Striking the Right Chord: Parameter Tuning in Memory Tiering Systems

Konstantinos Kanellis*
kkanellis@cs.wisc.edu
University of Wisconsin-Madison
Madison, WI, USA

Sujay Yadalam*
sujayyadalam@cs.wisc.edu
University of Wisconsin-Madison
Madison, WI, USA

Shivaram Venkataraman
shivaram@cs.wisc.edu
University of Wisconsin-Madison
Madison, WI, USA

Michael Swift
swift@cs.wisc.edu
University of Wisconsin-Madison
Madison, WI, USA

Abstract

Memory tiering achieves cost-effective scaling by adding multiple tiers of memory using new interconnect technologies, such as CXL, or denser alternatives, such as NVM. For maximum performance, frequently accessed (hot) data must be placed close to the CPU in faster tiers, while infrequently accessed (cold) data can be placed in farther, slower tiers. Existing tiering solutions employ heuristics and preconfigured thresholds to make data placement and migration decisions. As a result, these systems fail to adapt to different workloads and hardware, resulting in poor performance.

In this paper, we study how the thresholds and parameters used by tiering systems affect application performance. We find that applications are sensitive to these values, and that the optimal values vary significantly across workloads. Furthermore, factors such as application inputs, thread counts, fast–slow memory ratio, and memory latency and bandwidth influence the best parameter choices. Our evaluation shows that properly configuring these parameters can improve performance by up to 2x compared to default configurations.

CCS Concepts: • **Hardware** → **Memory and dense storage**; • **Software and its engineering** → **Memory management**; • **Computing methodologies** → *Machine learning*.

Keywords: Memory Tiering, Compute Express Link (CXL), Parameter Tuning, Bayesian Optimization

ACM Reference Format:
Konstantinos Kanellis, Sujay Yadalam, Shivaram Venkataraman, and Michael Swift. 2025. Striking the Right Chord: Parameter Tuning in Memory Tiering Systems. In *3rd Workshop on Disruptive Memory Systems (DIMES '25), October 13–16, 2025, Seoul, Republic of Korea.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3764862.3768172

*Both authors contributed equally to this work.

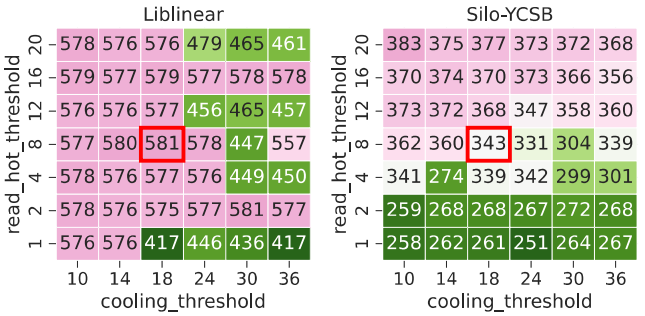

Figure 1. Execution time (in seconds) of *Liblinear* (left) and *Silo* (right), when we tweak two *HeMem* parameters. Default configuration execution time is shown in red box.

DIMES '25, October 13–16, 2025, Seoul, Republic of Korea

ACM ISBN 979-8-4007-2226-4/25/10
https://doi.org/10.1145/3764862.3768172

1 Introduction

Need for memory tiering. Modern data-intensive applications like graph processing, machine learning and in-memory databases demand large amounts of memory for high performance. Due to the scaling limitations of DRAM, this demand has led to memory becoming one of the most significant costs of datacenters [26, 40]. One way to alleviate this problem is to supplement existing DRAM with memory tiers consisting of new memory technologies such as CXL-attached memory [19] or Non-Volatile Memory (NVM) [8]. This enables building systems with vast amounts of memory in a cost-effective manner.

Newer memory technologies such as NVM or CXL memory have higher access latencies and lower bandwidths, which can lead to application performance degradation. To minimize the performance loss, memory tiering systems make smart data placement decisions based on memory access patterns. For instance, they keep frequently accessed (hot) data in faster tiers and infrequently accessed (cold) data in slower tiers. There have been numerous memory tiering systems that aim to smartly place and migrate data, transparent to running applications [13, 22, 25, 31, 43]. These systems

rely on hardware event sampling, page table scanning or page faults to infer application data access patterns, and use heuristics such as access frequency and/or recency to make page placement decisions.

Limitations of existing systems. Our experiments reveal that these systems fail to perform well under all circumstances; they either do not work well with certain applications or on certain hardware configurations. For example, we find that Memtis [22] performs poorly for iterative workloads with changing hot data sets (more in Section 3).

One of the main reasons for such poor performance is the use of pre-configured **static thresholds** used to make data placement decisions. For instance, HeMem [31] uses a parameter called `hot_threshold` to classify hot pages. When a page receives more than `hot_threshold` samples, HeMem considers the page hot and promotes it to the fast tier. Developers of tiering systems perform limited sensitivity studies to find the best values for these parameters. These default values provide reasonable performance for some workloads on certain hardware but are not optimized for specific workload behavior. As a result, HeMem fails to correctly identify all the hot pages and fails to migrate them in a timely manner for many workloads [22, 23]. As shown in Table 1, almost all of today's tiering systems employ such static thresholds to make placement and migration decisions.

Some systems such as Memtis [22] and ArtMem [44] attempt to address the above limitation by dynamically adjusting the *hot_threshold* for page promotion to ensure the correct set of hot pages are loaded in the fast tier. We observe that these systems still underperform because they uses static values for other parameters such as cooling period, adaptation period, and migration period.

To highlight the extent to which static thresholds affect memory tiering performance, we perform a simple parameter sensitivity study using two of HeMem's [31] parameters: `hot_threshold`, which is used to determine when a page is hot (default is 8 sampled accesses), and `cooling_threshold`, which controls when page counts are cooled, i.e., when page counts are halved to imitate an exponential moving average (default is 18 sampled accesses). Figure 1 shows the execution time for two workloads, Liblinear [14] and Silo [39] with a YCSB-C client, under different parameter values. From these experiments, we make two key observations. First, *workloads are sensitive to the thresholds.* Small changes in the values can result in large variations in performance. Second, the *best values of these parameters vary across workloads* and could be significantly different from the default values.

Based on the above observations, in this paper we try to answer the question: *how much performance improvement can be achieved by tuning all tiering-related parameters for each workload?* To this end, we conduct a tuning study of three tiering engines representative of most existing systems: ① HeMem [31], a frequency-based policy with all static thresholds, ② Memtis [22] which uses frequency heuristic and dynamically adjusts the `hot_threshold`, and ③ HMSDK [21], which uses both access frequency and recency data from DAMON [27] access monitoring. We perform tuning on different hardware, vary the number of threads and far:near memory size ratio, and change the application inputs. Instead of randomly exploring the search space, we leverage *Bayesian Optimization* (BO) to identify the best parameter values for each workload in different scenarios, as BO can identify the optimal value of a function with few samples [33]. Although researchers have used BO to determine the best set of knob values in other systems, such as databases [20] and storage systems [5], this work is the first to use BO to tune knobs for memory tiering systems.

	Frequency	**Frequency+Recency**
Static thresholds	**HeMem [31]**, M5 [37] Thermostat [3]	**HMSDK [21]**, TMTS [13] AutoNUMA [28], TPP [25] AutoTiering [43]
Static + dynamic thresholds	**Memtis [22]**, FlexMem [41] ArtMem [44], FreqTier [35]	IDT [6]

Table 1. Taxonomy of tiering systems based on access frequency/recency heuristics and static/dynamic thresholds. Systems studied in this work are highlighted in bold.

From our evaluation, we find that tuning tiering system parameters can yield as much as 2x improvement. Our analysis of the best-performing configurations reveals that tuning parameters leads to accurate classification of hot and cold pages, timely migrations, and a reduction in wasteful migrations. For instance, the Bayesian Optimizer recognizes workloads with streaming access patterns and finds parameter values that avoid wasteful migration of sequentially-accessed pages.

However, we also find that the best parameter values depend not only on the application but also on other factors such as input attributes, thread count, and fast-slow memory ratios. Furthermore, we find that tuning alone cannot help existing systems achieve optimal performance for two reasons. *First*, most tiering systems use simple, rigid heuristics such as access frequency or recency which are not suitable for certain memory access patterns. For example, frequency-based policies such as HeMem and Memtis cannot perform well with streaming/sequential access patterns (see Section 4 for more details). *Second*, most systems rely on noisy, inaccurate page access data from hardware event sampling (e.g. Intel PEBS) or page table scanning (e.g. DAMON [27]). As a result, they may fail to identify all the hot pages or perform wasteful migrations. Hence, for optimal performance, future tiering systems need to adapt to workloads: use suitable heuristics based on the access pattern, adjust thresholds dynamically, and use robust policies to handle the noise in the page access sampling mechanisms.

2 Tuning Methodology

We now describe the tuning environment and methodology, as well as the tiering engines we used for our study.

Specification	PMEM	NUMA
Number of cores	24	20
Processor generation	Icelake	Skylake
Processor frequency (GHz)	3	2.2
L3 cache size (MB)	18	13.75
Far memory type	Optane	NUMA
Max near memory size (GB)	96	96
Max far memory size (GB)	128	96
Max near mem BW (GB/s)	138	56
Max far mem Read/Write BW (GB/s)	7.45/2.25	22/20
Near memory latency (ns)	80	95
Far memory latency (ns)	150 - 250	180

Table 2. Specifications of machines used in our evaluation.

Workload	Inputs	RSS	Perf. Metric	Description
Liblinear	kddb	34.13	Train time	Logistic regression-based classifier.
GapBS-BC	kronecker twitter	78.13 13.08	Elapsed time	Compute the measure of centrality in a graph based on shortest paths.
GapBS-PR	kronecker twitter	71.29 12.32	Elapsed time	Compute PageRank score of a graph.
GapBS-CC	kronecker twitter	69.29 12.09	Elapsed time	Compute connected components of a graph using (Shiloach-Vishkin).
Silo	TPC-C YCSB-C	75.68 71.40	Throughput	In-memory transactional database.
Btree	-	12.13	Throughput	In-memory index lookup benchmark.
XSBench	-	64.97	Elapsed time	Compute kernel of MCNP algorithm.
GUPS	8 GiB hot	64.03	Updates/sec	Skewed accesses with dynamic hotset.

Table 3. Workloads Characteristics. RSS is in GiB.

Hardware Setup. Table 2 shows the hardware specifications of the two machines used in our evaluation. `PMEM` uses Intel Optane DC Persistent Memory DIMMs as the slow tier, whereas the `NUMA` machine (Cloudlab [12]) uses remote NUMA memory to emulate a slow tier.

Workloads. For our evaluation, we select a set of eight representative and diverse workloads shown in Table 3. Specifically, we employ a classifier training kernel from Liblinear [14], three graph processing algorithms from GapBS [4], an HPC workload (XSBench [38]), an in-memory database (Silo [39]), an in-memory index lookup (Btree [1]), and a popular memory intensive micro-benchmark, GUPS [29]. These workloads have been widely used to evaluate tiered memory systems in many prior works [22, 23, 31, 43]. By default, we run these workloads with 12 threads, large enough to saturate the memory bandwidth of each system.

Tiering Configuration. Similar to prior work [22], we configure the ratio of fast to slow tier memory size by setting the fast tier size to the corresponding proportion of the workload resident set size (RSS). Unless otherwise noted, experiments use a 1:8 memory size ratio. For example, for XSBench whose RSS is 65GiB, we set the fast-tier size to 7.22GiB (11%).

Tuning pipeline. To speed up the search for the best tiering parameters, we employ *Bayesian Optimization* (BO), which has been shown to identify the best-performing configurations in as few iterations as possible [33]. Here, we use the state-of-the-art SMAC framework [18], which uses a Random Forest as the surrogate to model the parameter space; this allows it to handle large high-dimensional spaces efficiently. SMAC has been used successfully to tune complex systems with hundreds of knobs [15, 20]. We configure SMAC with a budget of 100 iterations, allocating the first 20 to initial exploration; additionally, SMAC evaluates a random configuration 20% of the time. Each tuning session requires between 10 and 24 hours, depending on the workload and the input.

2.1 Tiering engines overview

HeMem. HeMem [31] is a user-space memory manager for tiered memory systems that is transparently linked to applications. HeMem has ten tunable parameters that control its run-time behavior and decision making.

HeMem monitors page accesses using hardware event sampling such as Intel PEBS [7]. HeMem samples LLC load misses as well as all store instructions. Based on the collected samples, HeMem maintains an access count per page. If the page access count exceeds a pre-defined threshold, HeMem classifies the page as hot; otherwise it considers the page cold. HeMem uses different thresholds for reads (`read_hot_threshold`) and writes (`write_hot_threshold`).

To adapt to workload hot-set changes, HeMem regularly cools the pages it is tracking. When the access count of any page reaches `cooling_threshold`, HeMem halves the access counts of all pages. If the access count falls below the hot threshold after cooling, the page is considered cold. To avoid scanning all pages, HeMem cools pages in batches whose size is determined by another parameter called `cooling_pages`. This is one of the (hidden) parameters that is not discussed in the paper but is part of the implementation.

HeMem includes a migration thread that runs periodically every `migration_period` and promotes hot pages from slower tiers and demotes cold pages from faster tiers. During page migration, pages are write-protected to avoid races between application writes and the data movement. Applications could therefore stall waiting for pages to be migrated.

Memtis. Memtis [22] builds on HeMem by dynamically adjusting the `hot_threshold` based on the workload's access distribution. It maintains a hotness distribution of all pages using a histogram of page access counts. Using the histogram, it selects the `hot_threshold` such that the hot set size is close to the fast tier capacity. To adapt to workload changes, Memtis adjusts the threshold every `adaptation_interval`.

Similar to HeMem, Memtis cools page access counts to decay the impact of old accesses. It performs cooling based on the number of sampled memory accesses. When the total PEBS samples reach a multiple of `cooling_threshold`, all page access counts are halved. Memtis also adjusts the PEBS sampling period to keep the overheads bounded (3% CPU).

Migration threads run periodically (`promotion_interval`, `demotion_interval`) to promote hot pages to the fast tier and demote cold pages to the slow tier.

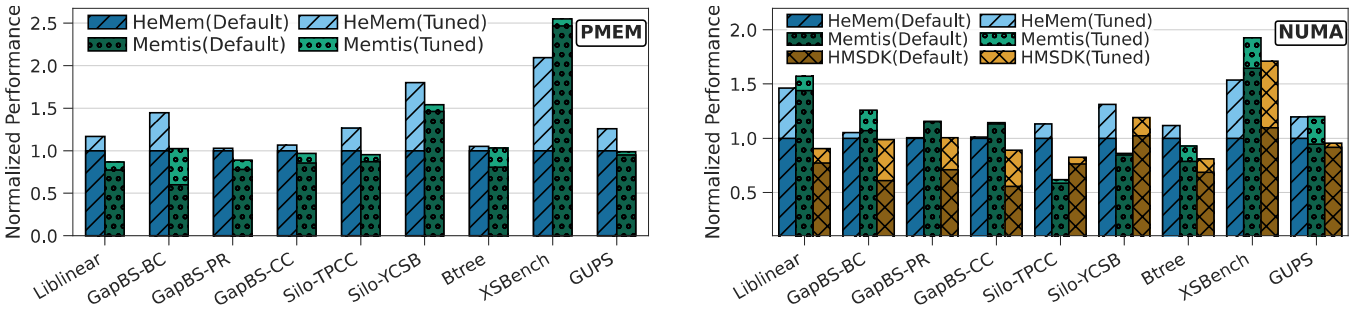

Figure 2. Performance of default and *tuned* configurations for all workloads on PMEM (left) and NUMA (right) nodes. Performance is normalized to HeMem default configuration. Top of the stacked bars show the performance improvement due to tuning.

HMSDK. The Heterogeneous Memory Software Development Kit, HMSDK [21], aims to enhance the efficiency of CXL memory, enabling expanded bandwidth and capacity. Unlike HeMem and Memtis, HMSDK does not rely on hardware event sampling. Instead, it uses the DAMON [27] monitoring framework in Linux for page access information. DAMON scans the Accessed (A) bits in page table entries periodically to identify hot and cold pages. To overcome the overhead of scanning every page table entries (PTE), DAMON divides the address space into regions and monitors only sampled pages in each region. It dynamically adjusts the number of regions by merging and dividing to create homogeneous regions such that pages within a region are accessed similarly.

DAMON includes thresholds related to the number of regions (e.g., `min_regions`, `max_regions`), and PTE scanning frequency (`sample_interval`, `aggr_interval`). Using DAMON's page access information, HMSDK promotes pages accessed at least `prom_accesses` times in the last `prom_age` seconds. Similarly, it demotes pages that have fewer than `dem_accesses` in the last `dem_age` seconds.

3 Results

3.1 Performance benefit of parameter tuning.

We begin by comparing the performance of the best parameter configurations found by the optimizer with that of the default configuration. Figure 2 show the performance benefits of tuning. HMSDK does not support NVM as the slow tier, so we do not run it on PMEM. We run all the three engines on NUMA. For almost all workloads, tuning the parameter values provides superior performance by 1.07-2.09x. HeMem and HMSDK benefit the most since all their thresholds are static. Although Memtis dynamically adjusts the `hot_threshold`, it still benefits from tuning of its other static thresholds.

Interestingly, we find that tuned HeMem outperforms Memtis for most workloads. There are three main reasons for this. First, Memtis reserves some free space at all times in DRAM for new allocations, which results in wasted useful DRAM space. Second, Memtis manages both basepages and hugepages which increases management overhead and migration costs. Third, Memtis performs cooling very infrequently (tens of seconds) compared to HeMem (tens of milliseconds). This results in delayed promotion of new hot pages which is explained below.

	Reason(s) for improvement	Important knobs
HeMem	Accurate hot page classification Promote hot pages earlier Reduce unnecessary migrations	hot_threshold cooling_threshold sampling_period
Memtis	React faster to hot set change Promote hot pages earlier	adaptation_period cooling_threshold
HMSDK	Accurate hot page classification Timely migrations	sample_interval aggr_interval promo:{accesses, age}

Table 4. Reasons for performance improvement with tuning and the most important knobs for each tiering system.

Understanding tuning benefits. As summarized in Table 4, tuned configurations achieve better performance due to:

① *Accurate hot and cold page classification*: With tuned sampling frequencies and hotness thresholds, HeMem and HMSDK can identify hot and cold pages correctly. Consider Silo with YCSB-C workload: default HeMem and HMSDK cannot identify all of the hot pages. The optimizer increases the sampling frequency and lowers the threshold to ensure the hot pages are correctly detected and promoted.

② *Adapting to hot set changes*: With default parameter values, we observe that all the three tiering engines react slowly to hot set changes. This happens mainly due to inaccurate cooling thresholds. In Memtis, due to inaccurate adaptation period and cooling threshold, previously hot pages continue to stay hot for too long, occupying the local tier. Memtis does not demote these pages quickly, so it cannot promote the newly hot pages to the fast tier.

③ *Eliminating unnecessary migrations*: Due to incorrect thresholds, HeMem promotes pages with fluctuating access counts and pages with short bursts of accesses. With GapBS-PR and GapBS-CC, that have streaming access patterns, default HeMem migrates many pages *after* they are accessed, which is not useful. When tuned, the thresholds are adjusted so that HeMem does not migrate pages that are part of the streaming accesses.

3.2 Parameter sensitivity to application inputs.

We find that the best parameter values vary not only across applications, but also across different inputs to the same

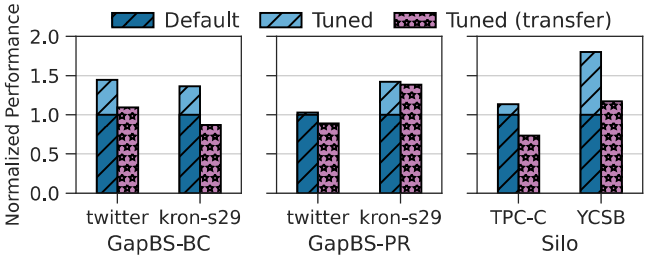

Figure 3. Performance of HeMem's best configuration compared to default, for two application inputs. We also show how each best configuration performed when HeMem used it while running on the alternative input (i.e., *transfer*).

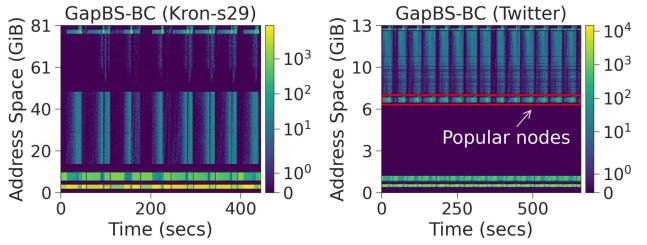

Figure 4. *GapBS-BC* memory access pattern over time, for Kronecker-s29 (left) and Twitter (right) input graphs.

application. This is intuitive because different inputs can cause applications to have different memory access patterns. For instance, changing the input graph of a graph algorithm can change the access pattern, and changing the number of threads can affect an application's bandwidth utilization.

To understand how the best values are dependent on the input, we use the tuned knob values obtained for one input with the other input. For example, we run *GapBS-BC* on a `Twitter` graph using the best configuration obtained for the Kronecker graph. Figure 3 shows the results. In most cases, the best configuration generated for one input does not perform well for the other input. In fact, we observe that the performance is worse than the default in some cases.

The above behavior occurs because even minor differences in memory access patterns require very different configurations. Consider the input graphs to GapBS-BC: Twitter graph and Kronecker graph. As shown in Figure 4, the access patterns are similar except for one difference. With the Twitter graph, there are a handful of popular nodes on a small set of pages as highlighted. This results in frequent accesses to these pages, whereas in Kronecker all pages receive almost equal number of accesses. The tuned configuration for Twitter graph uses thresholds that ensure the popular pages form the hot set, while the best configuration for `Kronecker` graph focuses on reducing wasteful migrations of pages with similar access intensities.

3.3 Parameter sensitivity to different thread counts.

We now measure application performance with varying application thread counts. Figure 5 shows that tuning can

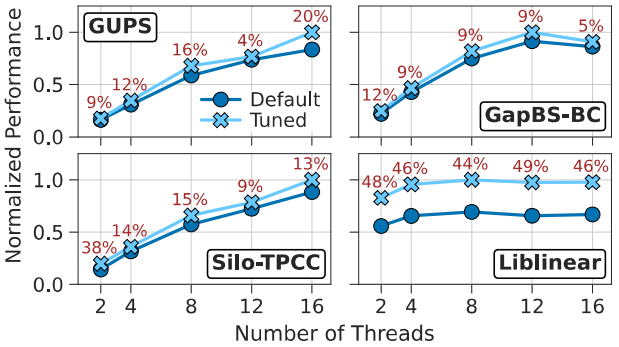

Figure 5. Performance of HeMem's default and tuned configuration when varying thread count. Performance normalized to the *best* tuned configuration. Improvement of tuned over default config. for each thread count is annotated in red.

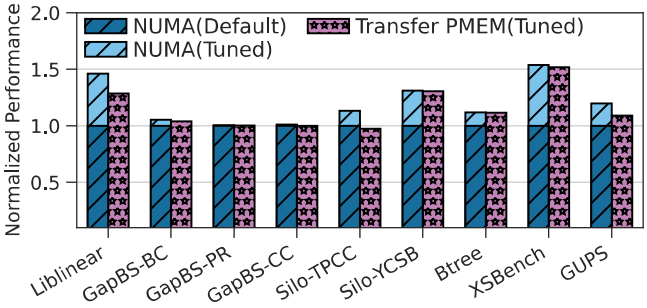

Figure 6. Performance gains of `PMEM`'s HeMem tuned configuration when deployed on NUMA node. We also show the performance of NUMA's default and tuned configuration.

provide performance benefits for all thread counts. Increasing the number of threads increases memory traffic, which usually results in more accesses per page. Therefore, the `hot_threshold` parameter should be adjusted accordingly.

We find that the relation between `hot_threshold` and thread count to be non-linear. Thus, manually identifying the right threshold can be challenging. In addition, other parameters such as `cooling_threshold` also affect performance significantly. For some workloads (i.e., Silo-TPCC), we observe that the best-performing configurations are similar at different thread counts. For other workloads (i.e., GapBS-BC, Liblinear), there is no single configuration that achieves best performance at all thread counts.

3.4 Parameter sensitivity to different hardware.

We run experiments to see if the tuned parameters for a workload on one machine can be used on another machine. Prior works have observed that tuning database parameters is challenging in cloud settings due to performance variability (noise) across hardware [15]. In contrast, we observe that the best-performing configurations on one machine (PMEM) generally also perform well on other machines (NUMA), as shown in Figure 6. This suggests that it is possible to transfer the tuned parameters between machines as long as their memory characteristics (latency and bandwidth) are similar.

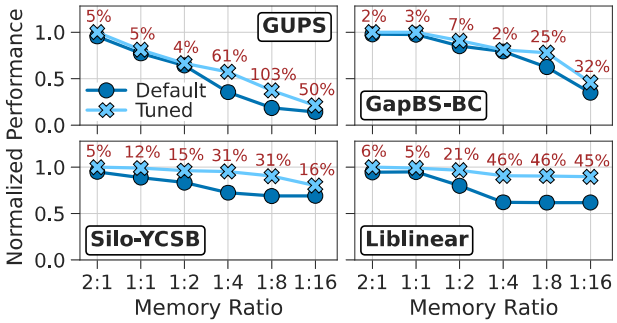

Figure 7. Performance of HeMem's default and tuned configuration when fast-slow memory ratio. Performance normalized to the *best* tuned config. Improvement of tuned over default config. for each memory ratio is annotated in red.

3.5 Tuning for different memory configurations.

Finally, we measure application performance with varying fast-slow memory tier size ratios and evaluate the importance of parameter tuning. Figure 7 shows the performance of GUPS, GapBS-BC, Silo-TPCC and Liblinear with HeMem in different ratios normalized to the 2:1 configuration.

We find that the best-performing HeMem configurations are stable within neighborhoods of similar ratios but shift abruptly at specific breakpoints: e.g., when the hot set no longer fits in the fast tier. With Silo-TPCC, the same HeMem configuration achieves best performance at 1:1, 1:2, and 1:4 configurations. At the 1:8 ratio, this configuration performs poorly and the best parameter values are much different. In particular, at lower ratios, the optimizer sets the `write_hot_threshold` to a higher value so that only a few of the hottest pages, those that reach the high threshold, are kept in the fast tier.

Second, we observe that tuning parameters helps more at smaller fast tier sizes compared to larger ones. At small fast-slow ratios, the tiering engine must rank pages with high precision because the admitted hot set is narrow; misclassifying even a few pages incurs a high opportunity cost. Identifying the top ~20% pages is inherently harder than identifying a broader ~50% set, so hotness thresholds, smoothing factors, and migration budgets require tighter tuning.

Since Memtis adjusts its hotness threshold based on the size of the fast tier, it can better adapt to different memory ratios than HeMem and HMSDK. Our experiments with Memtis reveal that a single tuned configuration provides performance benefits across all memory ratios, unlike HeMem and HMSDK.

4 Discussion

Offline tuning of the tiering parameters can yield significant performance gains, but can be very time-consuming, as it has to be performed for each workload, i.e., there is no one best configuration that works well in all scenarios. The best parameter values depend not only on the workload but also on the inputs, thread count, and memory configurations.

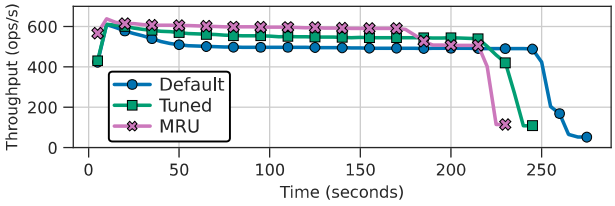

Figure 8. Silo throughput against a TPC-C client with 3 configurations: default HeMem, tuned HeMem, and a simple recency-based policy (MRU).

Furthermore, we find that tuning of existing tiering systems does not reach optimal performance in many cases due to the following shortcomings.

Different workload phases could require different parameter values. Many workloads have distinct execution phases. For instance, Liblinear classifier training has a data preparation phase, and then in every training iteration, forward and back propagation phases. The memory access behavior is different in each of these phases. Existing tiering systems that rely on static thresholds are unable to identify and adapt to the different phases. Furthermore, tuning parameters may also not help achieve the highest performance. A single tuned configuration may fail to accelerate all phases. The best parameters for one phase could be different from the best values for another phase.

A single simple heuristic does not work for all workloads. Most tiering systems use simple heuristics such as access frequency or recency, along with static thresholds, as shown in Table 1. We observe that different workloads work well with different heuristics. Tuning is insufficient in scenarios where an unsuitable heuristic is used with a workload. For example, we find that Silo running against a TPC-C workload with a simple, untuned recency-based policy performs better than even a tuned frequency-based policy, as shown in Figure 8. TPC-C follows the latest distribution: newly inserted tuples in the tables are the hottest, and they get colder over time [36]. Frequency-based policies such as HeMem, even when tuned, take longer to identify and promote pages with new hot tuples.

Input signals could be noisy, and existing systems are not designed to handle the noise. Most systems rely on hardware event sampling (Intel PEBS, AMD IBS) or page table scanning (DAMON) to track page accesses. The sampling/scanning frequency affects the accuracy and overhead. We find that PEBS sampling can be quite noisy at low frequencies, which can lead to wasteful migrations in tiering systems. For example, two pages accessed equally by the application may receive different sample counts over short time intervals. This is evident with XSBench that has the access pattern shown in Figure 9. Two adjacent pages in the region at the top (green) receive different samples. Tuning might help identify such scenarios and select high sampling/scanning frequencies. However, this does not solve the problem fully. PEBS cannot sample memory writes (LLC

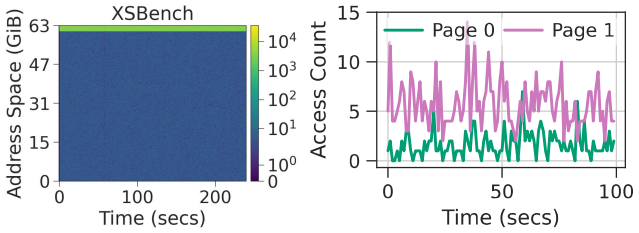

Figure 9. *XSBench*: Memory access pattern over time (left), and page access count for two pages that are accessed uniformly and equally (right). Due to PEBS sampling inaccuracies, one page (i.e., Page 1) might appear hotter than another.

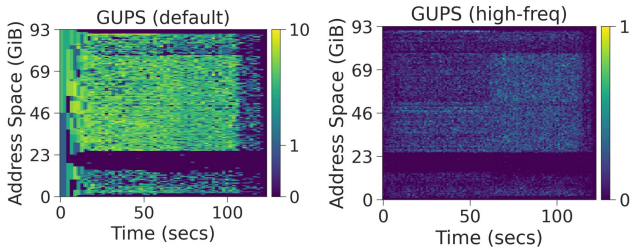

Figure 10. Heatmap of GUPS generated by DAMON, using default HMSDK scanning frequency (left) and high scanning frequency (right). Even at high scanning frequency, HMSDK cannot distinguish between hot and cold regions.

cacheline write-backs) and prefetches since these events are not tied to any instruction, so tiering policies do not have a full view of the memory accesses.

Similarly, DAMON also fails to identify the hot regions in workloads with a high memory access rate. Setting the scanning frequency to a high value does not help. Figure 10 shows how HMSDK with DAMON cannot distinguish between hot and cold regions in GUPS even at high scanning frequencies. Existing tiering systems are not equipped to handle such noisy data and can misclassify and migrate pages repeatedly. One way to overcome this is by building novel page-access tracking mechanisms, such as the CXL-driven hot page tracking proposed in M5 [37].

5 Related Work

Page-based memory tiering systems. Most proposed tiering systems perform page migrations transparently to the applications. Such systems are based on heuristics and use static thresholds to make policy decisions. Some use recency-based policies [25, 42], others use frequency-based policies [3, 13, 21, 31, 43], while some use both recency and frequency information to make data placement decisions [13, 17, 24]. We find that these systems are suboptimal as they do not adapt to the workload or the underlying hardware.

Dynamic behavior of tiering systems. Memtis [22] uses dynamic threshold adaptation for page hotness classification, which leads to better fast memory tier utilization. Cori [10] tunes the periodicity of data movement in hybrid memory systems by extracting data reuse patterns from the application. Other systems migrate pages at different granularities for different workloads [2, 32]. Instead of adjusting the parameters, some tiering systems use different policies for different workloads. Heo et. al., [17] propose a dynamic policy selection mechanism which identifies the best migration policy among LRU, LFU and random for a given workload. Yu et. al., [45] build bandwidth-aware tiering systems.

Machine learning for data placement. Sibyl [34] uses Reinforcement Learning (RL) for data placement in hybrid storage systems. IDT [6] and ArtMem [44] also use RL to dynamically adjust some tiering parameters. They still rely on some static parameters to make decisions. Kleio [9] uses deep neural networks to make smarter page placement decisions.

Hardware tiering and profile-guided data placement. To overcome the inaccuracies and inefficiencies in software profiling, Ramos et. al., [30] propose augmenting the memory controller hardware to monitor access pattern and migrate pages between memories which would be more efficient. X-Mem [11] and Mira [16] use profile-guided techniques to determine object hotness offline and make data placement decisions during allocation time. These approaches do not work well for applications whose hot-set changes over time.

6 Conclusion

Datacenters today serve a diverse mix of workloads, many of which demand large memory capacities and bandwidth. Memory tiering using CXL and NVM enables cost-effective memory expansion. Existing tiering policies aim to reduce the long latency accesses to slow tiers to limit performance degradation. In that process, they employ static thresholds to make data placement decisions, which leads to suboptimal performance. We find that tuning these thresholds per workload and hardware configuration can yield significant performance benefits (up to 2x).

However, tuning can be quite expensive as the best thresholds depend not just on the application but also on multiple factors: inputs, far-near memory ratio, thread count, and hardware configuration. Future tiering systems need to either adapt their thresholds to workload behavior and hardware characteristics or get rid of them. Although eliminating all thresholds might not be feasible, it is important to have as few parameters as possible that are robust and insensitive to the factors mentioned above.

Acknowledgments

We thank Fanchao Chen for sharing insights from his experiments. We thank the anonymous reviewers for their valuable comments that improved this paper. This work was supported by NSF "Expeditions in Computing: Learning Directed Operating System (LDOS)" Grant No. 2326576, and by PRISM, one of seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

References

[1] Reto Achermann and Ashish Panwar. 2020. Mitosis Workload Btree. https://github.com/mitosis-project/mitosis-workload-btree.

[2] Shashank Adavally and Krishna Kavi. 2021. Subpage migration in heterogeneous memory systems. In *Workshop on Heterogeneous Memory Systems (HMEM-2021)*.

[3] Neha Agarwal and Thomas F Wenisch. 2017. 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*.

[4] Scott Beamer, Krste Asanović, and David Patterson. 2015. The GAP benchmark suite. *arXiv preprint arXiv:1508.03619* (2015).

[5] Zhen Cao, Geoff Kuenning, and Erez Zadok. 2020. Carver: Finding Important Parameters for Storage System Tuning. In *18th USENIX Conference on File and Storage Technologies*.

[6] Juneseo Chang, Wanju Doh, Yaebin Moon, Eojin Lee, and Jung Ho Ahn. 2024. Idt: Intelligent data placement for multi-tiered main memory with reinforcement learning. In *Proceedings of the 33rd International Symposium on High-Performance Parallel and Distributed Computing*.

[7] Intel Corporation. 2018. Intel 64 and IA-32 Architectures Software Developer Manuals. https://software.intel.com/articles/intel-sdm. (2018).

[8] Ian Cutress and Billy Tallis. 2018. Intel Launches Optane DIMMs up to 512GB: Apache Pass is Here. https://www.anandtech.com/show/12828/intel-launches-optane-dimms-up-to-512gb-apache-pass-is-here.

[9] Thaleia Dimitra Doudali, Sergey Blagodurov, Abhinav Vishnu, Sudhanva Gurumurthi, and Ada Gavrilovska. 2019. Kleio: A hybrid memory page scheduler with machine intelligence. In *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*.

[10] Thaleia Dimitra Doudali, Daniel Zahka, and Ada Gavrilovska. 2021. Cori: Dancing to the right beat of periodic data movements over hybrid memory systems. In *2021 IEEE International Parallel and Distributed Processing Symposium*.

[11] 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 Eleventh European Conference on Computer Systems*.

[12] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. 2019. The Design and Operation of CloudLab. In *2019 USENIX Annual Technical Conference*.

[13] Padmapriya Duraisamy, Wei Xu, Scott Hare, Ravi Rajwar, David Culler, Zhiyi Xu, Jianing Fan, Christopher Kennelly, Bill McCloskey, Danijela Mijailovic, et al. 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*.

[14] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. 2008. LIBLINEAR: A library for large linear classification. *The Journal of machine Learning research* (2008).

[15] Johannes Freischuetz, Konstantinos Kanellis, Brian Kroth, and Shivaram Venkataraman. 2025. TUNA: Tuning Unstable and Noisy Cloud Applications. In *Proceedings of the Twentieth European Conference on Computer Systems* (Rotterdam, Netherlands) *(EuroSys '25)*. Association for Computing Machinery, New York, NY, USA, 954–973. https://doi.org/10.1145/3689031.3717480

[16] Zhiyuan Guo, Zijian He, and Yiying Zhang. 2023. Mira: A program-behavior-guided far memory system. In *Proceedings of the 29th Symposium on Operating Systems Principles*.

[17] Taekyung Heo, Yang Wang, Wei Cui, Jaehyuk Huh, and Lintao Zhang. 2020. Adaptive page migration policy with huge pages in tiered memory systems. *IEEE Trans. Comput.* (2020).

[18] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 2011. Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization: 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers 5*.

[19] Compute Express Link Consortium. Inc. 2020. CXL® Specification. https://computeexpresslink.org/cxl-specification/.

[20] Konstantinos Kanellis, Cong Ding, Brian Kroth, Andreas Müller, Carlo Curino, and Shivaram Venkataraman. 2022. LlamaTune: sample-efficient DBMS configuration tuning. *Proceedings of VLDB Endowment* (2022).

[21] KyungSoo Lee, Sohyun Kim, Joohee Lee, Donguk Moon, Rakie Kim, Honggyu Kim, Hyeongtak Ji, Yunjeong Mun, and Youngpyo Joo. 2024. Improving key-value cache performance with heterogeneous memory tiering: A case study of CXL-based memory expansion. *IEEE Micro* (2024).

[22] Taehyung Lee, Sumit Kumar Monga, Changwoo Min, and Young Ik Eom. 2023. MEMTIS: Efficient Memory Tiering with Dynamic Page Classification and Page Size Determination. In *Proceedings of the 29th Symposium on Operating Systems Principles*.

[23] Baptiste Lepers and Willy Zwaenepoel. 2023. Johnny Cache: the End of {DRAM} Cache Conflicts (in Tiered Main Memory Systems). In *17th USENIX Symposium on Operating Systems Design and Implementation*.

[24] Adnan Maruf, Ashikee Ghosh, Janki Bhimani, Daniel Campello, Andy Rudoff, and Raju Rangaswami. 2022. MULTI-CLOCK: Dynamic Tiering for Hybrid Memory Systems.. In *2022 IEEE International Symposium on High-Performance Computer Architecture*.

[25] Hasan Al Maruf, Hao Wang, Abhishek Dhanotia, Johannes Weiner, Niket Agarwal, Pallab Bhattacharya, Chris Petersen, Mosharaf Chowdhury, Shobhit Kanaujia, and Prakash Chauhan. 2023. 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*.

[26] Timothy Morgan. 2020. CXL And Gen-Z Iron Out A Coherent Interconnect Strategy. https://www.nextplatform.com/2020/04/03/cxl-and-gen-z-iron-out-a-coherent-interconnect-strategy/. In *The Next Platform*.

[27] SeongJae Park, Yunjae Lee, and Heon Y Yeom. 2019. Profiling dynamic data access patterns with controlled overhead and quality. In *Proceedings of the 20th International Middleware Conference Industrial Track*.

[28] [patch -v4 0/3] memory tiering: hot page selection. 2022. https://lwn.net/ml/linux-kernel/20220622083519.708236-1-ying.huang@intel.com/.

[29] Steven J Plimpton, Ron Brightwell, Courtenay Vaughan, Keith Underwood, and Mike Davis. 2006. A simple synchronous distributed-memory algorithm for the HPCC RandomAccess benchmark. In *2006 IEEE International Conference on Cluster Computing*.

[30] Luiz E Ramos, Eugene Gorbatov, and Ricardo Bianchini. 2011. Page placement in hybrid memory systems. In *Proceedings of the international conference on Supercomputing*.

[31] 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*.

[32] Jee Ho Ryoo, Lizy K John, and Arkaprava Basu. 2018. A case for granularity aware page migration. In *Proceedings of the 2018 International Conference on Supercomputing*.

[33] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. 2015. Taking the human out of the loop: A review of Bayesian optimization. *Proc. IEEE* (2015).

[34] Gagandeep Singh, Rakesh Nadig, Jisung Park, Rahul Bera, Nastaran Hajinazar, David Novo, Juan Gómez-Luna, Sander Stuijk, Henk Corporaal, and Onur Mutlu. 2022. Sibyl: Adaptive and extensible data

placement in hybrid storage systems using online reinforcement learning. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*.

[35] Kevin Song, Jiacheng Yang, Sihang Liu, and Gennady Pekhimenko. 2023. Lightweight frequency-based tiering for cxl memory systems. *arXiv preprint arXiv:2312.04789* (2023).

[36] Radu Stoica and Anastasia Ailamaki. 2013. Enabling efficient OS paging for main-memory OLTP databases. In *Proceedings of the Ninth International Workshop on Data Management on New Hardware*.

[37] Yan Sun, Jongyul Kim, Zeduo Yu, Jiyuan Zhang, Siyuan Chai, Michael Jaemin Kim, Hwayong Nam, Jaehyun Park, Eojin Na, Yifan Yuan, et al. 2025. M5: Mastering Page Migration and Memory Management for CXL-based Tiered Memory Systems. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*.

[38] John R Tramm, Andrew R Siegel, Tanzima Islam, and Martin Schulz. 2014. XSBench-the development and verification of a performance abstraction for Monte Carlo reactor analysis. *The Role of Reactor Physics toward a Sustainable Future* (2014).

[39] Stephen Tu, Wenting Zheng, Eddie Kohler, Barbara Liskov, and Samuel Madden. 2013. Speedy transactions in multicore in-memory databases. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*.

[40] Johannes Weiner, Niket Agarwal, Dan Schatzberg, Leon Yang, Hao Wang, Blaise Sanouillet, Bikash Sharma, Tejun Heo, Mayank Jain, Chunqiang Tang, et al. 2022. TMO: Transparent memory offloading in datacenters. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*.

[41] Dong Xu, Junhee Ryu, Kwangsik Shin, Pengfei Su, and Dong Li. 2024. {FlexMem}: Adaptive page profiling and migration for tiered memory. In *2024 USENIX Annual Technical Conference*.

[42] Zi Yan, Daniel Lustig, David Nellans, and Abhishek Bhattacharjee. 2019. Nimble page management for tiered memory systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*.

[43] Zhengyu Yang, Morteza Hoseinzadeh, Allen Andrews, Clay Mayers, David Thomas Evans, Rory Thomas Bolt, Janki Bhimani, Ningfang Mi, and Steven Swanson. 2017. AutoTiering: Automatic data placement manager in multi-tier all-flash datacenter. In *2017 IEEE 36th International Performance Computing and Communications Conference*.

[44] Xinyue Yi, Hongchao Du, Yu Wang, Jie Zhang, Qiao Li, and Chun Jason Xue. 2025. ArtMem: Adaptive Migration in Reinforcement Learning-Enabled Tiered Memory. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture*.

[45] Seongdae Yu, Seongbeom Park, and Woongki Baek. 2017. Design and implementation of bandwidth-aware memory placement and migration policies for heterogeneous memory systems. In *Proceedings of the International Conference on Supercomputing*.