCESSED` bit in PGD is not set, then none of the pages under this PGD subtree are accessed since the previous scan. Hence none of the pages under this subtree need to be moved to the new generation LRU list. Therefore, MGLRU-TS skips the entire PGD subtree. Skipping a PGD with `ACCESSED` bit not set makes the scanning efficient in terms of time and CPU utilization as several thousands of `ACCESSED` bits checks at leaf level are eliminated.

In case the `ACCESSED` bit at the PGD level is set then it implies that one or more data pages under this PGD subtree have been accessed since the previous scan. Hence such pages should be identified and moved to the new generation list. MGLRU-TS traverses down the PGD subtree if `ACCESSED` bit is set.

Similarly, at every level of the page table tree, the corresponding access bit is checked, and if not set the entire subtree below that level is skipped to improve scanning efficiency.

6 Evaluation

In this section, we compare and contrast Telescope with other state-of-the-art telemetry techniques. Our evaluation answers the following questions.

- How well does Telescope perform vis-a-vis the state-of-the-art in quickly and accurately identifying hot data?(§ 6.2).
- How do the incurred overheads of Telescope compare with other techniques? (§ 6.2.3).
- How well does Telescope perform on real-world big data applications? (§ 6.3).

6.1 Evaluation setup

We use a tiered memory system with an Intel Xeon Gold 6238M CPU having 4 sockets, 22 cores per socket, and 2-way HT for a total of 176 cores. It has a DRAM-based near memory tier with 768 GB capacity and a far memory tier with Intel's Optane DC PMM [4] configured in flat mode (i.e., as volatile main memory) with 6 TB capacity for a total of 6.76 TB physical memory. We run Fedora 30 and use Linux kernel 5.18.19 for our evaluation. We use 4 KB pages unless otherwise explicitly mentioned.

6.1.1 Telemetry techniques

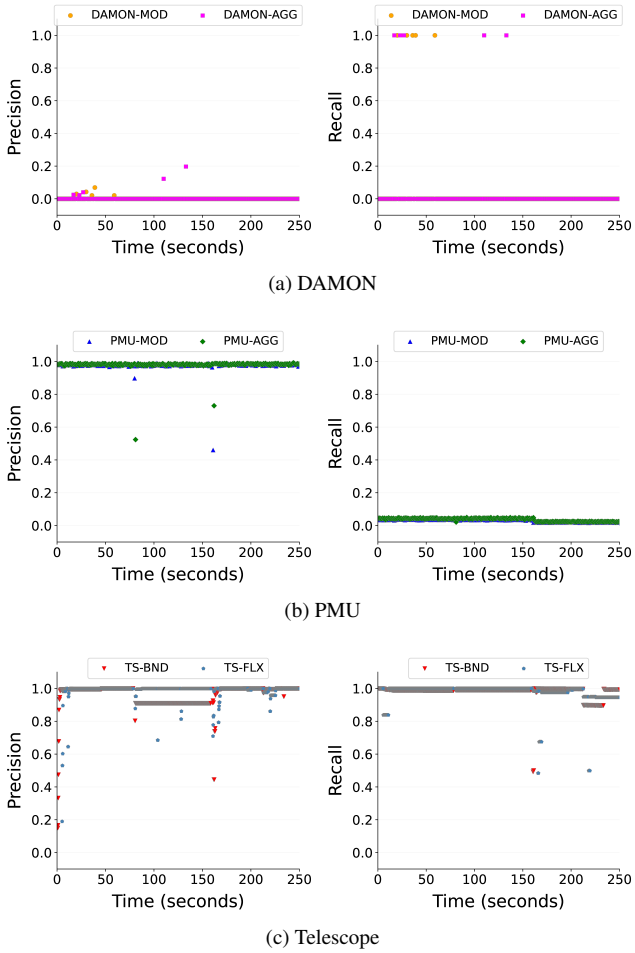
DAMON [45]. DAMON is a region-based sampling technique that is part of the Linux kernel. We use two configurations for DAMON – *moderate* and *aggressive*. *Moderate* (MOD) uses the default values of 5 ms sampling interval and 200 ms profile window (or aggregation interval) thus generating 40 samples per profile window. *Aggressive* (AGG) uses 1 ms sampling interval with 200 ms profile window generating 200 samples per profile window. *Aggressive* consumes more CPU cycles as it samples more frequently. The rest of the DAMON parameters are set to the Linux kernel default values. We do not include results with different profile windows because they provide no additional insights, as the trend remains the same.

PMU. We use Intel PEBS (Processor Event-Based Sampling), a hardware-based performance monitoring unit (PMU) available on Intel processors [28]. PEBS can monitor pre-defined hardware events and can capture additional information such as the virtual address that caused the event. We monitor `MEM_INST_RETIRED.ALL_LOADS_PS` and `MEM_INST_RETIRED.ALL_STORES_PS` [48] events that sample all the retired load and store instructions. We drive PEBS using the `perf` tool available in Linux. We evaluate PEBS with two different sampling frequencies: 10 kHz and 5 kHz for *aggressive* and *moderate* configurations, respectively. The higher the sampling frequency, the higher the overheads, as PEBS generates frequent interrupts.

Telescope. We evaluate two variants of Telescope: *bounded* and *flex*, which differ in the way a page table level is picked for profiling as explained in detail in the design section (§5). Both variants of Telescope are configured to use 5 ms sampling interval and 200 ms profiling window. We use different error thresholds at different levels of the page table tree for the *flex* variant. At PUD, the error threshold is kept low at 15% as it covers a larger region, while at PMD, it is set to 25%.

6.2 Microbenchmarks

To simulate different memory access patterns we use *memory access simulator*, or MASIM [46], a widely used utility by the Linux kernel developers [1,2,5,52]. We generate stable access patterns, as page access patterns remain stable for several minutes to hours in production workloads [41]. In addition, we also demonstrate the sensitivity of profiling techniques to changes in access patterns.



(a) DAMON

(b) PMU

(c) Telescope

Figure 6: Precision and recall for the *multi-phase* microbenchmarks

We fix a bug we found both in MASIM and DAMON, to support terabyte-scale workloads, by using a 64-bit random value instead of 32-bit to generate accesses to memory regions greater than 4 GB. We also optimize MASIM to perform multi-threaded memory allocation to reduce the initialization time.

Precision and recall. We quantify telemetry capabilities using two key metrics - *precision* and *recall* [51] for microbenchmarks, as we know the "ground reality", i.e., we know which pages are actually hot. Precision is the ratio of correctly identified hot pages to the total number of identified hot pages, i.e., the fraction of the memory identified as hot by the telemetry technique which is indeed hot as per the workload's actual access pattern. Recall is the ratio of correctly identified hot

pages to the number of actual hot pages in the workload i.e., the fraction of the workload's actual hot pages that were correctly identified as hot.

To compute precision and recall for DAMON and Telescope, we use the region data as reported during every profile window. PMU counters using PEBS do not report any region data, but report the virtual address of the profiled events. We use a 2 MB tracking granularity [48] to ensure that we do not underestimate the hot data regions of the application by tracking at finer granularities. Due to the 2 MB granularity, the precision can be slightly less than 1 in some cases where our tracked regions overshoot the boundary of the true hot region.

6.2.1 Multi phase (Multi-4K)

The goal is to test the three important hot data identification capabilities: (i) speed and accuracy in identifying hot regions, (ii) sensitivity and responsiveness to dynamically changing hot regions, and (iii) speed and accuracy in identifying multiple hot regions in the entire heap.

We configure MASIM to allocate 5 TB of heap and simulate access patterns in three different phases to test the capabilities mentioned above. In the first phase, MASIM performs data loads by randomly picking an address within a 10 GB region. The second phase is the same as the first phase but on a completely different 10 GB region. In the third phase, MASIM performs data loads by randomly picking an address from two different 10 GB regions. Data access patterns in real workloads generally remain stable for minutes to hours [41], so this microbenchmark is representative of real-world access patterns.

Precision and recall. Figure 6 shows the precision and recall for the multi-phase microbenchmark. DAMON's precision and recall are mostly 0 as it fails to detect any hot regions. This is because, at the terabyte scale, the probability of the sampled address belonging to the hot data set is low. PMU's precision values are always close to 1 as they include data only for the actual events (i.e., no false positives; events are not generated for cold pages). However, the recall for both variants of PMU is less than 0.1 because covering the entire hot region requires generating millions of events, which is not possible (as discussed before in limitations of hardware counters, §3.3).

Both variants of Telescope outperform both DAMON and PMU in all the phases. Telescope's precision and recall remain above 0.9 for all three benchmark phases. The precision for Telescope momentarily drops during the phase change (at around 80 and 160 seconds) but quickly recovers. This clearly demonstrates that only Telescope passes our versatility test.

Huge pages (Multi-2M). We repeat the experiments with transparent, huge pages or 2MB pages enabled to compare and contrast the efficiency of telemetry techniques. With huge pages, DAMON performs better than 4 KB pages with an average of 0.94 and 0.96 precision and 0.92 and 0.90 recall for the moderate and aggressive variants, respectively. Telescope achieved an average precision of 0.96 and a recall of 0.97 with both variants.

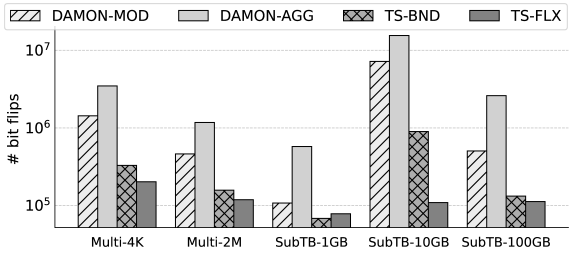


Figure 7: Total number of ACCESSED bit flipped by Telescope and DAMON. The Y-axis is in log scale

As discussed in §3.4, we expect the efficiency of DAMON with huge pages to drop as the memory footprint increases to several terabytes. Hence, using huge pages does not fundamentally solve the telemetry inefficiencies with DAMON. Nevertheless, as we show in §6.2.3, Telescope outperforms DAMON in terms of computational overheads even when huge pages are used.

6.2.2 Sub-terabyte (SubTB) workloads

The goal is to (i) test the capability of Telescope to identify hot regions even for low memory footprint or gigabyte-scale workloads and (ii) to demonstrate the memory footprint threshold at which DAMON and PMU start deteriorating. We configure MASIM to allocate 1 GB, 10 GB and 100 GB of heap and perform random loads within a 10% hot region.

Precision and recall. Figure 1 shows precision and recall plots for DAMON-MOD, PMU-MOD, and Telescope-BND for the SubTB workloads. We skip plotting the aggressive variants as they show similar results. For 1 GB workload, DAMON and PMU achieve 90%+ precision and recall. But at 10 GB, the recall drops significantly for both the variants of DAMON. For 100 GB workload, DAMON's precision and recall drop to zero. PMU's precision remains high as they include data only from the actual event samples. However, as the total number of hot pages increases, the hot data coverage drops significantly (as discussed before in limitations of hardware counters, §3.3). This clearly shows that both DAMON and PMU fail to precisely capture the hot data regions as we scale to large memory footprint applications. Both variants of Telescope outperform both DAMON and PMU in all the scenarios with high precision and recall. This shows the efficiency of Telescope for both small and large memory footprint applications.

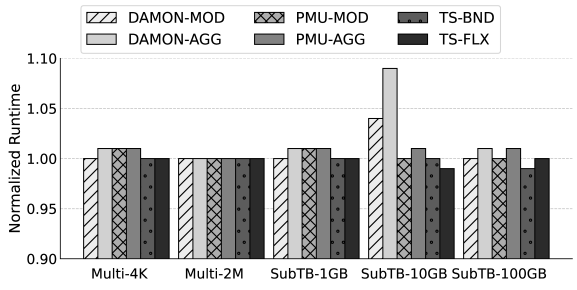


Figure 8: Impact on benchmark runtime with DAMON, Telescope, and PMU normalized to baseline with telemetry disabled.

Table 2: Cycles consumed and percentage of single core CPU utilization by the kernel thread for DAMON and Telescope.

Config.	Multi 4K	Multi 2M	SubTB 1GB	SubTB 10GB	SubTB 100GB
CPU cycles (in billions)					
DAMON-MOD	9.55	2.42	1.15	19.53	3.52
DAMON-AGG	24.27	11.94	5.80	68.22	18.91
Telescope-BND	2.25	2.09	0.83	3.20	1.27
Telescope-FLX	2.28	1.80	0.95	1.16	1.19
Single CPU core utilization in %					
DAMON-MOD	3.3	1.2	0.4	4.2	0.7
DAMON-AGG	9.1	6.1	2.1	14.7	3.5
Telescope-BND	0.9	1.1	0.3	0.8	0.7
Telescope-FLX	0.9	0.4	0.4	0.3	0.7

6.2.3 Performance overhead analysis

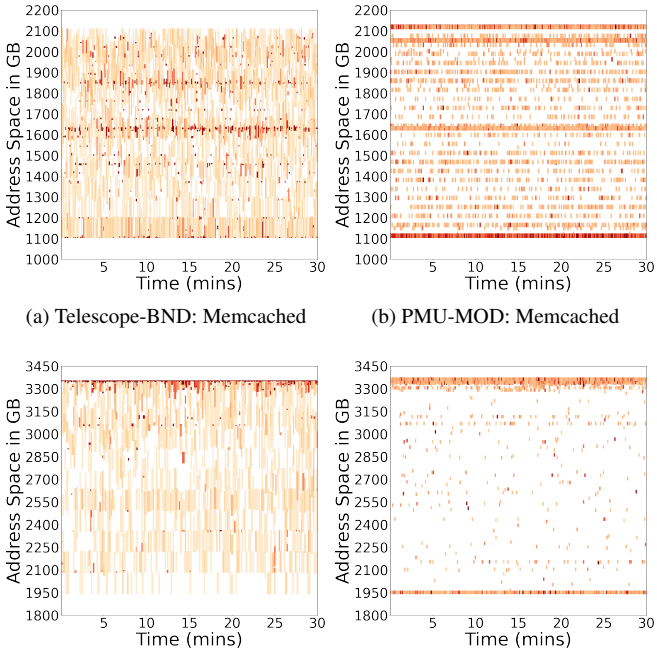
We present the computational overheads incurred by the telemetry techniques for the microbenchmarks described above.

Bit flips. The number of `ACCESSED` bits flipped by Telescope is significantly less than DAMON in all the microbenchmarks as shown in Figure 7. This is because, in most cases, a single access bit at the higher levels of the page table tree is sufficient to cover a significant portion of a region.

Computational overheads. Table 2 shows the number of cycles consumed by the kernel thread that performs the profiling for both variants of DAMON and Telescope. It can be observed that for all the microbenchmarks, both variants of Telescope consume significantly fewer CPU cycles.

In addition, Table 2 shows the average single CPU core utilization of the kernel thread for DAMON and Telescope. Both variants of Telescope have less than 0.9% of a single core utilized for the telemetry, while DAMON-MOD and DAMON-AGG have higher CPU overheads. PMUs have been excluded from this comparison as the profiling is taken care of by the hardware and not in a separate kernel thread.

Runtime impact. Figure 8 shows the execution time impact on the benchmarks, which excludes the memory initialization phase. Values are normalized to the baseline run where telemetry is disabled. We do not migrate any pages to measure pure telemetry overheads. It can be observed that Telescope does not impact the runtime of the microbenchmarks. PMU-AGG and DAMON-AGG impact the runtime of the microbenchmark in a few cases.



(a) Telescope-BND: Memcached (b) PMU-MOD: Memcached

(c) Telescope-BND: Redis (d) PMU-MOD: Redis

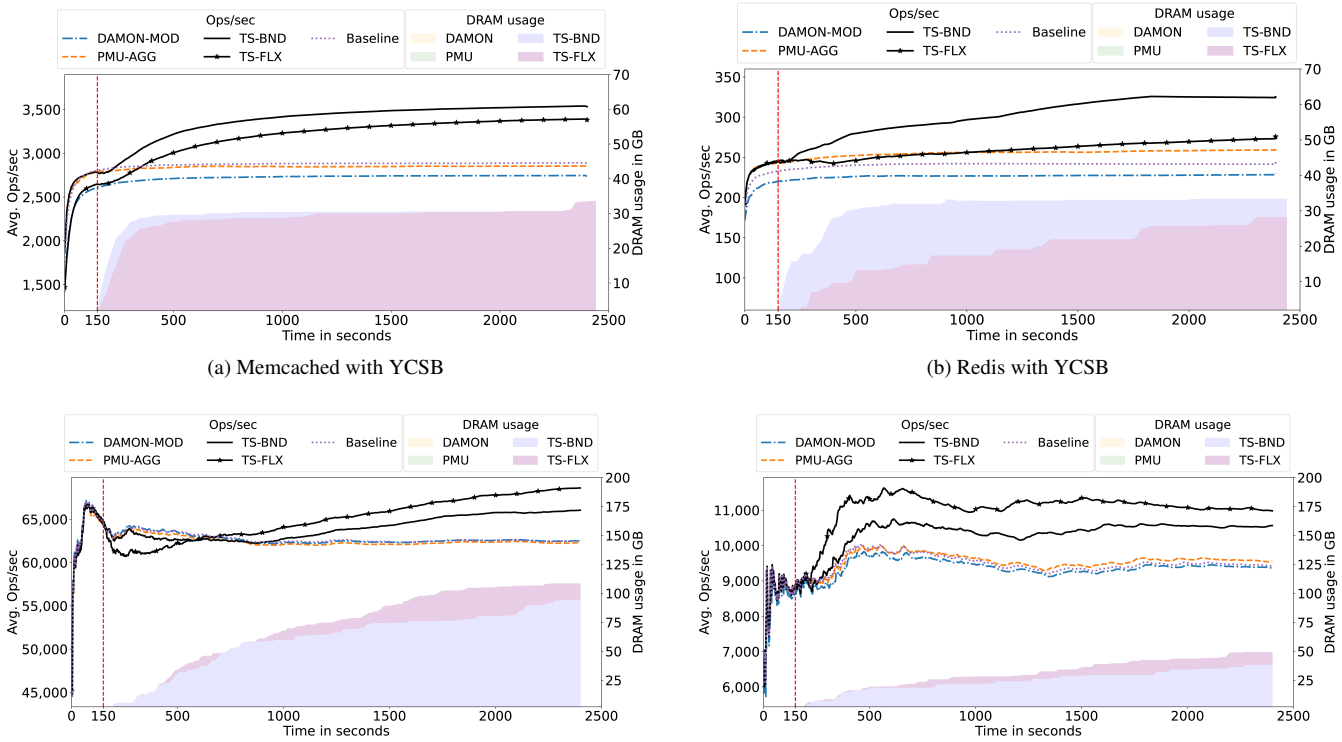
Figure 9: Heatmaps for the Memcached with Memtier for Telescope and PMU. We do not include DAMON as it fails to recognize any hot data resulting in an empty heatmap. Note that the data is generated on a fresh instance of the Memcached and Redis and hence the hot key-value pairs may be at a different virtual address. *This can result in different heatmaps for Telescope and PMU.*

6.3 Real-world application benchmarks

In this section, we present the results for large memory footprint real-world applications on a tiered memory system.

Table 3: memtier [33] and YCSB [15] configurations.

Parameter	Memtier (MT) [33]	YCSB [15]
Memory footprint	1 TB	2 TB
Number of keys	200K	1000 K
Key Value size	5 MB	2 MB
Number of threads	170	170
Execution time	40 mins	40 mins
Hot data distribution	Gaussian w/ Std. deviation 100	Hotspot (99% ops on 1% hot data)



(a) Memcached with YCSB

(b) Redis with YCSB

(c) Memcached with memtier

(d) Redis with memtier

Figure 10: Throughput improvement and DRAM usage for Memcached and Redis as hot data pages are migrated from far memory tier to near memory tier after a warmup period of 150 seconds.

We use the widely used *Memcached* [42], a commercial in-memory object caching system, and *Redis* [32], a commercial in-memory key-value store, as our real-world application benchmarks. We use Memtier [33, 49] and YCSB [15] as load generators that generate different access patterns [15, 19, 56] (refer Table 3 for configurations). In total we have four real world scenarios with the application and load generator combination.

6.3.1 Experiment setup

We initialize the data on the far memory tier (Optane NVM) using interleaved memory allocation policy [43]. Once the data is initialized, we execute the workloads for 40 minutes each. The first 150 seconds is the warmup phase, after which we start the telemetry technique to identify and migrate hot data to near memory tier. We compare the performance improvement over baseline (both telemetry and page migration disabled) with DAMON, PMU, and Telescope.

6.3.2 Hot page classification and migration.

The information generated by a typical telemetry technique is a set of pages (or regions), access timestamp, and the number of times they were accessed in a time window. Whether a particular page or region is hot or not is up to the user to define based on the application's behavior and requirements [41, 48, 51]. In addition, migrating pages across memory tiers have associated overheads and hence should be rate limited.

We use the following rules to classify and migrate hot pages (or regions) as also used in prior works [41, 48, 51]: ❶ we consider regions with access count greater than a threshold (set to 5) in a given time window of 120 seconds as hot, ❷ we skip large regions ($\geq$ 4 GB) to ensure hot pages are migrated at a finer granularity. Subsequent profiling windows will split these large hot regions, and hence these hot pages are eventually migrated, ❸ for the rest of the regions, we start migrating regions with the highest hotness score and stop once a limit of 10 GB is reached.

6.3.3 Results

Heatmaps. For realworld benchmarks we do not know the "ground reality" as the hot data pages depends on the access pattern of the applications. Hence, calculating precision and recall is not possible. Therefore, we generate heatmaps to visualize the telemetry data and then measure the performance efficiency of Telescope. Figure 9 shows the heatmap for Memtier workloads as per the data reported by the teleme-

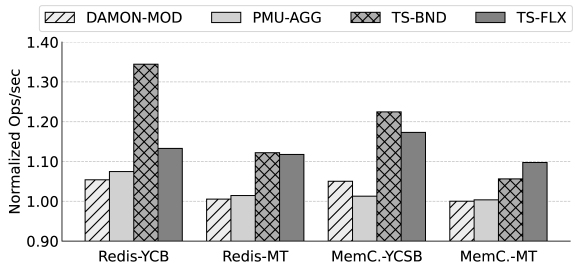


Figure 11: Normalized throughput improvement for Memcached and Redis.

Table 4: Latency impact with different telemetry techniques.

	Config.	95th percentile latency (ms)	
		YCSB	Memtier
Memcached	**DAMON-MOD**	881	11.2
	PMU-AGG	976	11.3
	Telescope-BND	867	10.8
	Telescope-FLX	824	10.5
Redis	**DAMON-MOD**	850	59.13
	PMU-AGG	757	57.50
	Telescope-BND	696	54.01
	Telescope-FLX	741	55.55

try techniques. The x-axis in the heatmap is the time, and the y-axis is the virtual address offset in the heap of the workload. For example, if the base virtual address of the heap is `addr`, then 1000 GB value on the y-axis represents `addr`+1000 GB. The dark red color represents hotter regions, yellow represents warm regions and white colour represents cold regions.

Performance. As shown in Figure 10, DAMON could not identify a single hot data page, but the profiling overheads resulted in decreased throughput compared to baseline. PMU identified only a small portion out of a few gigabytes of hot data set and hence resulted in marginal throughput improvement in some cases. Both variants of Telescope detected and migrated significant number of hot data pages to near memory tier, resulting in up to 34.4% throughput improvement (Figure 11) compared to the baseline. As Telescope outperformed PMU and DAMON in detecting hot pages, it better utilized the near memory tier, and thus performed better.

In addition, Table 4 shows latency values where Telescope outperforms both DAMON and PMU. This clearly indicates that Telescope is not interfering with the application execution, an important criteria for latency sensitive applications.

6.3.4 MGLRU-TS results

In this section, we present the results for MGLRU-TS. We run MGLRU experiments on a 2-socket system with Intel Xeon Gold 6354 CPU. The machine has a total of 72 cores and 1 TB of DRAM. We implemented MGLRU-TS in Linux kernel 6.5.0. We use Memcached with Memtier as a load generator (refer Table 3 for configuration) and populate around 976 GB

Table 5: Performance comparison: MGLRU vs. MGLRU-TS.

	MGLRU	**MGLRU-TS**
Scan time	14.57 sec	12.77 sec
CPU cycles (billions)	53.15	46.85
Memcached Tput	1 (normalized)	1.08

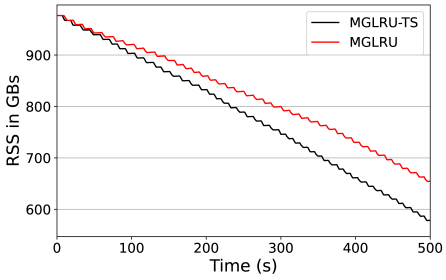


Figure 12: Resident Set Size (RSS) of Memcached with proactive reclaim to zswap.

of data. We assign Memcached to a cgroup [26] and trigger the MGLRU scan to identify hot and cold pages. After each scan, we trigger the proactive reclaim feature supported by MGLRU in the Linux kernel that reclaims and pushes 10 GB of cold data (from the oldest generation) to zswap compressed pool [6]. The scanning and proactive reclaim is repeated 100 times.

Scan time. A single scan to scan the entire address space takes around 14.57 seconds (geo mean) by MGLRU, while MGLRU-TS takes around 12.77 seconds (geo mean), an improvement of 12.38% per scan. The improvements are due to skipping page table subtrees whose `ACCESSED` bits are not set. Skipping subtrees in MGLRU-TS avoids the need to go deeper in the page table tree to the leaf level to identify cold pages.

Performance analysis. The improvement in the scanning time in MGLRU-TS directly translates into improvement in memory TCO savings due to early identification and reclamation of cold data pages. Figure 12 shows the memory footprint reduction with MGLRU-TS by pushing cold pages to zswap. The memory footprint gap increases with time for MGLRU-TS as every iteration of scanning completes faster than MGLRU resulting in early identification and reclamation of cold pages.

MGLRU-TS consumes 15.5% less CPU cycles to complete 100 scanning iterations (Table 5). The improvement in CPU cycles and scanning time are mainly due to skipping the scanning of subtrees at the high level page table entries. For example, checking the accessed bit and skipping a single PUD entry subtree from scanning saves time and CPU cycles required to check up to 262,144 `ACCESSED` bits at PTE level. MGLRU-TS also improves Memcached throughput by 8%.

7 Conclusion

Effectiveness of terabyte-scale tiered memory system depends on precise and timely telemetry data to identify hot/cold pages. Telescope future-proofs telemetry for tiered memory systems with memory capacity up to and beyond terabyte scale with a novel page table profiling technique and demonstrates the effectiveness for large memory footprint applications.

Acknowledgments

We thank the anonymous reviewers for their insightful comments. We would also like to thank Antonio Barbalace and Boris Grot for providing valuable feedback on an early draft of the paper.

References

[1] awslabs/damon-tests: Tests package for correctness verifications and performance evaluations of damon (https://damonitor.github.io). https://github.com/awslabs/damon-tests/tree/next. (Accessed on 07/19/2023).

[2] Damon: Data access monitor | hacklog. https://sjp38.github.io/post/damon/. (Accessed on 07/19/2023).

[3] Decadal plan for semiconductors. https://www.src.org/about/decadal-plan/decadal-plan-full-report.pdf.

[4] Intel® optane™ dc persistent memory product brief. https://www.intel.in/content/dam/www/public/us/en/documents/product-briefs/optane-dc-persistent-memory-brief.pdf. (Accessed on 08/01/2023).

[5] [patch 1/4] mm/damon/dbgfs: Implement recording feature - seongjae park. https://lore.kernel.org/linux-mm/20211008094509.16179-1-sj@kernel.org/. (Accessed on 07/19/2023).

[6] zswap — the linux kernel documentation. https://www.kernel.org/doc/html/latest/admin-guide/mm/zswap.html. (Accessed on 07/22/2023).

[7] Power isa version 3.1. https://openpowerfoundation.org/specifications/isa/, 2020.

[8] core.c - kernel/events/core.c - linux source code (v4.14.15) - bootlin, 2023.

[9] Neha Agarwal and Thomas F. Wenisch. Thermostat: Application-transparent page management for two-tiered main memory. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '17, page 631–644, New York, NY, USA, 2017. Association for Computing Machinery.

[10] Minseon Ahn, Andrew Chang, Donghun Lee, Jongmin Gim, Jungmin Kim, Jaemin Jung, Oliver Rebholz, Vincent Pham, Krishna Malladi, and Yang Seok Ki. Enabling cxl memory expansion for in-memory database management systems. In *Data Management on New Hardware*, DaMoN'22, New York, NY, USA, 2022. Association for Computing Machinery.

[11] Moiz Arif, Kevin Assogba, M. Mustafa Rafique, and Sudharshan Vazhkudai. Exploiting cxl-based memory for distributed deep learning. In *Proceedings of the 51st International Conference on Parallel Processing*, ICPP '22, New York, NY, USA, 2023. Association for Computing Machinery.

[12] Dhruba Borthakur. Petabyte scale databases and storage systems at facebook. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, page 1267–1268, New York, NY, USA, 2013. Association for Computing Machinery.

[13] Irina Calciu, M Talha Imran, Ivan Puddu, Sanidhya Kashyap, Hasan Al Maruf, Onur Mutlu, and Aasheesh Kolli. Rethinking software runtimes for disaggregated memory. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 79–92, 2021.

[14] Wenyan Chen, Kejiang Ye, Yang Wang, Guoyao Xu, and Cheng-Zhong Xu. How does the workload look like in production cloud? analysis and clustering of workloads on alibaba cluster trace. In *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 102–109, 2018.

[15] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 143–154, 2010.

[16] CXL. Compute express link. https://www.computeexpresslink.org/, 2023.

[17] Thaleia Dimitra Doudali, Sergey Blagodurov, Abhinav Vishnu, Sudhanva Gurumurthi, and Ada Gavrilovska. Kleio: A hybrid memory page scheduler with machine intelligence. In *Proceedings of the 28th International*

Symposium on High-Performance Parallel and Distributed Computing, HPDC '19, page 37–48, New York, NY, USA, 2019. Association for Computing Machinery.

[18] Samsung Electronics. Samsung electronics introduces industry's first 512gb cxl memory module. https://semiconductor.samsung.com/newsroom/news/samsung-electronics-introduces-industrys-first-512gb-cxl-memory-module/, 2022.

[19] Jerrin Shaji George, Mohit Verma, Rajesh Venkatasubramanian, and Pratap Subrahmanyam. go-pmem: Native support for programming persistent memory in go. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pages 859–872. USENIX Association, July 2020.

[20] Donghyun Gouk, Miryeong Kwon, Hanyeoreum Bae, Sangwon Lee, and Myoungsoo Jung. Memory pooling with cxl. *IEEE Micro*, 43(2):48–57, 2023.

[21] Graph500. Graph500 benchmark specification. https://graph500.org/?page_id=12, 2017.

[22] Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang G Shin. Efficient memory disaggregation with infiniswap. In *NSDI*, pages 649–667, 2017.

[23] Fei Guo, Seongbeom Kim, Yury Baskakov, and Ishan Banerjee. Proactively breaking large pages to improve memory overcommitment performance in vmware esxi. *SIGPLAN Not.*, 50(7):39–51, mar 2015.

[24] Vishal Gupta, Min Lee, and Karsten Schwan. Heterovisor: Exploiting resource heterogeneity to enhance the elasticity of cloud platforms. In *Proceedings of the 11th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '15, page 79–92, New York, NY, USA, 2015. Association for Computing Machinery.

[25] Christian Hansen. Linux idle page tracking. https://www.kernel.org/doc/Documentation/vm/idle_page_tracking.txt, 2018.

[26] Tejun Heo. Control group v2. https://www.kernel.org/doc/html/latest/admin-guide/cgroup-v2.html, 2015.

[27] Intel. Intel® 64 and ia-32 architectures software developer manuals, 2023.

[28] Intel. Pebs (processor event-based sampling) manual, 2023.

[29] Sudarsun Kannan, Ada Gavrilovska, Vishal Gupta, and Karsten Schwan. Heteroos: Os design for heterogeneous memory management in datacenter. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, page 521–534, New York, NY, USA, 2017. Association for Computing Machinery.

[30] Jonghyeon Kim, Wonkyo Choe, and Jeongseob Ahn. Exploring the design space of page management for Multi-Tiered memory systems. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 715–728. USENIX Association, July 2021.

[31] Sandeep Kumar, Aravinda Prasad, Smruti R. Sarangi, and Sreenivas Subramoney. Radiant: Efficient page table management for tiered memory systems. In *Proceedings of the 2021 ACM SIGPLAN International Symposium on Memory Management*, ISMM 2021, page 66–79, New York, NY, USA, 2021. Association for Computing Machinery.

[32] Redis Labs. Redis. https://redis.io/, 2020. (Accessed on 10/03/2020).

[33] Redis Labs. memtier_benchmark: Nosql redis and memcache traffic generation and benchmarking tool. https://github.com/RedisLabs/memtier_benchmark, 2023.

[34] 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. Software-defined far memory in warehouse-scale computers. ASPLOS '19, page 317–330, New York, NY, USA, 2019. Association for Computing Machinery.

[35] Taehyung Lee, Sumit Kumar Monga, Changwoo Min, and Young Ik Eom. Memtis: Efficient memory tiering with dynamic page classification and page size determination. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 17–34, 2023.

[36] Michael Lespinasse. V2: idle page tracking / working set estimation. https://lwn.net/Articles/460762/, 2023.

[37] Huaicheng Li, Daniel S. Berger, Lisa Hsu, Daniel Ernst, Pantea Zardoshti, Stanko Novakovic, Monish Shah, Samir Rajadnya, Scott Lee, Ishwar Agarwal, Mark D. Hill, Marcus Fontoura, and Ricardo Bianchini. Pond: Cxl-based memory pooling systems for cloud platforms. ASPLOS 2023, page 574–587, New York, NY, USA, 2023. Association for Computing Machinery.

[38] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. Scaling distributed

machine learning with the parameter server. In *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, pages 583–598, 2014.

[39] Lily Looi and Jianping Jane Xu. Intel optane data center persistent memory. In *Proc. HotChips: A Symp. High-Perform. Chips*, 2019.

[40] Chengzhi Lu, Kejiang Ye, Guoyao Xu, Cheng-Zhong Xu, and Tongxin Bai. Imbalance in the cloud: An analysis on alibaba cluster trace. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 2884–2892, 2017.

[41] Hasan Al Maruf, Hao Wang, Abhishek Dhanotia, Johannes Weiner, Niket Agarwal, Pallab Bhattacharya, Chris Petersen, Mosharaf Chowdhury, Shobhit Kanaujia, and Prakash Chauhan. TPP: 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, Volume 3*, ASPLOS 2023, page 742–755, New York, NY, USA, 2023. Association for Computing Machinery.

[42] Memcached. memcached - a distributed memory object caching system. https://memcached.org/, 2020. (Accessed on 10/03/2020).

[43] numactl. numactl - Linux manual page — man7.org. https://man7.org/linux/man-pages/man8/numactl.8.html, 2023. [Accessed 15-Apr-2023].

[44] Nikolaos Charalampos Papadopoulos, Vasileios Karakostas, Konstantinos Nikas, Nectarios Koziris, and Dionisios N. Pnevmatikatos. A configurable tlb hierarchy for the risc-v architecture. In *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*, pages 85–90, 2020.

[45] SeongJae Park, Yunjae Lee, and Heon Y. Yeom. Profiling dynamic data access patterns with controlled overhead and quality. In *Proceedings of the 20th International Middleware Conference Industrial Track*, Middleware '19, page 1–7, New York, NY, USA, 2019. Association for Computing Machinery.

[46] Song Jae Park. Masim: Memory access simulator. https://github.com/sjp38/masim, 2021.

[47] Ivy Peng, Roger Pearce, and Maya Gokhale. On the memory underutilization: Exploring disaggregated memory on hpc systems. In *2020 IEEE 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 183–190, 2020.

[48] Amanda Raybuck, Tim Stamler, Wei Zhang, Mattan Erez, and Simon Peter. 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 '21, page 392–407, New York, NY, USA, 2021. Association for Computing Machinery.

[49] Redis. memtier_benchmark: A high-throughput benchmarking tool for redis & memcached. https://redis.com/blog/memtier_benchmark-a-high-throughput-benchmarking-tool-for-redis-memcached/, 2023.

[50] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the Third ACM Symposium on Cloud Computing*, SoCC '12, New York, NY, USA, 2012. Association for Computing Machinery.

[51] Jie Ren, Dong Xu, Junhee Ryu, Kwangsik Shin, Daewoo Kim, and Dong Li. Mtm: Rethinking memory profiling and migration for multi-tiered large memory. In *Proceedings of the Nineteenth European Conference on Computer Systems*, EuroSys '24, page 803–817, New York, NY, USA, 2024. Association for Computing Machinery.

[52] Dongjoo Seo, Biswadip Maity, Ping-Xiang Chen, Dukyoung Yun, Bryan Donyanavard, and Nikil Dutt. Proswap: Period-aware proactive swapping to maximize embedded application performance. In *2022 IEEE International Conference on Networking, Architecture and Storage (NAS)*, pages 1–4, 2022.

[53] Linus Walleij. Arm32 page tables — linusw, 2023.

[54] Johannes Weiner, Niket Agarwal, Dan Schatzberg, Leon Yang, Hao Wang, Blaise Sanouillet, Bikash Sharma, Tejun Heo, Mayank Jain, Chunqiang Tang, and Dimitrios Skarlatos. Tmo: Transparent memory offloading in datacenters. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '22, page 609–621, New York, NY, USA, 2022. Association for Computing Machinery.

[55] Zi Yan, Daniel Lustig, David Nellans, and Abhishek Bhattacharjee. Nimble page management for tiered memory systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, page 331–345, New York, NY, USA, 2019. Association for Computing Machinery.

[56] Jiaqiao Zhang, Zhili Yao, and Jianlin Feng. Ncredis: An nvm-optimized redis with memory caching. In *International Conference on Database and Expert Systems Applications*, 2021.

[57] Yiying Zhang and Steven Swanson. A study of application performance with non-volatile main memory. In *2015 31st Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10. IEEE, 2015.

[58] Yun Zhang, F.N. Abu-Khzam, N.E. Baldwin, E.J. Chesler, M.A. Langston, and N.F. Samatova. Genome-scale computational approaches to memory-intensive applications in systems biology. In *SC '05: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, pages 12–12, 2005.

[59] Yu Zhao. Multigenerational lru framework. https://lwn.net/Articles/880393/, 2022.