Chrono: Meticulous Hotness Measurement and Flexible Page Migration for Memory Tiering

Zhenlin Qi
qizhenlin@sjtu.edu.cn
Department of Computer Science and Engineering, Shanghai Jiao Tong University
Shanghai, China

Shengan Zheng*
shengan@sjtu.du.cn
MoE Key Lab of Artificial Intelligence, AI Institute, Shanghai Jiao Tong University
Shanghai, China

Ying Huang
ying.huang@intel.com
Intel
Shanghai, China

Yifeng Hui
sinemora@sjtu.edu.cn
Department of Computer Science and Engineering, Shanghai Jiao Tong University
Shanghai, China

Bowen Zhang
bowenzhang@sjtu.edu.cn
Department of Computer Science and Engineering, Shanghai Jiao Tong University
Shanghai, China

Linpeng Huang*
lphuang@sjtu.edu.cn
Department of Computer Science and Engineering, Shanghai Jiao Tong University
Shanghai, China

Hong Mei
meih@sjtu.edu.cn
Shanghai Jiao Tong University
Shanghai, China

Abstract

As the memory demand continues to surge, the limitations of DRAM scalability have spurred the development of various new memory technologies in today's data centers. In order to harness the benefits of the heterogeneous memory architecture, tiering has become a widely adopted memory management paradigm. The effectiveness of a tiered memory management system primarily relies on its ability to accurately identify frequently accessed ("hot") pages and infrequently accessed ("cold") pages, and efficiently relocate them between tiers. However, existing systems rely on coarse-grained frequency measurement schemes that do not align with the performance characteristics of modern memory devices and memory-intensive applications. Additionally, these systems often incorporate rigid rules or manually configured parameters for page classification, resulting in inflexible migration strategies.

This paper introduces ***Chrono***, a novel OS-level tiering system that offers precise characterization of page access frequencies in different tiers and enables efficient migration of hot and cold pages. By leveraging timers instead of counters, Chrono achieves meticulous measurement of hot page access frequency with low overhead. This approach allows Chrono to automatically tune its page classification parameters, leading to flexible migration strategies that adapt to various workloads. Furthermore, Chrono includes a dynamic cold page identification subsystem, which balances the utilization and availability of tiered memory. We have implemented and evaluated Chrono on existing tiered memory platforms, and experimental results demonstrate that Chrono outperforms state-of-the-art tiering systems by a large margin.

*Corresponding author.

EuroSys '25, March 30-April 3, 2025, Rotterdam, Netherlands

ACM ISBN 979-8-4007-1196-1/2025/03
https://doi.org/10.1145/3689031.3717462

CCS Concepts: • **Computer systems organization** → **Heterogeneous (hybrid) systems**; • **Software and its engineering** → **Memory management**; • **Information systems** → *Information storage technologies.*

Keywords: Memory management, Heterogeneous memory, NUMA, Linux kernel

ACM Reference Format:
Zhenlin Qi, Shengan Zheng, Ying Huang, Yifeng Hui, Bowen Zhang, Linpeng Huang, and Hong Mei. 2025. Chrono: Meticulous Hotness Measurement and Flexible Page Migration for Memory Tiering. In *Twentieth European Conference on Computer Systems (EuroSys '25), March 30-April 3, 2025, Rotterdam, Netherlands.* ACM, New York, NY, USA, 19 pages. https://doi.org/10.1145/3689031.3717462

1 Introduction

In light of the rise of memory-intensive applications, such as in-memory database services and foundational model training [13, 26, 36, 50, 70, 73], there has been a notable surge in memory demands within contemporary data centers. However, the scalability of DRAM has proven challenging in meeting the escalating requirements [20, 43, 47, 56]. To address this issue, researchers have recently proposed various alternative technologies. Some focus on new memory devices [12, 25, 31, 80], such as production-grade non-volatile memory (NVM), which provides substantial gains in storage capacity and energy efficiency [17, 19, 21, 34] compared to DRAM. Others focus on new interconnection technologies [7, 28, 74, 77], such as Compute Express Link (CXL), which provides low-latency links for heterogeneous memory and drives the creation of next-generation memory pools [3, 22, 43, 76]. The advancement in hardware brings forth fresh opportunities for OS researchers, with requirements to exploiting the full potential of heterogeneous memory architectures.

The novel main memory technologies, including NVM and CXL memory, exhibit common characteristics [74, 75, 80, 82], which are byte-addressable and provide larger capacity with higher latency (150-270 ns) in comparison to DRAM (50-90 ns). In this paper, we refer to DRAM as "*fast memory*" and the newly introduced memory devices as "*slow memory*". The distinctive properties of slow memory establish it as a new tier situated between conventional SSD-based storage and the fast memory. In practice, it provides directly accessible physical memory to the kernel alongside the fast memory, without introducing architectural overhead such as memory mapping and swapping. With the heterogeneous memory architecture, ***memory tiering*** [49, 63, 78] has become a widely adopted paradigm as it exposes the full capacity of different tiers while maintaining transparency to applications.

To fully exploit the performance of fast and slow memory tiers, the OS needs to measure the ***hotness*** of pages and effectively migrate hot/cold pages between different tiers [1, 18, 33, 38, 61, 67]. Recent researches [27, 35, 39, 48, 49, 64, 72] have been dedicated to optimizing the hotness measurement methods.

However, existing solutions fail to incorporate both fine-grained frequency and high spatial resolution, leading to insufficient page classification ability for today's high performance memory tiering architecture. We have analyzed three common types of methods that utilize *software page fault*, *hardware access bit*, and *hardware event sampling* mechanisms to measure page hotness. Both software page faults and hardware access bits only provide coarse-grained access frequency statistics, whereas hardware event sampling, despite being more precise, is constrained by limited sampling capacity when dealing with fine-grained page sizes. Their inflexible migration criteria and manually tuned profiling schemes also hinder their capability to dynamically respond to evolving workload patterns.

Our analysis reveals the following insight: It is challenging for the *counter-based measurement schemes* to satisfy the requirement of fine-grained access frequency characterization, as their effective statistical scales rely on the measurement intensity, which is directly related to the system overhead. Fortunately, we discover that *utilizing timers to record the idle time of pages* offers meticulous measurement of the access frequency, decoupling frequency resolution from measurement rate, and enables adaptive classification criteria.

We propose ***Chrono***, a novel tiered memory system that provides fine-grained hotness measurement and flexible page migration with low overhead. Chrono leverages an innovative ***Captured Idle Time (CIT)*** method to accurately estimate page access frequency and provides meticulous frequency statistics. With CIT, Chrono significantly improves the hot page identification precision, while adding negligible overhead compared to the vanilla non-uniform memory access (NUMA) management in Linux. The fine-grained measurement scheme motivates us to design conditional promotion schemes, including candidate filtering and rate-limited migration, to identify hot pages stably and efficiently. Meanwhile, Chrono includes a proactive cold-page demotion scheme that balances the memory utilization and availability, with a monitoring method to avoid redundant page migrations and alleviate extra bandwidth consumption.

To exploit Chrono's flexibility in handling various application memory access patterns, we introduce automatic tuning schemes to adjust the critical system parameters at run-time. We design a ***Dynamic CIT Statistic Collection (DCSC)*** scheme that accurately depicts the distribution of page hotness across the fast and slow tiers. It enables Chrono to calculate and adjust its classification threshold and migration rate adaptively, thereby achieving transparent and flexible performance optimization.

In summary, we make the following contributions:

- We conduct a thorough analysis of existing tiered memory management systems, focusing on hotness measurement and migration criteria. It reveals that existing approaches lack the capability to discern fine-grained access frequency and employ inflexible migration strategies.
- We introduce Chrono, a novel tiered memory system that incorporates a meticulous hotness measurement scheme. It takes a timer-based method to precisely capture memory access frequency, with a hot page candidate filtering scheme. We provide theoretical analysis to validate the stability and efficiency of our method.
- We design adaptive parameter tuning methods based on a run-time page hotness distribution statistics subsystem, allowing Chrono to adjust its migration parameters transparently and adaptively. We further design

dynamic cold page identification schemes that optimize the fast-tier memory availability.
- We implement Chrono based on the Linux kernel and make it open-source. With thorough evaluation on various memory-intensive benchmarks, we show that Chrono outperforms the state-of-the-art tiered memory systems by a large margin.

2 Background and Motivation

In this section, we first review the existing NUMA-balancing scheme of Linux. Second, we show that memory-intensive workloads impose significant memory pressure on the slow tier, emphasizing the importance of accurate page hotness measurement. Third, we analyze recent research works focusing on hotness measurement and show their limitations on measurement granularity, with further experimental results demonstrating their deficiencies in hot page identification.

2.1 NUMA migration scheme

Current researches and industry solutions tend to reuse the NUMA management system to organize the new memory tiers [3, 43, 49], prioritizing system implementation simplicity and application transparency. In a multi-socket NUMA-aware system, the auto NUMA-balancing scheme [60] manages the cross-NUMA page migration procedure. It is designed to optimize the page placement according to the CPU sockets' memory localization. By periodically scanning the address space of a process and marking a range of pages as inaccessible, using the `PROT_NONE` flag, a page fault will occur when the process accesses a scanned page and the kernel intercepts it. The kernel then verifies the tag of the faulted page indicating which CPU performed the last access by checking the corresponding bits in the `struct page`. A migration occurs if the memory node of the page does not match the CPU node that triggered the fault. The auto NUMA-balancing scheme is expected to improve the performance when the applications cannot guarantee local memory allocation.

However, the original NUMA-balancing policy is unsuitable for page migration in tiered memory architecture. Since the newly-added slow tier appears as a CPU-less memory node, any memory access occurring in the poisoned address range will lead to page migration, which is equivalent to applying a most recently used (MRU) algorithm to identify hot pages. This MRU approach fails to capture access frequency, resulting in the misidentification of hot pages in the slow tier, as it may promote candidates that have remained idle for a considerable time before their most recent access. Therefore, it is imperative to implement a more precise hotness measurement mechanism that integrates frequency statistics to enhance the page migration accuracy.

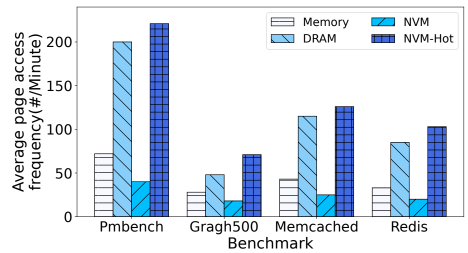


Figure 1. Per-page memory access frequency, with DRAM, NVM contribution, and top-10% hot region stats of NVM.

2.2 Memory pressure on fast-slow tiers

As fast-slow memory tiering offers increased capacity and bandwidth, the substantial memory traffic highlights the inadequacy of conventional coarse-grained techniques in accurately capturing page access frequency. Coarse-grained frequency measurement methods lead to inaccurate hot page identification, which will result in overall performance degradation. We conduct a quantitative analysis to demonstrate that only a precise hotness measurement method can distinguish hot pages effectively.

We run Pmbench [86], Graph500 [55], Memcached [53] and Redis [66] on a DRAM-NVM tiered system (see details in Section 5). We then use the PMU tool [6] (with processor event-based sampling (PEBS) [29] on x86) to capture memory accesses that target different memory regions. We calculate the average per-page access frequency for DRAM and NVM, respectively, dividing the number of memory accesses by the total number of pages. As shown in Figure 1, DRAM exhibits denser access patterns compared to NVM, due to the inherent differences in hardware characteristics Nevertheless, we observe that each NVM page also exhibits 20-40 accesses per minute on average, emphasizing the necessity for highly accurate hotness measurement strategies to enhance hot page selection in tiered memory systems.

Specifically, our analysis of the instruction samples reveals that the top 10% hot NVM pages exhibit access frequencies up to 5.5$times$ higher than the average access rate across the entire NVM region. As a result, an effective hotness measurement method, which is capable of handling access frequencies from tens to hundreds of accesses per minute, is essential for accurately distinguishing between hot, warm, and cold pages in the memory space. However, we analyze and find that existing solutions fail in achieving the precision needed for accurate hotness measurement.

2.3 Characteristics of existing solutions

We analyze the recent memory tiering research works, including Auto-Tiering [35], Multi-Clock [48], TPP [49], and Memtis [39], and summarize their characteristics in Table 1. As an overview, the majority of existing tiering systems adopt counter-based frequency measurement methods to

	Solution Type	Migration Criterion	Effective Frequency Scale	Default Page Size
Auto-Tiering	System-wide	Page-fault counters	0∼1 access/min	Base page
Multi-Clock	System-wide	Multi-level LRU lists	0∼1 access/min	Base page
Telescope	System-wide	Tree-structured PTE bits	0∼5 access/sec	Base page
TPP	System-wide	Page-fault + LRU lists	0∼2 access/min	Base page
Memtis	Process level	PEBS stats + Ratio config	0∼10 access/sec	Huge page
FlexMem	Process level	PEBS stats + Page fault	0∼10 access/sec	Huge page
Chrono [Ours]	**System-wide**	**Dynamic CIT stats**	**0∼1000 access/sec**	**Base page**

Table 1. Characteristics of the design and principles in recent tiered memory research works.

capture hotness, which tightly couple the effective frequency scale with the measurement intensity and lead to coarse-grained frequency statistics.

Software page fault. Many existing solutions utilize the software page fault mechanism to record memory access at the kernel level [1, 35, 49]. Auto-Tiering utilizes the software page faults, recording memory access history within the recent eight page-scanning periods as an 8-bit LAP (least accessed page) vector, to distinguish hot/cold pages [35]. TPP combines the page fault criterion provided by the NUMA-balancing scheme with the access recency criterion provided by the LRU mechanism, identifying hot pages by synthetic information [49], which is considered a software-hardware cooperated method.

However, these solutions fail to provide fine-grained page access frequency measurement resolution. The kernel performs page scan operations cyclically on the virtual address space of each process, where the default scan interval is set to one minute. Thus, existing solutions are unable to meet the precision required for effective hotness measurement. For instance, a page listed in the level-8 LAP only represents at least eight accesses over the last eight minutes. Adjusting the number of lists does not lead to improved precision in frequency measurement. While shortening the scanning period could theoretically refine frequency measurements, it also leads to substantial page-fault handling overhead.

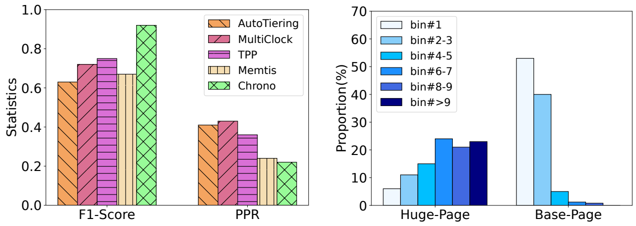
Hardware access bit. Other researchers propose to utilize conventional processor-managed bits and construct refined LRU lists to measure page hotness [18, 48, 57]. Leveraging the reference/dirty bits in page table entries (PTE), Multi-Clock optimizes the Linux page reclamation algorithm, named `clock`, by constructing multi-level LRU lists and selecting migration candidates from the top/bottom lists [48]. TMTS also adopts this mechanism to monitor page access and builds a hardware-based timely hot page selection algorithm [18]. Telescope, on the other hand, takes advantage of the tree-structured PTEs to enable a region-based profiling that is efficient for TB-level memory systems [57].

Nonetheless, they also fail to provide fine-grained hotness measurement. Hardware bits provide only *"accessed or not"* information over a fixed period, hindering the ability to capture nuanced access patterns. The reset intervals are determined by memory shortages and the size of the LRU list, often lasting from minutes to hours in today's data centers. While effective at identifying idle pages, the coarse-grained measurement fails to provide the precise access frequency statistics that are necessary for accurately tracking hot pages. The tree-structured PTE bits used in Telescope, although providing a scalable solution for large memory systems, also has a fixed profiling window (200ms) that limits its frequency resolution at each level of PTE tree.

PEBS counter. Recently, researchers have proposed to utilize the PEBS scheme to collect access frequency statistics [39, 64, 84]. HeMem [64] utilizes PEBS counters to represent the memory access frequency and classify hot and cold pages based on fixed thresholds. Memtis [39] further proposes a global histogram-based statistic mechanism with a fast-slow memory ratio configuration to adjust its classification criterion. FlexMem [84] integrates the PEBS-based method with the software page fault method to provide a synthetic classification criterion, which enhances Memtis with timely migration decisions.

Unfortunately, the PEBS-based tiering solutions are mainly optimized for huge-page (2MB page size) systems and face considerable obstacles when applied to base-page (4KB page size) systems. The root cause is that micro-operations in the sampling mechanism introduce non-negligible CPU and memory overhead to the users, across different hardware platforms [4, 9, 69]. This type of solutions adopt the sampling rate lower than *100000* samples per second, where the Linux kernel forces the upper-bound and system designers further restrict the sampling rate for performance consideration. Given the fact that a stable PEBS-based classification algorithm needs significant counter values (from 2^5 to 2^{15} in HeMem and Memtis) for each hot page within a cooling period (usually several seconds), the limited sampling rate prevents them from tracing large amount of pages. Consequently, these approaches face significant limitations in achieving fine-grained hotness identification in base-page systems, due to inherent hardware constraints in balancing address profiling resolution and system overhead.

Moreover, running traditional base-page oriented applications on huge-page based systems inevitably leads to memory bloat [54], which not only wastes memory space [37],



(a) Hot page identification. **(b)** PEBS bin distribution.

Figure 2. Characteristics of (a) hot page identification efficiency in existing solutions, and (b) PEBS bin distribution under different page granularity in Memtis.

but also degrades the classification accuracy due to hotness fragmentation [8, 42] that enlarges the identified hot region size. Memtis is able to reduce memory bloat ratio moderately by page splitting operations, which is, however, another performance bottleneck for tiered memory systems.

2.4 Limitations in hot page identification

To evaluate the effectiveness of hotness identification methods in existing works, we construct a memory-intensive benchmark with a skewed access pattern using the Pmbench tool. With a Gaussian access pattern and a stride step of 2, we run a 32-thread Pmbench workload on a platform with 256GB memory, with 25% of the memory composed of DRAM and the remaining as NVM. We stat the run-time memory access frequencies of the different memory nodes using the PMU tool and focus on two metrics. The first is the F1-score. Taking accesses that fall into the center 25% of the address space (hot region defined by the normal distribution in workload configuration) as actual positives, and accesses to DRAM as predicted positives since all the identified hot pages are promoted to the DRAM node, we calculate the harmonic means of precision and recall as F1-score. The second is the page promotion ratio (PPR), which is calculated as the ratio of the number of pages promoted to DRAM to the total number of accessed NVM pages. Results are shown in Figure 2a.

An ideal hotness identification method should have a high F1-score and a low PPR, which means that it can accurately identify hot pages and avoid unnecessary page migrations. As the results show, the existing page-fault based methods and hardware-bit based methods exhibit lower precision due to unnecessary migrations, while the PEBS-based methods (such as Memtis) provide lower recall due to hotness fragmentation caused by their use of huge pages, and the fact that our benchmark is base-page oriented.

We further analyze the limited generalizability of PEBS for the base-page system by counting the number of pages that fall into different hotness levels under PEBS-based statistical schemes and show them in Figure 2b. We run the same workload and collect the counter values in different bins representing various hotness levels, and repeat it under different page granularity settings. We observe a significant reduction in counter values within the base-page system, which leads to an unstable classification of hot and cold pages. Specifically, in the huge-page system over 80% PEBS counters fall in the 4th or higher bin (representing access counter value $\geq$ 8), while in the base-page system this ratio is reduced to under 7%. Statistically, a smaller PEBS counter value is related to a higher coefficient of variation, which indicates the instability of the PEBS-based hotness classification in the base-page system. It shows that the vast disparity between moderate sampling rate and large page count in the base-page system hinders PEBS from delivering meaningful statistical information for accurate hotness identification.

3 Design

In this section, we present Chrono, a novel OS-level tiered memory management system. Chrono employs a timer-based hotness measurement scheme that accurately tracks page access frequency with minimal overhead, enabling efficient system-wide hot and cold page identification while supporting flexible page migration strategies. Built on the mainline Linux kernel, Chrono employs the NUMA abstraction to architecturally separate fast and slow tiers. As shown in Figure 3, Chrono integrates three distinct components that collectively streamline page hotness tracking and migration within the tiering framework:

- **Meticulous Page Promotion**. We propose a timer-based page hotness measurement method based on page idle time capturing, facilitating accurate hot page identification for promotion with minimal overhead.
- **Adaptive Parameter Tuning**. We design two parameter tuning methods to manage page classification and migration within Chrono, with the first focusing on low overhead and the second enabling full automation via an integrated hotness statistic mechanism.
- **Proactive Page Demotion**. We introduce a new memory watermark to trigger page demotion proactively, achieving a balance between memory availability and hot page placement. We also propose a page thrashing monitor to mitigate unnecessary migrations.

3.1 Meticulous Page Promotion

The efficiency of the page promotion hinges on the accurate measurement of page hotness and the effective identification of hot pages for migration. Our meticulous approach utilizes **Captured Idle Time (CIT)**, which is the time gap between page scan and page fault, as a reliable metric that negatively correlates with access frequency. This allows us to estimate page hotness with minimal overhead while employing lightweight classification methods. To ensure robust hot page identification, we design a conditional promotion scheme with high stability and efficiency.

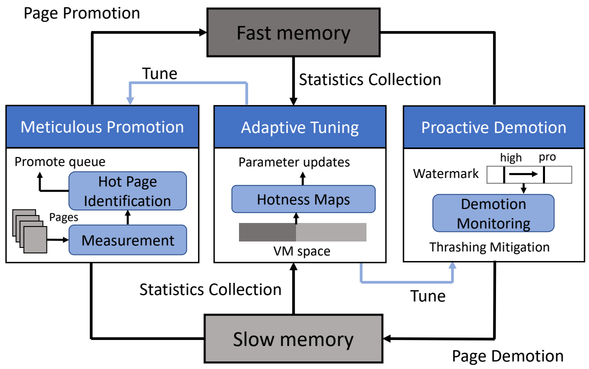


Figure 3. Chrono design overview.

3.1.1 Timer-based hotness measurement.

To measure page hotness effectively, we propose a *Ticking-scan* scheme that captures precise timestamps of page unmapping events during periodic scans. By combining them with consecutive page-fault timestamps, Chrono calculate CIT values for each page efficiently while maintaining bounded overhead.

Specifically, Ticking-scan employs a timer-based approach to monitor page hotness by periodically scanning the virtual memory space of active processes. Each scan marks a range of pages as inaccessible (by setting PTE bits to `PROT_NONE`), and records the scan timestamp for slow-tier pages. When an unmapped page is accessed, a page fault triggers Chrono to log the corresponding page-fault timestamp. CIT is then calculated as the difference between the page-fault timestamp and the prior scan timestamp. This metric is statistically proportional to the interval between consecutive accesses to the page, providing an accurate reflection of access frequency. Our experimental results (Subsection 5.1, Figure 10a) also support that a lower CIT correlates with a higher access frequency and vice versa.

By decoupling measurement resolution from the scan period, CIT provides an effective mechanism for capturing a wide range of access frequencies. Using millisecond-based timers, Chrono achieves a measurable frequency upper bound of 1000 accesses per second, making Ticking-scan particularly suited for fine-grained hot page identification in memory-intensive environments. CIT also allows Chrono to precisely identify hot pages with minimal system overhead, consisting of only timestamp recording and basic arithmetic calculations. In addition, the metadata required for CIT occupies only 4 bytes per page, imposes a negligible space cost, ensuring scalability in systems with very large memory capacities.

Chrono classifies hot and cold pages based on their corresponding CIT values and migrates them across memory tiers. A system-wide **CIT threshold** is employed as the classification boundary, ensuring optimal tier assignment. To adapt to various workloads, Chrono incorporates adaptive parameter tuning mechanisms (detailed in Subsection 3.2) that dynamically adjust the threshold in response to workload changes, enabling adaptive and responsive memory management.

3.1.2 Conditional page promotion.

Despite the high efficiency of CIT-based hot page identification, its accuracy can be compromised by the inherent randomness in scan timings and page access patterns, where a single-round CIT-based classification can occasionally result in unstable promotions. Additionally, a fixed CIT threshold does not responds effectively to various workloads. To prevent unnecessary migrations, we introduce a candidate filtering scheme to refine the identification process by evaluating multiple CIT rounds, thus reducing the likelihood of premature promotions. We also design a rate-limited promotion queue, which controls the frequency of page promotions, preventing excessive migrations and minimizing system overhead.

As illustrated in Figure 4, the candidate filtering procedure begins by logging the CIT values of all accessed pages within a Ticking-scan range. Pages with CIT values below the threshold are selected as candidates and stored in an XArray, which allows for low-latency access and minimal memory consumption. During the next scan cycle, these candidates undergo a second evaluation. If a page's CIT remains below the threshold, it is marked for promotion and added to the promotion queue. This two-round filtering mechanism ensures more accurate promotion decisions, minimizing unnecessary migrations and reducing system overhead.

Candidate filtering enhances both the stability and efficiency of the promotion process. By using the maximum value of two CIT samples, it reduces the chance of incorrectly classifying pages as hot, lowering the likelihood of premature promotions (Appendix B.1). Additionally, limiting the rounds of sampling ensures that the resource footprint remains low, avoiding the page-fault overhead associated with excessive sampling. Both our theoretical analysis (Appendix B.2) and experimental results confirm that two-round page selection strikes a balance between stability and efficiency without compromising system performance.

Chrono initiates asynchronous page migration for candidate pages that are deemed ready for promotion. This involves remapping the pages and copying data across memory tiers, followed by their removal from the promotion queue. To prevent excessive migrations, we introduce a **promotion rate limit**, which regulates the number of migrations and reduces system overhead. Chrono regularly updates the rate limit based on a running count of enqueue and dequeue events, tuning it dynamically to match workload intensity (Subsection 3.2). Through the adaptive adjustment of migration rate, Chrono ensures timely page migrations while maintaining low overhead, achieving a delicate balance between migration responsiveness and system stability.

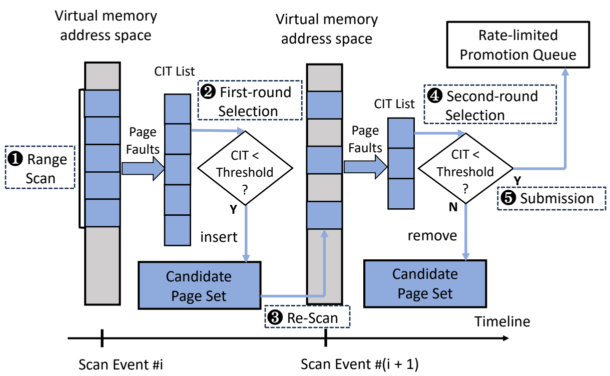


Figure 4. The candidate filtering scheme.

3.2 Adaptive Parameter Tuning

Chrono's page promotion mechanism relies heavily on the coordination between the CIT threshold and the promotion rate limit. The *CIT threshold* serves as a dynamic upper bound, ensuring that only the hottest pages (those with CIT values shorter than the threshold) are selected as promotion candidates, while the *promotion rate limit* modulates the number of pages promoted during each scan cycle, preventing overloading of the fast memory. By adaptively controlling both the quality and quantity of promoted pages, Chrono is dynamically adjusted to fit various workload patterns.

To optimize its performance at run-time, Chrono incorporates two parameter tuning methods. The semi-automatic method adjusts only the CIT threshold dynamically based on workload characteristics, providing stable performance with negligible overhead. The fully automatic method, which serves as Chrono's default option, utilizes a sophisticated dynamic CIT collection and statistical monitoring system. This approach continuously adapts to shifts in workload memory access patterns, offering a completely automated solution that fine-tunes both the CIT threshold and promotion rate dynamically.

3.2.1 Semi-auto parameter tuning. Chrono's semi-auto parameter tuning method provides a user-guided approach to memory tier management. Users manually configure the promotion rate limit, while Chrono automatically adjusts the CIT threshold to align with dynamic memory access patterns. This approach is particularly suited for users with a deep understanding of their applications' memory behavior, striking a balance between user control and system adaptability.

During each Ticking-scan period, Chrono continuously monitors the promotion enqueue rate and compares it to the rate limit, recalibrating the CIT threshold to maintain balance. If too many pages enter the promotion queue, the threshold is reduced to slow the promotion rate, and vice versa. Specifically, the adjustment is determined by a coefficient r, calculated as the ratio between the promotion rate limit and the promotion enqueue rate. With an adaption step δ, the threshold TH update process is represented as:

$$r_i = \frac{Rate\ Limit[i]}{Enqueue\ Rate[i]}, \quad TH_{i+1} = (1 - \delta + \delta \cdot r_i) TH_i.$$

Guided by the adjustment coefficient r_i, Chrono ensures that the enqueue rate of promotion queue converges to the rate limit, without overwhelming the promotion queue or underutilizing available memory resources.

Chrono's semi-auto tuning method strikes a balance between simplicity and precision by using only two counters to dynamically decide the CIT threshold. A well-balanced threshold ensures that hot pages are promoted with optimal accuracy and promotion rate. A short CIT threshold yields too few promotion-ready pages, resulting in prolonged adjustments, while a long threshold can cause overflow with excessive hot page promotions. By averaging the enqueue rate within each Ticking-scan period, Chrono ensures smooth and predictable adjustments with minimal system overhead.

3.2.2 Statistics-based parameter tuning. The semi-auto parameter tuning offers a lightweight yet effective way to classfy hot pages by automatically fine-tuning the CIT threshold. While this method provides a degree of flexibility, its reliance on user-provided rate limits may pose challenges for workloads with unfamiliar memory access behaviors. Moreover, due to the periodical and gradual nature of adjustments, the method may exhibit delayed responsiveness to rapidly changing memory demands.

To resolve these challenges, Chrono introduces a statistic-based fully automatic parameter tuning method that is leveraging a **Dynamic CIT Statistic Collection (DCSC)** approach to adaptively adjust both the CIT threshold value and promotion rate limit. DCSC periodically performs the Ticking-Scan procedure to sample a randomly selected small memory portion from different tiers, generating heat maps that reflects the overall CIT distributions of each tier. Chrono then compares the heat maps of different memory tiers to determine the misplacement ratio and control the migration rate, enabling a transparent and adaptive page management system without manual configuration.

The DCSC scheme is depicted in Figure 5. To begin, Chrono randomly selects a small portion ($P\%$) of the virtual memory space allocated to the process, designating these pages as victim pages. These pages are marked as inaccessible using a special flag (`PG_probed`) to differentiate them from those subjected to a regular Ticking-scan. Subsequently, the CIT values of the probed pages are gathered using a two-round CIT generation mechanism, allowing for a global standard for both the statistics and hot page identification schemes. The overall distribution of page hotness is represented in a heat map organized into B buckets corresponding to different frequency ranges, facilitating for efficient detection of overlapping hotness levels. For each memory tier, a statistic

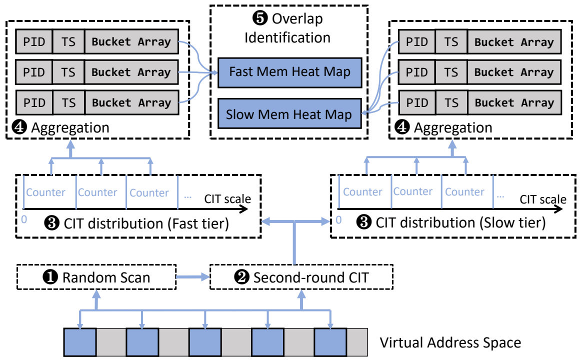


Figure 5. Dynamic CIT statistic collection scheme.

table is created that contains the process PID, timestamp, and a CIT range counter array. As processes update their entries, Chrono dynamically aggregates and updates the heat maps for each tier, leading to a transparent and real-time memory hotness capturing system.

Chrono leverages heat maps to track page hotness across memory tiers, dynamically identifying overlaps and adjusting migration policies to optimize efficiencies. By comparing the hot pages in the slow-tier with the cold pages in the fast-tier, Chrono identifies overlaps on the same hotness levels in different tiers. The number of overlapping pages is used to calculate the misplacement ratio, which is then multiplied by the memory consumption and divided by the Ticking-scan period to determine the proper promotion rate limit. Meanwhile, the CIT threshold is dynamically recalibrated based on the overlap point, enabling Chrono to continuously fine-tune its page classification criterion in response to dynamic workload patterns.

The statistical scan procedure operates independently from the Ticking-scan, employing a randomized sampling order to facilitate rapid parameter tuning. While Ticking-scan sequentially scans the entire address space, the statistical scan selectively probes a small memory subset, enabling frequent per-second scans without imposing significant overhead. Meanwhile, the randomness of the selected memory pages ensures a statistically accurate representation of memory access patterns while providing a stable basis for parameter adjustments. By leveraging the DCSC approach, Chrono offers a fully automated memory management solution that adapts to changing workload patterns without user intervention, ensuring optimized performance and efficiency across a wide range of applications.

3.3 Proactive Page Demotion

Effective page demotion is essential for Chrono's goal of maintaining an optimized and responsive tiered memory system. To facilitate timely and effective page migration, Chrono introduces a promotion-aware memory watermark that triggers the demotion of cold pages from the fast tier. Pages in the fast-tier inactive list are selected based on an LRU algorithm for demotion to the slow tier. Moreover, Chrono continuously monitors for page thrashing, adjusting its migration policies dynamically to prevent unnecessary migrations and optimize memory utilization.

3.3.1 Watermark-based page demotion. To maintain memory availability of fast-tier memory and avoid promotion delays, Chrono integrates a proactive demotion scheme triggered by a promotion-aware watermark. We extend the Linux memory reclamation mechanism by introducing a promotion-aware watermark (`pro`), which resides above the original `high` watermark, to ensure that sufficient memory is always available for hot page promotions. When fast-tier memory availability falls below the `high` watermark, demotion is triggered to free space until the amount of available memory reaches the `pro` watermark. The gap between the `high` and `pro` watermarks is dynamically configured to ensure ample space for page promotions, calculated as twice the default scan interval multiplied by the promotion rate limit. Demotion candidates are chosen from the inactive list of the fast-tier memory using an LRU algorithm, providing a lightweight and scalable approach to managing cold pages, while ensuring efficient memory reclamation and avoids unnecessary page thrashing.

Overall, Chrono ensures that fast-tier memory pages are efficiently managed without the need for disk-based reclamation. Instead, DRAM cold pages are proactively demoted to the slow tier, maintaining the performance of demand paging by keeping sufficient available fast-tier memory capacity. Meanwhile, the slow-tier pages could be promoted to DRAM, and also could be swapped out to disk if necessary. It also enables Chrono to accommodate user-defined memory limits (e.g., `cgroups memory.limit`), while prioritizing the retention of hot pages in the fast tier. When memory limits are reached, Chrono initiates slow-tier reclamation to relieve memory pressure while maintaining the placement for hot pages, without sacrificing application performance.

3.3.2 Page thrashing monitor. Page thrashing occurs when recently demoted pages are prematurely promoted back to the fast tier, leading to redundant page migrations, unnecessary page faults, and wasted memory bandwidth. To mitigate this, Chrono incorporates a page thrashing monitor that tracks the hotness of recently demoted pages.

Chrono marks each recently demoted page with a new flag, *demoted*, and immediately makes them inaccessible by changing the corresponding PTE bits to `PROT_NONE`. The demotion timestamp is stored as a substitution for its Tiering-scan timestamp, and the page is re-evaluated under the same promotion criteria as other slow-tier pages. A thrashing event is recorded if a *demoted* page is marked as promotion candidate again within a scan period. By periodically comparing the thrashing rate with the overall promotion rate, Chrono dynamically adjusts the promotion rate limit. If the thrashing ratio exceeds a preset threshold (e.g. 20%), Chrono responds by halving the promotion rate limit for the next scan period, which effectively mitigates the impact of thrashing without compromising system responsiveness.

3.4 Huge Page Support

Chrono also supports huge pages, which are commonly used in memory-intensive applications to reduce TLB misses and improve memory management efficiency. The Linux kernel provides a mechanism to allocate different page sizes, including 2MB and 1GB huge pages, which can be used in both fast and slow tiers.

To support huge pages, Chrono extends the candidate filtering mechanism and DCSC approach to handle different page sizes effectively. Ticking-scan is adapted to measure the hotness of huge pages by logging the CIT values of each huge page in the same way of base-page, and using an adjusted CIT threshold to classify hot and cold items for huge pages. For example, the CIT threshold for 2MB huge pages is set to $TH_{2MB} = \frac{TH_{4KB}}{512}$, and the CIT threshold for 1GB huge pages is set to $TH_{1GB} = \frac{TH_{4KB}}{512\times512}$. It ensures that the hotness measurement and promotion mechanism are consistent across different page sizes. The DCSC approach is also adaptive to huge pages, where the huge-page related CIT values are counted into the CIT heat map by eventually distributing the calculated accesses to the corresponding 4KB pages. For example, a 2MB huge page falling into the i−th CIT bucket will be counted as 512 base pages in the $(i+9)$-th CIT bucket (assuming that adjacent CIT buckets representing 2x access frequency). This approach ensures that the hotness statistic mechanism remains fair and consistent across different page sizes, enabling seamless integration of huge pages into the timer-based tiering framework.

4 Implementation

We implement Chrono based on the Linux kernel v5.18, with 1.9k SLOC code changes, and have made it publicly available on GitHub[1] and Zenodo[2]. We add a new `numa_tiering` option in `sysctl` to enable Chrono.

All the configurable parameters in Chrono are summarized in Table 2. For Ticking-scan, we set the default manner identical to the Linux kernel's NUMA scan mechanism. The additional CIT metadata allocated in the `extended struct page` structures consumes 0.2% of the physical memory space, and thus incurs only a modest space overhead. Regarding the XArray indexing the hot page candidates, we allocate new slots in the kernel space, which consume less than 32 KB memory on average for each active process across their lifetime as there is a limited number of pages selected as promotion candidates by the DCSC design.

Name	Default	Description
Scan step	256 MB	• Marked page set size of a Ticking-scan event.
Scan period	60 sec	• Period for Ticking-scan to loop over address space.
P-victim	0.003%	• Ratio of pages sampled in the DCSC scheme.
B-bucket	28	• Number of different CIT-levels in DCSC stats.
δ-step	0.5	• Adaption step for CIT threshold adjustment.
CIT threshold	1000 ms	• Auto-tuned.
Rate limit	100 MBps	• Auto-tuned.

Table 2. Summary of the parameter default values in Chrono.

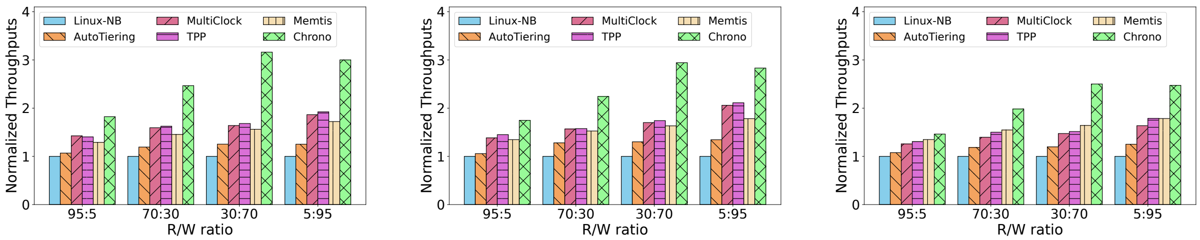
In the DCSC-based tuning schemes, we choose a small portion P of pages to be sampled as the victims, where 0.003% is corresponding to about 8 MB in our 256 GB platform. The default victim ratio should be decreased when applied to a larger memory system, to avoid scanning too many pages in the statistics collection process. For CIT-level buckets, the finest CIT level is 1 ms and i−th bucket contains the CIT values in the range of $[2^{i-1}, 2^i)$ millisecond. Setting the finest granularity as 1 ms is sufficient to hotness representation, as the CIT values are not used for precise time measurement but for frequency estimation, and pages with a CIT value above 2^{27} ms (which indicates at least 37.3 hours untouched) will not be considered as key points in hot-page selection. We have also developed `procfs` controllers that allow system managers to configure parameters manually as they need. Details about the impact of these parameters are provided in the evaluation section.

5 Evaluation

Our evaluation testbed is equipped with an Intel Xeon Gold 6348 CPU running at 2.6 GHz. The fast memory is composed of four local 16 GB DDR4 DRAM modules. We configure two 128 GB Intel Optane PM modules in a CPU-less NUMA node as slow memory, following the community tendency [23]. It has about 200 ns memory load/store latency, which is also similar to CXL memory specification [74, 75].

For comparison, we choose the Linux kernel v5.18 with NUMA-balancing [52] (Linux-NB), Auto-Tiering [35], Multi-Clock [48], TPP [49], and Memtis [39]. For Auto-Tiering we use opportunistic and background mode (OPM-BD) to get the best performance. Memtis running on a base-page system performs similar to vanilla Linux, such that we keep the huge-page options as its suggested setting ("`always`"). We use Pmbench [86], Graph500 [55], Memcached [53] and

[1] https://github.com/SJTU-DDST/chrono-project
[2] https://doi.org/10.5281/zenodo.14875828



(a) 50 processes, 5 GB working set. **(b)** 32 processes, 8 GB working set. **(c)** 32 processes, 4 GB working set.

Figure 6. The pmbench throughput under different concurrency levels and working set sizes.

Redis [66] as benchmarks, and lkp-test [46] tools for lightweight kernel-level feature characterization.

5.1 Microbenchmark: Pmbench

We use Pmbench [86] to construct memory-intensive workloads, profiling the throughput and latency. To elaborate the cause of performance differences, we analyze the kernel characteristics at run-time. We also construct a multi-tenant workload to evaluate the hot/cold identification effectiveness. Moreover, we track the parameter values during the benchmark execution time to examine the effectiveness of our adaptive parameter tuning methods.

5.1.1 Throughput/Latency profiling. We use memory-intensive workloads with a skewed and sparse access pattern. With 50 concurrent Pmbench tasks, each having 5 GB private working sets, we configure the `pattern` as `normal_ih` and `stride` as 2, resulting in scattered Gaussian distributed accesses over the address space.

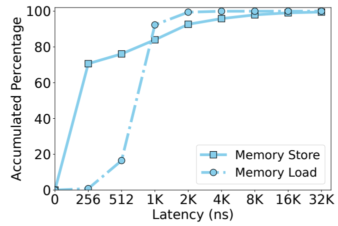
The throughput results are shown in Figure 6. Chrono provides higher throughput under various read-write ratios, outperforming Linux-NB, Auto-Tiering, Multi-Clock, TPP, and Memtis by 216%, 152%, 92%, 90%, and 102%, respectively. The absolute throughput of Linux-NB is 71.5 Mop/s, and the working set includes 62.5M pages for a base-page system, such that the average frequency is 1.14 access per second. Auto-Tiering, Multi-Clock, and TPP do not discern such a high frequency resolution so that they fail to identify the real hot pages. Memtis suffers from hotness fragmentation, where each 2MB huge-page has only a half 4KB-regions being accessed, and its splitting strategy is too conservative to mitigate this problem. Chrono provides fine-grained frequency measurement on base-page granularity, thus it is able to relocate hot/cold pages precisely.

We also change the concurrency level and the working set size to evaluate the adaptiveness of different systems in Figure 6b and Figure 6c. Memtis performs better under smaller resident sizes, since the increased fast-tier memory ratio alleviates the bottleneck caused by hot page fragmentation. Chrono optimizes the system better under write-intensive workloads, which comes from the biased read/write performance of Optane PM, indicating that Chrono avoids intensive memory load/store to the slow-tier pages. We also find that Chrono is more efficient under high memory utilization, and the system-wide statistics method yields stable results for different concurrency levels.

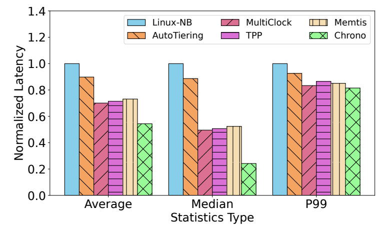
We then concentrate on the 50-process workload and profile its latency distribution under the Linux-NB system in Figure 7a. We find more improvement space at median latency for read, and at tail latency for write. The latency characteristics of all the tested systems are shown in Figure 7. Chrono achieves lower latency than the existing systems; it reduces the average latency and the P99 latency by as much as 68% and 79% respectively. Auto-Tiering fails to distinguish hot and cold pages precisely, because it incurs high kernel-level overhead by maintaining the LAP lists that include only coarse-grained frequency statistics. Similarly, Multi-Clock and TPP distinguish hot pages by LRU lists with a one-minute time window, which fails to capture the fine-grained frequency difference. Memtis also achieves more limited improvements in latency than Chrono because some hot pages are migrated out of the fast memory due to hot region bloat. Chrono measures the page access frequency effectively and selects hot page candidates precisely, leading to lower overall access latency.

5.1.2 Performance attribution. To investigate the reasons for the performance improvement, we collect run-time characteristics including the fast-tier access ratio, kernel level overhead and context switch rate, and show the results in Figure 8. Generally, Chrono places more hot pages to the fast-tier with moderate kernel overhead, while reducing page-fault handling time by precise migration.

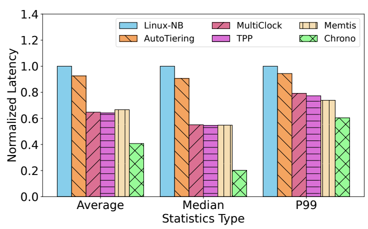
We compute the fast-tier memory access ratio (FMAR) by sampling and dividing the size of memory access records to fast and slow-tier memory. Higher FMAR indicates more hot page identified properly. Chrono improves the FMAR from 49% to 77%, which is significantly higher than other systems. Auto-Tiering spends over 14% of the execution time in kernel, which is 2.2× the overhead of the Linux-NB baseline, reducing its optimization effectiveness. Chrono adds 2.1% kernel time cost compared to Linux-NB, where 1.8% comes from the DCSC scheme. Memtis avoids extra page fault and metadata management overhead owing to the huge-page system, and



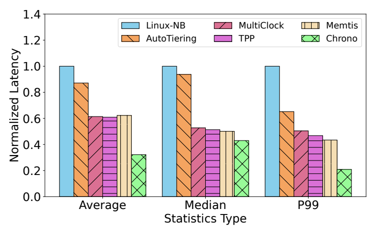
(a) Baseline latency profiling.



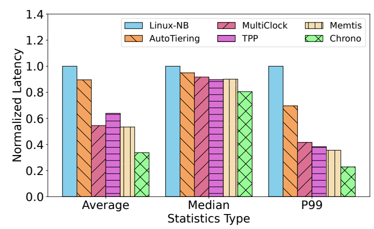
(b) R/W ratio = 95:5



(c) R/W ratio = 70:30



(d) R/W ratio = 30:70



(e) R/W ratio = 5:95

Figure 7. The pmbench latency under various read to write ratio, normalized to Linux-NB.

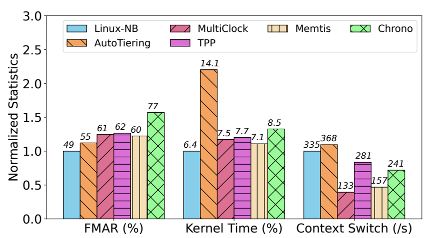


Figure 8. Run-time characteristics comparison.

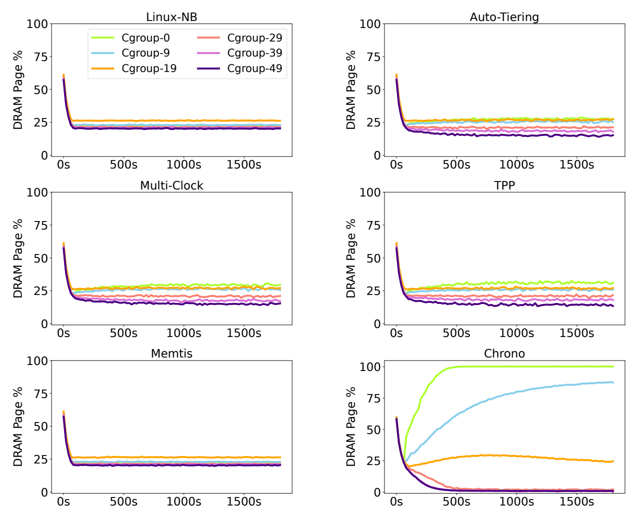


Figure 9. The DRAM page percentage history of multi-process benchmarks with different hotness levels.

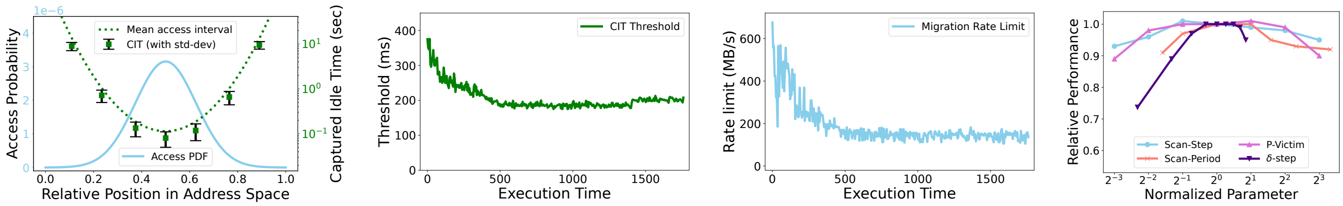
it incurs sampling and statistic overhead, leading to 0.7% total increase. We also observed that the majority of context switches happen due to page faults. Multi-Clock has the lowest context switch rate because it adopts LRU lists without forcing page faults. Chrono reduces the context switch rate compared to Auto-Tiering and TPP, because it selects hot pages precisely and avoids redundant page migration.

5.1.3 Hot/cold page identification. We construct a synthetic workload with various frequency levels to illustrate the effectiveness of the hot/cold page identification schemes. We profile 50 `Cgroups` and conduct one Pmbench process in each with random access pattern. To generate different frequencies, we use the `delay` parameter to add stall time before every memory access, with i unit(s) (50 cycles) of delay for i-th process. Throughput decreases with the increasing of access delay, where cgroup-0 has 2.8× throughput of cgroup-49 under Linux-NB.

We monitor the `sysfs/numa_stat` of each cgroup to collect the number of pages allocated to different memory tiers, and define the DRAM page percentage as:

$$\frac{\#FastTier\ page}{(\#FastTier\ page + \#SlowTier\ page)} \times 100\%,$$

which captures the process' page distribution. The DRAM page percentage logs are shown in Figure 9. They demonstrate that the NUMA-balancing scheme is not able to distinguish different access frequencies. All the processes allocate approximately a 25% ratio of DRAM pages, which is the average fast-tier memory ratio in the workloads. Auto-Tiering, Multi-Clock, TPP, and Memtis show results similar to Linux-NB. The first three have coarse-grained hotness measurement and fail to distinguish differences in access frequencies



(a) Collected CITs. **(b)** CIT threshold history. **(c)** Rate limit history. **(d)** Sensitivity analysis.

Figure 10. The parameter tuning effectiveness and sensitivity analysis.

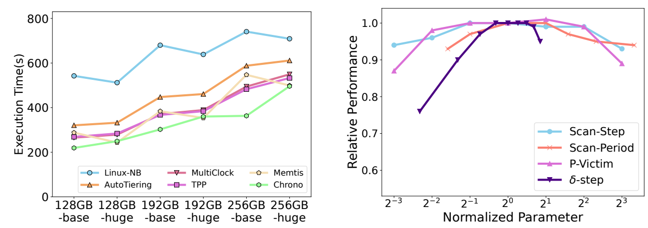
at sub-second granularity. Memtis is a process-level solution which does not distinguish the different levels of hotness between processes. With Chrono, the hottest instance gets nearly all pages allocated in DRAM, while the cold ones gradually release their DRAM pages and consume more NVM. It shows that the hot/cold page identification and migration schemes of Chrono are fine-grained and effective.

5.1.4 Parameter tuning and Sensitivity test.

To demonstrate the statistical correlation between CIT and page access frequency, we collect CIT values through the address space of a Pmbench process with a Gaussian memory access pattern. Figure 10a shows the CIT distribution at different addresses, and the profiled access probability density function (PDF) within the address space. The dashed line shows the mean value of the access time interval (in log-scale), which is negatively correlated with the access probability. CIT values are distributed around the mean access interval, which indicates that the CIT correctly reflects the page access frequency.

To verify the effectiveness of the adaptive parameter tuning, we track the CIT threshold and rate limit. The results are shown in Figure 10b, 10c. We find that the CIT threshold converges to about 200 ms which is close to the access interval upper bound of the hottest 25% pages. Given that the fast-tier memory consists of 25% of the memory capacity, we conclude that the automatically tuned CIT threshold classifies hot/cold pages with high precision. It is worth noting that the 200 ms CIT threshold represents a 300 access/minute frequency, surpassing the measurement ability of Auto-Tiering and TPP. We also find that the rate limit decreases and turns stable during the execution time. The page placement needs more intense adjustment at the beginning of execution, where Chrono discovers it and adopts an aggressive migration strategy. After a long-term page migration during execution, the hot and cold page distribution tends to be optimal thus Chrono adopts a lower and stable migration rate.

We further conduct the sensitivity analysis to other parameters by changing their values and observe the performance change. Results are shown in Figure 10d. The scan-step parameter has impact on the page-fault rate, where a larger value leads to higher kernel-level overhead and lower throughput, and a smaller scan-period has a similar impact.



(a) Execution time. **(b)** Sensitivity.

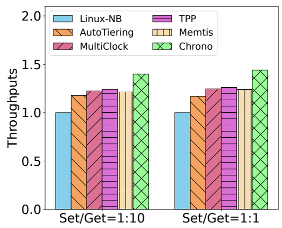
Figure 11. Graph 500 macrobenchmark: (a) execution time with various working set sizes and page granularities, and (b) sensitivity analysis.

CIT-based measurement scheme decouples the frequency granularity from the scan-step and scan-period, such that our system is able to set moderate values for these two parameters. The P-victim parameter controls the sampling ratio in DCSC, where a too small sample set is not representative of the overall distribution, and a too large one leads to increased overhead. For the δ-step parameter used in the semi-auto tuning scheme, a smaller value leads to slower convergence procedure and decreased performance.

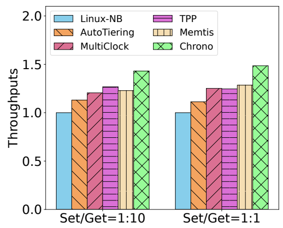
5.2 Macrobenchmark: Graph500

The Graph500 benchmark is used to analysis the performance of breadth first search (BFS) and single-source shortest path (SSSP) algorithms on a weighted, undirected graph [14]. It employs a scalable data generator through which we control the memory consumptions.

We run multi-processes Graph500 test with different working set sizes from 128 GB to 256 GB and enforce a base-page setting for all the systems. The speedup results are shown in Figure 11a. We find that under different memory pressure, Chrono has better speedup ratios compared to Auto-Tiering, Multi-Clock, TPP, and Memtis. It outperforms the Linux-NB by 2.49×, 2.29×, and 2.05× under different working set sizes. The main reason is that the graph searching algorithm produces hot regions following the various edge degree distribution, of which the hotter items and the colder items have mild access frequency difference. The methods based on page-fault counters fail to identify the real hot pages from the warm pages, because their inadequate frequency



(a) Memcached.



(b) Redis.

Figure 12. In-memory database application throughput.

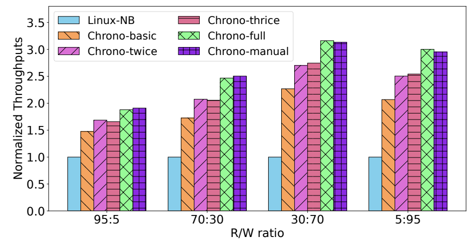


Figure 13. Design choice analysis.

resolution leads to a poor ability to distinguish the hotness borderline. Memtis, a PEBS-based solution, achieves performance improvements similar to the LRU-based methods, as it has limited sampling capacity to trace the large amount of base pages. Chrono is able to measure frequency precisely and migrate the hotter items stably to the fast-tier.

We also conduct the same experiments with huge-page settings, and show the results in Figure 11a. The huge-page system with Linux-NB has a 8% performance gain compared to its base-page counterpart, as it mainly benefits from the reduced page-fault handling overhead. Memtis has a significant performance gain under huge-page settings, and outperforms Chrono by 1.03×, but is slower than Chrono under base-page settings due to the hot region bloat issue. Chrono has slightly higher system overhead under huge-page settings, though it still outperforms Linux-NB by 2.06× because it includes an adaptive hot/cold page identification strategy with respect to different page sizes.

To analyze the sensitivity of the system parameters on the Graph500 benchmark, we conduct the experiments with different parameter values and show the results in Figure 11b. Similarly, the scan-step, scan-period and the P-victim have impact on the page-fault rate, and the δ-step has impact on the convergence speed of the semi-auto tuning scheme. With all parameters initialized in a reasonable range centered on the default value, Chrono is able to maintain stable performance with different settings, which indicates that our CIT-based measurement scheme and DCSC approach are adaptive to different system environments.

5.3 Applications: Memcached and Redis

We use the popular in-memory database applications, Memcached and Redis, to evaluate the tiered memory systems. For key-value generation and performance statistics, we use the standard benchmark named memtier-benchmarks [65].

We construct a key-value store including 500 M items, which consumes 160 GB memory. To maintain the same initial page distribution, we start the database and perform sequential initialization on all the items, with Gaussian distributed SET/GET ops for performance statistics. Results are collected as the normalized throughput values shown in Figure 12. As the results show, Chrono generally provides better overall throughput on Memcached and Redis. These in-memory databases generate massive page access operations, whose active working set sizes are larger than the DRAM space. The systems with coarse-grained frequency measurement schemes are not able to distinguish real hot pages from the slow tier under memory-intensive environment. For Memtis, the huge-page system leads to memory bloat with an average bloat rate of 145%, such that the fast-tier memory pages are not fully utilized, and the actual hot region is significantly smaller than the identified one. Chrono manages to perform meticulous hotness measurement and flexible page migration, such that it identifies and relocates hot and cold pages precisely.

5.4 Design Choice Analysis

To understand the benefit of different design parts, we first implement Chrono-*basic* which adopts the one-round CIT filtering, and the semi-auto parameter tuning scheme (with a 120 MB/s rate limit, the stable state in adaptive tuning). To evaluate the candidate filtering design, we implement Chrono-*twice* and Chrono-*thrice*, which use a 2-round and a 3-round scan for hot page selection respectively. To evaluate the parameter tuning scheme, we implement Chrono-*full*, which adopts 2-round candidate filtering and DCSC schemes simultaneously. To show the potential of the low-overhead semi-auto tuning, we also consider a Chrono-*manual* configuration, which is based in the semi-auto tuning scheme and for which we configure the rate limit parameters as the average of adaptive tuning results per minute.

We use Pmbench for the evaluation and we show the results in Figure 13. The improvement from Linux-NB to Chrono-*basic* shows the benefit of timer-based measurement scheme, indicating that timers provide more precise frequency resolution to classify hot/cold pages better. Comparing Chrono-*twice* with Chrono-*basic*, we conclude that the filtering scheme improves Chrono by reducing the measurement deviation and improving the hot page migration efficiency. The similar performance of Chrono-*thrice* and Chrono-*twice* indicates that a 2-round selection is enough, as more rounds have marginal impact on performance while consuming more resources. The improvement between Chrono-*twice* and Chrono-*full* comes from the DCSC-based tuning scheme, which adjusts key parameters adaptively and further reduces redundant page migration. Last but not least, Chrono-*manual* exhibits a comparable performance, with slightly lower hot page identification efficiency and more user-level execution time. It shows that the low-overhead semi-auto tuning is viable when ideal manual configuration is provided.

6 Related Work

There have been many studies dedicated to heterogeneous memory system design. Existing works focus on distinct important fields including memory expansion [3, 24, 59, 62, 87], disaggregation and pooling [5, 22, 30, 51, 76, 83], and next-generation storage computing technologies [10, 15, 32]. They pay more attention to the system functionality requirements and interface construction methods. Our work focuses on optimizing such heterogeneous memory architectures, where memory hotness identification and page migration matter.

The topic of placement optimizations for heterogeneous memory architectures has been extensively studied. Apart from the studies we analyzed in this paper, there are some user-level page classification and migration researches [16, 45, 58, 68]. User-level management is able to utilize more specific application statistics but loses transparency, while kernel-level tiered memory management has inherent limitations caused by restricted resource. Meanwhile, some researchers also reconsidered the architecture of the memory hierarchy, including optimizations to the existing caching paradigm [41], and the design of non-exclusive memory tiers [81]. These works are orthogonal to our research, as they focus on the memory hierarchy organization principles and the page accessing mechanisms.

There are also some studies focusing on page migration optimizations [44, 71, 79, 85]. They encompass advanced mechanisms to accelerate the page migration procedure in the kernel space, leaving the page migration criterion unchanged. Such optimization methods include symmetric migration, batched migration, huge-page-aware migration, etc. These optimized migration techniques are orthogonal to the contributions regarding hotness measurement and hot/cold identification.

7 Conclusion

In this paper, we present Chrono, an OS-level tiered memory management system that precisely captures page access frequencies in different tiers and migrates hot/cold pages efficiently. We propose to use CIT as a meticulous page hotness measurement, enhancing the kernel-level access frequency assessment ability. We design adaptive parameter tuning methods to enable flexible hot page migration, combining them with a proactive demotion scheme to stabilize overall performance. We implement Chrono and evaluate it using various memory-intensive benchmarks and applications. Experimental results show that Chrono outperforms state-of-the-art tiering systems by a large margin.

Acknowledgments

We thank our shepherd Renaud Lachaize and the anonymous reviewers for their valuable feedback and insightful suggestions. This work is supported by National Key Research and Development Program of China (Grant No. 2023YFB4502902), National Natural Science Foundation of China (NSFC) (Grant No. 62332012, 62227809, 62302290), the Fundamental Research Funds for the Central Universities, Shanghai Municipal Science and Technology Major Project (Grant No. 2021SHZDZX0102), Natural Science Foundation of Shanghai (Grant No. 22ZR1435400), and Huawei Innovation Research Plan.

References

[1] Neha Agarwal and Thomas F Wenisch. Thermostat: Application-transparent page management for two-tiered main memory. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 631–644, 2017.

[2] A Agawal and Anoop Gupta. Memory-reference characteristics of multiprocessor applications under mach. In *Proceedings of the 1988 ACM SIGMETRICS conference on Measurement and modeling of computer systems*, pages 215–225, 1988.

[3] Minseon Ahn, Andrew Chang, Donghun Lee, Jongmin Gim, Jungmin Kim, Jaemin Jung, Oliver Rebholz, Vincent Pham, Krishna Malladi, and Yang Seok Ki. Enabling cxl memory expansion for in-memory database management systems. In *Data Management on New Hardware*, pages 1–5. 2022.

[4] Soramichi Akiyama and Takahiro Hirofuchi. Quantitative evaluation of intel pebs overhead for online system-noise analysis. In *Proceedings of the 7th International Workshop on Runtime and Operating Systems for Supercomputers ROSS 2017*, pages 1–8, 2017.

[5] Hasan Al Maruf and Mosharaf Chowdhury. Memory disaggregation: advances and open challenges. *ACM SIGOPS Operating Systems Review*, 57(1):29–37, 2023.

[6] andikleen. Intel pmu profiling tools, 2024. https://github.com/andikleen/pmu-tools.

[7] Brad Benton. Ccix, gen-z, opencapi: Overview & comparison. In *OpenFabrics Workshop*, 2017.

[8] Shai Bergman, Priyank Faldu, Boris Grot, Lluís Vilanova, and Mark Silberstein. Reconsidering os memory optimizations in the presence of disaggregated memory. In *Proceedings of the 2022 ACM SIGPLAN International Symposium on Memory Management*, pages 1–14, 2022.

[9] Georgios Bitzes and Andrzej Nowak. The overhead of profiling using pmu hardware counters. *CERN openlab report*, pages 1–16, 2014.

[10] Anthony M Cabrera, Aaron R Young, and Jeffrey S Vetter. Design and analysis of cxl performance models for tightly-coupled heterogeneous computing. In *Proceedings of the 1st International Workshop on Extreme Heterogeneity Solutions*, pages 1–6, 2022.

[11] George Casella and Roger L Berger. *Statistical inference*. Cengage Learning, 2021.

[12] An Chen. A review of emerging non-volatile memory (nvm) technologies and applications. *Solid-State Electronics*, 125:25–38, 2016.

[13] Cheng Chen, Jun Yang, Mian Lu, Taize Wang, Zhao Zheng, Yuqiang Chen, Wenyuan Dai, Bingsheng He, Weng-Fai Wong, Guoan Wu, et al. Optimizing in-memory database engine for ai-powered on-line

decision augmentation using persistent memory. *Proceedings of the VLDB Endowment*, 14(5):799–812, 2021.

[14] Graph 500 Steering Committee. Graph 500 benchmark specification, 2022. https://graph500.org/?page_id=12.

[15] Thomas M Coughlin and William R Tonti. Computing nearer to data. *Computer*, 55(7):82–87, 2022.

[16] Thaleia Dimitra Doudali, Sergey Blagodurov, Abhinav Vishnu, Sudhanva Gurumurthi, and Ada Gavrilovska. Kleio: A hybrid memory page scheduler with machine intelligence. In *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*, pages 37–48, 2019.

[17] Subramanya R Dulloor, Amitabha Roy, Zheguang Zhao, Narayanan Sundaram, Nadathur Satish, Rajesh Sankaran, Jeff Jackson, and Karsten Schwan. Data tiering in heterogeneous memory systems. In *Proceedings of the Eleventh European Conference on Computer Systems*, pages 1–16, 2016.

[18] Padmapriya Duraisamy, Wei Xu, Scott Hare, Ravi Rajwar, David Culler, Zhiyi Xu, Jianing Fan, Christopher Kennelly, Bill McCloskey, Danijela Mijailovic, et al. 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*, pages 727–741, 2023.

[19] Assaf Eisenman, Darryl Gardner, Islam AbdelRahman, Jens Axboe, Siying Dong, Kim Hazelwood, Chris Petersen, Asaf Cidon, and Sachin Katti. Reducing dram footprint with nvm in facebook. In *Proceedings of the Thirteenth EuroSys Conference*, pages 1–13, 2018.

[20] Amir Gholami, Zhewei Yao, Sehoon Kim, Coleman Hooper, Michael W Mahoney, and Kurt Keutzer. Ai and memory wall. *IEEE Micro*, 2024.

[21] Gurbinder Gill, Roshan Dathathri, Loc Hoang, Ramesh Peri, and Keshav Pingali. Single machine graph analytics on massive datasets using intel optane dc persistent memory. *Proceedings of the VLDB Endowment*, 13(8).

[22] Donghyun Gouk, Sangwon Lee, Miryeong Kwon, and Myoungsoo Jung. Direct access, high-performance memory disaggregation with directcxl. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, pages 287–294, 2022.

[23] Pmem Google Group. Cxl1.1 and memory tiering, 2022. https://groups.google.com/g/pmem/c/iXNTTIxI3j8.

[24] Zhiyuan Guo, Zijian He, and Yiying Zhang. Mira: A program-behavior-guided far memory system. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 692–708, 2023.

[25] Mohammad Hossein Hajkazemi, Mohammad Khavari Tavana, and Houman Homayoun. Wide i/o or lpddr? exploration and analysis of performance, power and temperature trade-offs of emerging dram technologies in embedded mpsocs. In *2015 33rd IEEE International Conference on Computer Design (ICCD)*, pages 62–69. IEEE, 2015.

[26] Kostas Hatalis, Despina Christou, Joshua Myers, Steven Jones, Keith Lambert, Adam Amos-Binks, Zohreh Dannenhauer, and Dustin Dannenhauer. Memory matters: The need to improve long-term memory in llm-agents. In *Proceedings of the AAAI Symposium Series*, volume 2, pages 277–280, 2023.

[27] Taekyung Heo, Yang Wang, Wei Cui, Jaehyuk Huh, and Lintao Zhang. Adaptive page migration policy with huge pages in tiered memory systems. *IEEE Transactions on Computers*, 71(1):53–68, 2020.

[28] Seokbin Hong, Won-Ok Kwon, and Myeong-Hoon Oh. Hardware implementation and analysis of gen-z protocol for memory-centric architecture. *IEEE Access*, 8:127244–127253, 2020.

[29] C Intel. Intel® 64 and ia-32 architectures software developer's manual volume 3 (3a, 3b, 3c & 3d): System programming guide. *Denver,[2006]*, 2023.

[30] Junhyeok Jang, Hanjin Choi, Hanyeoreum Bae, Seungjun Lee, Miryeok Kwon, and Myoungsoo Jung. CXL-ANNS: Software-Hardware collaborative memory disaggregation and computation for Billion-Scale approximate nearest neighbor search. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pages 585–600, 2023.

[31] Hongshin Jun, Jinhee Cho, Kangseol Lee, Ho-Young Son, Kwiwook Kim, Hanho Jin, and Keith Kim. Hbm (high bandwidth memory) dram technology and architecture. In *2017 IEEE International Memory Workshop (IMW)*, pages 1–4. IEEE, 2017.

[32] Myoungsoo Jung. Hello bytes, bye blocks: Pcie storage meets compute express link for memory expansion (cxl-ssd). In *Proceedings of the 14th ACM Workshop on Hot Topics in Storage and File Systems*, pages 45–51, 2022.

[33] Sudarsun Kannan, Ada Gavrilovska, Vishal Gupta, and Karsten Schwan. Heteroos: Os design for heterogeneous memory management in datacenter. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 521–534, 2017.

[34] Hiwot Tadese Kassa, Jason Akers, Mrinmoy Ghosh, Zhichao Cao, Vaibhav Gogte, and Ronald Dreslinski. Improving performance of flash based Key-Value stores using storage class memory as a volatile memory extension. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 821–837, 2021.

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

[36] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626, 2023.

[37] Youngjin Kwon, Hangchen Yu, Simon Peter, Christopher J Rossbach, and Emmett Witchel. Ingens: Huge page support for the os and hypervisor. *ACM SIGOPS Operating Systems Review*, 51(1):83–93, 2017.

[38] Andres Lagar-Cavilla, Junwhan Ahn, Suleiman Souhlal, Neha Agarwal, Radoslaw Burny, Shakeel Butt, Jichuan Chang, Ashwin Chaugule, Nan Deng, Junaid Shahid, et al. Software-defined far memory in warehouse-scale computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 317–330, 2019.

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

[40] Erich Leo Lehmann and Henry Scheffé. Completeness, similar regions, and unbiased estimation—part ii. In *Selected Works of EL Lehmann*, pages 269–286. Springer, 2012.

[41] Baptiste Lepers and Willy Zwaenepoel. Johnny cache: the end of DRAM cache conflicts (in tiered main memory systems). In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 519–534, 2023.

[42] Chuandong Li, Sai Sha, Yangqing Zeng, Xiran Yang, Yingwei Luo, Xiaolin Wang, Zhenlin Wang, and Diyu Zhou. Taming hot bloat under virtualization with HUGESCOPE. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pages 999–1012, 2024.

[43] Huaicheng Li, Daniel S Berger, Lisa Hsu, Daniel Ernst, Pantea Zardoshti, Stanko Novakovic, Monish Shah, Samir Rajadnya, Scott Lee, Ishwar Agarwal, et al. Pond: Cxl-based memory pooling systems for cloud platforms. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 574–587, 2023.

[44] Felix Xiaozhu Lin and Xu Liu. Memif: Towards programming heterogeneous memory asynchronously. *ACM SIGPLAN Notices*, 51(4):369–383, 2016.

[45] Haikun Liu, Renshan Liu, Xiaofei Liao, Hai Jin, Bingsheng He, and Yu Zhang. Object-level memory allocation and migration in hybrid memory systems. *IEEE Transactions on Computers*, 69(9):1401–1413, 2020.

[46] LKP-test. Lkp-tests - a linux* kernel performance test and analysis tool, 2021. https://github.com/intel/lkp-tests.

[47] Chen Luo and Michael J Carey. Breaking down memory walls: adaptive memory management in lsm-based storage systems. *Proceedings of the VLDB Endowment*, 14(3):241–254, 2020.

[48] Adnan Maruf, Ashikee Ghosh, Janki Bhimani, Daniel Campello, Andy Rudoff, and Raju Rangaswami. Multi-clock: Dynamic tiering for hybrid memory systems. In *HPCA*, pages 925–937, 2022.

[49] Hasan Al Maruf, Hao Wang, Abhishek Dhanotia, Johannes Weiner, Niket Agarwal, Pallab Bhattacharya, Chris Petersen, Mosharaf Chowdhury, Shobhit Kanaujia, and Prakash Chauhan. Tpp: Transparent page placement for cxl-enabled tiered-memory. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 742–755, 2023.

[50] Syed Akbar Mehdi, Deukyeon Hwang, Simon Peter, and Lorenzo Alvisi. ScaleDB: A scalable, asynchronous In-Memory database. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 361–376, 2023.

[51] Pankaj Mehra and Tom Coughlin. Taming memory with disaggregation. *Computer*, 55(9):94–98, 2022.

[52] Gorman Mel, Molnar Ingo, and Torvalds Linus. Mm: numa: Document automatic numa balancing sysctls, 2021. https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?h=v5.15&id=10fc05d0e551146ad6feb0ab8902d28a2d3c5624.

[53] Memcached. Free and open source, high-performance, distributed memory object caching system, 2022. https://memcached.org/.

[54] Theodore Michailidis, Alex Delis, and Mema Roussopoulos. Mega: Overcoming traditional problems with os huge page management. In *Proceedings of the 12th ACM International Conference on Systems and Storage*, pages 121–131, 2019.

[55] Richard C Murphy, Kyle B Wheeler, Brian W Barrett, and James A Ang. Introducing the graph 500. *Cray Users Group (CUG)*, 19:45–74, 2010.

[56] Onur Mutlu, Saugata Ghose, Juan Gómez-Luna, and Rachata Ausavarungnirun. A modern primer on processing in memory. In *Emerging computing: from devices to systems: looking beyond Moore and Von Neumann*, pages 171–243. Springer, 2022.

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

[58] Dimitrios S Nikolopoulos, Theodore S Papatheodorou, Constantine D Polychronopoulos, Jesús Labarta, and Eduard Ayguadé. User-level dynamic page migration for multiprogrammed shared-memory multiprocessors. In *Proceedings 2000 International Conference on Parallel Processing*, pages 95–103. IEEE, 2000.

[59] Geraldo F Oliveira, Saugata Ghose, Juan Gómez-Luna, Amirali Boroumand, Alexis Savery, Sonny Rao, Salman Qazi, Gwendal Grignou, Rahul Thakur, Eric Shiu, et al. Extending memory capacity in modern consumer systems with emerging non-volatile memory: Experimental analysis and characterization using the intel optane ssd. *IEEE Access*, 2023.

[60] OpenSUSE. Automatic non-uniform memory access (numa) balancing, 2022. https://doc.opensuse.org/documentation/leap/tuning/html/book-tuning/cha-tuning-numactl.html.

[61] Mark Oskin and Gabriel H Loh. A software-managed approach to die-stacked dram. In *2015 International Conference on Parallel Architecture and Compilation (PACT)*, pages 188–200. IEEE, 2015.

[62] SJ Park, H Kim, KS Kim, J So, J Ahn, WJ Lee, D Kim, YJ Kim, J Seok, JG Lee, et al. Scaling of memory performance and capacity with cxl memory expander. In *2022 IEEE Hot Chips 34 Symposium (HCS)*, pages 1–27. IEEE Computer Society, 2022.

[63] Pmem.io. Intel optane persistent memory module provisioning., 2022. https://docs.pmem.io/ipmctl-user-guide/provisioning.

[64] Amanda Raybuck, Tim Stamler, Wei Zhang, Mattan Erez, and Simon Peter. Hemem: Scalable tiered memory management for big data applications and real nvm. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pages 392–407, 2021.

[65] Redis. Memtier benchmark: A high-throughput benchmarking tool for redis and memcached, 2022. https://github.com/redislabs/memtier_benchmark.

[66] Redis. The open source, in-memory data store used by millions of developers as a database, cache, streaming engine, and message broker., 2022. https://redis.io/.

[67] Jie Ren, Dong Xu, Junhee Ryu, Kwangsik Shin, Daewoo Kim, and Dong Li. Mtm: Rethinking memory profiling and migration for multi-tiered large memory. In *Proceedings of the Nineteenth European Conference on Computer Systems*, pages 803–817, 2024.

[68] Jee Ho Ryoo, Lizy K John, and Arkaprava Basu. A case for granularity aware page migration. In *Proceedings of the 2018 International Conference on Supercomputing*, pages 352–362, 2018.

[69] Muhammad Aditya Sasongko, Milind Chabbi, Paul HJ Kelly, and Didem Unat. Precise event sampling on amd versus intel: Quantitative and qualitative comparison. *IEEE Transactions on Parallel and Distributed Systems*, 34(5):1594–1608, 2023.

[70] Abu Sebastian, Manuel Le Gallo, Riduan Khaddam-Aljameh, and Evangelos Eleftheriou. Memory devices and applications for in-memory computing. *Nature nanotechnology*, 15(7):529–544, 2020.

[71] Vivek Seshadri, Yoongu Kim, Chris Fallin, Donghyuk Lee, Rachata Ausavarungnirun, Gennady Pekhimenko, Yixin Luo, Onur Mutlu, Phillip B Gibbons, Michael A Kozuch, et al. Rowclone: Fast and energy-efficient in-dram bulk data copy and initialization. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 185–197, 2013.

[72] Sai Sha, Chuandong Li, Xiaolin Wang, Zhenlin Wang, and Yingwei Luo. Hardware-software collaborative tiered-memory management framework for virtualization. *ACM Transactions on Computer Systems*, 42(1-2):1–32, 2024.

[73] Anil Shanbhag, Nesime Tatbul, David Cohen, and Samuel Madden. Large-scale in-memory analytics on intel® optane™ dc persistent memory. In *Proceedings of the 16th International Workshop on Data Management on New Hardware*, pages 1–8, 2020.

[74] Debendra Das Sharma. Compute express link®: An open industry-standard interconnect enabling heterogeneous data-centric computing. In *2022 IEEE Symposium on High-Performance Interconnects (HOTI)*, pages 5–12. IEEE, 2022.

[75] Debendra Das Sharma and Siamak Tavallaei. Compute express link 2.0 white paper. *Tech. Rep.*, 2020.

[76] Joonseop Sim, Soohong Ahn, Taeyoung Ahn, Seungyong Lee, Myunghyun Rhee, Jooyoung Kim, Kwangsik Shin, Donguk Moon, Euiseok Kim, and Kyoung Park. Computational cxl-memory solution for accelerating memory-intensive applications. *IEEE Computer Architecture Letters*, 2022.

[77] Sajjad Tamimi, Florian Stock, Andreas Koch, Arthur Bernhardt, and Ilia Petrov. An evaluation of using ccix for cache-coherent host-fpga interfacing. In *2022 IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 1–9. IEEE, 2022.

[78] Alexander Van Renen, Lukas Vogel, Viktor Leis, Thomas Neumann, and Alfons Kemper. Persistent memory i/o primitives. In *Proceedings of the 15th International Workshop on Data Management on New Hardware*, pages 1–7, 2019.

[79] Hao Wang, Jie Zhang, Sharmila Shridhar, Gieseo Park, Myoungsoo Jung, and Nam Sung Kim. Duang: Fast and lightweight page migration in asymmetric memory systems. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 481–493. IEEE, 2016.

[80] Michèle Weiland, Holger Brunst, Tiago Quintino, Nick Johnson, Olivier Iffrig, Simon Smart, Christian Herold, Antonino Bonanni, Adrian Jackson, and Mark Parsons. An early evaluation of intel's optane dc persistent memory module and its impact on high-performance scientific applications. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*, pages 1–19, 2019.

[81] Lingfeng Xiang, Zhen Lin, Weishu Deng, Hui Lu, Jia Rao, Yifan Yuan, and Ren Wang. Nomad: Non-Exclusive memory tiering via transactional page migration. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 19–35, 2024.

[82] Lingfeng Xiang, Xingsheng Zhao, Jia Rao, Song Jiang, and Hong Jiang. Characterizing the performance of intel optane persistent memory: a close look at its on-dimm buffering. In *Proceedings of the Seventeenth European Conference on Computer Systems*, pages 488–505, 2022.

[83] Chuhao Xu, Yiyu Liu, Zijun Li, Quan Chen, Han Zhao, Deze Zeng, Qian Peng, Xueqi Wu, Haifeng Zhao, Senbo Fu, et al. Faasmem: Improving memory efficiency of serverless computing with memory pool architecture. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 331–348, 2024.

[84] Dong Xu, Junhee Ryu, Kwangsik Shin, Pengfei Su, and Dong Li. FlexMem: Adaptive page profiling and migration for tiered memory. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pages 817–833, 2024.

[85] Zi Yan, Daniel Lustig, David Nellans, and Abhishek Bhattacharjee. Nimble page management for tiered memory systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 331–345, 2019.

[86] Jisoo Yang and Julian Seymour. Pmbench: A micro-benchmark for profiling paging performance on a system with low-latency ssds. In *Information Technology-New Generations*, pages 627–633. Springer, 2018.

[87] Shao-Peng Yang, Minjae Kim, Sanghyun Nam, Juhyung Park, Jinyong Choi, Eyee Hyun Nam, Eunji Lee, Sungjin Lee, and Bryan S Kim. Overcoming the memory wall with CXL-Enabled SSDs. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pages 601–617, 2023.

Appendix A Artifact

A.1 Abstract

The artifact of this paper contains the prototype implementation of Chrono, a novel tiered memory system that provides fine-grained hotness measurement and flexible page migration with low overhead. The artifact is based on the Linux kernel v5.18.0, with 1.9k SLOC code changes. It also includes the compiling and installing instructions and part of the raw experimental data used in the evaluation section.

A.2 Description & Requirements

How to access. You can access the source code of Chrono at this **GitHub repository**: https://github.com/SJTU-DDST/chrono-project. The artifact is released under the GPLv2 license, which follows the license of the Linux kernel. We have also uploaded the artifact to Zenodo, and the DOI is https://doi.org/10.5281/zenodo.14875828.

Hardware requirements. To download and compile the artifact, you need a machine with at least 8GB of free disk space and 4GB of DRAM. Moreover, we highly recommend using a server with physical Intel Optane DC Persistent Memory (PMem) modules to better evaluate the performance of Chrono on a tiered memory system. A server with 2 Intel Xeon Gold 6348 CPUs and 128GB of DRAM and 256GB of PMem is used in our evaluation.

Software dependencies. Any Linux distribution with kernel v5.15 or later is theoretically compatible with Chrono. Moreover, we recommend using Ubuntu 20.04 LTS as the host system, which make the compiling process and installation of dependencies easier.

Additionally, the following tools are required to construct a tiered memory system:

- **ndctl** is a utility for managing the Non-Volatile Memory Device Control and NVDIMM subsystem.
- **ipmctl** is a utility for managing Intel Optane DC Persistent Memory modules.
- **daxctl** is a utility for managing Device-DAX devices.

You can check the installation instructions by referring to the official documentation at https://docs.pmem.io/.

Benchmarks. The suggested benchmarks for evaluating Chrono are listed in the evaluation section of the paper. You can download the source code of these benchmarks from the following links:

- **PmBench**: github repo at https://github.com/blakecaldwell/pmbench
- **Graph500**: github repo at https://github.com/graph500/graph500
- **Memcached**: github repo at https://github.com/memcached/memcached

A.3 Set-up

The compilation and installation of Chrono are almost identical to the process of compiling the Linux kernel.

Step 1: Download the source code. You can download the source code of Chrono from the GitHub repository.

```
git clone [our repository]
cd chrono-project
```

Step 2: Compile the kernel. We provide a script to compile the Chrono kernel.

```
bash compile-install.sh
```

Make sure that current user has sudo privilege, and the .config file is successfully saved during the menuconfig step.

Step 3: Reboot the system. Before rebooting, make sure that the kernel is correctly installed, by checking the boot directory:

```
ls /boot
```

While rebooting, make sure to select the correct kernel version from the boot menu.

Step 4: Install PMem Tools. Install the NDCTL, IPMCTL, and DAXCTL tools to manage the PMem modules.

Step 5: Configure Tiered Memory. The persistent memory is configured as DAX devices by default. To make the persistent memory as system RAM

```
daxctl reconfigure-device -m system-ram daxX.Y
```

Check our README document in github repo for detailed instructions, where `daxX.Y` should be replaced with the actual device name.

Step 6: Enable Chrono. To enable the Chrono features:

```
echo 1 > /sys/kernel/mm/numa/demotion_enabled
echo 2 > /proc/sys/kernel/numa_balancing
```

Then check our README document in github repo for more detailed instructions to adjust the parameters of Chrono.

A.4 Evaluation workflow

We mainly show an example of using PmBench to evaluate Chrono in this section.

Step 1: Install LKP tool. The LKP (Linux Kernel Performance) is a benchmarking tool that can be used to evaluate the performance of the Linux kernel, which is available at https://github.com/intel/lkp-tests.git.

Step 2: Run PmBench. We provide samples in test directory.

```
cd test/pmbench
sudo lkp run ./60G-4G-64-tiering.yaml
```

You should be able to see the benchmark results as json files in the test/pmbench directory. Our repo also includes the raw logs for a baseline kernel. More details about the json results can be found in the LKP documentation.

We suggest running the benchmark using `numactl` and `taskset` to get stable results. More detailed instructions can be found in our documentation.

Appendix B Theoretical Analysis

Here we provide the theoretical analysis supporting our candidate filtering scheme.

B.1 Lower measurement variance.

If we perform multiple rounds of scan and get multiple CIT values of a page, we need to estimate the access period accurately. A naive choice is to calculate the mean-value as an estimation. Our candidate filtering design is equivalent to a maximum-value estimator. Here we show that the maximum value is a better choice when compared to the mean value. Specifically, the former has a lower variance.

Assume that we are measuring a page with inherent access period T_0. During the run-time we scan it n times and get a series of CIT values, denoted as $t_1, t_2, \ldots, t_n$. Because the scan events occur independently of the application execution, we have that t_i are i.i.d. following a uniform distribution:

$$t_i \sim \mathbf{U}[0, T_0]. \tag{1}$$

For an mean-value estimator, it uses

$$\widetilde{T_1} = \frac{2}{n}\sum_{i=1}^{n} t_i \tag{2}$$

to estimate T_0. We calculate the mean and variance of $\widetilde{T_1}$ as:

$$\begin{aligned} \mathsf{E}(\widetilde{T_1}) &= \frac{2}{n}\sum_{i=1}^{n}\mathsf{E}(t_i) = \frac{2}{n}\cdot n\cdot\frac{T_0}{2} = T_0, \\ \mathsf{D}(\widetilde{T_1}) &= \frac{4}{n^2}\sum_{i=1}^{n}\mathsf{D}(t_i) = \frac{4}{n^2}\cdot n\cdot\frac{T_0^2}{12} = \frac{T_0^2}{3n}. \end{aligned} \tag{3}$$

Meanwhile, for a maximum-value estimator, it uses

$$\widetilde{T_2} = \frac{n+1}{n}\max_i t_i \tag{4}$$

to estimate T_0. Denote the variable $\max_i t_i$ as M, and we have the cumulative distribution function of M is

$$F_M(m) = P(M \le m) = \left(\frac{m}{T_0}\right)^n. \tag{5}$$

We then calculate the mean and variance of $\widetilde{T_2}$ as:

$$\begin{aligned} \mathsf{E}(\widetilde{T_2}) &= \frac{n+1}{n}\int_{m=0}^{T_0} m\cdot F_M'(m)\,dm = T_0, \\ \mathsf{D}(\widetilde{T_2}) &= \frac{(n+1)^2}{n^2}\left(\int_{m=0}^{T_0} m^2\cdot F_M'(m)\,dm - \mathsf{E}^2(M)\right) \\ &= \frac{1}{n(n+2)}T_0^2. \end{aligned} \tag{6}$$

Comparing equation 3 and equation 6, we conclude that the maximum-value estimator has lower variance. Actually, we are able to prove that the maximum-value estimator is the minimum variance unbiased estimator in our case, following the lehmann-scheffe theorem [11, 40].

B.2 Higher selection efficiency.

When we consider cold pages (i.e., whose access period is greater than the CIT threshold), they also have a chance to be measured as hot because of the randomness of CIT. A classification method is better if it ensures a higher real-hot-page ratio in the selected hot pages. On the other hand, the scan procedure is executed in kernel mode consuming CPU and memory. We can model the promotion efficiency by the real-hot-page ratio and the cost.

Denote all the pages as a set $\{pg_i\}$, where T_i is the access period of pg_i, and the CIT threshold is TH. The real hot page number is $N_h = |\{i | T_i < TH\}|$. If we adopt n-round scan, the probability of a page pg_i to be identified as hot page is

$$P_{hot}(pg_i) = \begin{cases} 1 & , T_i < TH \\ \left(\frac{TH}{T_i}\right)^n & , T_i \ge TH \end{cases}. \tag{7}$$

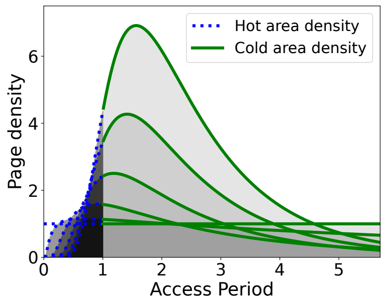


Figure B1. Image of function h, where α changes in {0.25, 0.3, 0.4, 0.6, 0.9, 1}.

Then we model the page hotness distribution using a cumulative function and its normalized density:

$$F(t) \triangleq |\{i|T_i < t\}|, \ where\ t \in (0, \infty).$$
$$f(x) = \frac{1}{N_h \cdot TH} F'(x), \ where\ x = \frac{t}{TH}. \tag{8}$$

So that the real-hot-page ratio is calculated as

$$R_f(n) = \frac{1}{1 + S_f(n)},$$
$$where\ \ S_f(n) = \int_{x=1}^{\infty} f(x) \left(\frac{1}{x}\right)^n dx. \tag{9}$$

Intuitively, $S_f(n)$ represents the number of miss-classified cold pages. Further taking the cost of the n-round scan into consideration, we define the hot page selection efficiency as:

$$E_f(n) = \frac{1}{n} R_f(n). \tag{10}$$

With equation 10, we can calculate the efficiency $E_f(n)$ for any given page hotness distribution f and round number n. The realistic distribution of access period should be bounded. Some existing work [2] has shown that the distribution is generally dense in the hot region, and sparse in the cold region.

We use a class of function $h(x, \alpha)$ in place of $f(x)$ to capture the feature, where h is defined as

$$h(x, \alpha) = \frac{1}{C_\alpha} \cdot x^{1-\frac{1}{\alpha}} \cdot \alpha^{\alpha x + \frac{1}{\alpha x}}, \ where\ 0 < \alpha \leq 1. \tag{11}$$

The C_α is a coefficient ensuring $\int_{x=0}^{1} h(x, \alpha)\, dx = 1$, required by the normalization property of $f(x)$. Figure B1 shows the image of function $h(x, \alpha)$ with some fixed α value between 0.25 and 1. The maximum of $h(x, \alpha)$ get higher value when α is smaller.

We first analyze the case $\alpha = 1$, where $h(x, \alpha)$ becomes a constant function $h(x) = 1$, indicating totally random page distribution over access period. Then the efficiency $E_h(n)$ is calculated as

$$E_h(n) = \frac{1}{n} \frac{1}{1 + \int_{x=1}^{\infty} (\frac{1}{x})^n\, dx} = \frac{n-1}{n^2}, \ \ n = 1, 2, 3 \ldots \tag{12}$$

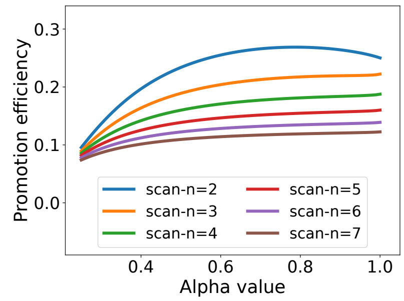


Figure B2. Numeric calculation results of $E_{h(x,\alpha)}(n)$, where n changes from 2 to 7.

It is obvious that $E_h(n)$ has a maximum value at $n = 2$ in equation 12. In another word, under a workload with random page access period distribution, the two-round filtering has the best efficiency.

We also analyze the case of $E_{h(x,\alpha)}(n)$ with various α values by the numeric integral method. Figure B2 plots the image of promotion efficiency v.s. α value. The results show that round number $n = 2$ generally gets a higher efficiency. It is worth noting that the format of h function and domain of α value are of our choice. If one settles the cold page density value as $h(x) = 10$ for $x > 1$, or $\alpha = 0.01$, they will get other results where a round number choice $n > 2$ has the best efficiency. However, those page distribution hypotheses are not realistic. We have also provided the comparison in the evaluation section, to show that two-round filtering is proper for real-world applications.