Published: 15 December 2025

Middleware '25: 26th International Middleware Conference
December 15 - 19, 2025
TN, Nashville, USA

RESEARCH-ARTICLE

MTAT: Adaptive Fast Memory Management for Co-located Latency-Critical Workloads in Tiered Memory System

MINHO KIM, Daegu Gyeongbuk Institute of Science and Technology, Daegu, South Korea

SEONGGYU HAN, Daegu Gyeongbuk Institute of Science and Technology, Daegu, South Korea

GYEONGSEO PARK, Electronics and Telecommunications Research Institute, Daejeon, South Korea

DAEHOON KIM, Yonsei University, Seoul, South Korea

Open Access Support provided by:

Daegu Gyeongbuk Institute of Science and Technology

Electronics and Telecommunications Research Institute

Yonsei University

MTAT: Adaptive Fast Memory Management for Co-located Latency-Critical Workloads in Tiered Memory System

Minho Kim*
Daegu Gyeongbuk Institute of Science and Technology
Daegu, South Korea
mhkim@dgist.ac.kr

Seonggyu Han*
Daegu Gyeongbuk Institute of Science and Technology
Daegu, South Korea
sghan@dgist.ac.kr

Gyeongseo Park
Electronics and Telecommunications Research Institute
Daejeon, South Korea
gspark@etri.re.kr

Daehoon Kim†
Yonsei University
Seoul, South Korea
daehoonkim@yonsei.ac.kr

Abstract

Modern data centers increasingly employ multi-tenant deployment models in which multiple applications or virtual machines share a single physical server. However, existing tiered memory management schemes classify pages solely by access frequency to govern promotions and demotions across memory tiers without accounting for the distinct access patterns of latency-critical (LC) and best-effort (BE) workloads. LC workloads demand low-latency service yet lack sustained high-frequency access; consequently, frequency-based tiering demotes LC data to slower memory (SMem), degrading responsiveness and violating service-level objectives (SLOs).

To address these challenges, we propose MTAT, an adaptive tiered memory management framework that guarantees the SLO of LC workloads while maintaining overall system performance for BE workloads. Rather than relying solely on hotness-based page placement, MTAT employs distinct policies for LC and BE workloads by isolating them into dedicated fast memory (FMem) partitions. Specifically, MTAT employs reinforcement learning to identify the minimal FMem capacity necessary to satisfy stringent SLOs, supporting rapid response to sudden demand surges, and uses a simulated annealing algorithm to allocate the remaining FMem fairly among co-located BE workloads. Compared to state-of-the-art tiered memory page-placement solutions, MTAT improves the maximum throughput of LC workloads by up to 1.7× and enhances BE workloads' fairness by up to 3.3×, all while incurring only a 19% throughput penalty at worst.

CCS Concepts

• **Software and its engineering** → *Main memory*; • **Computer systems organization** → *Processors and memory architectures.*

*Both authors contributed equally to this research.
†Corresponding author.

Keywords

Tiered memory system, Memory management, Latency-critical workload, Workload co-location, Machine learning

ACM Reference Format:
Minho Kim, Seonggyu Han, Gyeongseo Park, and Daehoon Kim. 2025. MTAT: Adaptive Fast Memory Management for Co-located Latency-Critical Workloads in Tiered Memory System. In *26th ACM Middleware Conference (Middleware '25), December 15–19, 2025, Nashville, TN, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3721462.3770767

1 Introduction

Modern data centers increasingly employ multi-tenant deployment models in which multiple applications or virtual machines share a single physical server [1, 14]. To maximize resource utilization, providers consolidate workloads onto fewer machines and adopt tiered memory architectures [19, 25, 44] that combine a small, high-speed tier (*e.g.*, DRAM) with a larger, lower-speed tier (*e.g.*, Compute Express Link-based memory [3], Non-Volatile Memory [12]). Although accesses to the slower tier (SMem) incur higher latencies, these architectures mitigate such penalties by retaining frequently accessed ("hot") pages in the fast tier (FMem) and relegating infrequently accessed ("cold") pages to SMem. Consequently, current tiered memory systems present the heterogeneous memory pool as a unified, large-capacity FMem, thereby minimizing overall latency impacts.

The issue is that existing tiered memory management schemes predominantly rely on page hotness–quantified by access frequency–to govern promotions and demotions across memory tiers. This frequency-driven approach delivers simplicity and strong performance for memory-intensive best-effort (BE) workloads, but it undermines latency-critical (LC) applications in consolidated, multi-tenant settings. LC workloads inherently exhibit irregular, bursty access patterns due to their direct interaction with end users, causing their pages to appear cold relative to the steady, high-intensity usage of BE workloads [13, 41]. As a result, LC pages are frequently demoted to SMem, incurring substantial latency penalties and making it increasingly difficult to satisfy the strict service-level objectives (SLOs) of LC workloads when competing with BE workloads for memory resources.

To address these challenges, we introduce MTAT, an adaptive tiered-memory management framework that guarantees SLO compliance for LC workloads while maintaining overall performance for BE tenants. Rather than relying on a uniform hotness-driven policy, MTAT isolates LC and BE workloads into dedicated FMem partitions. For LC workloads, it leverages reinforcement learning to predict the minimal FMem allocation required to meet stringent SLOs, thereby enabling rapid adaptation to sudden request surges. The remaining FMem is then assigned among co-located BE workloads using a fairness-oriented simulated annealing (SA) algorithm. MTAT periodically revisits these partitioning decisions to adapt to the LC workload's fluctuating demands, always prioritizing its SLO requirements. Between repartition intervals, MTAT continuously monitors FMem and employs workload-specific page exchanges between FMem and SMem to ensure that each workload's hottest data remains resident in FMem.

We implement an MTAT prototype by separating its functionality into two components: a user-space daemon that determines the memory partitioning policy and a kernel-space daemon that enforces the actual partition assignment. Our experiments confirm that MTAT adjusts the FMem allocation for the LC workload in response to varying demands, consistently satisfying the workload's SLO. Compared with the state-of-the-art tiered memory system page-placement solutions, TPP [25] and MEMTIS [23], MTAT increases the LC workload's maximum throughput by 73% and 34%, respectively, while improving BE workloads' fairness by 3.3× and 1.5×, respectively. Even when MTAT prioritizes FMem for the LC workload—thereby reducing FMem allocation for BE workloads–its overall throughput decreases by at most 19%.

The key contributions are the following:

- We demonstrate that purely hotness-driven tiered memory management is ill-suited for latency-critical workloads colocated with best-effort workloads in multi-tenant environments, leading to substantial SLO violations.
- We propose MTAT, an adaptive tiered-memory management framework that explicitly addresses the heterogeneous access patterns of LC and BE workloads, isolating fast memory partitions to guarantee LC SLO while improving overall system performance.
- We evaluate MTAT against state-of-the-art page placement solutions and demonstrate its superiority by achieving up to 1.7× higher throughput for LC workloads and 3.3× improved fairness for BE workloads, all while incurring minimal overhead.

2 Motivation

2.1 Tiered Memory System in Multi-tenant environments

Modern data centers increasingly employ multi-tenancy, allowing applications and virtual machines to share CPU, memory, and storage on the same server to enhance utilization and reduce operational overhead [1, 14, 17, 24]. To keep pace with rising workload demands without incurring proportional hardware expenditures, providers are consolidating an ever-growing number of tenants onto fewer machines, steadily elevating consolidation ratios. While this strategy yields substantial efficiency gains and cost savings, it also intensifies memory pressure–a direct consequence of the inherent scalability limits of modern DRAM modules [35].

To address this memory bottleneck, many providers now adopt tiered memory architectures as a foundational element of their server designs. In such systems, a high-performance "fast" tier (FMem), typically composed of DRAM modules, is paired with a higher-capacity "slow" tier (SMem)–often realized via NVM or CXL-attached memory. Latency-sensitive portions of an application's working set reside in the FMem, while less-frequently accessed data is placed in the SMem. However, because FMem capacity has not kept pace with demand, multiple applications must compete for constrained memory resources. With rising consolidation ratios in modern multi-tenant data centers, FMem has become an ever-more highly contended shared resource.

2.2 FMem Contention in Multi-Tenant LC/BE Co-Locations

The heterogeneous workload characteristics typical of multi-tenant environments exacerbate FMem contention. Prior work on tiered memory systems has predominantly focused on identifying the hottest pages and designing lightweight mechanisms to migrate them between FMem and SMem [19, 23, 25, 26, 29, 32, 34, 39]. Commonly, these mechanisms periodically scan page-access counters to isolate high-frequency pages and apply migration policies driven by recency or frequency thresholds. Such approaches implicitly assume that servers execute memory-intensive, compute-bound batch workloads (*i.e.*, BE workloads) with relatively stable and predictable demands over long intervals. In contrast, practical multi-tenant deployments frequently co-locate these BE workloads with short-lived, latency-critical workloads (*i.e.*, LC workloads) that prioritize rapid user responsiveness [21, 31, 43]. Although LC workloads require low-latency access, they may not generate sustained high-frequency memory accesses. Consequently, conventional frequency-based tiering can inadvertently evict latency-critical data into SMem, impairing LC responsiveness and risking violations of SLOs.

Ultimately, LC data eviction arises from the inherently irregular access patterns of LC workloads. Figure 1 illustrates this effect by showing maximum throughput across different FMem–SMem allocation ratios, with load intensity varied to evaluate its impact on performance. Our experiments employ a tiered memory system with 32 GB of DRAM as FMem and 256 GB of emulated CXL memory as SMem, managed by MEMTIS [23], a state-of-the-art page placement policy. We subject four LC workloads–Redis, Memcached, MongoDB, and Silo–to uniformly distributed requests for 60 s each. Although each workload's absolute throughput differs according to its access profile, all four exhibit a clear trend: throughput degrades monotonically as available FMem diminishes. In contrast to BE workloads, for which hot pages remain stable over long intervals, LC workloads undergo abrupt, real-time-driven shifts in data access. To preserve LC performance, it is essential to retain their datasets in FMem as much as possible. Current tiered-memory policies, however, rely solely on access-frequency heuristics; they uniformly classify LC data as "cold" and evict it to SMem.

To quantify LC workload performance degradation when colocated with a BE workload, we run Redis alongside the GAP Benchmark Suite's single-source shortest-path (SSSP) benchmark under MEMTIS, measuring Redis's 99th-percentile latency (P99) while

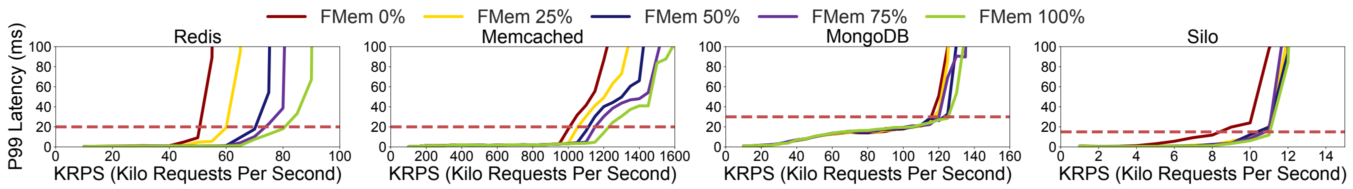


Figure 1: Tail latency of LC workloads as Kilo Requests Per Second (KRPS) increases. The red lines indicate the SLOs, determined by the knee point of each curve under 100% FMem utilization (FMem 100%).

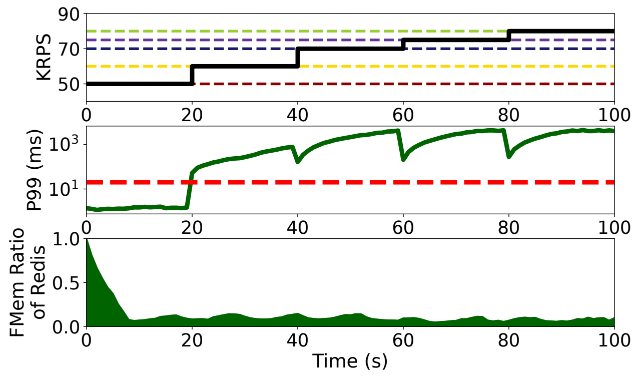


Figure 2: Performance of Redis co-located with SSSP under MEMTIS-managed tiered memory. The top plot shows the imposed load on Redis in KRPS, with colored lines indicating the maximum throughput at different FMem allocation levels (0%, 25%, 50%, 75%, and 100%). The middle plot presents the P99 latency in milliseconds, with a red line representing the SLO. The bottom plot illustrates the ratio of Redis data stored in FMem over time.

monitoring its FMem allocation. Redis initially occupies 100% of available FMem, then receives increasing load corresponding to the maximum throughputs observed at FMem allocations of 0%, 25%, 50%, 75%, and 100% (per Figure 1). Figure 2 presents the resulting load pattern, P99 latency, and FMem usage. At the start of each run, MEMTIS promptly fills FMem with the SSSP dataset, causing Redis's resident pages in FMem to drop below 10%. Once the load reaches the peak throughput corresponding to 25% FMem, Redis's P99 latency surges sharply–violating its SLO–even though it could sustain this load with 25% FMem allocation. Even as the load increases further, the proportion of Redis pages resident in FMem remains consistently low. Thus, although Redis runs in a tiered memory system with both FMem and SMem available, its performance degrades to match that of operating exclusively on SMem.

In summary, current tiered memory management policies prioritize pages by access frequency under the assumption of stable, memory-bound BE workloads. Such frequency-based schemes, however, disadvantage LC workloads: their irregular access patterns and requirement for sustained FMem residency preclude existing systems from meeting SLOs. Hence, an advanced tiered memory policy is urgently needed to preserve LC performance–particularly in consolidated servers hosting concurrent workloads.

3 Design

3.1 Overview

Driven by the need to support LC and BE workloads concurrently in multi-tenant environments, we introduce MTAT–an adaptive tiered memory management framework that guarantees LC SLOs while preserving overall system performance among co-located BE workloads. The core principle of MTAT is the dynamic partitioning and isolation of FMem, allocating a dedicated capacity to each workload to prevent interference and ensure predictable performance. Figure 3 presents an overview of the MTAT framework.

MTAT consists of two primary components: the Partition Policy Maker (PP-M), which determines per-workload FMem allocations, and the Partition Policy Enforcer (PP-E), which implements these allocations. PP-M adopts individualized strategies for LC and BE workloads. For LC workloads–where memory demands fluctuate over time–it leverages reinforcement learning to model dynamic access patterns and adjust FMem allocations in real time. Conversely, for BE workloads–characterized by sustained, repetitive memory usage–PP-M relies on offline profiling to estimate marginal performance gains and provision FMem accordingly. PP-E implements the PP-M's decisions via fine-grained FMem adjustments, prioritizing the LC partition plan, followed by progressive BE allocation reconfiguration. Between policy-update intervals, PP-E continuously promotes and demotes pages between FMem and SMem to maintain the designated "hot" working set.

MTAT determines and enforces its memory partitioning policy through the following steps. First, PP-M leverages RL inferences—derived from system memory conditions such as memory traffic and migration overhead—to predict the minimal FMem allocation required to meet the SLOs of LC workloads. It then reserves this FMem capacity for LC workloads and distributes any remaining FMem among BE workloads in a manner that ensures uniform performance improvement across all BE workloads. Once the partition sizes for all workloads are finalized, PP-M conveys the FMem partitioning policy to PP-E, which then reconfigures the FMem partitions accordingly. Specifically, PP-E identifies the workloads that require additional FMem and those that must release some portion of their existing FMem allocation. Based on this classification, PP-E carries out the memory tier exchange, where pages belonging to workloads that need reduced FMem are migrated to SMem, while pages from workloads demanding increased FMem

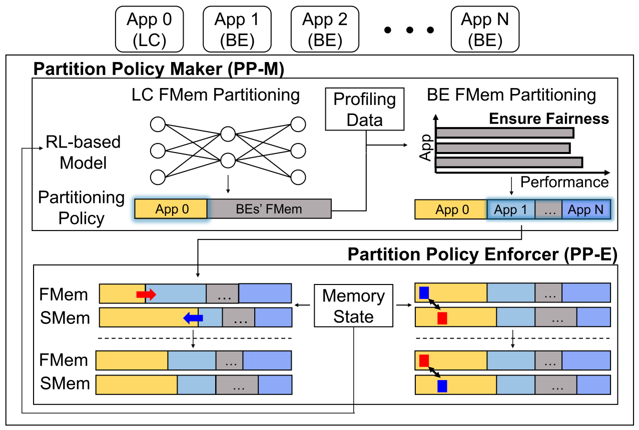


Figure 3: Overview of MTAT.

are simultaneously migrated from SMem to FMem. After the exchange, PP-E continuously monitors the memory access patterns of each workload until the next partitioning interval, promoting frequently accessed pages from SMem to FMem while ensuring that all migrations remain within the workload's allocated FMem and SMem to preserve isolation.

3.2 Partition Policy Maker

3.2.1 RL-based FMem Partitioning for LC Workload. PP-M employs an RL model that adaptively determines the amount of FMem to allocate in real-time to accommodate the SLO of the LC workload under unpredictable demand fluctuations. The LC partitioning policy in PP-M exhibits a clear causal relationship between FMem allocation and SLO compliance outcomes, and it follows an iterative decision-making structure that adapts to evolving loads over time. Consequently, this setup aligns with the canonical four-stage Markov Decision Process—state (MDP) observation, action selection, reward assignment, and policy update—rendering reinforcement learning a suitable choice. The formulation below details how the MDP is defined for LC Fmem partitioning in PP-M.

State. We select three primary metrics—FMem Usage Ratio, FMem Access Ratio, and Memory Access Count—to characterize the state of LC workload under FMem allocation, as they directly influence SLO violations.

- **FMem Usage Ratio.** Defined as the fraction of an LC workload's total memory usage residing in FMem, the FMem Usage Ratio indicates whether sufficient FMem is available to house frequently accessed data. This metric serves as a critical determinant of potential response time violations due to its strong correlation with SLO satisfaction.
- **FMem Access Ratio**. Measured as the percentage of total memory accesses directed to FMem, the FMem Access Ratio quantifies the efficiency of LC workload utilizing allocated FMem. Analyzed alongside the FMem Usage Ratio, it provides complementary insights into the adequacy of FMem allocation and its operational effectiveness.
- **Memory Access Count.** Representing the number of memory accesses performed by an LC workload per unit time, the Memory Access Count assesses workload intensity. This metric aids in detecting potential under- or over-provisioning of FMem resources in response to varying load conditions.

Action. In each partitioning policy decision interval, PP-M may increase or decrease the amount of FMem allocated to an LC workload. We formalize this decision as a single scalar action α representing the net change in FMem (e.g., measured in gigabytes). A positive α indicates an expansion of the FMem region, whereas a negative α signifies a reduction.

Any action must be completed before the next interval begins. Due to the bandwidth constraint of the tiered memory system, only a limited amount of data can be reconfigured during a single interval. Consequently, the magnitude of α is upper-bounded by $\frac{M}{2t}$, reflecting the bidirectional nature of memory reconfiguration with simultaneous promotion and demotion of data, yielding an action space

$$\alpha \in \left[-\frac{M}{2t}, \frac{M}{2t}\right]. \tag{1}$$

Here, M denotes the maximum data movement capacity (e.g., bytes per second) of the tiered memory subsystem, and t is the length of the policy interval. The constraint ensures that all changes to FMem allocation are completed within the available bandwidth during a single partition policy decision interval, guaranteeing a fully consistent allocation for subsequent decisions.

Reward. We define the reward r at each decision interval as follows:

$$r = \begin{cases} 1 - fmem\_ratio, & \text{if } p99 \le SLO, \\ -1, & \text{otherwise,} \end{cases} \tag{2}$$

where $fmem\_ratio$ represents the fraction of total memory usage that resides in FMem (i.e., FMem Usage Ratio), and $p99$ denotes the 99th-percentile response time. If $p99$ exceeds the specified SLO, the reward becomes -1. This binary penalty enforces strict compliance with latency requirements, ensuring that solutions resulting in SLO violations incur a sharply negative outcome. Conversely, when the SLO is satisfied, the reward is given by $1 - fmem\_ratio$. By subtracting the proportion of FMem usage from unity, the agent is encouraged to minimize FMem allocation while still meeting the latency constraint. As a result, the reward design concurrently targets two objectives: enforcing hard SLO adherence through a large penalty on violations and incentivizing FMem efficiency by reducing the reliance on FMem whenever the SLO remains satisfied.

RL model. The RL model for PP-M, based on the defined state, action, and reward representations, is implemented using the Soft Actor-Critic (SAC) algorithm [20]. SAC adopts an actor-critic framework, wherein the actor is the policy network responsible for selecting actions, and the critic consists of Q-networks that estimate the value of state-action pairs. Algorithm 1 outlines the pseudocode of this approach.

The model initializes the Q-networks (Q_1, Q_2), the policy network π, and the replay buffer $\mathcal{D}$, then retrieves the current state—comprising the FMem Usage Ratio, FMem Access Ratio, and Memory Access Count. Next, the policy outputs a continuous action α, indicating how much FMem to add or remove, which is clipped to the permissible range $\left[-\frac{M}{2t}, \frac{M}{2t}\right]$. The environment applies this

Algorithm 1: LC workload's FMem partition determination based on Soft Actor-Critic algorithm.

```
1:  Initialize:
      • $Q_1, Q_2$: twin Q-networks (Critic)
      • $\pi$: policy network (Actor)
      • $\mathcal{D}$: replay buffer to store transitions
      • $\mathcal{E}$: environment providing
        $s$ = (UsageRatio, AccessRatio, AccessCount)
2:  Define action range: $\alpha \in \left[-\frac{M}{2t}, \frac{M}{2t}\right]$
3:  while not done do
4:      $\alpha \leftarrow \pi(s)$
5:      $\alpha_{\text{clip}} \leftarrow \text{clip}(\alpha, \alpha_{\min}, \alpha_{\max})$
6:      $(s_{\text{next}}, p99, \text{done}) \leftarrow \mathcal{E}.\text{step}(\alpha_{\text{clip}})$
7:      if $p_{99} \leq SLO$ then
8:          $r \leftarrow 1 - s.\text{UsageRatio}$
9:      else
10:         $r \leftarrow -1$
11:     end if
12:     $\mathcal{D}.\text{store}(s, \alpha_{\text{clip}}, r, s_{\text{next}}, \text{done})$
13:     $s \leftarrow s_{\text{next}}$
14:     if it is time to update then
15:         Sample a mini-batch $(s, \alpha, r, s')$ from $\mathcal{D}$
16:         Optimize $Q_1$ and $Q_2$ by minimizing error between their
            predictions and the target
17:         Update $\pi$ to maximize the critic-estimated returns
18:     end if
19: end while
```

action, checks whether the 99th-percentile latency remains below the specified SLO, and returns a reward of $1 - fmem\_ratio$ if compliant or -1 otherwise. Upon accumulating a sufficient number of action-reward pairs, empirically determined as 50 iterations, the PP-M updates Q_1, Q_2 and π using a mini-batch sampled from $\mathcal{D}$. Each transition (s, α, r, s') is placed into $\mathcal{D}$, and the Q-networks are trained to minimize the mean-squared error relative to a soft Bellman target. The policy network π is adjusted to maximize the Q-value estimates, thereby balancing efficient FMem usage with consistent SLO satisfaction.

3.2.2 *FMem Partitioning for BE workloads.*

PP-M allocates the remaining FMem capacity among the BE workloads, excluding the portion reserved for the LC workload. In order to evenly distribute any performance losses resulting from the LC workload's dynamic FMem occupancy, PP-E employs a fairness-based approach when allocating FMem to the BE workloads.

Fairness-driven BE partitioning. To ensure fairness in performance across BE workloads, PP-M seeks to minimize performance imbalance among individual workloads. In particular, it aims to elevate the performance of the "worst-off" workload as close as possible to that of the "best-off" workload, thereby equalizing the performance degradation caused by using SMem instead of FMem. Formally, let $Perf^{i}_{full}$ denote the per-second throughput of workload i when it has exclusive access to 100% of the FMem. Under a specific memory allocation, let $Perf^{i}_{alloc}$ be the corresponding per-second throughput of workload i. The performance degradation

Algorithm 2: FMem partitioning for BE workloads via Simulated Annealing-based search.

```
1:  Initialize:
      • $M_{\text{total}}$: total FMem capacity
      • $M_{\text{LC}}$: reserved memory for LC workload
      • $T_0$: initial temperature
      • $\gamma$: temperature decay factor $(0 < \gamma < 1)$
      • $P(\mathbf{M})$: performance degradation metric function
      • $\mathbf{M} \leftarrow [M_1, \ldots, M_n]$, where
            $M_i \leftarrow \frac{M_{\text{total}} - M_{\text{LC}}}{n}, \quad \forall i \in \{1, \ldots, n\}.$
      • $P^* \leftarrow P(\mathbf{M})$, $\mathbf{M}^* \leftarrow \mathbf{M}$
      • $T \leftarrow T_0$, $iter \leftarrow 0$
2:  while $iter < iter_{\max}$ and $T >$ threshold do
3:      Randomly select distinct $i, j \in \{1, \ldots, n\}$
4:      $\Delta m \leftarrow \{+1, -1\}$ (in GB) at random
5:      $\mathbf{M}' \leftarrow \mathbf{M}$ with $M'_i \leftarrow M_i + \Delta m$, $M'_j \leftarrow M_j - \Delta m$
6:      $\Delta P \leftarrow P(\mathbf{M}') - P(\mathbf{M})$
7:      if $\Delta P > 0$ or rand$(0,1) < \exp\left(\frac{\Delta P}{T}\right)$ then
8:          $\mathbf{M} \leftarrow \mathbf{M}'$
9:      end if
10:     $T \leftarrow \gamma \cdot T$
11:     $iter \leftarrow iter + 1$
12:     if $P(\mathbf{M}) > P(\mathbf{M}^*)$ then
13:         $\mathbf{M}^* \leftarrow \mathbf{M}$
14:     end if
15: end while
16: Return $\mathbf{M}^*$ and $P(\mathbf{M}^*)$
```

NP for workload i is then defined as:

$$NP_{[i]} = \frac{Perf^{i}_{alloc}}{Perf^{i}_{full}} \tag{3}$$

To achieve the highest level of fairness, PP-M must find a memory allocation such that, for any two workloads i and j, the ratio $\frac{NP_{[i]}}{NP_{[j]}}$ is as close to 1 as possible. In practice, PP-M maximizes the minimum ratio $NP_{[i]}$ across workload i.

Simulated Annealing-Based Search. PP-M employs a Simulated Annealing (SA) approach [15] (Algorithm 2) to determine an FMem allocation that optimizes fairness among BE workloads. Initially, PP-M reserves the necessary capacity for the LC workload and then distributes the remaining FMem evenly across the BE workloads. Let $M = [M_1, \cdots, M_n]$ denote this initial allocation vector, and $P(M)$ represent the corresponding performance degradation metric. During each iteration, PP-M randomly selects two workloads i and j and applies a memory shift of Δm, chosen randomly from $\{-1, +1\}$ GB. The allocation vector is updated accordingly: $M_i \leftarrow M_i + \Delta m$ and $M_j \leftarrow M_j - \Delta m$, producing a new allocation M'. PP-M evaluates this new allocation by computing $\Delta P = P(M') - P(M)$. If $\Delta P > 0$, the new allocation is automatically accepted; otherwise, M' is accepted with probability $exp^{\frac{\Delta P}{T}}$ to prevent premature convergence to a local optimum, where T is the current temperature. With each iteration, T is reduced by a factor γ, such that $T \leftarrow \gamma T$. The search

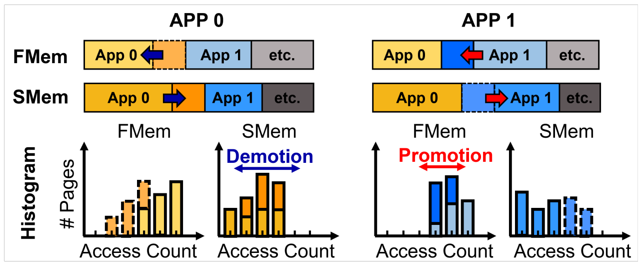
Algorithm 3: Detailed partition adjustment procedure by PP-E.

```
1:  **Initialize:**
      • $M_{\mathrm{LC}}^{(t)}, M_{\mathrm{LC}}^{(t+1)}$: LC memory states at $t$ and $t+1$.
      • $M_i^{(t)}, M_i^{(t+1)}$: BE memory states for job $i$ at $t$ and $t+1$.
      • $p_{\max}$: Maximum pages per time slice.
      • $\Delta_{\mathrm{LC}} \leftarrow M_{\mathrm{LC}}^{(t+1)} - M_{\mathrm{LC}}^{(t)}, \quad \Delta_i \leftarrow M_i^{(t+1)} - M_i^{(t)}$
      • $\mathrm{PromoteSet} \leftarrow \{i \mid \Delta_i > 0\}, \mathrm{DemoteSet} \leftarrow \{i \mid \Delta_i < 0\}$
      • $P_{\mathrm{promote}} \leftarrow \sum_{i \in \mathrm{PromoteSet}} \Delta_i + \max(0, \Delta_{\mathrm{LC}})$
      • $P_{\mathrm{demote}} \leftarrow \sum_{i \in \mathrm{DemoteSet}} |\Delta_i| + \max(0, -\Delta_{\mathrm{LC}})$
      • $\mathrm{remainingPages} \leftarrow \max(P_{\mathrm{promote}}, P_{\mathrm{demote}})$
2:  **while** $\mathrm{remainingPages} > 0$ **do**
3:      $p \leftarrow \min(p_{\max}, \mathrm{remainingPages})$
4:      Allocate LC movement:
          $m_{\mathrm{LC}} \leftarrow \begin{cases} \min(\Delta_{\mathrm{LC}}, p), & \Delta_{\mathrm{LC}} > 0, \\ -\min(|\Delta_{\mathrm{LC}}|, p), & \Delta_{\mathrm{LC}} < 0, \\ 0, & \Delta_{\mathrm{LC}} = 0. \end{cases}$
5:      Determine BE memory movement $m_i^*$:
6:      **if** $m_{\mathrm{LC}} > 0$ **then**
7:          Distribute $m_{\mathrm{LC}}$ pages proportionally to $|\Delta_i|$ across DemoteSet.
8:      **else if** $m_{\mathrm{LC}} < 0$ **then**
9:          Distribute $|m_{\mathrm{LC}}|$ pages proportionally to $\Delta_i$ across PromoteSet.
10:     **else**
11:         Distribute $p$ pages each to PromoteSet and DemoteSet proportionally to their respective demands.
12:     **end if**
13:     $\mathrm{remainingPages} \leftarrow \mathrm{remainingPages} - p$
14:     Update $m_{\mathrm{LC}}$ and $m_i^*$ accordingly.
15: **end while**
16: **Return** Page movement schedule $\{m_{\mathrm{LC}}, m_i^*\}$.
```

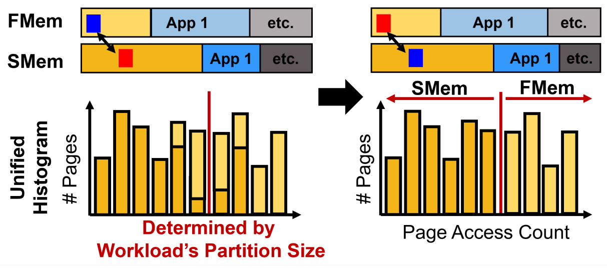
terminates when the maximum number of iterations is reached or when T falls below a predefined threshold (determined experimentally to maintain the periodic cycle). Lastly, upon completing the search for the optimal allocation M' and its corresponding performance value $P(M')$, PP-M establishes the partitioning policy for the subsequent cycle based on the finalized allocation M' and the predefined memory M_{LC} designated for the LC workload.

3.3 Partitioning Policy Enforcer

3.3.1 LC-First Adjustment for Partitioning Policy. PP-E implements the partitioning policy determined by PP-M by directly reconfiguring FMem allocations. Since FMem demand remains saturated at full utilization across all workloads, any partition modification entails migrating data between FMem and SMem, making overall performance highly sensitive to the sequence of migrations. To mitigate this sensitivity, PP-E subdivides the update interval specified by PP-M into finer time slices, enabling granular execution of partition changes that preserve priority for LC workloads while concurrently optimizing BE workload performance.



(a) Partitioning policy adjustment



(b) Refining FMem contents

Figure 4: PP-E partitioning policy adjustment and FMem refinement based on page access histograms.

Algorithm 3 provides a detailed procedure for adjusting memory partitions. First, PP-E computes the discrepancies between the current and desired memory allocations for both the LC workload (Δ_{LC}) and each BE workload (Δ_i). Based on these discrepancies, each workload is classified into either the set requiring additional memory as PromoteSet or the set capable of releasing some of its allocated memory as DemoteSet. Subsequently, PP-E determines the total number of pages to be migrated, denoted *remainingPages*. This total is then divided into multiple time slices, each constrained by the maximum number of pages that can be relocated in a single slice, p_{max}. For each time slice, PP-E calculates the number of pages to be moved, p, identifies which workloads require memory movement, and specifies how this movement is carried out. When the LC workload requires a memory adjustment (*i.e.*, $m_{LC} \neq 0$)—whether through promotion or demotion—its page migrations take precedence. Specifically, during each time slice, the memory promotion or demotion demands induced by the LC workload are proportionally allocated among the BE workloads in PromoteSet or DemoteSet based on each BE workload's respective promotion or demotion demands, ensuring that the overhead of memory migration is fairly distributed among the BE workloads. If no changes are required for the LC workload, a similar procedure is applied solely among the BE workloads, with page exchanges between PromoteSet and DemoteSet proportionally allocated based on the demands of the workloads within each set.

3.3.2 Hotness-Aware Page Placement. To maintain FMem in a high-activity ("hot") state during partitioning adjustments, PP-E continuously collects per-page access frequencies through hardware counter sampling and leverages these metrics to guide page promotions and demotions. Specifically, PP-E tracks access frequencies separately for pages residing in FMem and SMem and then performs page migration by employing a histogram-based scheme–commonly used in state-of-the-art tiered memory systems [23, 39]–where histogram bins increase exponentially, doubling at each step (from 2^0 to 2^n), thereby categorizing pages according to their observed access frequencies. Figure 4 illustrates the overall depiction by which PP-E maintains per-workload access frequency histograms and leverages them to identify promotion and demotion pages. During the partitioning policy adjustments, as depicted in Figure 4a, when a workload requires additional FMem capacity, PP-E promotes pages from SMem to FMem by selecting those in the highest frequency bin. Conversely, when FMem requires eviction, pages are demoted from FMem to SMem following the lowest-frequency bin. Because the set of frequently accessed pages can shift over time, PP-E periodically 'ages' the recorded access frequencies by half at each partitioning-policy update interval determined by PP-M.

Once the memory adjustments prompted by partitioning are complete, as shown in Figure 4b, PP-E continues to refine FMem contents based on these access frequencies until the following partitioning policy is determined. During this process, PP-E represents all pages in a unified histogram based on their access frequencies and allocates them to FMem up to the workload's partition size, leaving the remaining pages in SMem. Crucially, each workload's allocated FMem partition size is preserved, regardless of disparities in access frequencies across workloads. Page replacement is performed strictly within the confines of each workload's assigned FMem partition, thus ensuring the isolation required for PP-E to fulfill the performance objectives set by PP-M.

4 Implementation

We implement a prototype of the MTAT framework on a Linux 6.1.53 kernel, which comprises two primary components–a user-space daemon for PP-M and a kernel-space daemon for PP-E–that communicate through the cgroup interface [2] using file I/O operations. PP-E periodically aggregates per-workload memory statistics and publishes them via its cgroup interface; PP-M reads these measurements, updates the partitioning policy, and writes the revised policy back for PP-E to enforce. The tunable parameters of PP-E and PP-M (e.g., the number of data points PP-E collects per partition-update window and the action-reward interval in PP-M) regulate the reinforcement-learning adaptation rate, with faster adaptation incurring higher CPU overhead. In our prototype, we confine RL execution to BE cores, thereby ensuring that LC workloads see no impact on latency and throughput. To empirically balance adaptation speed and CPU overhead, we set the parameters so that partition updates occur once per minute in our environment.

PP-M is developed in Python [8], incorporating PyTorch for its RL modeling. Whenever 50 new data points (comprising state–action-reward tuples) are collected, PP-M performs an incremental training step to update its RL model. PP-M employs offline profiling data for BE workload partitioning, which measured their throughput under varying FMem allocations, ranging from 0 GB to higher capacities in 1 GB increments.

Table 1: LC benchmarks characteristics.

Benchmark	RSS (GB)	SLO (ms)	Max Load (KRPS)
Redis	33.6	20	80
Memcached	31.4	20	1220
MongoDB	33.2	30	125
Silo	30.4	15	11

Table 2: BE benchmarks characteristics.

Benchmark	RSS (GB)	Description
SSSP	35.5	Finds the shortest paths from a single source node.
BFS	35.2	Explores all nodes at the current depth level.
PR	36.0	Assigns importance scores to nodes in a directed graph.
XSBench	31.7	Simulates the computational workload of Monte Carlo neutron transport calculations.

PP-E leverages Intel's Process Event-Based Sampling (PEBS) to capture detailed page-level access patterns at a 1 ms action-reward interval. Specifically, it samples `MEM_LOAD_L3_MISS_RETIRED.LOCAL_DRAM` and `MEM_LOAD_L3_MISS_RETIRED.REMOTE_DRAM` events to classify Read operations as FMem or SMem, respectively, and `MEM_INST_RETIRED.ALL_STORES` event to record Write operations, partitioning them by address ranges into FMem or SMem categories. PP-E accumulates these read-and-write counts in page-level histograms, following a method similar to MEMTIS [23] and FlexMem [39]: PP-E uses each page's page frame number in its Page Table Entry (PTE) to store access counts, then aggregates them into FMem, SMem, and unified histograms. Each histogram bin is linked to a list of physical pages that fall within that bin's access count range, making it straightforward to identify specific pages and correlate them with their memory locations.

5 Evaluation

System Configuration. We set up an experimental environment for the client-server using two physical machines. The client machine, which generates load, is equipped with an AMD Ryzen Threadripper 3970X CPU operating at 3.70 GHz, featuring 32 cores and 32 GB of DRAM. The server machine, which receives the load, comprises two sockets of Intel Xeon Gold 6342 CPUs, each operating at 2.80 GHz and containing 24 cores per socket. The server's memory is distributed across two NUMA nodes: node 0 with 32 GB of DRAM and node 1 with 256 GB of DRAM. To prevent the network bandwidth from becoming a performance bottleneck, the client and server machines are interconnected via a 200 Gbps link, and each is equipped with a Mellanox ConnectX-6 network interface card. Following prior work [23, 37], we emulate CXL memory by utilizing NUMA-based remote memory. Specifically, only the CPU connected to Node 0 is used for computation, while the DRAM attached to Node 1 is designated solely as remote memory. Measurements using the Intel Memory Latency Checker show that the memory access latencies for Node 0 and Node 1 are approximately

73 ns and 202 ns, respectively—consistent with typical access times for local DRAM and CXL memory [25].

Benchmarks. We employ four representative LC workloads: Redis [9] and Memcached [5] (in-memory key-value stores), MongoDB [6] (a NoSQL database), and Silo [36] (an in-memory transactional database). For client programs, Redis and MongoDB uses YCSB [11] workload C (100% reads), Memcached uses Mutilate [7], and Silo uses TailBench [22]. Redis runs in a single-threaded configuration on a dataset comprising 13.5 million records (each 1 KB, with 100 B per field and 10 fields). MongoDB runs with eight threads on a dataset of 23.3 million records (also 1 KB each, with 100 B per field and 10 fields). Memcached is configured to use eight threads with 7.1 million items (each item having a 100 B key and a 4 KB value). Silo runs in single-threaded mode, using the TPC-C dataset scaled to 320 warehouses. The SLO for each LC workload is determined by the maximum KRPS at which the workload can reliably handle the load without an exponential increase in latency, along with the corresponding P99 latency. As for BE workloads, we select a set of applications commonly employed in tiered-memory evaluations: the Single-Source Shortest Paths (SSSP), Breadth-First Search (BFS), and PageRank (PR) benchmarks from the GAPBS [4], along with the HPC workload XSBench [10]. Table 1 and 2 provide a detailed benchmark description for LC and BE workloads, respectively.

Comparisons. We evaluate our proposed MTAT against two state-of-the-art tiered-memory page management policies, MEMTIS and TPP, as well as two static approaches, FMEM_ALL and SMEM_ALL.

- **MEMTIS** uses a page access histogram to identify frequently accessed 'hot' pages and migrates them to FMem.
- **TPP** employs an active/inactive list, similar to traditional swapping systems, leveraging page faults to decide which pages should reside in FMem or SMem.
- **FMEM_ALL** configures the LC workload to exclusively occupy and utilize FMem.
- **SMEM_ALL** forces the LC workload to use only SMem.

To comprehend how FMem allocation policies for LC and BE workloads affect MTAT 's performance, we separately evaluate two variants of MTAT:

- **MTAT (LC Only)** exclusively allocates FMem only to the LC workload, with the remaining memory shared among BE workloads through competitive allocation.
- **MTAT (Full)** determines/allocates an appropriate amount of FMem for all workloads.

Experimental Methodology. We allocate a total of 24 cores from the server machine to specific workloads by pinning them to designated CPU cores. Eight cores are dedicated to the LC workload, while the remaining 16 cores are evenly distributed among the four BE workloads, assigning four cores to each BE application. All page sampling and migration activities are executed on cores designated for BE workloads to minimize interference with the LC workload's performance. Further, to mitigate potential cache interference, we employ Cache Allocation Technology (CAT) [27] to partition the LLC ways between LC and BE workloads. Of the 12 available LLC ways, four are allocated to the LC workload, while each BE workload is assigned two ways.

5.1 Overall Performance

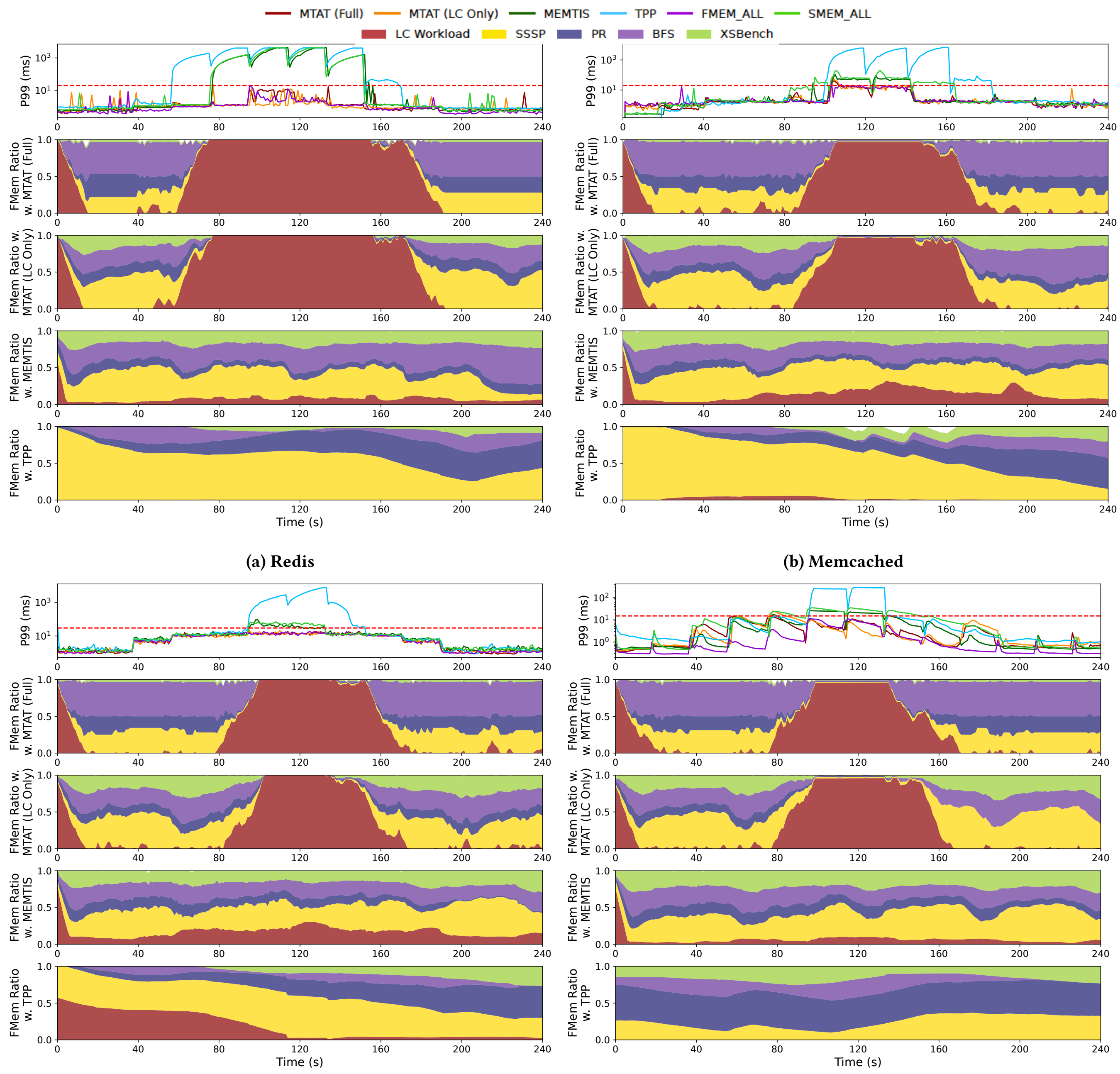
To assess how effectively MTAT manages FMem in a multi-tenant environment where LC and BE workloads share the same server, we measure both the time-varying FMem allocation per workload and the corresponding performance under dynamic load conditions. In each experiment, we co-locate each LC workload with four different BE workloads. The load applied to the LC workload follows the pattern shown in Figure 7, rising until it reaches the maximum capacity that FMEM_ALL can handle and then decreasing thereafter.

Figure 5 depicts the FMem allocation over time and the 99th-percentile (P99) latency of the LC workload, with the red line indicating the SLO. We define this SLO at the point where the LC workload's latency diverges. Although TPP and MEMTIS are capable of allocating FMem to the LC workload, both still violate the SLO for all LC workload types, similar to SMEM_ALL. Moreover, TPP experiences even more severe latency degradation than SMEM_ALL. This violation arises because LC workloads require their data to be present in FMem at the time of request, whereas frequency-based page management (used by TPP and MEMTIS) only promotes data to FMem after each request completes, offering no timely benefit once LC data has been displaced to SMEM by BE workloads. In contrast, MTAT (LC Only) and MTAT (Full) both successfully satisfy the SLO by adaptively allocating FMem in response to growing LC workload demand. During low-load periods (before 60 seconds and after 180 seconds), only a small portion of FMem is allocated to the LC workload, allowing the BE workloads to utilize FMem similarly to SMEM_ALL. During the high-load interval (100–140 seconds), the LC workload occupies nearly the entire FMem capacity, akin to FMEM_ALL, thereby handling the surge effectively.

Moving on to the BE workloads, Figure 6 presents their performance with respect to fairness and throughput. We utilize the fairness metric as the smallest performance degradation ratio (NP) derived from Equation 3 and the throughput metric as the sum of the per-second throughput across all BE workloads. As illustrated in the FMem allocation graph for MTAT (Full) in Figure 5, MTAT (Full) substantially improves fairness by distributing FMem among the BE workloads according to performance degradation ratios, thereby ensuring equitable resource sharing while still prioritizing the LC workload. In particular, MTAT (Full) improves fairness nearly threefold compared to TPP and 1.4× relative to MEMTIS. Although prioritizing FMem for LC workloads results in a throughput reduction of approximately 18% compared to MEMTIS, this decrease is considered an acceptable trade-off given the substantial improvement in fairness. Moreover, MTAT (LC Only)—which adopts a frequency-based policy for BE workloads while retaining LC priority—reduces this gap to approximately 7%.

5.2 Throughput of LC Workloads

To quantitatively evaluate how effectively MTAT guarantees the performance of LC workloads when co-executed with BE workloads, we compare MTAT and comparisons against FMEM_ALL by measuring the maximum load that can be sustained without violating the LC workload's SLO. The experimental setup follows the same methodology as the overall performance evaluation: each LC workload is co-located with four BE workloads, and the SLO violation is checked to see whether it is breached during periods of



(a) Redis

(b) Memcached

(c) MongoDB

(d) Silo

Figure 5: Performance comparison of MTAT and baseline approaches under dynamic load. This figure presents the impact of dynamic load on various LC workloads—(a) Redis, (b) Memcached, (c) MongoDB, and (d) Silo—when co-located with four BE workloads. The top plot in each subfigure shows the P99 latency over time, with red lines indicating the SLO. The subsequent plots illustrate the FMem ratio under MTAT (Full), MTAT (LC Only), MEMTIS and TPP.

high load. Figure 8 presents the results. On a geometric mean basis, TPP delivers roughly 70% of FMEM_ALL's throughput that lags behind SMEM_ALL, which relies solely on SMem for the LC workload; similarly, MEMTIS reaches only about 85% of the throughput. In contrast, MTAT achieves an overwhelming throughput by dynamically tracking the load and allocating a dedicated portion of FMem that the LC workload can fully utilize, incurring less than a 1% loss relative to FMEM_ALL, regardless of LC workload types.

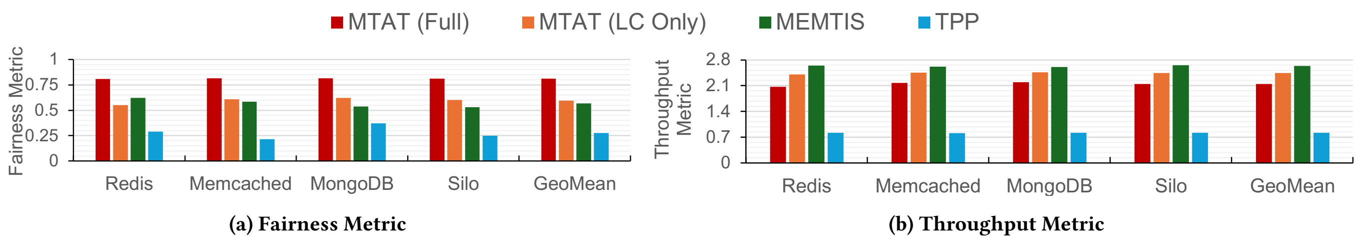


Figure 6: (a) Fairness and (b) throughput metric comparison of BE workloads under MTAT (Full), MTAT (LC Only), MEMTIS and TPP. Each metric is computed based on the performance of four BE workloads, which are co-located with one of the four LC workloads.

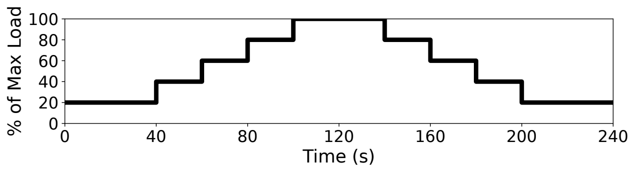


Figure 7: Dynamic load pattern applied to the LC workload. The load starts at 20% of Max Load, increases to 100% in increments of 20% every 20 seconds, and then decreases back to 20% following the same pattern.

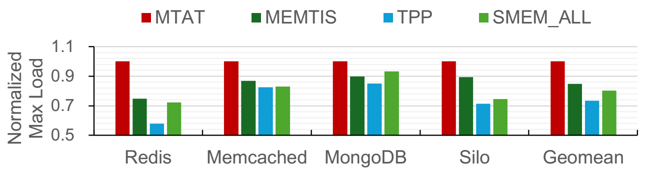


Figure 8: Maximum load of LC workloads under MTAT, MEMTIS, TPP and SMEM_ALL, normalized to FMEM_ALL.

5.3 Performance of BE Workloads

To quantitatively evaluate the distribution of remaining FMem among BE workloads after reserving FMem for the LC workload, we compare the allocation ratios for each workload and the resulting fairness and throughput with SLO violation rates while progressively increasing the load intensity of LC to 20%, 50%, and 80% of its maximum throughput. In these experiments, Redis serves as the LC workload under uniform load, while four BE workloads are executed concurrently. Figure 9a and 9b respectively present the fairness and throughput across different load levels, and Table 4 reports the corresponding rates of SLO violations. Despite allocating all available FMem to BE workloads, TPP exhibits the lowest fairness and throughput along with the highest SLO violation rate due to continuous page-fault-induced migration and severe FMem contention. MEMTIS, on the other hand, achieves the highest throughput regardless of the load level since it does not manage FMem for Redis. Therefore, as the load increases, the lack of management results in a significant increase in SLO violations; at 80% of the maximum load, 99% of requests miss their SLO target. Conversely, MTAT (Full) consistently delivers the highest fairness at all load levels without any SLO violations. Under heavy load conditions, MTAT dynamically reallocates FMem to maintain the SLO of Redis, while under minimal load conditions, it allocates only the necessary FMem to satisfy Redis's SLO and aggressively assigns the remaining memory to BE workloads. By flexibly adapting to the demands of both LC and BE workloads, MTAT significantly improves overall system performance.

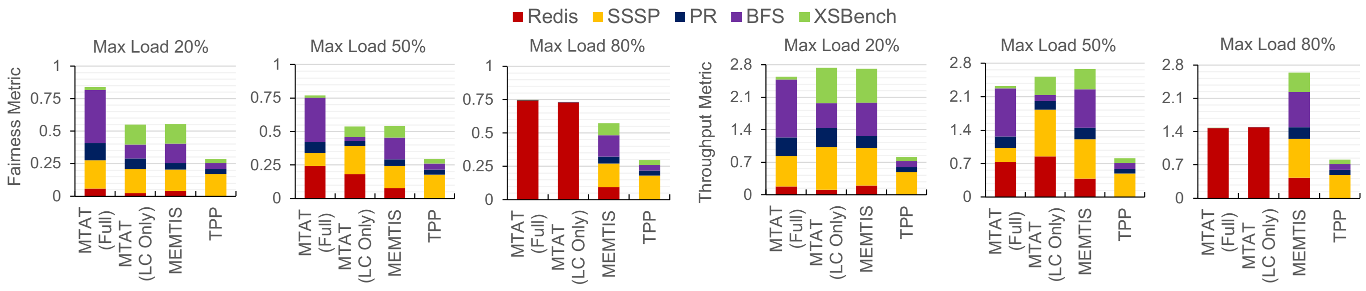
5.4 Effectiveness under Varying Settings

To assess whether MTAT performs reliably across diverse settings, we vary the ratio of LC to BE cores and the number of BE workloads and, for each setting, measure the maximum throughput achievable without SLO violations and evaluate BE fairness and throughput at loads of 20%, 50%, and 80% of that setting's maximum. We report results for both MTAT (Full) and MTAT (LC Only); the LC workload is Memcached, and the BE workload set comprises SSSP and PR for the setting with two BE workloads and SSSP, BFS, PR, and XSBench for the setting with four BE workloads. We normalize maximum throughput to FMEM_ALL, in which LC exclusively uses FMem, and normalize BE fairness and throughput to MEMTIS. Table 3 summarizes these results.

Across all settings, MTAT satisfies the LC performance guarantee. Regardless of the core partitioning or the workload mix, it attains maximum throughput between 98% and 99% of FMEM_ALL. Even as LC performance is preserved, at low loads of 20% and 50%, MTAT sustains BE throughput approximately 80% to 90% of MEMTIS. Moreover, for MTAT (LC Only), eliminating unnecessary LC promotions to FMem can yield BE throughput that exceeds MEMTIS. In the same regimes, MTAT (Full) improves BE fairness by at least 10% over MEMTIS, and MTAT (LC Only) exhibits BE fairness comparable to MEMTIS. At 80% load, since both MTAT variants reserve most FMem for LC, BE throughput falls below 75% (about 53% in the most adverse setting); however, the overhead from this reduction is equitably apportioned across BE applications, yielding up to a 77% increase in fairness. In sum, MTAT consistently meets its design objective of preserving LC performance and operates effectively across a wide range of settings.

5.5 MTAT overhead

We evaluate the overhead incurred in maintaining the MTAT framework by categorizing it into PP-M and PP-E. The measurements are conducted as part of the overall performance evaluation (§5.1) using Redis. The primary overhead for PP-M arises from CPU usage during the RL inference process to determine the partitioning policy and the PEPS sampling procedure. Our measurements show



(a) Fairness Metric (b) Throughput Metric

Figure 9: (a) Fairness and (b) throughput metrics of four BE workloads co-located with Redis under MTAT (Full), MTAT (LC Only), MEMTIS, TPP. Results are shown for three different load levels (Max Load 20%, 50%, and 80%). The colors within each bar represent the FMem distribution across all co-located workloads.

Table 3: Summary of experimental results for MTAT (Full) and MTAT (LC Only) across varying settings (x, y, z). Each triple specifies x cores allocated to the LC workload and y total cores shared among z BE workloads. The LC max load is normalized to FMEM_ALL, whereas BE fairness and BE throughput are normalized to MEMTIS.

Setting	Config.	LC max load	Max Load 20%		Max Load 50%		Max Load 80%	
			BE fairness	BE throughput	BE fairness	BE throughput	BE fairness	BE throughput
(4, 20, 2)	MTAT (Full)	0.98	1.10	0.91	1.17	0.88	1.30	0.65
	MTAT (LC Only)	0.99	1.06	1.00	1.06	0.98	1.26	0.66
(4, 20, 4)	MTAT (Full)	0.99	1.16	0.93	1.19	0.87	1.19	0.57
	MTAT (LC Only)	0.99	1.05	0.98	1.00	1.00	1.18	0.58
(10, 14, 2)	MTAT (Full)	0.99	1.32	0.88	1.30	0.88	1.25	0.72
	MTAT (LC Only)	0.99	1.02	0.99	1.01	0.99	1.22	0.73
(10, 14, 4)	MTAT (Full)	0.99	1.18	0.94	1.33	0.84	1.52	0.54
	MTAT (LC Only)	0.99	1.00	1.00	0.99	1.01	1.51	0.56
(16, 8, 2)	MTAT (Full)	0.99	1.33	0.87	1.37	0.86	1.31	0.71
	MTAT (LC Only)	0.99	1.04	0.99	1.02	0.99	1.31	0.73
(16, 8, 4)	MTAT (Full)	0.98	1.23	0.92	1.44	0.83	1.76	0.52
	MTAT (LC Only)	0.98	1.00	1.01	0.97	1.02	1.77	0.51

Table 4: SLO violation rates of MTAT (Full), MTAT (LC Only), MEMTIS, and TPP at varying load levels.

Config.	SLO violation rate (%)		
	Max Load 20%	Max Load 50%	Max Load 80%
MTAT (Full)	0	0	0
MTAT (LC Only)	0	0	0
MEMTIS	0	11.6	99
TPP	0	30.7	100

that the combined CPU overhead remains below 7% of a single core, which is actually negligible. The main overhead for PP-E derives from memory bandwidth consumption during partitioning replacement. On average, this process consumes about 4 GB/s of memory bandwidth. Although our experimental server is equipped with a single-channel DDR4-3200 memory module—offering a maximum bandwidth of 25.6 GB/s—server-grade machines typically feature 6 to 8 memory channels, achieving an aggregate bandwidth of approximately 200 GB/s. Thus, the 4 GB/s consumption imposes only a minimal impact on overall system performance.

6 Related Work

Page management in tiered memory system. Several studies have attempted to improve the performance of tiered memory systems [19, 23, 25, 26, 29, 33, 34, 39, 40]. Telescope [26] extends the region-based sampling technique, DAMON (Data Access Monitor) [29], to utilize the access bits of multiple levels of page table entries, thereby effectively capturing page hotness in terabyte-scale memory environments. NOMAD [38] reduces page migration overhead by placing data non-exclusively across memory tiers and providing transactional page migration. FlexMem [39] combines NUMA hint faults and PEBS to enable more accurate identification of hot pages. TMTS [19] focuses on environments that employ Optane DIMMs as SMem, demoting only data that has not been accessed for several minutes to SMem. vTMM [34] defines a 'hot set size' for each VM as the number of pages exceeding a base access threshold and allocates FMem to each VM in proportion to its hot set size. ArtMem [42] treats the DRAM hit ratio as the system state and employs reinforcement learning to jointly tune how many pages to migrate and which pages qualify as hot. PET [18] performs proactive demotion and selective promotion at allocation-unit granularity rather than page granularity. These studies share a common principle of retaining highly accessed pages in FMem and propose strategies for efficient FMem utilization. However, MTAT demonstrates that the access-frequency-based approach fails to handle LC workloads exhibiting irregular memory access patterns and addresses the issue by allocating FMem based on each workload's intrinsic characteristics.

Resource management in multi-tenant environments with co-located LC/BE workloads. Extensive research has explored strategies for allocating resources to ensure that LC workloads meet their SLOs while optimizing system performance in multi-tenant environments [16, 17, 24, 28, 30]. Heracles [24] and PARTIES [17] employ heuristic-based partitioning of shared resources (e.g., CPU cores, LLC, DRAM bandwidth, and network bandwidth) by monitoring the tail latency of LC workloads, thereby maintaining LC SLOs and improving overall system throughput. Similarly, Twig [28] utilizes a multi-agent Branching Dueling Q-network (BDQ) reinforcement learning model to partition shared resources between co-located LC and BE workloads, while CLITE [30] leverages Bayesian optimization for resource partitioning, and OLPart [16] adopts a contextual multi-armed bandit (CMAB) approach for online resource allocation. However, these approaches do not account for tiered memory systems and, therefore, do not consider FMem as a contested resource. In contrast, MTAT demonstrates that system performance can degrade due to contention over FMem and mitigates it by determining the appropriate FMem allocation per workload, subsequently isolating FMem accordingly.

7 Discussion

LC/BE co-location work with MTAT. Prior work on LC and BE workload management has focused on dynamically adjusting latency-sensitive resources–specifically CPU core, DRAM capacity, and last-level cache–to accommodate fluctuations in LC demand. These schemes operate orthogonally to MTAT's FMem/SMem ratio adjustments, and because MTAT's RL model rewards solely on SLO compliance, the resource fluctuations they introduce are inherently absorbed into the learning process, thereby enabling complementary operation. However, integrating them in this manner would necessitate a thorough analysis and empirical validation of how the resources monitored by each scheme and their partitioning between LC and BE workloads influence MTAT 's ultimate performance contribution. This undertaking lies beyond the scope of this paper, where the primary focus is to establish the necessity of differentiated FMem usage for LC and BE workloads. We therefore defer the extension of existing LC/BE allocation techniques to a tiered-memory context and their integration with MTAT to future work.

Additional bandwidth-aware memory management policy with MTAT. Bandwidth-aware memory-management policies extend traditional frequency-based page-hotness classification and replacement schemes by dynamically adjusting FMem/SMem priorities and hot-page placement according to real-time bandwidth utilization–for instance, inverting access priority when FMem bandwidth becomes saturated. MTAT also operates orthogonally to such bandwidth-aware extensions, allowing its existing partitioning logic to function in parallel without interference. Since that mechanism fundamentally governs the intra-workload distribution of hot pages, it can be incorporated directly into MTAT's PP-E FMem page refinement. Moreover, MTAT's strategy of allocating FMem first to the LC workload ensures that any increase in LC demand automatically enlarges its FMem partition and proportionally reduces BE occupancy, thereby preventing LC performance degradation due to bandwidth saturation. As a result, the dynamic accommodation of LC workload leaves insufficient residual FMem for BE tenants to saturate the bandwidth, eliminating the need for an extended evaluation of this policy extension in this paper.

8 Conclusion

We propose MTAT, an adaptive tiered memory system that leverages per-workload FMem isolation to enable the consolidation of LC and BE workloads on multi-tenant servers–while strictly preserving the LC workload's SLO and ensuring fair performance for BE workloads. MTAT dynamically tracks the LC workload memory access patterns, reserves FMem determined through RL inference accordingly, and distributes the remainder among BE workloads based on their FMem-to-SMem ratio in relation to throughput. Compared to state-of-the-art tiered memory page-placement solutions, MTAT improves the maximum throughput of LC workloads by up to 1.7× and enhances BE workload fairness by up to 3.3× while ensuring only a minor decrease in throughput.

Acknowledgments

This research was supported in part by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grants funded by the Korea government (MSIT), under Grant No.RS-2025-09942968 (AI Semiconductor Innovation Lab(Yonsei University)), Grant No.RS-2025-02263869 (Development of AI Semiconductor Cloud Platform Establishment and Optimization Technologies), Grant No.RS-2024-00396013 (DRAM PIM Hardware Architecture for LLM Inference Processing with Efficient Memory Management and Parallelization Techniques), Grant No.RS-2024-00459797 (Development of ML compiler framework for on-device AI), Grant No.2022-0-00498 (Development of high-efficiency AI computing SW core technology for high-speed processing of large learning models), and Grant No.2018-0-00503 (Researches on next generation memory-centric computing system architecture).

References

[1] [n. d.]. Amazon Elastic Compute Cloud (EC2). Accessed: 2025. https://aws.amazon.com/ec2/.
[2] [n. d.]. Cgroup interface. Accessed: 2025. https://man7.org/linux/man-pages/man7/cgroups.7.html.
[3] [n. d.]. Compute Express Link (CXL). Accessed: 2025. https://computeexpresslink.org/.
[4] [n. d.]. GAP Benchmark Suite (GAPBS). Accessed: 2025. https://github.com/sbeamer/gapbs.
[5] [n. d.]. Memcached. Accessed: 2025. https://memcached.org/.
[6] [n. d.]. MongoDB. Accessed: 2025. https://www.mongodb.com/.
[7] [n. d.]. Mutilate. Accessed: 2025. https://github.com/leverich/mutilate.
[8] [n. d.]. Python. Accessed: 2025. https://www.python.org/.
[9] [n. d.]. Redis. Accessed: 2025. https://redis.io/.
[10] [n. d.]. XSBench. Accessed: 2025. https://github.com/ANL-CESAR/XSBench.
[11] [n. d.]. Yahoo! Cloud Serving Benchmark (YCSB). Accessed: 2025. https://github.com/brianfrankcooper/YCSB.
[12] Anandtech. [n. d.]. Intel Launches Optane DIMMs Up To 512GB: Apache Pass Is Here! Accessed: 2025. https://www.anandtech.com/show/12828/intel-launches-optane-dimms-up-to-512gb-apache-pass-is-here.
[13] Berk Atikoglu, Yuehai Xu, Eitan Frachtenberg, Song Jiang, and Mike Paleczny. 2012. Workload analysis of a large-scale key-value store. In *Proceedings of the 12th*

ACM SIGMETRICS/PERFORMANCE joint international conference on Measurement and Modeling of Computer Systems. 53–64.

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

[15] Dimitris Bertsimas and John Tsitsiklis. 1993. Simulated annealing. *Statistical science* 8, 1 (1993), 10–15.

[16] Ruobing Chen, Haosen Shi, Yusen Li, Xiaoguang Liu, and Gang Wang. 2023. Olpart: Online learning based resource partitioning for colocating multiple latency-critical jobs on commodity computers. In *Proceedings of the Eighteenth European Conference on Computer Systems*. 347–364.

[17] Shuang Chen, Christina Delimitrou, and José F Martínez. 2019. Parties: Qos-aware resource partitioning for multiple interactive services. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 107–120.

[18] Wanju Doh, Yaebin Moon, Seoyoung Ko, Seunghwan Chung, Kwanhee Kyung, Eojin Lee, and Jung Ho Ahn. 2025. PET: Proactive Demotion for Efficient Tiered Memory Management. In *Proceedings of the Twentieth European Conference on Computer Systems*. 854–869.

[19] 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*. 727–741.

[20] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. 2018. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905* (2018).

[21] Călin Iorgulescu, Reza Azimi, Youngjin Kwon, Sameh Elnikety, Manoj Syamala, Vivek Narasayya, Herodotos Herodotou, Paulo Tomita, Alex Chen, Jack Zhang, et al. 2018. {PerfIso}: Performance isolation for commercial {Latency-Sensitive} services. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. 519–532.

[22] Harshad Kasture and Daniel Sanchez. 2016. Tailbench: a benchmark suite and evaluation methodology for latency-critical applications. In *2016 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 1–10.

[23] 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*. 17–34.

[24] David Lo, Liqun Cheng, Rama Govindaraju, Parthasarathy Ranganathan, and Christos Kozyrakis. 2015. Heracles: Improving resource efficiency at scale. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*. 450–462.

[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*. 742–755.

[26] Alan Nair, Sandeep Kumar, Aravinda Prasad, Ying Huang, Andy Rudoff, and Sreenivas Subramoney. 2024. Telescope: telemetry for gargantuan memory footprint applications. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*. 409–424.

[27] Khang T Nguyen. [n. d.]. Introduction to Cache Allocation Technology in the Intel® Xeon® Processor E5 v4 Family. Accessed: 2025. https://www.intel.com/content/www/us/en/developer/articles/technical/introduction-to-cache-allocation-technology.html.

[28] Rajiv Nishtala, Vinicius Petrucci, Paul Carpenter, and Magnus Sjalander. 2020. Twig: Multi-agent task management for colocated latency-critical cloud services. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 167–179.

[29] 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*. 1–7.

[30] Tirthak Patel and Devesh Tiwari. 2020. Clite: Efficient and qos-aware co-location of multiple latency-critical jobs for warehouse scale computers. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 193–206.

[31] Aidi Pi, Xiaobo Zhou, and Chengzhong Xu. 2022. Holmes: SMT interference diagnosis and CPU scheduling for job co-location. In *Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing*. 110–121.

[32] 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*. 392–407.

[33] Jie Ren, Dong Xu, Junhee Ryu, Kwangsik Shin, Daewoo Kim, and Dong Li. 2024. MTM: Rethinking Memory Profiling and Migration for Multi-Tiered Large Memory. In *Proceedings of the Nineteenth European Conference on Computer Systems*. 803–817.

[34] Sai Sha, Chuandong Li, Yingwei Luo, Xiaolin Wang, and Zhenlin Wang. 2023. vtmm: Tiered memory management for virtual machines. In *Proceedings of the Eighteenth European Conference on Computer Systems*. 283–297.

[35] Yan Sun, Yifan Yuan, Zeduo Yu, Reese Kuper, Chihun Song, Jinghan Huang, Houxiang Ji, Siddharth Agarwal, Jiaqi Lou, Ipoom Jeong, et al. 2023. Demystifying cxl memory with genuine cxl-ready systems and devices. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*. 105–121.

[36] 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*. 18–32.

[37] Midhul Vuppalapati and Rachit Agarwal. 2024. Tiered Memory Management: Access Latency is the Key!. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*. 79–94.

[38] Lingfeng Xiang, Zhen Lin, Weishu Deng, Hui Lu, Jia Rao, Yifan Yuan, and Ren Wang. 2024. Nomad:{Non-Exclusive} Memory Tiering via Transactional Page Migration. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. 19–35.

[39] 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 (USENIX ATC 24)*. 817–833.

[40] 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*. 331–345.

[41] Juncheng Yang, Yao Yue, and KV Rashmi. 2021. A large-scale analysis of hundreds of in-memory key-value cache clusters at twitter. *ACM Transactions on Storage (TOS)* 17, 3 (2021), 1–35.

[42] 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*. 405–418.

[43] Laiping Zhao, Yanan Yang, Kaixuan Zhang, Xiaobo Zhou, Tie Qiu, Keqiu Li, and Yungang Bao. 2020. Rhythm: component-distinguishable workload deployment in datacenters. In *Proceedings of the Fifteenth European Conference on Computer Systems*. 1–17.

[44] Yuhong Zhong, Daniel S Berger, Carl Waldspurger, Ishwar Agarwal, Rajat Agarwal, Frank Hady, Karthik Kumar, Mark D Hill, Mosharaf Chowdhury, and Asaf Cidon. 2024. Managing Memory Tiers with CXL in Virtualized Environments. In *Symposium on Operating Systems Design and Implementation*.