**Understanding and Optimizing Tiered Memory and Storage Systems**

A Dissertation Proposal presented

by

**Tyler Estro**

to

The Graduate School

in Partial Fulfillment of the

Requirements

for the Degree of

**Doctor of Philosophy**

in

**Computer Science**

Stony Brook University

**Technical Report FSL-26-01**

**January 2026**

**Stony Brook University**
The Graduate School

Tyler Estro

We, the thesis committee for the above candidate for the
degree of Doctor of Philosophy, hereby recommend
acceptance of this thesis proposal

**Erez Zadok - Dissertation Advisor**
**Professor, Computer Science Department**

**Mike Ferdman - Chairperson of Dissertation Proposal**
**Associate Professor, Computer Science Department**

**Anshul Gandhi**
**Associate Professor, Computer Science Department**

**Carl Waldspurger**
**Consultant, Carl Waldspurger Consulting**

Abstract of the Dissertation Proposal

**Understanding and Optimizing Tiered Memory and Storage Systems**

by

**Tyler Estro**

**Doctor of Philosophy**

in

**Computer Science**

Stony Brook University

**January 2026**

Tiered memory and storage systems have become increasingly complex, combining heterogeneous devices across multiple layers to balance performance and cost. They underpin critical infrastructure, from operating system page caches and database buffers to large-scale AI training pipelines. These systems expose a vast configuration space, as many configurable parameters and design choices compound across tiers. As hierarchies deepen and workloads diversify, identifying optimal configurations has become both essential and increasingly difficult. Compute Express Link (CXL) further amplifies this challenge by adding a shared, byte-addressable memory tier accessible by multiple hosts, expanding the tiering space and requiring new techniques to effectively exploit it.

This thesis proposal has three thrusts. In the first thrust, we examined cache analysis trends and found that many techniques focus primarily on performance, analyze tiers in isolation, and overlook cost-performance trade-offs. To address these limitations, we implemented a multi-tier cache simulator that evaluates configurations across a broad range of parameters. Simulations on real-world traces revealed that cost-aware, multi-tier analysis can overturn common assumptions about device choice, hierarchy complexity, and policy effectiveness.

In the second thrust, we developed methods for efficiently exploring large tiered storage and memory configuration spaces. We designed a multi-stage frame-

work for analyzing cache miss-ratio curves that combines hash-based sampling, curve simplification, and knee detection, including a novel multi-knee detection technique "Z-Method". To further support this exploration, we developed advanced interactive visualization techniques for analyzing large, high-dimensional configuration spaces. Lastly, we developed statistical performance models based on distribution fitting to better characterize and predict storage workload behavior.

In the third thrust, we propose two methods that apply CXL-based tiering to improve live virtual machine migration. The first method copies and transparently remaps guest memory to a shared memory device in a single pass, eliminating the need for dirty page tracking and retransmission. The second method partitions guest memory between DRAM and shared memory so that only DRAM-resident memory requires transfer. Together, these approaches aim to reduce migration time, blackout duration, and total data transferred compared to traditional migration methods.

It is our thesis that tiered memory and storage systems expose a vast configuration space with the potential for significant performance and cost optimizations. Fully realizing these benefits requires efficient techniques for both exploring and exploiting this space, particularly as the introduction of CXL shared memory adds new and powerful opportunities for tiering.

# Contents

# List of Algorithms

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Modern storage and memory hierarchies have become increasingly complex, forming multi-tier systems comprising heterogeneous devices in a wide range of topologies. From operating system page caches to large-scale distributed systems, tiering is employed to improve performance while reducing total cost of ownership (TCO). Cloud providers often dynamically reconfigure memory and storage hierarchies to meet service-level objectives (SLOs), balance loads, or in response to hardware faults [164, 72, 49, 27, 28]. The configuration space for tiering systems is vast, spanning device types, number of tiers, tier capacities, and tiering management policies, each with tunable parameters that influence performance and cost [38]. To navigate this space effectively, techniques must be efficient in (i) identifying high-quality configurations and (ii) adapting quickly to changing locality and reuse behavior as workload access patterns vary over time [121, 147, 116, 10, 24, 152]. Within this evolving landscape, Compute Express Link (CXL) introduces a cache-coherent interconnect that allows multiple hosts to access a shared, byte-addressable memory tier [45]. CXL significantly complicates the tiering space by introducing this fundamentally new paradigm while enabling novel techniques to exploit its potential.

This work addresses two distinct challenges of tiering. The first is the efficient exploration of the complex design space in tiered systems to identify optimal configurations. The second is the effective implementation of tiering to fully realize its benefits in modern systems. Addressing both challenges is essential for achieving high performance while maintaining cost efficiency.

In the first thrust, to address the challenge of efficiently exploring the complex design space in tiered systems, we examined cache analysis trends and identified key limitations in existing evaluation techniques [38]. We found that past tech-

niques often focused solely on performance, analyzed tiers in isolation, and ignored trade-offs such as cost versus performance. Current cache simulators were similarly restrictive, supporting only fixed hierarchies and limited metrics. We addressed these gaps by extending PyMimircache [155] to create a general $n$-level cache simulator capable of modeling arbitrary hierarchies, capturing both performance and cost, and enabling the analysis of trade-offs across multiple metrics. We uncovered surprising insights through simulations using real-world traces: (a) when total cost was held constant, lower-priced DRAM outperformed high-end DRAM by providing greater capacity; (b) aging SSDs reduced performance enough to favor simpler designs; and (c) write-back policies delivered up to 6$\times$ the throughput of write-through on identical hardware.

In the second thrust, we further addressed the challenge of efficient exploration by developing a framework to efficiently characterize multi-tier caches [37]. Miss-ratio curves (MRCs) are common analysis tools for evaluating cache performance, but generating them at fine granularity for every tier and configuration can be prohibitively expensive. Our approach focuses on identifying knee points in MRCs, where a small increase in cache size yields a disproportionately large drop in miss ratios, indicating promising candidates for further evaluation. The framework applies hash-based sampling [141], curve simplification [110], and adapts any single-knee detection algorithm to find multiple knee points [8, 117, 129]. In addition, we introduced a novel multi-knee detection algorithm, called Z-Method, that employs statistical outlier detection to identify points robustly and efficiently. We evaluated our framework using 106 diverse real-world workloads, and were able to reduce the number of simulations needed to find optimal two-tier hierarchies by 5.5$\times$ for ARC and 7.7$\times$ for LRU. We also applied our framework to seed the initial population of evolutionary algorithms that we used to optimize multi-tier cache configurations, achieving an overall convergence-acceleration rate of 34% across a broad range of configurations and datasets [37].

To further support exploration, we developed a set of advanced interactive visualizations designed to help analysts interpret large, high-dimensional configuration spaces. First, we created an interactive configuration explorer that allows users to examine how categorical design choices influence system behavior and to compare competing configuration families through guided visual analysis [132]. Second, we built a parallel-coordinate axes-reordering framework that enables users to uncover structural patterns and relationships across configuration parameters by automatically arranging axes to emphasize informative pairings [133]. Lastly, we introduced an empty-space analysis technique that identifies promising, previously unsampled configurations by detecting sparsely populated gaps in

the design space that may contain high-value alternatives [163].

In addition to these visualization tools, we developed statistical models that apply distribution fitting to characterize storage workload behavior [140]. Through an extensive evaluation of real-world traces, we found that the two-phase Hyper-exponential provides the best empirical fit to storage workload distributions. This result enables substantially more accurate queuing models that provide performance predictions for cache and storage hierarchies.

In our third thrust, to address the challenge of effectively implementing tiering in modern systems, we propose to apply tiering to live virtual machine (VM) migration using CXL. Unlike traditional migration approaches that rely on repeated memory transfers and dirty page tracking [29], CXL enables multiple hosts to directly access the same byte-addressable memory region. We explore two designs: (1) a CXL-based migration mechanism that eliminates dirty tracking entirely and (2) a tiered-memory approach that reduces migration cost by placing part of the VM's memory in shared CXL space. Both approaches aim to reduce total migration time, blackout duration, and data movement compared to state-of-the-art techniques such as RDMA.

The first proposal of this thesis is a new CXL-based migration mechanism implemented in QEMU. In this approach, the source machine performs a one-time copy of the VM's memory from local DRAM into CXL memory while the VM continues executing. After each page is copied to CXL, the hypervisor transparently remaps the guest's physical address pointer to point to CXL memory. As a result, all subsequent VM writes transparently go directly to CXL, and there is no need to track dirty pages. This eliminates the complexity and performance penalty of traditional dirty tracking mechanisms. Once all pages have been transferred to CXL, the destination host resumes the VM immediately from shared CXL memory, without waiting for memory to be copied to its local DRAM. From there, the destination may choose to migrate memory lazily into DRAM for performance, or continue executing from CXL.

The second proposal of this thesis is to implement memory tiering and the migration of tiered memory within QEMU. In this model, the guest VM's memory is split between local DRAM and a shared CXL device. Since the CXL memory is shared and visible to both hosts, pages that reside in CXL require no data movement during migration. Only the portion of memory that is located in local DRAM must be transferred. This significant reduction in data movement directly translates to a much shorter total migration time. We also plan to evaluate hybrid methods, where local memory is migrated using traditional methods such as TCP or RDMA—to explore if there are scenarios where this is more optimal than pure

CXL-based migration.

It is our thesis that tiered storage and memory systems expose a vast configuration space with the potential for significant performance and cost optimizations. Fully realizing these benefits requires efficient techniques for exploring and exploiting this space, particularly as the introduction of CXL shared memory and tiering adds new and powerful opportunities for tiering.

The rest of this thesis proposal is organized as follows: Chapter 2 provides additional background on live migration, RDMA, CXL, as well as motivation for this work. Chapter 3 reviews prior work relevant to live VM migration and CXL. Chapter 4 examines cache analysis techniques and introduces our multi-tier cache simulator. Chapter 5 presents our work on accelerating multi-tier storage cache simulations using knee detection. Chapter 6 presents interactive visualization tools and workload modeling techniques that support exploring and analyzing complex system design spaces. Chapter 7 presents our proposed work on CXL-based migration and tiered migration. Chapter 8 discusses future work outside of the scope of this thesis. Chapter 9 concludes the proposal and outlines future research directions. Chapter 10 acknowledges the contributions of collaborators, institutional support, and funding agencies.

# Chapter 2

# Background and Motivation

In this chapter, we provide background information relevant to live VM migration and describe our vision and motivations for applying CXL-based tiering to migration. We begin by describing the different live VM migration techniques. Then we give some background on RDMA and discuss how it is used for live VM migration. Lastly, we provide some background on CXL.

## 2.1 Live VM Migration Overview

Live VM migration involves transferring the VM state from the source machine to the target machine. Much of this state is the contents of the VM's memory pages, which presents two main problems: (1) increasing memory sizes mean an increasingly large amount of data must be transferred to the target host, and (2) as the VM continues running, the memory contents at the source host continue to change. There are three primary paradigms for live-VM migration that have been developed to mitigate these issues: pre-copy, post-copy, and hybrid—described next.

The pre-copy technique comprises three phases. In the first phase, all of the VM's memory pages are copied to the target machine while the VM continues to run on the source host. Pages can be dirtied by the workload on the source host during this time because the VM is still active. The second phase iteratively copies the dirtied pages until a stopping condition is met. The final stop-and-copy phase begins when this condition is met; the VM is paused on the source host, all remaining pages are copied to the target machine, and the hypervisor resumes VM execution on the target.

The post-copy technique takes the opposite approach. The minimum VM state required to resume execution is first copied to the target machine, and then VM execution is immediately resumed on the target without copying over any memory pages. Memory is then handled using demand paging, where page faults require transmission from the source machine. Several optimizations have been proposed to reduce the number of page faults, including active push, pre-paging, and Dynamic self-ballooning [26].

The third technique is a hybrid that combines both pre- and post-copy methods. The pre-copy portion runs for a single round, rather than the iterative approach used in traditional pre-copy. Some versions of this technique only copy over a fraction of the memory and storage, such as adaptive live VM migration [161]. After the pre-copy phase, the minimal VM state is copied to the target machine and VM execution is immediately resumed. Finally, the rest of the memory pages are handled using demand paging as in traditional post-copy.

All of the existing migration techniques have significant CPU, memory, storage, and network overheads. The migration process heavily degrades the performance of guest workloads and adversely affects the efficiency of the data center [26]. The overheads are typically classified as either blackout or brownout phases. A blackout phase occurs during traditional pre-copy, where VM execution is halted as the last dirty pages are copied over. A brownout phase refers to the time during migration while the VM is still active and performance is degraded. Brownout phases are particularly problematic, as they can significantly degrade performance for multiple minutes.

## 2.2 Remote Direct Memory Access (RDMA)

RDMA allows data to be transferred directly between the memory of two machines over a network without involving their operating systems or CPUs. RDMA provides low latency and high throughput by bypassing the CPU, making it ideal for applications requiring rapid access to remote memory, such as distributed databases, big data analytics, and cloud services. While this technology offers tangible benefits, RDMA still has at least three *architectural limitations*.

The first challenge with RDMA is the complexity and overhead of its memory management protocols. Before using memory for RDMA operations, each machine must register that memory with their RDMA NICs (RNICs) as an RDMA "memory region" (MR). During registration, the operating system reserves physical memory and ensures that the memory pages in their MRs are "pinned." Mem-

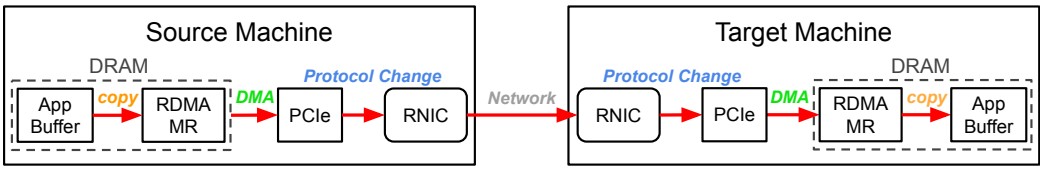



Figure 2.1: A high-level overview of data movement over RDMA. Data is first copied from the source machine's application buffer to its RDMA memory region (MR). Next, the source machine's RNIC transfers the memory from its DRAM to a PCIe channel utilizing DMA. The RNIC driver then converts the transaction to the appropriate network protocol (*e.g.*, InfiniBand). The data is then transferred over the network to the target machine, where all previous steps are executed in the reverse order until the data reaches the target machine's application buffer.

ory pinning involves locking memory pages in physical memory so they cannot be swapped out by the operating system, ensuring the RNICs can reliably access them. Registration also involves translating virtual addresses to physical addresses and then registering them with the RNIC. Depending on the size and fragmentation of the memory, this registration process can take from microseconds to milliseconds, involving system calls and coordination between the CPU, OS, and hardware [96]. Frequent pinning introduces significant latency and increased CPU usage, greatly reducing the benefits of RDMA [76]. When an application is done with the MR, that region can be deregistered.

Several techniques have been developed to reduce the overhead of RDMA registration and deregistration, but also come with their own disadvantages and have limited applications [154]. For example, a commonly used technique is to pre-allocate and reuse MRs [76, 111, 150, 83]. Reusing MRs improves the overall performance of RDMA by avoiding de/registration overheads, but requires redundant data copies in the DRAM of both machines—thus adding some overhead and complexity. This is because data need to be copied from their original location on the source machine into a pre-allocated MR, and then from a pre-allocated MR on the target machine to its final destination.

The second RDMA issue is the inefficiency of data movement over RDMA. Figure 2.1 depicts a high-level overview of an RDMA operation using pre-allocated MRs on both machines. Three main issues contribute to the inefficiency: First, the data must be copied between the application buffers and MRs of both machines, resulting in redundant data and the overhead of the copy. Next, RDMA utilizes efficient DMA to transfer data to and from DRAM, but this requires at least two

DMA transfers because transfers need to be performed by both RNICs. Lastly, transfers between PCIe-attached RNICs take place over a high-speed network (e.g., InfiniBand), introducing additional overheads from the required protocol/interface changes [45].

The third issue is that RDMA requires additional hardware and potential infrastructure changes. The initial investment of adopting RDMA can be prohibitively expensive due to the need for specialized RNICs and lossless network infrastructure [97]. While RDMA is often promoted for its low latency and reduced CPU usage, it still imposes some processing overhead and struggles with congestion control. Many of these challenges are addressed by using smart RNICs, which offload tasks from the CPU and optimize data flow. However, these advanced RNICs can even further increase costs, with prices reaching up to 5.7X more than commodity RNICs [112]. Integrating RDMA into existing infrastructures may also involve replacing traditional Ethernet networks, requiring significant system modifications. These investments make it challenging to identify appropriate applications and design systems that properly leverage RDMA performance gains to justify the initial expenses [103, 55, 48].

## 2.3 Compute Express Link (CXL)

Compute Express Link (CXL) is an open, industry-supported interconnect standard designed to provide high-performance, cache-coherent memory access between processors, memory expansion devices, and accelerators such as GPUs and FPGAs. CXL reduces overall system costs, simplifies software development, and minimizes access latencies by enabling resource sharing through an optimized data path, efficient coherency protocols, and minimizing redundant data copies [30].

CXL is built on the PCIe interface and defines three separate protocols for interacting with CXL devices:

- CXL.io: functionally equivalent to the PCIe protocol; it provides non-coherent load/store I/O access and is used for fundamental operations such as initialization, device discovery, and interrupts,
- CXL.cache: allows CXL devices to coherently access and cache host CPU memory, and

- CXL.mem: allows the host CPU to coherently access CXL device memory for both volatile memory and persistent storage.

Furthermore, CXL devices are classified into three different types:

- Type 1 Devices: specialized accelerators that lack their own local memory (e.g., SmartNICs),
- Type 2 Devices: general-purpose accelerators such as GPUs and FPGAs with their own local DDR and/or HBM that provides coherent two-way access between host CPU memory and device memory, and
- Type 3 Devices: memory expansion devices that provide hosts access to disaggregated memory or byte-addressable persistent storage.

The first CXL Specification 1.0 was released in March 2019. It is based on PCIe 5.0 and allows hosts to coherently access the memory of directly attached accelerators and memory expansion devices [146]. CXL 1.1 was released in June 2019, comprising some errata and a new compliance chapter defining how interoperability testing between the host processor and an attached CXL device can be performed [119].

The next generation CXL 2.0 was released in November 2020, introducing single-level CXL switching, memory pooling, and CXL IDE (Integrity and Data Encryption) [31]. CXL 2.0 switches enable multiple hosts to connect to multiple CXL devices, either through a CXL switch or via direct connect. With CXL 2.0 memory pooling, CXL Type 3 memory expansion devices called multi-logical devices (MLDs) can be partitioned into logical devices (LDs), with up to 16 different hosts exclusively accessing the LDs of a single device.

CXL 3.0 doubles the bandwidth of CXL 2.0 up to 64 GT/s while maintaining the same latency, and introduces several new features that greatly differentiate it from traditional RDMA memory access. It implements memory sharing through an enhanced coherency protocol, replacing the bias-based coherency used in previous generations. This model enables snoop filter implementation and allows devices with their own memory to back-invalidate a host machine's cache. For example, Type 2 accelerators can fetch data from the host and save it in their own cache, perform work on the data, and then update the host's cache upon completion. This new coherency protocol also facilitates peer-to-peer connectivity between devices. CXL 3.0 devices are able to directly access each other's memory

without needing to go through the host, enabling a higher level of disaggregation and more flexible topologies.

CXL 3.0 also expands on memory pooling by introducing coherent memory sharing. With 2.0 memory pooling, Type 3 memory expansion devices can be partitioned to multiple hosts with each partition belonging to only a single host. In contrast, multiple hosts or devices can coherently access shared regions of memory with CXL 3.0 memory sharing. This feature gives us the opportunity to design systems that reduce unnecessary data movement and redundant data copies, resulting in less overhead and better resource utilization than RDMA.

## 2.4 Thesis Statement

This thesis proposal aims to utilize CXL shared memory to address three problems that stem from RDMA's *migration-specific inefficiencies*: (1) the significant overhead of dirty-page tracking, which is required by traditional RDMA-based migration; (2) retransmission of pages that are dirtied after their initial copy, which increases both total migration time and the amount of data transferred; and (3) extended blackout periods during the stop-and-copy phase, with duration increasing as VM memory size grows.

To address these problems, we propose two CXL-based techniques that overcome fundamental limitations of state-of-the-art RDMA-based live VM migration. The first is a CXL-based migration approach that avoids dirty tracking and transfers each page only once, reducing the overhead of traditional multi-pass techniques. The second is a transparent CXL memory tiering migration strategy that dramatically reduces both the total migration time and data that needs to be migrated.

The goal of this work is to implement both techniques in QEMU, evaluate their performance against RDMA-based and hybrid migration approaches, and demonstrate that CXL migration and memory tiering can significantly reduce total migration time, blackout duration, and the amount of data transferred, particularly for large, memory-intensive VMs.

# Chapter 3

# Related Work

In this chapter, we survey related works about live migration algorithms, RDMA-based migration, shared-memory migration, and CXL remote memory and tiering.

## 3.1 Live Migration Algorithms

Clark *et al.* first proposed the standard pre-copy approach to live migration in 2005, in which memory pages are copied iteratively while the VM continues running. They showed that their prototype could migrate an 800 MB guest over 100 Mbps Ethernet in around 20–180 seconds while keeping blackout times between roughly 40–270 ms, depending on the workload [29]. Biswas *et al.* evaluated migration over 10 Gbps Ethernet and showed that transferring a 400 MB VM finishes in about 15 seconds with blackout times of approximately 300–400 ms [14]. More recently, in 2018 Google reported blackout times of around 50–200 ms for 30 GB VMs using 25 Gbps Ethernet, with total migration taking a few minutes [113]. These results demonstrate that even as network bandwidth improves, the effective speed of migration remains limited and blackout times show no improvement, underscoring fundamental limitations of traditional live-migration methods.

Researchers have proposed many techniques that make traditional migration more efficient. Ibrahim *et al.* add a stop-and-copy switch that adapts to memory-dirtying rate and link speed, cutting slowdown on HPC jobs by up to 10× [61]. Haris *et al.* replace hand-tuned thresholds with a k-nearest-neighbor predictor that halts pre-copy once the predicted blackout meets an SLA, trimming total migration time by 86% and downtime by 65% [53]. Eswaran *et al.* preserve copy-on-write sharing among templated VMs during migration, cutting network

traffic by up to 92% and total migration time by 95% [39]. Liu *et al.* compress outgoing pages with Intel's In-Memory Analytics Accelerator, achieving 4.5× compaction and restoring memory 55 percent faster with no extra CPU load [75]. Song *et al.* shard copy, checksum, and I/O across all cores and NIC queues, shrinking downtime by up to approximately 280× using 10 Gbps Ethernet [120]. Despite these optimizations, each technique still depends on dirty-page tracking and retransmitting dirtied pages, and therefore cannot fully eliminate the fundamental inefficiencies of traditional migration.

## 3.2 RDMA-Based Migration

State-of-the-art migration systems often utilize RDMA to reduce both migration time and CPU overhead. Huang *et al.* first demonstrated that InfiniBand RDMA can reduce total migration time by up to two orders of magnitude compared with TCP, inaugurating the zero-copy paradigm [59]. Follow-up work by Isci *et al.* recorded similar gains on enterprise traces while maintaining blackout under 200 ms [62]. Nomad snapshots RDMA queue pairs and restores them on a peer RNIC at the destination, enabling live migration of VMs with active RDMA connections [60]. Live migration over InfiniBand with single-root I/O virtualization (SR-IOV) support, which allows virtual machines to directly access virtual functions exposed by a physical NIC, achieves blackout times below 140 ms for 8 GB guests but requires device-specific coordination to transfer hardware state between source and destination [47].

While RDMA improves performance, it still relies on dirty-page tracking, retransmission of dirtied pages, and has architectural limitations (see Chapter 2.2). As a result, blackout times have seen little improvement and convergence remains slow under write-heavy workloads.

## 3.3 Shared Memory Migration

Researchers have begun to investigate shared memory as a more efficient alternative to network-based VM migration. Recent work by Grapentin *et al.* developed an IBM POWER9-based prototype that uses ThymesisFlow [105] to enable peer-to-peer, disaggregated memory access between servers. This work showed that key performance metrics from the perspective of applications running in the virtual machine, such as memory latency and throughput, were improved by up to

three orders of magnitude during the migration process [46]. Ran *et al.* evaluated migration using distributed shared memory (DSM), a system that allows multiple machines to share a unified memory space over a network. By preloading the VM's memory into the DSM layer before hand-off, they reduced total migration time by about 70 percent [66]. Finally, we note that there was a BoF discussion about how CXL could be used for VM migration at the Linux Storage, Filesystem, Memory Management & BPF Summit in May 2023, along with a website titled "nil migration" [122]; however, no code has been released and there have been no updates on the website since. These works show that shared-memory migration is feasible on proprietary fabrics and motivate the development of CXL-based migration.

## 3.4 CXL Remote Memory and Tiering

Recent evaluations demonstrate that CXL 2.0 Type-3 memory-expansion devices achieve a significant portion of PCIe's theoretical bandwidth, with latency about twice as high as local DRAM. Zhong *et al.* measure 97 ns idle latency for on-board DDR5 and 219 ns for a CXL device on Intel Sapphire Rapids, while per-core bandwidth tops out at 48 GB/s on an x16 PCIe 5.0 link [164]. Unal *et al.* report similar results in a HotOS 2025 study, measuring 112 ns latency for DRAM and 237 ns for CXL, with bandwidth just under 50 GB/s [134]. Weisgut *et al.* evaluate Micron CXL devices on Genoa servers and record a median 255 ns load latency together with 46 GB/s sustained copy bandwidth per device, scaling to 92 GB/s with four cards [145].

A recent survey by Sharma *et al.* thoroughly describes the architectural sources of CXL latency and bandwidth [32]. They state that CXL latency is composed of a protocol component and a queuing component, which depends on load. Protocol latency comes from two full traversals of the CXL port stack, each adding around 21–25 ns, plus approximately 15 ns of wire and retimer flight time, resulting in roughly 57 ns per memory access. Queueing latency is incurred only under load as requests pile up in the link-layer and memory-controller queues. Ultimately, this results in the average latency of CXL being roughly double that of local DRAM. As for bandwidth, 6–9% is lost to link-layer overheads, including bytes used for flit headers, CRC/FEC, SKP and ordered-set blocks, as well as flow-control bookkeeping based on credits.

Emerging research shows that CXL-attached DRAM can serve as a high-capacity second tier with only single-digit performance cost when guided by smart

hardware or OS policies. Transparent Page Placement (TPP) extends Linux NUMA balancing to demote cold pages to CXL and promote hot ones, keeping a tiered system within 1% of an all-DRAM baseline and outperforming stock Linux by up to 18% [92]. Intel Flat Memory Mode plus the Memstrata allocator moves tiering into the memory controller at cache-line granularity and enforces per-tenant isolation, keeping 82% of 115 Azure traces within 5% of local DRAM and trimming the worst slowdown from 34% to below 6% [164]. Nomad retains shadow copies and performs transactional page migration, cutting thrashing and delivering up to 6× speed-ups over TPP under heavy memory pressure [151]. Alto introduces a memory-level parallelism-aware amortized off-core latency metric to suppress unnecessary migrations, improving performance by up to 12× compared to TPP, Nomad, and two other policies across both NUMA and CXL hardware [86]. Together, these results paint an increasingly optimistic picture: CXL memory can reliably expand server capacity while keeping latency-sensitive workloads within a few percent of local DRAM performance.

# Chapter 4

# Desperately Seeking ... Optimal Multi-Tier Cache Configurations

Modern cache hierarchies are tangled webs of complexity. Multiple tiers of heterogeneous physical and virtual devices, with many configurable parameters, all contend to optimally serve swarms of requests between local and remote applications. The challenge of effectively designing these systems is exacerbated by continuous advances in hardware, firmware, innovation in cache eviction algorithms, and evolving workloads and access patterns. This rapidly expanding configuration space has made it costly and time-consuming to physically experiment with numerous cache configurations for even a single stable workload. Current cache evaluation techniques (*e.g.*, Miss Ratio Curves) are short-sighted: they analyze only a single tier of cache, focus primarily on performance, and fail to examine the critical relationships between metrics like throughput and monetary cost. Publicly available I/O cache simulators are also lacking: they can only simulate a fixed or limited number of cache tiers, are missing key features, or offer limited analyses.

It is our position that best practices in cache analysis should include the evaluation of multi-tier configurations, coupled with more comprehensive metrics that reveal critical design trade-offs, especially monetary costs. We are developing an $n$-level I/O cache simulator that is general enough to model any cache hierarchy, captures many metrics, provides a robust set of analysis features, and is easily extendable to facilitate experimental research or production level provisioning. To demonstrate the value of our proposed metrics and simulator, we extended an existing cache simulator (PyMimircache). We present several interesting and counter-intuitive results in this paper.

## 4.1 Introduction

The vast configuration space of multi-tier caching enables the design of very complex systems. Several tiers of cache and persistent storage can be allocated in numerous arrangements. Moreover, devices can be partitioned into many differently sized cache segments for separate applications. All of these devices can be implemented within, and interact with, any number of independent, large-scale infrastructures (*e.g.*, cloud services, virtual machines, big data warehouses, distributed systems). Furthermore, new storage technologies are constantly emerging (*e.g.*, NVM, 3D flash), introducing additional complexity, greater capacities, and different cost/performance profiles. Our ability to dynamically change hardware in live systems (*e.g.*, adding or deleting RAM, SSD, NVM) has also been increasing, particularly in cloud environments and virtual machines [49, 27, 28], making it significantly easier to reconfigure a cache hierarchy. Workloads continue to evolve as well, with complex and diverse access patterns that affect the frequency of data reuse and the size of working sets, two of the most influential factors in any caching system [121, 147, 116, 10, 24].

Research in cache algorithms and policies is also trying to keep up with these changes. Machine learning and similar techniques that leverage historical data are being incorporated into caching systems to bolster prefetching [156], dynamically switch between replacement algorithms [114, 139, 116], or enhance existing eviction policies [5]. I/O classification has been used to enforce caching policies and improve file system performance [95]. Multi-tier cache eviction algorithms that are aware of some or all layers in the hierarchy at any given time are being developed [24]. The challenges of cache resource allocation and provisioning are being investigated as well [71, 13]. Zhang *et al.* introduced CHOPT, a choice-aware, optimal, offline algorithm for data placement in multi-tier systems [160]. Algorithms such as CHOPT are promising solutions for efficiently finding optimal multi-tier configurations, but their bounding assumptions and inability to model all parameters limit the configuration space they can explore.

Physically experimenting with various cache configurations is costly and time-consuming, with so many parameters to consider (*e.g.*, number of tiers, device types and models, caching policies). A well-known technique for evaluating cache performance without running experiments is Miss Ratio Curve (MRC) analysis [141, 54, 15, 58]. MRCs plot the cumulative miss ratio of all requests in a given workload for some cache eviction algorithm(s) as a function of cache size. Cache size usually ranges from one data block to the size required to store every unique block accessed in the workload, also known as the *working set*. This technique

has many uses, such as comparing eviction algorithms' performance for a given workload or identifying optimal cache size allocations. However, MRCs evaluate the performance of only a single cache and are not capable of accurately modeling the complicated interactions between devices in a multi-tier cache. Recent studies have shown that traditional MRCs are even sub-optimal for resource allocation in a single layer, since they admit data with poor locality into the cache. [41].

It is vital that our methods of evaluating caches mature as storage technologies and cache hierarchies continuously evolve. For example, examining performance metrics such as latency or using an MRC to analyze miss ratio as cache size increases may be misleading without also considering the monetary cost of purchasing and using the cache. Cost has a non-linear, positive correlation with cache size, and is fundamentally the primary constraint when deciding how much cache to include in a system. If this were not the case, everyone would cache all data in copious amounts of the fastest DRAM money can buy and back it up with a huge battery. Furthermore, improved performance does not directly translate into cost efficiency, especially in a multi-tier system where devices' cost and performance characteristics can vary wildly. The purchase cost of hardware is a simple example. Ideally, we should be evaluating more comprehensive metrics such as the total cost of ownership, which combines other metrics such as power consumption, the cost of labor to maintain a system, and the projected lifetime of devices given access patterns. It is also essential that we can freely evaluate the relationship between metrics (*e.g.*, throughput/$) so we can make educated design decisions with full awareness of the inherent trade-offs.

The most complete solution would be an $n$-level I/O cache simulator that could quickly and accurately evaluate many configurations. While there are some advanced CPU cache simulators available [63, 35, 104, 143, 91], storage cache simulators are scarce and lacking. State-of-the-art storage cache simulators are mostly outdated; they either can simulate only a single layer or some fixed set of layers, have limited analysis features, are not easily extendable, or are simply not released to the public [148, 51, 1]. PyMimircache [155] is a popular open-source storage simulator with several useful features that is actively maintained. However, even this simulator is inadequate; it also can simulate only a single layer of cache with no implementation of back-end storage, has no concept of write policy, and its analysis features are limited. The main strength of PyMimircache is its ability to perform MRC analysis on multiple cache replacement algorithms.

It is our position that best practices in cache research need to be broadened to reflect the growing multi-tier configuration space. This paper makes the following contributions:

1. We explore current trends in cache analysis and propose that best practices in cache research including the analysis of multi-tier configurations and a more comprehensive set of evaluation metrics (*e.g.*, monetary cost).

2. We describe the critical features an $n$-level I/O cache simulator should have and outline the design of a simulator we began to develop.

3. We extended PyMimircache to function as a multi-tier cache simulator, experimented with many configurations on a diverse set of real-world traces, and present initial results that support our position.

## 4.2 Cache Analysis

The fundamental strategy in engineering a cache hierarchy involves placing faster and typically lower-capacity devices in front of slower devices to improve the overall latency of accessing frequently reused data. There is a tangible dollar cost per byte increase when purchasing hardware with better performance attributes. Therefore, it follows that the cache size and speed are closely correlated with the purchase cost. Straightforward logic dictates that performance is constrained by cost, so unless money is in endless supply, the best practice should be to evaluate these metrics together. Surprisingly though, cost is often overlooked during analysis in favor of performance metrics such as raw throughput, latency, or hit/miss ratio [144, 27, 24, 109, 41, 23, 19].

The argument can be made that any improvement in cache performance translates into a reduction in cost when designing a cache, such that the relationship between cost and performance does not necessarily need to be considered. This is situationally true, particularly when evaluating performance in a single-tier caching system. However, in a more realistic, multi-tier storage or CPU cache hierarchy, the large configuration space and complex interactions between tiers produce scenarios where the relative performance per dollar between two configurations is vastly different, necessitating a more complex analysis (see Section 4.4 for examples).

Performance metrics have long been the standard in cache analysis. Recently, additional metrics that are more relevant and informative for specific applications have gained popularity in storage research. The $95^{th}$ (P95) or $99^{th}$ (P99) percentile latency, often referred to as *tail latency*, is an important quality of service (QoS) metric for cloud [153, 125] and web [64, 34, 52, 13] services, as well as at the hardware level [77, 18, 82, 36]. Inter-cache traffic analysis has been used to design

more efficient cache hierarchies in modern microprocessors [102]. Reducing the energy consumption of storage systems is beneficial for the environment, lowers operation costs, and promotes advancements in hardware design [20, 80, 118, 137]. Even the total cost of ownership (TCO) can be difficult to calculate when considering all the factors that contribute to capital and operational expenditures (CapEx and OpEx) [81, 79].

It is our position that cache analysis should be conducted using a diverse set of metrics whenever possible. These metrics should be evaluated at various level of granularity: at each individual layer, some subset of layers, or globally. Moreover, we need to create complex metrics (*e.g.*, throughput/$) that allow for analysis of their informative relationships and reveals critical design trade-offs.

## 4.3 Multi-tier Cache Simulation

**Simulator Design** A general, *n*-level I/O cache simulator with a rich set of features is necessary to thoroughly explore the multi-tier caching configuration space and analyze our proposed metrics. We are developing such a simulator that includes (but is not limited to) the following capabilities: **(1)** *Write policy* that determines where data is placed upon write requests. We will support traditional write policies (*e.g.*, write through, write back, write around), but also allow user-defined policies. **(2)** *Admission policy* that controls if and how data is promoted and demoted throughout the hierarchy by request size, address space, or simply whether layers are inclusive or exclusive of each other. **(3)** *Eviction policy* that decides which data to evict when a cache is full and new data needs to be brought in. There will be support for single-layer or global policies, as well as the ability to easily add new policies. **(4)** *Trace sampling* techniques (*e.g.*, Miniature Simulations [142]) that reduce the size of a trace to greatly decrease simulation time while maintaining similar cache behavior. **(5)** *Prefetching* to retrieve data before it is requested with techniques like MITHRIL [157] that exploit historical access patterns.

The associated API will fully expose all data structures at request-level granularity or for any given real timestamp or virtual ones (where the trace has only ordered records without their original timing). This will allow users to perform important analysis such as examining clean and dirty pages at any level, measure inter-reference recency, calculate stack distance metrics when relevant, or perform any type of analysis offered by our simulation framework on a subset of a trace. The simulator will also be coupled with modern visualization tools that enable

users to efficiently explore the large amount of data it produces.

**Multi-tier Cache Reconfiguration** A major motivation for simulation is seeking optimal cache configurations. However, efficiently reconfiguring a multi-tier cache hierarchy is another challenging problem. In this work, we analyze various physical devices for simplicity, but manually swapping out devices is often not a feasible solution. More likely, multi-tier caches may be dynamically reconfigured in cloud, distributed, and virtual environments, where storage can more easily be allocated through virtualization abstractions. For example, distributed memory caching systems (*e.g.*, Memcached) can greatly benefit from automatically reconfiguring cache nodes in response to changes in workload; but this process can significantly degrade performance as nodes are retired and data is migrated. Hafeez *et al.* developed ElMem, an elastic Memcached system that uses a novel cache-merging algorithm to optimize data migration between nodes during reconfiguration [50]. Moving between configurations in any caching system has a *temporarily* negative impact on performance, until the new caches are fully warmed [166, 22]. Therefore, efficient reconfiguration methods are essential to fully leverage any techniques that find optimal configurations (including simulations).

**PyMimircache Extension** To demonstrate the utility of our proposed simulator, we extended PyMimircache [155], a storage cache simulator with an easily extendable Python front-end and efficient C back-end. We made several simplifying assumptions for this extension and experimented with a subset of the possible features we are proposing. **(1)** We implemented a traditional write-through policy and an "optimistic" write-back policy as global write policies. The write-through policy is consistent and reliable: a block is written to every cache layer and the back-end storage whenever there is a write request. Our write-back policy is optimistic: it only writes to the first layer and assumes this data will be flushed to persistent storage at some point in time, outside of the critical path where it does not affect performance (*i.e.*, we do not account for the write in any other layer). This simplified version of write-back models the best-case performance scenario, which we found useful for exploring the potential effects of write policy. A more realistic write-back would require asynchronous functionality that is not available in PyMimircache, and is a limitation of this work. **(2)** All evicted blocks are discarded rather than demoted (moved or copied) to some lower layer of cache or back-end storage. **(3)** Layers of DRAM are included in our simulations even

though we are using block traces, which capture requests for data that was not found in DRAM. This is a limitation of the traces we are using; the simulator we implement will be able to operate on any data item from any trace that includes some form of address accesses. The simulator will support traces obtained from networks (*e.g.*, NFS, HTML), distributed systems (*e.g.*, HPC, Memcached), system calls, block traces, and potentially more. **(4)** Throughput is limited by the system where traces were actually captured since we experiment with block traces. For demonstration purposes, we ignore this limitation and assume requests are fed as fast as possible without using the original request timestamps. This allows us to show how we can potentially evaluate throughput when using different hardware configurations. **(5)** We consider each layer to have a portion of its capacity partitioned for caching to emulate various cache sizes at each layer using the specifications of a single device.

A high-level description of how we extended PyMimircache is as follows: **(1)** We feed an original block I/O trace to an instance of PyMimircache, this is the top layer ("L1") of our cache hierarchy. **(2)** This instance generates two output files: i) A log file "L1-log" containing counters for the following: read hits, write hits, read misses, write misses, data read, data written. ii) We specify a new trace file called "L1-trace" which contains read requests that missed in L1, as well as all write requests. As per our assumptions, write requests are not included when using write-back policy and evicted blocks from L1 are never included. These intermediate trace files are stored in memory using Python-based virtual files to avoid disk I/O costs. **(3)** After the L1 instance of PyMimircache completes, we feed the generated "L1-trace" from step 2 as input into another, separate instance of PyMimircache. This emulates our L2 layer. **(4)** We repeat steps 2–3 for L2, L3, etc. **(5)** When all layers have been processed, we aggregate all the log data into a single log file for that experiment. **(6)** We have a higher-level script that we pass parameters to for each layer's device: purchase cost, capacity, and average read and write latencies. This script records and calculates the following metrics for the cache configuration of an experiment: total purchase cost, partitioned device capacities, miss ratio per layer, and total read and write latency incurred. It can be [re]run at any time using previously obtained simulation logs, and is separate from the actual simulation process.

## 4.4 Evaluation

**Workloads** In this section, we evaluate simulation results gathered using the Microsoft Research (MSR) traces. These 36 traces, each about a week long, were collected from 36 different volumes on 13 production servers at MSR in Cambridge, Massachusetts, as described in detail by Narayanan *et al.* [99]. The percentage of total requests that access unique blocks (*i.e.*, data used for the first time) in these traces range from 1% to 97%, which is representative of the frequency of data reuse. The percentage of total requests that are writes range from nearly 0% to almost 100%, and is ideal for evaluating the effects of write policies.

We are continuing to run additional experiments using 9 traces from the Department of Computer Science at Florida International University (FIU) [137] and 106 traces from CloudPhysics [141], but do not present results here due to space limitations.

**Experimental setup** We ran simulations on the MSR traces using between 1 and 3 layers of cache, in addition to the back-end storage device. Each simulation consisted of a configuration of several parameters: cache and back-end sizes, eviction algorithms, and global write policy. The capacity required to hold the entire working set of a trace dictated the cache and back-end storage sizes for every configuration. The back-end size was always fixed to be the same size as the working set, since the data initially resides in the back-end. The cache sizes selected for the first layer of cache are 100 evenly spaced sizes between 1 block (512 bytes) and the size of the working set for that trace. 100 is the default number of points for plotting MRCs with PyMimircache. The second and third layers of cache are 10 evenly spaced sizes within the same range. Using 10 cache sizes for these layers rather than 100 drastically reduced the time required to complete each experiment while still revealing the entire range of metrics (albeit with fewer data points within that range).

In this work, we only present results for configurations using a Least Recently Used (LRU) eviction policy at every layer, although we are varying these policies in our ongoing simulations. We simulated each of the MSR traces using our extension of PyMimircache (see Section 4.3) and then calculated cost and performance metrics using the device specifications described in Table 4.1.

While these comprehensive traces represent a wide variety of workloads, they have only a relatively small working-set size that can easily fit in a modern server's RAM. Therefore, to simulate larger workloads (*e.g.*, bigdata, HPC), we treat the original MSR traces as if they were scaled-down spatial samples of larger traces.

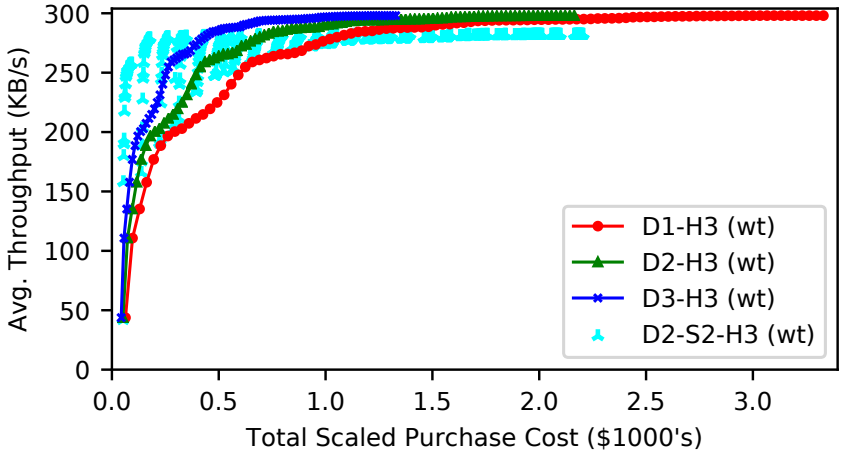



Figure 4.1: Effects of an intermediate SSD tier (Workload MSR hm-1)

We call this technique *reverse-mini-sim*: the reverse of the miniature simulations technique for down-scaling traces introduced by Waldspurger *et al.* [142]. Miniature simulations was shown to be fairly accurate at a sampling rate of 0.001 on the MSR traces, so we multiply the purchase cost (X axis) by a factor of 1,000 times: this simulates a workload whose working set size is 1,000× larger.

Each data point in our figures represents a configuration with some set of cache sizes. We assume that each layer consists of an independent device with a portion of its capacity partitioned for caching and the remaining capacity as unused. For example, a cyan triangle in Figure 4.1 at Total Scaled Purchase Cost of around $235 represents the average throughput of all requests in a single simulation of the hm-1 trace with an L1 LRU cache of 61,865 blocks partitioned in device D2, an L2 LRU cache of 199,344 blocks partitioned in device S2, and back-end storage of device H3 partitioned to fit the working set of 687,396 blocks.

**Cache hierarchy depth** Figure 4.1 shows (D1-H3, red) that too little RAM hurts performance but too much wastes money. Adding a bit of SSD cache (D2-S2-H3, cyan) between DRAM and HDD (D2-H3, green) can help, but *not* always (some cyan dots are *below* the green line). Consider the knee of D1-H3 (around X=$500): there are D2-S2-H3 configurations that provide higher throughput for the same cost, same throughput for less cost, and even *both* higher throughput and less cost. Surprisingly, we also see that purchasing more of a cheaper DRAM (D3-

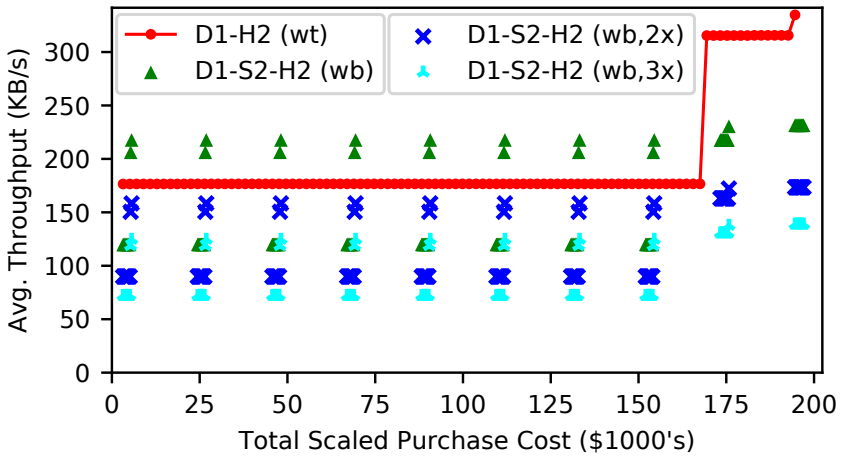



Figure 4.2: SSD Aging Effects (Workload MSR src2-1). 2× and 3× indicate configurations where S2 has 2–3× increased latency due to the potential effects of SSD aging

H3, blue) for the same cost of a more expensive DRAM (D1-H3) yields overall better performance. Therefore, we can sacrifice DRAM performance for a larger amount of DRAM to get better results.

**Solid-state drive (SSD) degradation** Storage devices have an expected lifetime which is typically defined by some amount of I/O. For example, it is well-known that the memory cells within SSDs can only be written to a finite number of times before they are no longer usable [65, 88, 101]. While the lifespan of devices is a parameter that should be considered when estimating the total cost of ownership of a storage system over some period of time, it is also important to evaluate the performance impact this aging process can have. Studies have shown that SSD aging can increase average latency by around 2–3× [67]. To simulate this effect, we multiplied the latency specifications of device S2 and analyzed the results alongside simulations using its original specifications. Figure 4.2 shows that while a new SSD (D1-S2-H2, green triangles) improves performance when inserted into a D1-H2 tier (red), when the SSD is aged (blue and cyan), performance is actually worse than not having the SSD at all. For users with write-heavy workloads or infrastructures where these devices are expected to receive a lot of I/O traffic over a short period of time, choosing to exclude SSDs completely may not only save

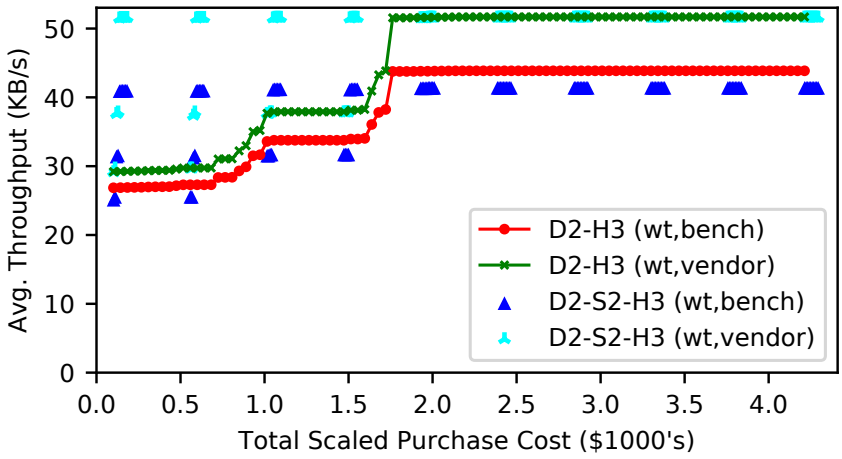



Figure 4.3: Variation between vendor-reported specs and independently operated benchmarks (Workload MSR web-3)

money, but also yield a similar or better average throughput over time.

**Device specification variance** Storage vendors want to convince consumers that their latest device is competitive. They do so by publishing many device specifications: storage capacity, physical dimensions, hardware interface, durability, energy consumption, and performance metrics. While most specifications are fairly standard, a wide variation of performance metrics can be found, even amongst the same type of device and vendor. Some commonly found metrics are the minimum, average, median, or maximum values for latency, bandwidth, or throughput. These metrics may also be further refined as random or sequential workloads, or separated by reads and writes. These measurements are obtained via benchmarks using some specific workload(s), software environment, and hardware configuration, which are sometimes disclosed at varying levels of detail. This poses a significant problem for consumers, who often are unable to reproduce vendors' performance results. Given such a vast configuration space of variables that can affect performance and the understandable motivation for vendors to publish optimistic results, how can storage devices be reliably compared for their own usage? A handful of independent, reputable websites have emerged by fixing these variables and benchmarking devices from different vendors, and producing realistic, trustworthy specifications: AnandTech [6], Tom's

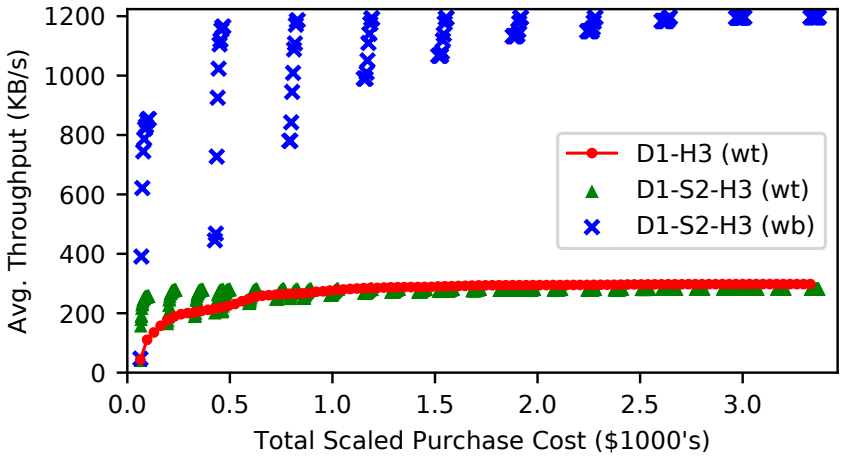



Figure 4.4: Write-through vs. Write-back policy effects (Workload MSR hm-1)

Hardware [131] and UserBenchmark [135].

In this experiment we show the difference between numbers reported by vendors and others. Figure 4.3 shows that inserting an SSD tier between DRAM and HDD provides equal or better performance when using vendor reported specifications (green and cyan). However, specifications obtained from Anandtech [6] (red and blue) show that the majority of the configurations yield worse average throughput.

**Write Policy** The write policy of a cache hierarchy determines how and where data is written whenever there is a write request. Write-through policy ensures data consistency by writing data to every cache and storage device in the hierarchy. However, this incurs the write latency of every device and negatively impacts overall performance. The write-back policy improves performance over write-through by only writing to the cache and then flushing data to back-end storage at a more favorable time. The downside of write-back is that data is at risk of being lost in the event that a cache device fails or whole system loses power. If reliability is more important, a write-through policy is the obvious choice, but how much impact will this have on performance? Figure 4.4 compares write-through and write-back policies (policy implementations described in Section 4.3). Using an optimistic write-back (wb) policy we achieve up to 6× better throughput for the same cost as write-through (wt) with the same devices. Note that a more accurate

write-back policy will account for the delayed writes, which will tie up the storage devices even during idle times.

## 4.5 Conclusion

Designing and evaluating cache hierarchies has become incredibly complex due to the expanding multi-tier configuration space. In this work, we analyzed the deficiencies of single-tier cache analysis and common cache evaluation metrics. We propose that best practices in cache research should include the analysis of multi-tier systems, as well as the evaluation of a more comprehensive set of metrics (particularly monetary cost) and their relationships. We are developing an $n$-level I/O cache simulator with a rich set of features and analysis tools that is capable of modeling any cache hierarchy. We extended PyMimircache to function as a multi-tier cache simulator and experimented with a wide variety of workload. We presented interesting and counter-intuitive results that demonstrate the need for our proposed simulator and multi-tier analysis.

| ID | Device | Type | Price | Capacity | Average Latency (Benchmark Source) |
|---|---|---|---|---|---|
| D1 | G. Skill TridentZ DDR4 3600 MHz C17 | DRAM | $150 | 16GB | 0.0585$\mu$s r/w (UserBenchmark) |
| D2 | G. Skill TridentZ DDR4 3000 MHz C15 | DRAM | $97 | 16GB | 0.0642$\mu$s r/w (UserBenchmark)<br>0.01$\mu$s r/w (Vendor) |
| D3 | Corsair Vengeance LPX DDR4 2666 MHz C16 | DRAM | $59 | 16GB | 0.0726$\mu$s r/w (UserBenchmark) |
| S2 | HP EX920 M.2 NVMe | SSD | $118 | 1TB | 292$\mu$s read 1,138$\mu$s write (AnandTech)<br>20$\mu$s read 22$\mu$s write (Vendor) |
| H2 | WD Black 7200 RPM | HDD | $60 | 1TB | 2,857$\mu$s read 12,243$\mu$s write (AnandTech) |
| H3 | Toshiba MK7559GSXP | HDD | $65 | 750GB | 17,000$\mu$s read 22,600$\mu$s write (Tom's HW)<br>17,550$\mu$s read 17,550$\mu$s write (Vendor) |

Table 4.1: Device specifications and parameters. Each device is denoted with a letter and number for brevity (1 is high-end, 2 is mid-range, and 3 is low-end). Devices S1, S3, and H1 are skipped for space considerations. Prices were obtained from Amazon in September 2019. Benchmarked specifications were correlated from device vendors, AnandTech [6], Tom's Hardware [131], and UserBenchmark [135].

# Chapter 5

# Accelerating Multi-Tier Storage Cache Simulations Using Knee Detection

Storage cache hierarchies include diverse topologies, assorted parameters and policies, and devices with varied performance characteristics. Simulation enables efficient exploration of their configuration space while avoiding expensive physical experiments. Miss Ratio Curves (MRCs) efficiently characterize the performance of a cache over a range of cache sizes, revealing "key points" for cache simulation, such as knees in the curve that immediately follow sharp cliffs. Unfortunately, there are no automated techniques for efficiently finding key points in MRCs, and the cross-application of existing knee-detection algorithms yields inaccurate results.

We present a multi-stage framework that identifies key points in *any* MRC, for both stack-based (*e.g.*, LRU) and more sophisticated eviction algorithms (*e.g.*, ARC). Our approach quickly locates candidates using efficient hash-based sampling, curve simplification, knee detection, and novel post-processing filters. We introduce *Z-Method*, a new multi-knee detection algorithm that employs statistical outlier detection to choose promising points robustly and efficiently.

We evaluated our framework against seven other knee-detection algorithms, identifying key points in multi-tier MRCs with both ARC and LRU policies for 106 diverse real-world workloads. Compared to naïve approaches, our framework reduced the total number of points needed to accurately identify the best two-tier cache hierarchies by an average factor of approximately $5.5\times$ for ARC and $7.7\times$ for LRU.

We also show how our framework can be used to seed the initial population for evolutionary algorithms. We ran 32,616 experiments requiring over three million cache simulations, on 151 samples, from three datasets, using a diverse set of population initialization techniques, evolutionary algorithms, knee-detection algorithms, cache replacement algorithms, and stopping criteria. Our results showed an overall acceleration rate of 34% across all configurations.

## 5.1 Introduction

A cache's miss ratio is one of the most important predictors of its performance. A miss-ratio curve (MRC) for a given cache and replacement algorithm plots the cumulative miss ratio for all accesses as a function of the cache size, providing a powerful tool for analyzing the performance of live systems and dynamically adjusting cache configurations as workload conditions change [142, 13]. MRCs can also inform offline evaluations such as comparing caching algorithms or analyzing monetary cost vs. storage-system performance [38].

There are many efficient techniques for generating MRCs [93, 126, 149, 141, 128, 57, 142, 41]. MRC-reported miss ratios are good indicators of expected performance (*e.g.*, throughput), but real system performance can vary due to additional factors including device characteristics, write policies, and admission policies [38]. Alas, repeatedly reconfiguring and testing a real caching system, with all possible cache sizes, is prohibitively expensive due to the slowness of storage I/O.

Since experimenting with physical devices is costly and time-consuming, simulation offers a more practical way to explore this large search space and evaluate trade-offs such as latency vs. cost. A common first step is to sample a workload: approximation algorithms enable accurate simulation of cache behavior using only a fraction of the original trace data. Small sampled traces can then be used to construct an MRC accurately, enabling quick evaluation of cache performance [141, 142]. Many storage-cache simulators have been developed that replay traces while attempting to faithfully reproduce real system behavior [1, 89, 155]. However, even simulations can be too expensive to allow exploring a large number of configurations or optimizing live systems in real time. For example, consider a cache with a maximum size of 100GB. Simulating every 1GB size step would require 100 experiments. In a multi-tier setup, the number of simulations grows with the number of tiers; a two-tier configuration would require $100^2$ experiments, three tiers would need $100^3$, and so on. Thus, it is essential to

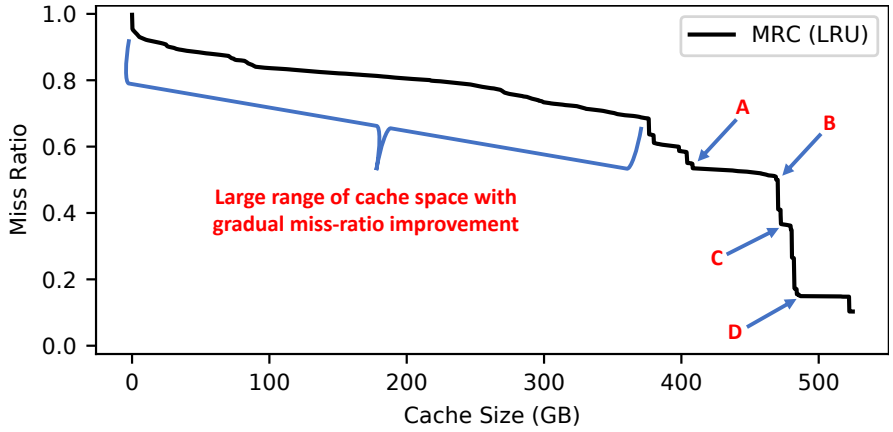



Figure 5.1: MRC for trace w10, annotated to illustrate several key points: useful "knees" (points A, C, and D), a useless "cliff" (B), and a large range of cache sizes with relatively gradual miss-ratio improvement.

explore this vast configuration space efficiently.

Creating an MRC requires a sequence of cache references. In a multi-tier cache, references to level $n+1$ come from misses in—and write evictions and flushes from—level $n$; thus the MRC for $n+1$ directly depends on the cache size chosen for level $n$. A naïve exploration of *multi-tier* configurations would require a separate simulation for each point in level $n$'s MRC to identify misses that become references at level $n+1$, and hence to compute the level $n+1$ MRC. Since an MRC may contain hundreds of points (one for each potential cache size), this approach quickly becomes intractable. Thus, a crucial second step for evaluating multi-tier caches is to limit the number of simulations by intelligently selecting the cache sizes to evaluate at each level.

Intuitively, the most promising candidates are points where a little extra cache space produces a relatively large drop in the miss ratio; such points are often visible as "knees" in MRCs—*e.g.*, points A, C, and D in Figure 5.1. (Note that although B has sharp curvature, it is not of interest since C provides a much lower miss rate.) Given enough computational resources, we may be interested in also selecting some points in the large gradually sloping regions that cover a significant range of cache sizes. We refer to both types of points as *key points* from here on.

In this article we describe a multi-stage framework designed to pick an appropriate yet small number of key points in MRCs: **(1)** We first approximate the MRC accurately using a hash-based sampling technique [141, 142]; **(2)** Next, we use

the Ramer-Douglas-Peucker (RDP) line simplification algorithm [110] to reduce noise by eliminating minor variations in the curve; **(3)** We then run a multi-knee detection algorithm on the remaining points to find cache sizes that provide the greatest miss-ratio improvement for the lowest cost. Our framework currently implements eight different knee-detection algorithms, including our novel Z-Method and modified, multi-knee versions of five widely used single-knee detection algorithms [8, 117, 115, 129]; and **(4)** Finally, in post-processing we remove less interesting points, add points in gradually-sloped regions (if desired), and then select the final points based on a ranking that uses hierarchical clustering and relevance metrics.

This article is an extension of our previous work [37]: we additionally provide a more in-depth analysis of our Z-Method algorithm and we demonstrate how our framework can accelerate the optimization of multi-tier caching systems using evolutionary algorithms. This collective work makes several contributions:

1. We establish the novel methodology of using multi-knee detection to efficiently identify optimal multi-tier cache configurations;

2. We present a framework that combines several techniques to find a minimal number of key points in MRCs for both stack and non-stack caching algorithms;

3. We introduce Z-Method, a new multi-knee detection algorithm that uses statistical outlier detection;

4. We demonstrate that, compared to naïve approaches, our framework significantly reduces the number of 2-tier cache evaluations needed to identify good configurations by a factor of $5.5\times$ for ARC and $7.7\times$ for LRU;

5. We evaluate our framework for the additional application of seeding the initial population of evolutionary algorithms. Our results show an overall acceleration rate of 34% across a highly diverse set of configurations and datasets; and

6. We release the code library containing all techniques used in this work [37].

The next section provides some background on MRCs, knee-detection algorithms, and MRC cliff removal techniques. Section 5.3 presents the point-selection techniques used in our framework, leading to the design of the Z-Method algorithm in Section 5.4. We evaluate all of our techniques ability to find key

points in MRCs in Section 5.5. We then present an additional evaluation of how our framework can be used to optimize multi-tier caching systems using evolutionary algorithms in Section 5.6. Finally, we summarize our conclusions and highlights in Section 4.5.

## 5.2 Background

### 5.2.1 Miss Ratio Curves (MRCs)

A key feature of some MRCs is their monotonicity. Cache replacement algorithms such as LRU are stack-based, which means they satisfy the *cache-inclusion property:* the content of a cache of size $n$ is always a subset of a cache of size $n + 1$. Ultimately, this property ensures that the miss ratio will never degrade as we increase the size of the cache, producing a monotonically decreasing curve. However, more sophisticated algorithms such as ARC [94] are not stack-based and thus the inclusion property does not hold, causing them to produce MRCs that may contain both convex and concave regions [142]. Thus, non-stack-based MRCs need not be strictly decreasing.

### 5.2.2 Knee-Detection Algorithms

Many heuristic algorithms have been developed that find a single knee in a curve, although the precise definition of a "knee" varies. One can define a knee point as the point with the maximum curvature in a function. For continuous functions, curvature [44] is mathematically defined as follows:

$$K_{f(x)} = \frac{f''(x)}{(1 + f'(x)^2)^{\frac{3}{2}}} \tag{5.1}$$

However, knee-detection algorithms are applied to discrete sets of points, instead of a well-defined continuous function. As such, there are several methods to measure the curvature of the discrete sequence. Menger curvature [129, 117] defines the curvature for a sequence of three points as the curvature of the circle circumscribed by those points. This method relies only on a local criterion, using only three points to estimate the knee point without considering any others. As such, noisy data can lead to poor accuracy when estimating the knee point. We use this method as a baseline reference to compare with other methods.

The $L$-method [115] fits two straight lines from the head of a curve to a candidate point, and from the candidate point to the curve's tail. The candidate that minimizes the Root Mean Squared Error (RMSE) between the straight lines and the points of the curve is returned as the knee point; this represents the sharpest angle in the curve.

Similar to the $L$-method, Dynamic First Derivative Thresholding (DFDT) [8, 9] tries to identify the point where the function has a sharp angle. Instead of fitting two straight lines, this method relies on the first derivative of the curve. After computing that derivative, a thresholding algorithm is used to identify the value that separates the derivative values as "high" or "low." The knee is then the point with a derivative value that is closest to the previously computed threshold.

Kneedle [117] uses the point on the curve that is furthest away from a line defined by the head and tail points of the curve. Both axes of the original curve are normalized to $[0, 1]$ to easily find the point with maximum curvature. Kneedle was designed for single or multi-knee detection in a streaming scenario where new data is arriving continuously. The authors used this technique to detect relevant points for network congestion control and latency.

There are several algorithms that can find multiple knee points in a curve, but they have limitations that make them unsuitable for MRCs. The Kneedle algorithm's primary use case is anomaly detection, where it serves as an initial filter to reduce the number of candidates needing further analysis. As such, for Kneedle, recall is more important than precision: it aggressively captures all anomalies, producing many false positives. In some cases it is possible to reduce the number of false positives, but doing so requires extensive tuning of its sensitivity parameter.

A few multi-knee detection algorithms have been developed for use in multi-objective optimization problems, where the notion of a knee guides the exploration of meaningful candidate solutions [158]. However, these problems use a stricter definition of a knee that assumes a set of well-behaved, Pareto-optimal points. Several other knee-detection methods [115, 8, 9, 129, 7] are only effective at finding a *single* knee in a small and relatively smooth set of points. In contrast, MRCs can consist of a relatively large number of points, can be noisy or non-monotonic, and commonly contain more than one significant knee. In this work, we had to develop techniques to overcome these limitations (see Section 5.3) by enabling these algorithms to find multiple knee points.

### 5.2.3 Cliff Removal Techniques

An alternative to detecting knees in an MRC is to modify the underlying cache-replacement policy so that it does not have any cliffs, yielding a *convex* MRC. Talus [12] removes cache performance cliffs by dividing the cache into two *shadow partitions*, each receiving a fraction of the input load. Varying the sizes and input loads of each partition emulates the behavior of smaller or larger caches. Given an MRC as input, Talus computes the partition sizes and their respective input fractions to ensure that their combined aggregate miss ratio lies on the convex hull of the original MRC. Originally proposed for processor caches, Talus inspired the SLIDE [142] technique for removing performance cliffs from software caches that employ sophisticated non-LRU replacement policies. CliffHanger [28] applied a similar idea to key-value web caches, but instead estimated the MRC gradient without explicitly constructing one.

The recent eMRC [87] technique generalized Talus's cliff removal to multi-dimensional *miss ratio functions*, such as the three-dimensional miss-ratio surface for a two-tier cache. The eMRC convex-hull approximation technique leverages the absence of cliffs to efficiently generate the miss ratio function for a multi-tier cache. However, eMRC *requires* convexity, which limits its applicability to modeling multi-tier cache systems that employ cliff removal. As real-world multi-tier cache systems do not yet perform cliff removal, eMRC is unable to approximate their non-convex MRCs. In contrast, our approach does not require convexity to accelerate multi-tier cache evaluations, making it broadly applicable to production deployments of existing caches.

### 5.2.4 Evolutionary Algorithms: Population Initialization

The initial population of an evolutionary algorithm functions as the first guess at a set of good solutions to an optimization problem. The quality of this first set can significantly influence the quality of the final solution and the speed at which an algorithm converges [3, 68]. Studies have shown that some evolutionary algorithms, such as Genetic Algorithms, are more sensitive to the initial population, while other algorithms like Particle Swarm Optimization are less dependent on the initial population [40]. This sensitivity has also been shown to be problem-dependent, such that an algorithm may be influenced by the initial population for certain functions, numbers of dimensions, or population sizes [3].

There are several categories of population-initialization techniques. Common stochastic variants, such as random initialization, are favored for their simplicity,

generic nature, and applicability to a wide range of problems [90]. They are also popular because they produce a relatively uniform distribution as the population size increases. But statistical methods such as Latin Hypercube sampling [98] and quasi-random sequences (*e.g.*, Halton sequence [136]) have been shown to outperform random initialization for most problems [11].

There are also application-specific techniques aimed at specific, narrow problems. They often exploit domain knowledge or use a problem's characteristics with specific evolutionary algorithms. These methods have been applied to improve convergence speed in problems such as grammar-guided genetic programming [42] and flexible job-shop scheduling [159]. Initialization techniques in this class typically perform better than more generic variants, but they are usually limited to specific problems.

Lastly, there are domain-agnostic, general heuristics applicable to problems that meet some set of conditions. Examples include approaches developed to optimize any two-stage stochastic mixed-integer problem [130]. Such techniques can be applied to any problem where some of the variables are constrained to integer values.

The techniques described in this article focus on a domain-agnostic, generic approach that can be applied to any problem where a curve that is correlated to the solution can be derived from features of the input data. We show how such techniques can be used on miss ratio curves to optimize a cache with evolutionary algorithms.

## 5.3 Point Selection Techniques

In this section, we introduce our framework for finding multiple key points in MRCs. We first designed a pre-processing stage to deal with the large volume of data (Section 5.3.1). Next, we made substantial modifications to each point-selection technique, enabling them to output a set of multiple knees instead of just one (Section 5.3.2). Finally, we added a post-processing stage (Section 5.3.3) that filters and ranks knees based on an appropriate definition.

### 5.3.1 Pre-Processing

A curve can contain an arbitrary number of data points. The largest MRC that we evaluated contains 276K points even after sampling-based size reduction [142];

the original MRC is 10,000× larger. However, the knee-detection algorithms evaluated in this work were originally designed to work with small or partial data, such as for clustering optimizations. Our main idea is to reduce the number of points while preserving those that follow the shape of the curve; this greatly reduces the computational costs of subsequent steps while also improving knee-detection accuracy by minimizing irrelevant fine-grained variation.

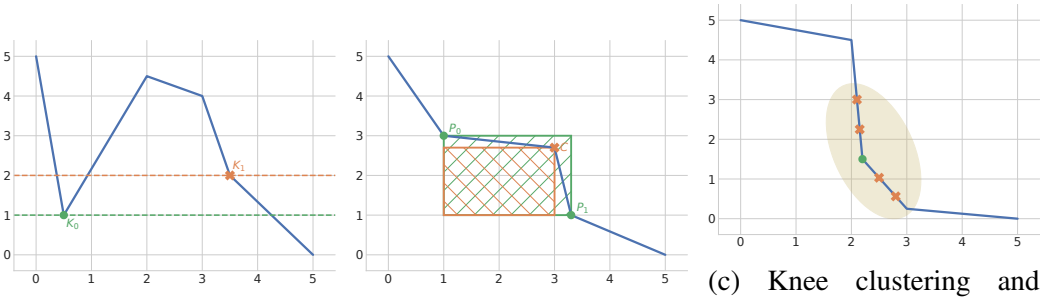
The Ramer-Douglas-Peucker (RDP) algorithm modifies a curve by finding a similar one with fewer points [110]. RDP fits a line between the curve's endpoints and then finds the point in between that is farthest from this line. If the distance between that point and the line is over a given threshold, the curve is split there and the algorithm is reapplied recursively on the two new segments. Once the distance is smaller than the threshold, *all* intermediate points are removed. The main drawback of RDP is the need to define a threshold, which can be understood as the maximum allowed reconstruction error. The choice of threshold is difficult because it depends on the curve's complexity.

We modified the original RDP algorithm to address this difficulty. Instead of defining a threshold for the maximum allowed perpendicular distance between a point and the fitted straight line, we use a relevance-based cost metric that computes the difference between the fitted straight line and the data points in the current segment.

We evaluated four different metrics that assess how far our linear reduction is from the original data: Root Mean Squared Log Error (RMSLE), Root Mean Squared Percentage Error (RMSPE), Relative Percent Difference (RPD), and symmetric mean absolute percentage error (SMAPE). Of these four, the best performance came from SMAPE: it found the smallest set of points that minimized the reconstruction error.

### 5.3.2 Methods

Except for Kneedle, the algorithms we evaluate in this work (see Section 5.2.2) were not designed to detect multiple knees. Thus, we developed a recursive algorithm that can be used to adapt any single-knee detection technique to handle multiple knees. The basic idea is to use a single-knee technique to select the best knee in a segment. We then split the current segment at that knee, and for each new segment check whether it is sufficiently linear (computed using the SMAPE metric). If not, we repeat the process recursively. Apart from applying this recursive generalization, we do not alter the core knee-detection technique, using it as a black box. All of the methods we evaluated, even Kneedle, require our pre- and




(a) Undesirable knee filter. (b) Cliff-point removal filter. (c) Knee clustering and ranking.

Figure 5.2: Graphical representation of the post-processing methods. (a) Representation of the filter that removes unwanted knees, knee $K_1$ is removed since knee $K_0$ achieved better performance. (b) Representation of the overlapping rectangles used by the corner detection algorithm. Point $C$ is identified as a cliff point and then removed. (c) Representation of the clustering and ranking elements. The orange ellipse represents the cluster of knee candidates. The orange candidates are filtered out, while the green knee is selected as the best representative based on Equation 5.2.

post-processing methods to work properly on MRCs.

### 5.3.3 Post-Processing

Given the differences between single- and multi-knee detection and the large number of points produced by using our recursive strategy on some of the knee-detection algorithms, we developed three different filters to further reduce and select the most relevant knees.

The first filter, shown in Figure 5.2a, removes useless knees. When dealing with non-monotonic curves, a knee-detection algorithm can incorrectly choose a knee that is *above* a previously detected one. We remove such knees since they are sub-optimal and do not add useful information.

The second filter, shown in Figure 5.2b, removes cliff points located after a smooth, near-horizontal area that precedes a sharp descent. These points are found using a corner-detection algorithm that computes the overlapping area of two rectangles. The first rectangle, shown in green, is drawn from the neighbor points $P_0$ and $P_1$ (assuming that RDP pre-processing was used, these are the previous and following points) that are adjacent to the knee candidate point $C$. The second rectangle, drawn in orange, has its corners placed at $C$ and the lower left of

the green rectangle. The filter computes the percentage overlap between these two rectangles, and a knee candidate is removed if the overlap exceeds a threshold.

The third and final filter, shown in Figure 5.2c, uses a hierarchical clustering algorithm to group knees by their distance along the $x$-axis, using a percentage of the $x$ range as a threshold. After grouping the knees into clusters, the knees within each cluster are ranked based on their relevance score, computed from two metrics: (i) the improvement given by each knee (*i.e.*, how much it decreases on the $y$-axis from the highest knee in the cluster) and (ii) the smoothness of the improvement, computed using the coefficient of determination ($R^2$). Specifically, the relevance score $S$ is given by Equation (5.2):

$$S(K_i, \vec{L}) = |K_h - K_i| \cdot R^2(\vec{L}), \tag{5.2}$$

where $K_i$ is the $i^{th}$ knee, and $K_h$ is the knee with the highest value on the $y$-axis (in a single cluster). $\vec{L}$ is a vector containing all the knees in the cluster up to and including the $i^{th}$ one: $\vec{L} = [K_0, ..., K_i]$. The highest-ranked knee in each cluster is selected as its representative knee.

## 5.4 Z-Method

### 5.4.1 Design Concepts

Our design for Z-Method was inspired by the DFDT [8] and DSDT [9] knee-detection algorithms, which use first and second derivatives, respectively. In statistics, a *z-score* (also known as a *standard score*) is a transformation that normalizes a data value by quantifying how many standard deviations away it is from the mean; typically, a point whose z-score has an absolute value greater than three is considered an outlier [2]. For the purpose of detecting knees, such outliers in the second derivative indicate a significant change in the y-axis. The foundation of our Z-Method technique is in detecting such outliers and intelligently selecting knees among them.

Although the second derivative is useful, we found that large and small knees often tend to cluster, causing several points in close proximity to be selected, rather than the single most optimal knee in the vicinity. To remedy this, we introduced two hyper-parameters, $dx$ and $dy$, that specify the minimum $x$ and $y$ distances, respectively, between all selected points. These parameters limit the total number of knees selected and give users control over the algorithm. For ex-

ample, users interested only in large knees can give relatively high values for $dx$ and $dy$ to minimize the number of points.

Z-Method was designed to function independently of the techniques described in Sections 5.3.1 and 5.3.3. As such, it works for curves that are non-monotonic, with both convex and concave regions (see Section 5.2.1).

### 5.4.2 Algorithm Description

**Algorithm 1:** Z-Method Multi-Knee Detection

**Input:** Data $D$ with $(x, y)$ points, $dx, dy, dz$
**Output:** List of $(x, y)$ points corresponding to knees

$\Delta x \leftarrow \text{length}(D) \cdot (dx/100)$
$\Delta y \leftarrow (\max(y) - \min(y)) \cdot (dy$ / 100)
$D'' \leftarrow$ calculate second derivative of $D$
$Z \leftarrow$ calculate z-scores for $D''$
$K \leftarrow$ empty list
$zLimit \leftarrow 3$ # standard outlier threshold [2]
**while *TRUE* do**
    $C \leftarrow$ points in $Z$ with z-score $\geq zLimit$
        **and** at least $\Delta x$ and $\Delta y$ apart from all points in $K$
    $Z \leftarrow Z$ - $C$
    **if** $zLimit \leq 0$ ***and*** $\text{length}(C) == 0$ **then**
        Remove points from $K$ to ensure that $y$ always decreases as $x$ increases
        **return** $K$
    $G \leftarrow$ group all points in $C$ such that all adjacent points in each group are $< \Delta x$ apart
    sort $G$ in descending order by max z-score of each group
    **foreach** *group in* $G$ **do**
        $p \leftarrow$ point in $group$ with the lowest $y$ value
        **if** $p$ is at least $\Delta y$ from all points in $K$ **then**
            $K.append(p)$
    $zLimit \leftarrow zLimit - dz$

As shown in Algorithm 1, Z-Method takes as input a discrete curve *D* consist-

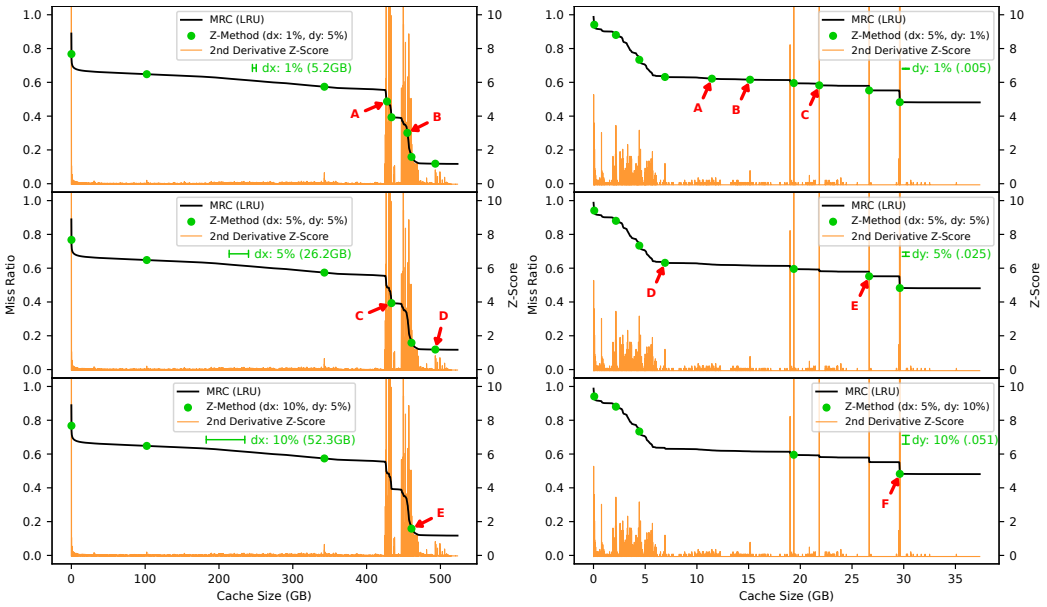
ing of an ordered list of $(x, y)$ points, along with parameters $dx$, $dy$, and $dz$. The parameters $dx$ and $dy$ both influence the size and number of selected knees, while $dz$ controls the maximum number of iterations in the main selection loop (lines 7–20).

We first convert $dx$ and $dy$, specified as percentages, into absolute values $\Delta x$ and $\Delta y$ for the input curve (lines 1–2). This normalization ensures that these parameters function similarly for different curves. We then approximate a list of second derivatives of the curve, $D''$, using second-order polynomial fitting [21]; next we calculate the z-scores of all points in $D''$ as $Z$, both of which are found in linear time (lines 3–4). We initialize a list $K$ to contain all selected knees, and set our starting value of $zLimit$ to 3, since a z-score $\geq 3$ is a widely accepted value for outliers [2] (lines 5–6).

We then enter the main selection loop (lines 7–20), which selects points and progressively decrements the $zLimit$ value. First, we create a new list $C$ that contains candidate points: points in $Z$ that have a z-score greater than the current $zLimit$ and are at least a minimum $\Delta x$ and $\Delta y$ distance from all other already-selected points (lines 8–9). The complexity of this step is $O(|C| \times |K|)$. All candidate points $C$ are removed from $Z$ so that we will not consider them again in future iterations (line 10). The termination clause is then checked (lines 11–13) to ensure that we have candidate points to operate on.

We next group the candidate points $C$ into $G$, such that the adjacent points in each group are less than $\Delta x$ apart, based on the $dx$ parameter constraint (line 14). This takes $O(|C|)$ time, effectively forming groups of points such that there is at least $\Delta x$ distance between every group. We then sort the groups in $G$ in descending order by the maximum z-score of each group (line 15). Here, we are sorting the location of each group in the list of groups $G$ rather than the points within each group. From each group, we select the point with the lowest $y$ value (line 17); we then check that the selected point is not within a minimum $\Delta y$ distance from other points that have already been selected, enforcing the $dy$ parameter (line 18). The complexity of this loop is $O(|G| \times |K|)$. A point is added to the list of knees $K$ if it satisfies this constraint (line 19). We then decrement the $zLimit$ by $dz$ and continue with the next loop iteration (line 20).

This loop terminates only after we have reached a $zLimit \leq 0$ and there are no remaining points that can be selected given the $dx$ and $dy$ parameters (line 11). At $zLimit = 0$, we consider all points in $D$ that have not already been considered in previous iterations. By starting at z-score $\geq 3$ and iteratively approaching 0, we select the largest knees first and gradually lower our threshold for how big a knee should be.




(a) Effects of Z-Method parameter $dx$. (b) Effects of Z-Method parameter $dy$.

Figure 5.3: (a) Effects of $dx$ = 1% (top panel), 5%, and 10% on trace w09. Red arrows denote points in a panel that were not selected when the $dx$ value increased (next panel down). (b) Effects of $dy$ = 1% (top panel), 5%, and 10% on trace w62. Red arrows denote points in a panel that were not selected when the $dy$ value increased (next panel down). The minimum distance that Z-Method enforces between points is shown below the legends in green.

Finally, we eliminate any points that may have been poorly selected due to non-monotonicity in the curve. A final pass removes points where increasing the $x$ value makes the $y$ value worse (line 12); in our MRC application, such points are clearly undesirable. This simple pass requires time linear in the size of $K$. The overall complexity of this algorithm is therefore $\mathcal{O}(|D| \times |K|)$, where $D$ is the size of the input curve and $K$ is often a trivially small value. For example, with $dx$ set to 5%, enforcing at least 5% distance on the $x$-axis between each selected knee point, the maximum size of $K$ would be 20.

### 5.4.3 Parameters

We present qualitative evaluations of the algorithm's parameters $dx$ and $dy$, as well as its overall success at finding *key points*. Furthermore, we demonstrate that

Z-Method is effective for both stack and non-stack algorithms, by evaluating with LRU and ARC cache replacement policies.

**Parameter: *dx*** The $dx$ parameter has several functions within Z-Method. It is provided as a percentage of the maximum cache size in the given MRC. The most transparent effect of $dx$ is that it constrains the minimum $x$ distance, or cache size, between selected points. Since no two points can have an $x$ distance less than $dx$ between them, this provides an upper bound on the total number of selected points, and also influences the number of points that are actually selected. Because it affects the "grouping" stage of the algorithm, $dx$ also effectively defines the width of the knees.

Figure 5.3a shows the effects of $dx$ on workload w09 with LRU cache replacement, with $dy$ fixed and $dx$ set to 1%, 5%, and 10%. The black line in each plot represents the MRC for LRU cache replacement. The green dots are the points selected by Z-Method when using the $dx$ and $dy$ parameters indicated in the legend. The vertical orange lines show the second derivative z-score of the MRC at each cache size. Because the z-score values have a large, workload-dependent dynamic range, we truncate them at 10 in this plot and for the remainder of this article. A z-score range up to 10 is sufficient to identify all points considered as outliers (*e.g.*, z-score $\geq 3$).

For the MRC plotted in Figure 5.3a, we will focus on the knee(s) in the region of cache sizes between approximately 425GB and 475GB. In the top plot with $dx = 1\%$, Z-Method considers this region to contain four separate knees, since their distances from each other are at least 1% of the maximum cache size. When we move from 1% to 5% in the middle plot, we can see that points A and B from the top plot have been removed. Those points are no longer within $dx$ of each other, so they are grouped together; we are now left with two points at wider, more prominent, knees.

A similar effect is seen when we increase $dx$ from 5% to 10% in the bottom plot. The two knees at points C and E are grouped together and C is removed. Point D is also removed, since its cache size is less than 10% away from point E. Significantly, the knee point E was favored rather than the less interesting point D.

**Parameter: *dy*** The $dy$ parameter is also specified as a relative percentage, which is then converted into an absolute value for the given MRC. It functions similarly to $dx$, except that it constrains the $y$ distance, or delta miss ratio, be-

tween any two selected points. This effectively influences the height of knees and how many points are selected, while providing an upper bound on the total number of points that can be selected.

Figure 5.3b shows the effects of $dy$ using workload w62 with LRU cache replacement by fixing $dx$ and varying $dy$ between 1%, 5%, and 10%. The format is otherwise the same as in Figure 5.3a. In the top and middle plots, the most interesting change occurs at point C. With $dy = 1\%$ in the top plot, this very small knee is considered significant and is selected. However, when we increase $dy$ from 1% to 5% in the middle plot, points A, B, and C are removed, as the $y$ distance between these points and adjacent points is no longer less than $dy$. Similarly, point E is removed when we move from 5% to 10% in the bottom plot, while the taller knee point F is retained. We can also see that point D is removed as well, as increasing $dy$ reduces the number of selected points.

It is important to note that for both of these parameters, we are not guaranteed to *always* have a point that is $dx$ or $dy$ apart from every other point. Enforcing a hard separation rule would add a great deal of complexity and would provide little benefit, since we already select points by their order of importance.

**Parameter: *dz*** The $dz$ parameter controls the amount that the $zLimit$ variable is decremented in each iteration of Z-Method. This affects the overall running time by influencing the total number of iterations. It can also affect the size of candidate point groups. For example, with a starting $zLimit$ of 3, $dz = 0.1$ would only consider points with a z-score $\geq 2.9$ on the second iteration, but $dz = 0.5$ would consider a potentially larger set of points that have a z-score $\geq 2.5$. While this may seem significant, the $dx$ and $dy$ parameters are still the predominant influence on how groups are formed, so we did not observe any trends or significant changes when modifying $dz$.

**Finding key points** In Figure 5.4, we show the points selected by Z-Method with $dx$ and $dy$ set to 5%, for multiple workloads using both LRU and ARC cache replacement policies. We evaluated these plots based on whether or not they selected all of the points that we consider *key points*. To reiterate, Z-Method should first select the largest knee points and then eventually select those within any regions that cover at least 5% of the $x$ and $y$ axes. The first row of plots (LRU1-3) shows examples where Z-Method performed well for LRU. All prominent knees were selected and large ranges of cache space with gradual decreases in miss ratio also contained an adequate number of points. The second row of plots (LRU4-6)

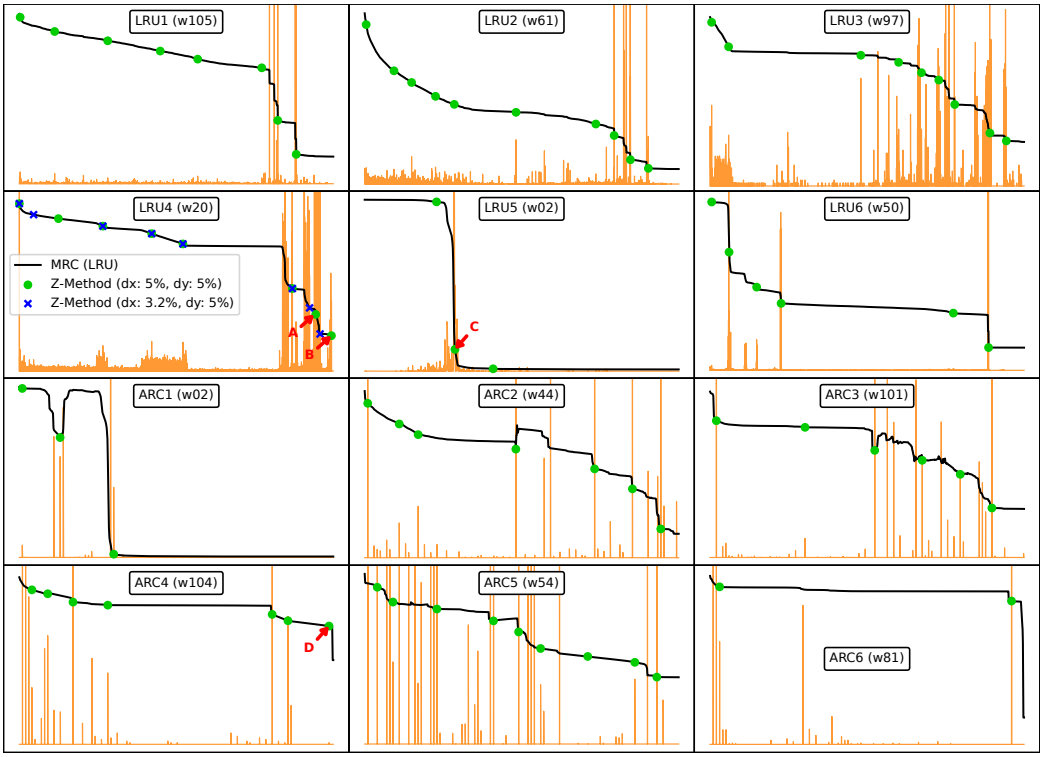



Figure 5.4: From the top: the first row shows LRU plots where Z-Method picked fairly good points; the second row has LRU plots where Z-method missed a few, better points. The third and fourth rows are the same but for ARC (third row good points selected; fourth row missed some points). The plot labeled LRU4 shows points selected by Z-Method using a $dx$ of both 5% (green) and 3.2% (blue). Sub-optimal points A and B in LRU4 were selected using $dx$ of 5%, missing nearby knee points. The knees were appropriately selected when $dx$ was lowered to 3.2%. Sub-optimal point C in LRU5 was selected due to the extreme shape of the curve and the standard z-score threshold of 3. The knee was appropriately selected when lowering the threshold. Sub-optimal point D in ARC4 was selected due to the ARC MRC being generated with too few points. The knee point to the immediate right was selected when the number of points in the MRC was increased from 100 to 220.

shows examples where Z-Method missed key points. For example, in plot LRU4, points A and B missed the knee points directly to their left. There were similar issues in LRU5 and LRU6.

We show how to improve the lower-quality points A and B in LRU4 by modifying Z-Method parameters. The green points were selected using the default $dx$ of 5%, while the blue points were selected using a $dx$ of 3.2%. These blue points more accurately capture these two knees, and are more optimal than A and B.

There were also cases where Z-Method could pick slightly less optimal points due to extreme shapes in a curve and the nature of the z-score metric. This can be observed in plot LRU5, which exhibits a nearly flat region followed by a massive, steep knee, then another nearly flat region. Point C is not quite at the bottom of the knee because the z-score at the very bottom was slightly below the standard threshold for an outlier of 3. This could be remedied by lowering the threshold (and modifying Z-Method's parameters if needed). It should be noted that Z-Method will still pick a point that is relatively close to the knee in these edge cases.

The third row of plots (ARC1-3) shows where Z-Method performed well for ARC. In addition to always selecting prominent knees and points in gradually sloped regions, we also see that points were never selected in concave regions where the miss ratio increased due to the non-monotonicity of ARC. A key feature of Z-Method is that it will never select points with a higher miss ratio that any other previously selected points with a lower cache size.

The fourth row of plots (ARC4-6) depicts a situation where Z-Method missed key points. In cases such as ARC5, modifying the parameters was sufficient for identifying higher quality points, but plots ARC4 and ARC6 exhibit a problem that is unique to non-stack-based cache eviction algorithms (*i.e.*, ARC). Unlike stack-based algorithms (*e.g.*, LRU), we cannot easily generate a fine-grained MRC that includes every potential cache size. Instead, best practice is to sample the workload [142] and then generate the MRC using a subset of points that still preserves the shape of the curve. This is typically done using 100 points. We used 100 points to generate all ARC MRCs during the point selection process throughput this work, but plotted the z-score and points selected against MRCs generated using 1000 points to better show how Z-Method selects points in MRCs that more closely represent the "true" curves. This value worked well for the majority of our workloads, but there were edge cases (*e.g.*, ARC4 and ARC6) where the z-score did not accurately capture important knees present at the very end of the curve. For example, point D in ARC4 was selected because there was a very small knee with a positive z-score at the top of the cliff, but there was a

much larger knee to its immediate right. This could be remedied by increasing the number of points used to generate the MRC. 220 points were enough to remedy this value for both ARC4 and ARC6 (not shown in the plots due to complexity). The task of generating an MRC is separate from Z-Method, so we did not include this value as a parameter.

In all cases where Z-Method missed key points, there were slight modifications that enabled those points to be be selected. We fixed $dx$ and $dy$ to 5%, the z-score outlier threshold to 3, and generated ARC MRCs using 100 points for this evaluation, since we do not yet have a way of automatically selecting the ideal values. Even so, the overwhelming majority of MRCs we looked at still found all key points with these values.

## 5.5 Evaluation: Miss Ratio Curves

In this section, we evaluate our framework's ability to find key points in miss ratio curves. We first compared the accuracy of 8 different knee-detection algorithms, including Z-Method, for identifying knees in single-tier MRCs, and then evaluated our framework's ability to quickly find optimal multi-tier configurations.

### 5.5.1 Experimental Setup

We evaluated our techniques on 106 real-world block traces collected by CloudPhysics [141], each representing a week of virtual disk activity from production VMware environments. We used hash-based spatial sampling [141, 142], with a size-based sampling rate ranging from 0.1 to 0.0001, to reduce these workloads and thus the running time while maintaining an accurate representation of the originals. We dynamically varied the rate by powers of 10, such that each sampled trace was guaranteed to contain between 100K and 1M requests. The traces contain heterogeneous request sizes, so we also transformed all requests into 4KB block-aligned operations to facilitate accurate sampling, consistent with previous work [87].

To evaluate multi-tier systems, we extended PyMimircache [155], a cache simulator with an easily modifiable Python front end and an efficient C back end. Our extension generates two-tier MRCs by simulating an L1 cache with the original sampled trace, then simulating L2 with the requests that missed in L1. L1 cache sizes were selected using the MRC of the original trace, while L2 sizes were chosen using the MRCs of each intermediate trace. The total miss ratio of the two-tier

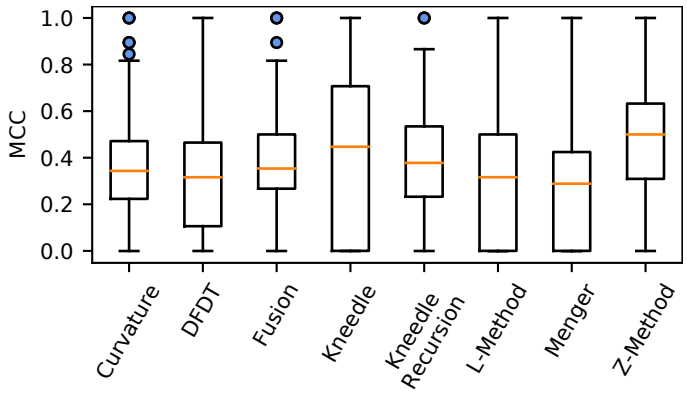



Figure 5.5: An MCC evaluation of 8 knee detection algorithms using our optimized hyper-parameters for accurately identifying knees that were manually selected by experts. Higher MCC values and lower standard deviations are better. Kneedle and Z-Method have the highest median MCC values of 0.45 and 0.5, respectively, with Z-Method achieving much tighter bounds.

configuration was calculated as the product of the miss ratios of L1 and L2. We modeled a simple write-back policy by treating both reads and writes as cache references, as done in previous work [87]. The cache eviction policy was configured as either LRU or ARC and was the same in both tiers. We generated two-tier MRCs for each trace, for both LRU and ARC replacement policies, resulting in a total of 212 MRCs.

### 5.5.2 Knee-Detection Algorithms

We evaluated the accuracy of our framework using Z-Method and several other knee-detection algorithms: Curvature, DFDT, Kneedle, L-Method, and Menger. We also included the *Fusion* method, which considers all points retained by RDP and relies on our post-processing filters to select relevant knees.

Kneedle finds knees using peak-detection methods, and can be used for single-knee detection by selecting only the highest possible peak; we call that approach *Kneedle Recursion*. We analyzed each method's ability to find knee points that had been manually curated in the 212 single-tier MRCs by four domain experts.

Most of our techniques have one or more hyper-parameters that can influence which points are selected. To achieve the best performance for each MRC, it is necessary to tune the hyper-parameters of each algorithm appropriately. While

the default parameters offered acceptable performance, a more complete evaluation requires optimized parameters [17]. Therefore, we developed a cost function and ran an optimization algorithm for each knee-detection method using all 212 MRCs.

When designing the cost function, we carefully considered the target use case of our framework. We want to find the ideal parameters that have the lowest error globally across all MRCs, while keeping the number of knees as close as possible to the number of knees identified manually. Constraining the number of knees is necessary since our framework is designed to pick only the most relevant points. A technique that finds parameters that correctly identify all knee points but also selects many non-relevant points, while having a high precision score, would be inefficient for our use case.

We use the Matthews correlation coefficient (MCC) [25] as the basis of our cost function. MCC measures classification quality by considering the balance ratios of the four confusion matrix categories: true positives and negatives, and false positives and negatives. Although the knee-detection problem is better modeled as a regression, we based our evaluation on binary classification, since we wanted to control the impact of false positives and negatives (*i.e.*, non-relevant points being classified as knees and vice-versa). Prior work [25] has shown that the MCC is more informative than an F1-score for evaluating accuracy in binary classification problems. As such, the cost function we used was the following:

$$\mathit{Cost}(E, K) = \frac{1}{n}\sum_{i=0}^{n} \mathit{MCC}(E_i, K_i) + \max\left(\left(\frac{1}{n}\sum_{i=0}^{n}|K_i|\right) - K_t, 0\right), \quad (5.3)$$

where $E$ represents the expected (manually selected) knees for all MRCs, and $K$ represents knees picked for all MRCs ($n$ defines the number of MRCs) by our framework using some configuration of hyper-parameters and a knee-detection algorithm. $\mathit{MCC}(E_i, K_i)$ represents the MCC computed from the expected and detected knees of the $i^{th}$ MRC. Finally, $|K_i|$ represents the number of knees detected in the $i^{th}$ MRC, and $K_t$ represents a threshold for the acceptable number of knees.

Figure 5.5 shows our evaluation. Three techniques stand out: Fusion, Kneedle, and Z-Method. Fusion achieves tighter margins than all other techniques, spanning only 0.23 MCC between the upper and lower quartiles, suggesting that the expected performance in unseen traces would be well-bounded. Kneedle and Z-Method achieve the highest median MCC values of 0.45 and 0.50, respectively, with Z-Method having a smaller standard deviation when compared with Kneedle.

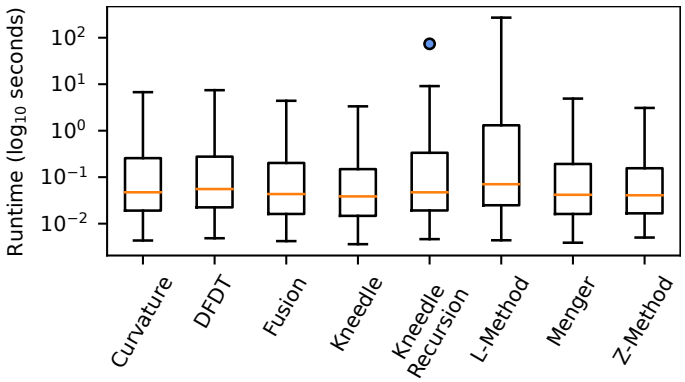



Figure 5.6: Running times for 8 knee-detection algorithms. The y-axis scale is logarithmic. L-Method displays a considerably higher running time than other methods, due to its high complexity. Other methods are comparable, with Kneedle and Z-Method having the lowest median running times.

The much tighter bounds of Z-Method are more significant than the improvement in median, making Z-Method the ideal candidate for our multi-tier evaluation.

We also experimentally measured the time and memory usage of our framework using each knee-detection algorithm for all MRCs. We used the default hyper-parameters for each algorithm, as they do not significantly impact the overheads. The results of the running-time benchmarks are shown in Figure 5.6.

All of the algorithms have comparable execution times across all MRCs with the exception of L-method. This is especially true for the upper quartile of L-Method, which is 1307.79 ms. The second highest upper quartile is Kneedle Recursion, which is 74.4% lower than L-Method at 334.62 ms. This is expected since L-method uses straight-line fitting ($O(n^2)$) for each point to detect the ideal knee, leading to a time complexity of $O(n^3)$. Combined with our recursive algorithm that enables it to detect multiple knees, the expected time complexity for L-method is $O(n^3 \log n)$. Note that Kneedle (non-recursive version) and Z-Method have the lowest median running times of 38.59 ms and 40.81 ms, respectively.

The memory overhead is nearly linear in the file size for each MRC, ranging from approximately 53KB to 71MB after sampling. There were no significant differences across any of the techniques, so we do not present any further memory overhead analysis.

### 5.5.3 Multi-Tier MRCs

Miss-ratio curves are typically used to find configurations that minimize both miss ratio and cache size(s). We seek multiple configurations that are optimal in two or more objectives.

Consider designing a multi-tier cache, with many device options, for a large workload. With an unlimited budget, one could simply purchase enough DRAM to hold the entire data set, but that is rarely economical. Instead, most system administrators will want to trade cost off against performance, meaning that they will be interested in *Pareto-optimal* solutions, *i.e.*, those where a given objective cannot be improved without making one or more others worse.

Only a subset of all possible cache configurations are Pareto-optimal. When a set contains every Pareto-optimal configuration for a given workload and no others, it is called the *true Pareto-optimal front*. Any point in this front minimizes the cache size(s) and the miss ratio; the front as a whole can be considered to mark the "best" points.

However, it is often not feasible to find the true Pareto front for a large configuration space. Instead, the space can be sampled in an attempt to find optimal points, creating a *Pareto approximation*. Our work aims to find a minimal number of key points in MRCs. Thus, we are trying to find the most significant Pareto-optimal points by efficiently generating accurate Pareto approximations of multi-tier MRCs.

There are multiple metrics for evaluating the quality of Pareto approximations [78]; the most commonly used is the *HyperVolume Indicator (HVI)* [16], which measures the size of the space between the points in a front and a user-defined reference point; a larger space is better.

Figure 5.7 shows an example of how HVI is measured in a 3-dimensional space. The blue shape represents a simple linear series descending from (0,0,10) to (10,10,0). If this were a two-tier MRC, the $x$-axis would be the L1 size, $y$-axis the L2 size, and the $z$-axis the miss ratio. The hypervolume is the volume between points on the Pareto front (here, the blue shape) and a user-defined *reference point*, here the *nadir point*[1] at (10,10,10), where all objectives are maximized. To find the hypervolume of the point at (5,5,5), we draw a rectangular prism from it to the reference point. The resulting $5 \times 5 \times 5$ cube has a hypervolume of 125. If we were to instead find the hypervolume of the point (4,4,4), we would have a $6 \times 6 \times 6$

[1] Although the reference point is placed at the largest coordinates, prior literature on hypervolume indicators uses the term "nadir" rather than "zenith" because it represents the worst performance; we follow that convention.

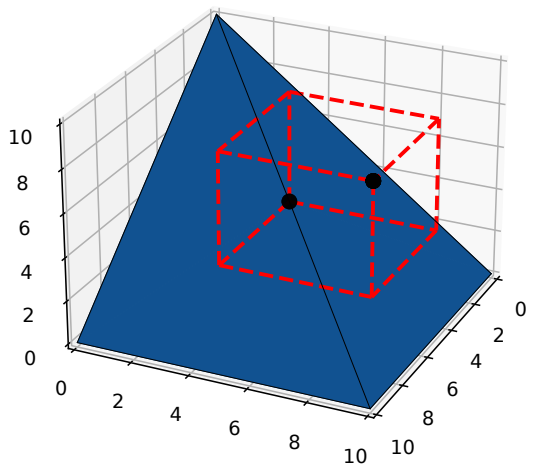



Figure 5.7: An example of how the *HyperVolume Indicator (HVI)* is calculated for a single point of a 3-dimensional data series. A cube is drawn from the reference point (10,10,10) to the data point (5,5,5) creating a $5 \times 5 \times 5$ cube with a hypervolume of 125.

cube with a hypervolume of 216. Thus, configurations with lower cache sizes and miss ratios result in larger hypervolumes. The total hypervolume of a dataset is the non-overlapping hypervolume of all points on its Pareto front, making HVI a useful metric for our multi-knee detection framework.

Another metric highly relevant to our problem is the Ratio of Non-Dominated Individuals (RNI) [127], which is the fraction of dataset points that are on the Pareto front. As discussed earlier, points not on the front represent sub-optimal configurations, so a higher ratio is better. RNI does not measure the magnitude of quality; instead, it informs us of a point selection technique's efficiency. Therefore, evaluating HVI and RNI together is a comprehensive approach to analyzing techniques that find the minimal number of key points in MRCs.

We evaluated our framework across all 212 two-tier MRCs using Z-Method, compared to a naïve approach of selecting evenly-spaced points. We also tried geometrically-spaced points, but this yielded worse results than even spacing so we omit them from this analysis. It was not practical to evaluate every point in MRCs containing thousands of points, so we used 50 evenly-spaced points (Even50) as a reasonable approximation of the full configuration space and the true Pareto front. We varied the number of evenly-spaced points to most closely match Z-Method's average HVI or number of points, resulting in Even4, 10, and

| Method | Avg. Points | | Avg. HVI % | | Avg. RNI | |
|---|---|---|---|---|---|---|
| | ARC | LRU | ARC | LRU | ARC | LRU |
| Even4 | 20 | 20 | 59.63 | 61.00 | 0.30 | 0.28 |
| Even10 | 110 | 110 | 86.43 | 83.31 | 0.39 | 0.40 |
| Even13 | 182 | 182 | 90.41 | 91.07 | 0.41 | 0.34 |
| Even50 | 2550 | 2550 | 100.00 | 100.00 | 0.33 | 0.34 |
| Z-Method | 23.33 | 20.33 | 86.99 | 90.75 | 0.94 | 0.97 |

Table 5.1: Evaluation results of our framework using Z-Method across 2-tier ARC and LRU MRCs, derived from 106 real-world block traces collected from CloudPhysics. The averages of 3 metrics are presented for each algorithm: number of points (lower is better), HyperVolume (HVI) as a percentage of Even50's HyperVolume (higher is better), and Ratio of Non-Dominated Individuals (RNI) (higher is better).

13.

In Table 5.1, we show the averages across all 212 MRCs of the number of points selected, HVI as a percentage of Even50's HVI, and RNI. When measuring the efficiency of a method, a lower number of points and a higher RNI are better; when measuring the accuracy of a method, higher HVI is better. The number of points for even spacing is always constant, calculated as $X + X^2$ for two-tier MRCs using Even*X*. Z-Method has HVI similar to that of Even10 for ARC and to Even13 for LRU, but Z-Method evaluates $5.5\times$ fewer points for ARC and $7.7\times$ for LRU to get those results. This efficiency is also reflected in Z-Method's RNI of 0.94 for ARC and 0.97 for LRU. Conversely, the RNI of the evenly-spaced methods ranges from 0.28 to 0.41, meaning that the majority of points they select are sub-optimal and uninteresting to explore.

In Figure 5.8, we show box plots for all 212 MRCs. Figure 5.8a displays the number of points selected by each technique. We can see that Z-Method and Even4 selected approximately the same median number of points. A significant result is that in the worst case, Z-Method still picked fewer points than Even10. We can also see cases where Z-Method picked very few points. There are times when such a low number of points is appropriate, but this can also represent cases where the default parameters were too conservative, resulting in too few points and a low HVI.

Figure 5.8b displays the HVI as a percentage of Even50's HVI. This figure

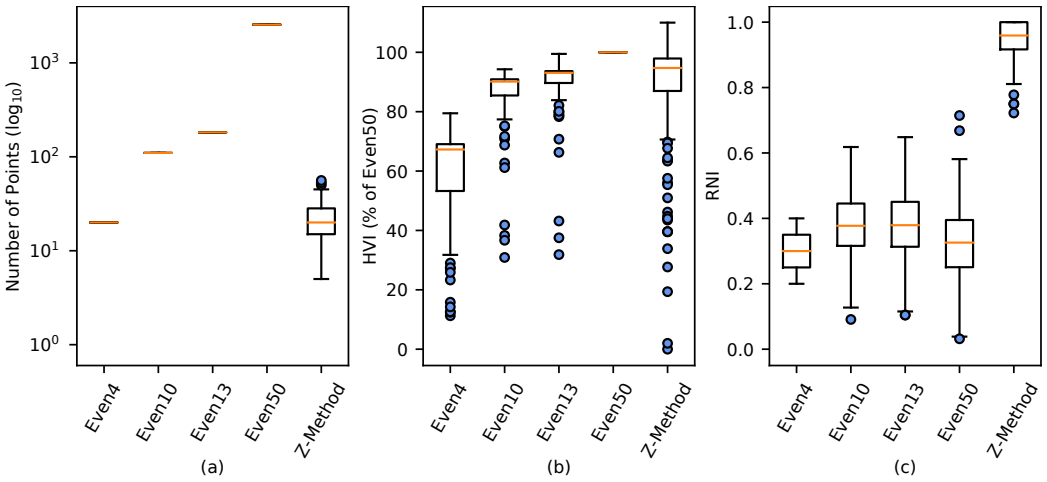



Figure 5.8: Evaluation results of our framework using Z-Method across 2-tier ARC and LRU MRCs, derived from 106 real-world block traces collected from CloudPhysics. Box plots of 3 metrics are presented: (a) number of points (lower is better), (b) HyperVolume (HVI) as a percentage of Even50's HyperVolume (higher is better), and (c) Ratio of Non-Dominated Individuals (RNI) (higher is better).

reveals several outliers where Z-Method performed poorly, but also many cases where it had a higher HVI than Even50. These results inform us about Z-Method's sensitivity to its hyper-parameters. The default parameters worked well for the majority of our workloads, but needed to be tuned better for others. With the right parameters, Z-Method performed better than naïve approaches while selecting a minimal number of points. Lastly, Figure 5.8c displays the RNI. Z-Method consistently had a greater RNI than all of the evenly spaced methods, indicating that it properly identified key points. We can also see that there were diminishing returns when increasing the number of evenly spaced points. The median RNI decreased from Even13 to Even50, meaning that Even50 selected many points that did not contribute to the Pareto front.

In Figure 5.9, we show visualizations of the points chosen by each method for a few selected two-tier MRCs with fairly different characteristics.[2] Figure 5.9a (top row) displays the MRCs for workload *w04* using LRU replacement, where several knees of various sizes are followed by gradually-sloped regions. We can

[2]A similar figure that appeared as Figure 4 in an earlier version of this paper [37] inadvertently showed visualizations for Even5 instead of Even4.

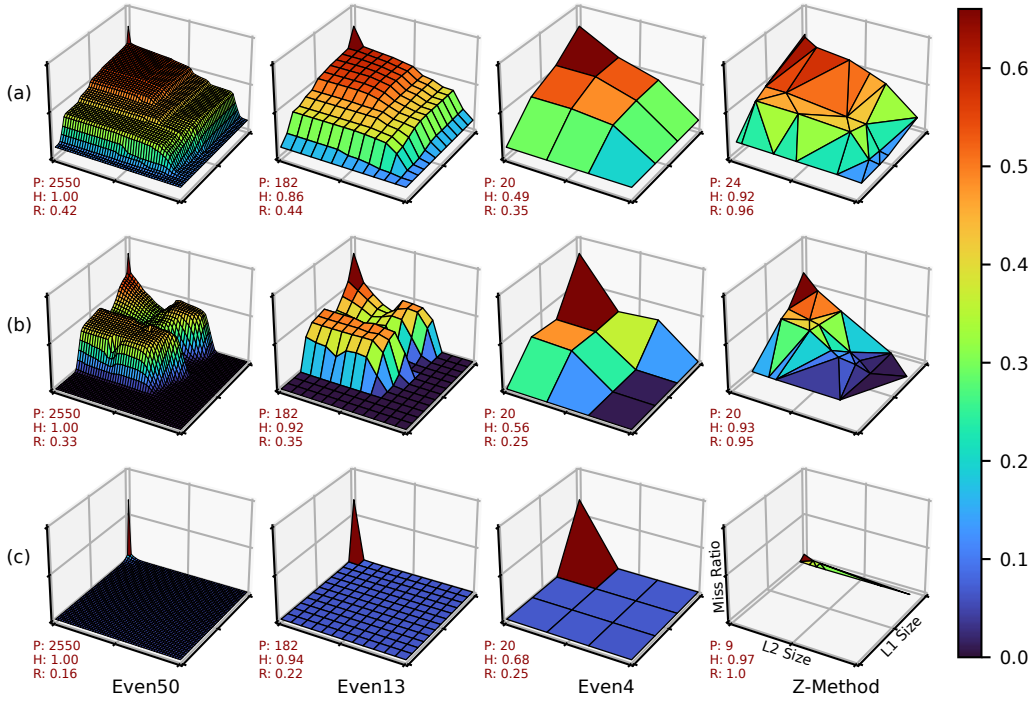



Figure 5.9: Examples of point selection on two-tier MRCs that highlight three different commonly observed scenarios. Each point represents the total miss ratio of a configuration of some L1 and L2 sizes. The $x$ and $y$ axes are the normalized L1 and L2 sizes, respectively, while the $z$-axis is the miss ratio. Axis labels are omitted to reduce clutter. Each row contains MRCs of a single workload using 4 different point-selection methods, listed at the bottom of each column. The $P$ value indicates the total of number of points (lower is better), $H$ is the HyperVolume as a percentage of Even50 (higher is better), and $R$ is the Ratio of Non-Dominated Individuals (higher is better).

see that Z-Method accurately selects each knee, achieving 92% of Even50's HVI while evaluating over $100\times$ *fewer* points. Conversely, Even13 and Even4 perform poorly, selecting points at the tops of the cliffs before the knees, resulting in lower HVI's of 86% and 49%, respectively. When several knees are present, Z-Method has more opportunities to exploit these significant improvements in miss ratio, performing much better than evenly-spaced points.

Figure 5.9b (middle row) displays the MRCs of workload *w66* using ARC replacement, which exhibits large amounts of non-monotonicity, creating several hilly regions. Z-Method finds the interesting knee points at the hill bottoms, while the post-processing filter prevents selecting any points at the hilltops. Z-Method is even more efficient here than in the previous figure while still being highly accurate, selecting only 20 points and achieving 93% of Even50's HVI. Even13 gets close to Z-Method's HVI, but requires $9.1\times$ more points.

Finally, Figure 5.9c (bottom row) displays the MRCs of workload *w06* using ARC replacement, which contains only a couple of interesting points at the very beginning of the plot. Z-Method finds 3 points in this tiny space that are more optimal than those found by Even4 or Even13; it also does not waste time exploring the large, flat MRC region that offers almost no improvement in miss ratio. With only 9 points, Z-Method achieves 97% of Even50's HVI, while Even13 evaluates $20.2\times$ *more* points but achieves only 94% of Even50's HVI. MRCs that contain only a handful of good points are fairly common, even in multi-tier settings, and our framework dramatically reduces the time spent exploring them.

## 5.6 Evaluation: Population Initialization

In this section, we show how our multi-tier knee detection framework can also be applied to population initialization for evolutionary algorithms, to search large configuration spaces more efficiently [43, 33]. In many cases, evaluating the fitness of a configuration is an expensive operation, making the speed of convergence particularly important. The initial population of an evolutionary algorithm functions as the first guess at a set of good solutions, so the population's quality can significantly influence the quality of the final solution and the speed at which an algorithm converges [3, 68]. As such, heuristics to intelligently select a population have been developed for a variety of scenarios and optimization problems [136, 11]. Evaluating multi-tier caching systems fits this scenario well; replaying a workload repeatedly on numerous cache configurations can be costly in both time and money. We demonstrate how the key points found by our multi-knee

detection framework can be used to seed the initial population of evolutionary algorithms.

### 5.6.1 Experimental Setup

We configured each experiment with choices for an input I/O trace, a population-initialization technique, an evolutionary algorithm, a knee-detection algorithm, a cache-replacement algorithm, and two parameters controlling the stopping criteria for the optimization. For each configuration, we analyzed the convergence of an evolutionary algorithm with each of the population-initialization techniques and with our multi-knee detection framework.

We used our PyMimircache [155] cache simulator extension (see Section 5.5.1) for all experiments. Simulation enabled us to study a wide variety of configurations, as trace replay on real hardware would be far too slow and would limit the configuration space we could explore. We optimized a single variable, cache size, for the performance metric of I/O operations per second (IOPS) per dollar ($), or IOPS/$. We calculated theoretical values for the IOPS using the same methodology as eMRC [87], and computed dollar costs from a given configuration's cache size and current market values for that type of device [6, 131]. We normalized both the IOPS and dollar cost and then combined them to determine IOPS/$.

We evaluated this use case on three different sets of real-world block traces obtained from CloudPhysics [141] and the publicly available FIU [138] and MSR [100] traces, for a total of 151 individual traces. We used uniform randomized spatial sampling [141, 142] with a size-based sampling rate $R$ (ranging from 0.1 to 0.0001) on the larger traces to reduce the running time while maintaining an accurate representation of the original trace. Our sampling produced a fairly diverse set of MRC sizes, with a mean of $70,446 \pm 110,014$ blocks, ranging from 263 to 829,424 blocks.

We evaluated the speed of convergence of evolutionary algorithms using four population-initialization techniques: our multi-knee detection framework, random initialization, Latin Hypercube sampling (LHS) [98], and Halton sequences [136]. For techniques that include randomization (all but multi-knee), we ran them three times with different random seeds to obtain stable results. Three seeds is generally considered the minimum acceptable number for this type of analysis. However, given the size of our dataset and configuration space, even three random seeds resulted in experiments that required significant running time while still remaining viable. We ran a total of over **3M** experiments, which sufficiently covers the search space, allowing us to evaluate our proposed solution with statistical confi-

dence. We experimented with three types of evolutionary algorithms: Generalized Differential Evolution 3 (GDE3) [73], a Genetic Algorithm (GA) [56], and Particle Swarm Optimization (PSO) [69]. We focused on a subset of the knee-detection algorithms available in our framework: Menger, Kneedle, and Z-Method. We selected these three because Menger represents a baseline knee-detection method that uses a local feature, Kneedle is a well-known algorithm that greatly benefits from our framework, and Z-Method is our novel algorithm designed for this specific application.

We used both Adaptive Cache Replacement (ARC) and Least-Recently Used (LRU) cache replacement policies. These two popular policies present interesting scenarios for multi-knee detection since the MRCs produced by LRU are guaranteed to be monotonically decreasing, while ARC's MRCs can contain both convex and concave regions (see Section 5.2.1). Lastly, we enforced two types of stopping criteria for the optimization: (1) the number of evaluations and (2) an objective value that was some percentage of the "best" value for that configuration. The number-of-evaluations stopping criterion was fixed at 300. We found this value sufficient to allow approximately 99% of our experiments to converge. To handle the objective-based stopping criterion, for each trace we simulated 1,000 cache sizes evenly spaced from the minimum to the maximum and calculated their IOPS and dollar costs. We obtained the maximum IOPS/$ from these simulations, and treated it as the "best" value when calculating the objective stopping criterion for all optimizations involving that trace.

A rule of thumb for the population size is to use ten times the number of parameters in the solution [123]. Since we are only trying to optimize a single parameter (cache size), this implies a minimum population of size 10. If our framework picks fewer points, we iteratively selected points in the center of the largest gap in the curve until we reached 10. It is also possible to optimize the hyper-parameters of the techniques in our framework to select a desired number of points, but we did not explore hyper-parameter optimization in this work.

### 5.6.2 Acceleration Rate

We evaluated our experiments using the overall *acceleration rate* ($\overline{AR}$) [108] to quantify the increased convergence speed when using our framework to select an initial population for evolutionary algorithms. This metric compares the number of function calls (NFCs) made by two separate sets of optimization problems. For our purpose, the NFCs will correspond to the number of epochs (iterations) an evolutionary algorithm takes to converge. Each optimization problem uses an

evolutionary algorithm to optimize the IOPS/$ of a cache and has several parameters: an input trace, an evolutionary algorithm, a knee-detection algorithm, a population-initialization technique, a threshold for the value-based stopping criterion, and a cache-replacement algorithm. The $\overline{AR}$ compares two sets of problems and reports the percent difference in convergence speed. For all of our evaluations, we compared a set of problems $PM$ that use our multi-knee detection framework for population initialization, against a set $PO$ that uses some other technique for initialization. The $\overline{AR}$ is calculated as follows:

$$\overline{AR} = \left(1 - \frac{\sum_{i=1}^{|PM|} NFC(PM_i)}{\sum_{i=1}^{|PO|} NFC(PO_i)}\right) \times 100\% \tag{5.4}$$

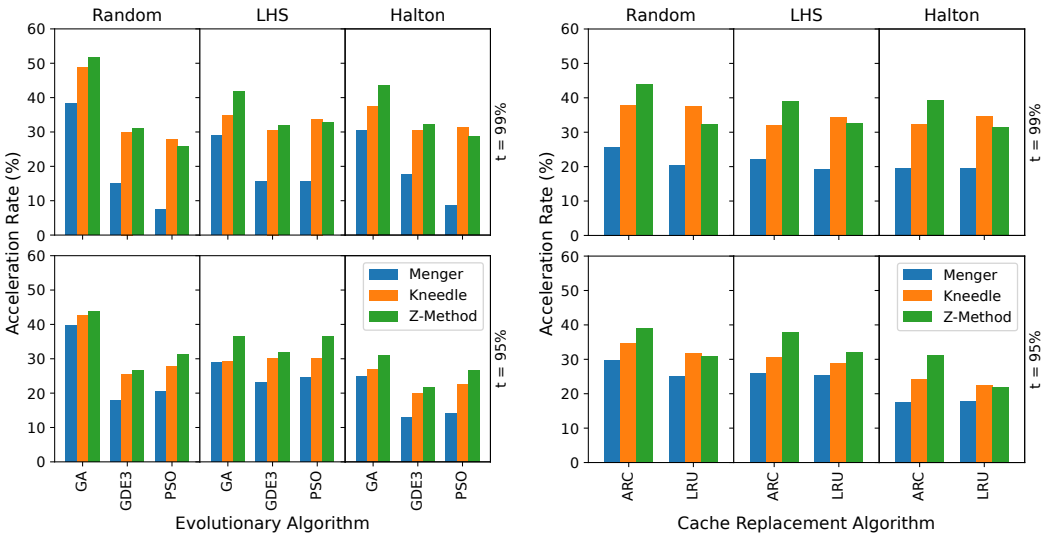
*i.e.*, an x% $\overline{AR}$ implies an x% reduction in running time.

We evaluated 151 traces, four population initialization techniques, two cache-replacement algorithms, and three variations of all other parameters, resulting in 32,616 total problems and over three million cache simulations. The overall $\overline{AR}$ achieved using our multi-knee detection framework for population initialization was 34%. In the remainder of this section, we show the effects of each configuration parameter by creating subsets of the total problems via parameter constraints.

Figure 5.10 presents bar plots showing the $\overline{AR}$ achieved using our multi-knee detection framework for population initialization. Each bar represents a comparison between two sets of optimization problems, $PM$ and $PO$, which are different sets for each bar, depending on the constraints of the axes and legend. $PM$ uses our multi-knee detection framework for population initialization with a knee-detection algorithm identified by the color of the bar (Menger, Kneedle, or Z-Method). $PO$ uses the initialization technique shown on the x2-axes (top of figure): a pseudo-random number generator (Random), Latin Hypercube sampling (LHS), or Halton sequences. The value of the objective threshold $t$ stopping criterion for both problem sets is shown on the y2-axes.

In each plot, we show three knee-detection algorithms: Menger, Kneedle, and Z-Method. The bars labeled "Menger" represent a baseline knee-detection method; it was the least accurate of those depicted. "Kneedle" is the robust version that we optimized for this scenario; we employed Global RDP (gRDP) before using Menger or Kneedle, reducing noise in the MRCs and improving knee detection. After the knee-detection phase, we applied the post-processing filters described in Section 5.3.1 to refine the selected knees.

We varied the objective-threshold stopping criterion *t* between 99%, 98% (omitted in Figure 5.10 for brevity), and 95% for all configurations. We observed




(a) Acceleration rates using GA, GDE3, and PSO algorithms.

(b) Acceleration rates using ARC and LRU cache replacement.

Figure 5.10: The acceleration rate ($\overline{AR}$) achieved using our multi-knee detection framework for population initialization vs. other techniques. The height of each bar represents the acceleration rate (y-axis), where higher is better. The bottom row represents experiments using the objective threshold $t = 95\%$; the top row is those using $t = 99\%$ (the most challenging threshold to meet). Each group of bars, from left to right, shows the $\overline{AR}$ using our framework for population initialization with the baseline (Menger, in blue), with Kneedle (orange), and with our Z-Method (green), each evaluated against three different initialization methods on the x2-axis (Random, LHS, and Halton, at the top of the figure). (a) shows the $\overline{AR}$ for three different Evolutionary Algorithms on the x-axis (GA, GDE3, and PSO), with results included from both LRU and ARC cache replacement algorithms. (b) shows the same results as in (a) but separated by LRU vs. ARC, with results included from all three Evolutionary Algorithms.

that a lower threshold usually yielded a lower $\overline{AR}$. A higher threshold makes optimization more difficult by requiring a solution closer to the most optimal value. Thus, a lower threshold increases the number of acceptable solutions, giving less-informed population initialization techniques a better chance at finding a solution.

Figure 5.10a shows the effects of using different evolutionary algorithms. The optimization problems analyzed in each group of bars are shown on the x-axis: Genetic Algorithm (GA), Generalized Differential Evolution 3 (GDE3), and Particle Swarm Optimization (PSO). Results in this figure include problem sets with configurations using both ARC and LRU cache replacement. Therefore, in the upper leftmost subfigure of Figure 5.10a, the first blue bar on the left represents, for the 151 traces, the $\overline{AR}$ for all optimization problems using: (1) our framework with Menger for population initialization vs. Random, (2) using a Genetic Algorithm (GA), (3) with either ARC or LRU cache replacement, and (4) an objective threshold $t = 99\%$.

Z-Method is our novel method designed for this task; it outperformed Kneedle in most cases. Kneedle was still competitive, usually only a few percent behind Z-Method and even winning by a small margin in three out of the 18 configurations displayed. (It should be noted that Kneedle benefited greatly from our framework's pre- and post-processing filters, and that Kneedle on its own yielded much poorer results; see Section 5.2.2). We also saw a wide range of $\overline{AR}$ values in these configurations, from under 10% $\overline{AR}$ (Menger with PSO) to over 50% $\overline{AR}$ (Z-Method with GA). Lastly, ranking the $\overline{AR}$ from highest to lowest yields GA (best), GDE3, and then PSO. We attribute this ranking to the difference in efficiency of the evolutionary algorithms. The more efficient algorithms have a lower $\overline{AR}$ because they are less sensitive to the initial population.

Figure 5.10b shows the effects of using either ARC or LRU cache replacement. Results in this figure include problem sets with configurations using any of the three evolutionary algorithms. The most relevant difference between ARC and LRU is that LRU MRCs always decrease monotonically, while ARC can have both convex and concave regions (*i.e.*, can go up and down). This poses a unique challenge for knee detection, since a curve can have a knee that is less optimal than a previous point. Said differently, in ARC and similar non-stack cache algorithms, counterintuitively, adding more cache can actually *hurt* performance. This non-monotonic property can also distort many metrics that knee-detection algorithms use to detect or rank points. Z-Method significantly outperformed Kneedle for all ARC cases in Figure 5.10b, while Kneedle beat Z-Method somewhat for all but one of the LRU configurations. This is to be expected, as Z-Method was originally designed with cache optimization for non-monotonic MRCs in mind,

enforcing constraints in a grid-like pattern to prevent proximal knees. Conversely, Kneedle was designed under the assumption of monotonicity. Again, we note that our pre- and post-processing filters were essential to Kneedle's good results.

The population-initialization techniques did not alter any of the previous trends we observed in Figure 5.10, but we did see an approximately 5% difference in $\overline{AR}$ between the worst and best case: Random performed the worst, followed by Halton, and then LHS. These results are consistent with those seen in previous studies [98, 136, 11].

## 5.7 Conclusion

The many configurations of multi-tier caching systems produce a wide range of performance and costs. As the configuration space continues to grow due to advancements in caching and storage technology, exploring the space through physical experiments or traditional simulation becomes infeasible.

We introduced the novel concept of applying multi-knee detection to MRCs using a framework for selecting key points, reducing the cost of exploration significantly. We present Z-Method, an algorithm that robustly and efficiently identifies multiple key points in MRCs with minimal overhead. We also designed a recursive algorithm that enables any single-knee-detection algorithm to find multiple knees. We demonstrated that our framework using Z-Method can be applied to reduce the total number of points required to identify optimal two-tier cache configurations by an average factor of approximately $5.5\times$ for ARC and $7.7\times$ for LRU compared to naïve approaches. Finally, we evaluated our framework across a highly diverse set of configurations and datasets for the additional application of seeding the initial population of evolutionary algorithms, showing an overall acceleration rate of 34% compared to commonly used population-initialization techniques.

# Chapter 6

# Visual Analytics and Performance Modeling

This chapter presents additional techniques for exploring and analyzing complex system design spaces. It focuses on interactive visualization methods for analyzing high-dimensional configuration spaces, along with modeling techniques that capture workload behavior to support performance reasoning. Together, these approaches support systematic reasoning about performance trade-offs and guide more efficient system design.

## 6.1 Advanced Interactive Visualizations

To further support exploration, we developed advanced interactive visualizations, including an interactive configuration explorer for studying how categorical parameters affect system performance, a parallel-coordinate axes-reordering framework that reveals patterns and relationships among configuration parameters, and an empty-space search algorithm to uncover promising configurations hidden in sparsely sampled gaps of the space.

### 6.1.1 ICE: An Interactive Configuration Explorer for High Dimensional Categorical Parameter Spaces

Modern tiered storage and memory systems expose large, high-dimensional configuration spaces dominated by categorical parameters, making it difficult to reason about trade-offs using traditional plots or automated search alone. To address

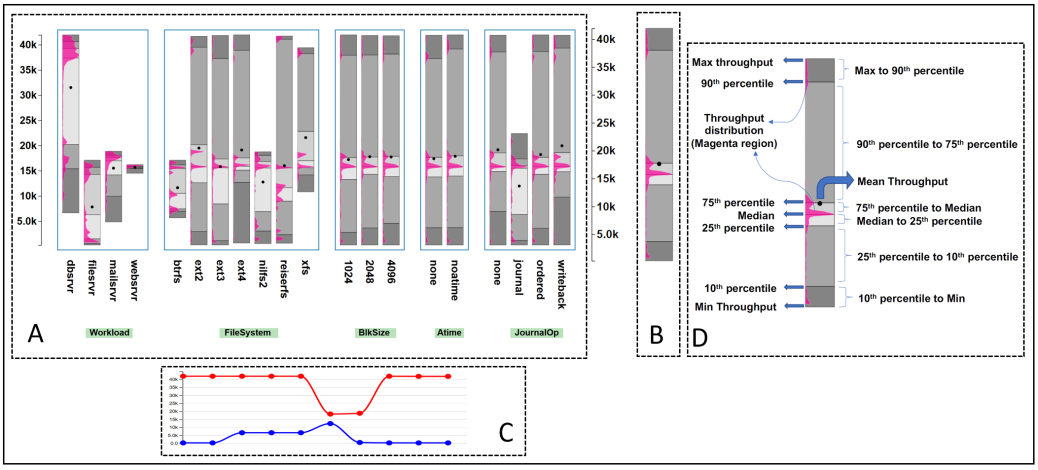



Figure 6.1: Interface of our Interactive Configuration Explorer (ICE) tool used to explore high dimensional parameter spaces. This example shows the use of the ICE in a computer systems performance optimization scenario. A is the Parameter Explorer. It shows the distribution and statistics of the numerical target variable in the context of the various categorical variables (or parameters), labeled by the green buttons at the bottom of the interface (e.g., Workload, File System). Each parameter has levels e.g., Workload has 4 levels (dbsrvr, filesrvr, mailsrvr, and websrvr), and each level has an associated bar displaying the statistical information about the numerical target variable (here, system throughput) for this level. Analysts can interactively deselect (and select) parameter levels to filter out the associated parameter configurations throughout. B is the Aggregate View, which visualizes the joint distributions of all currently selected parameter levels. C is the Provenance Terminal, to keep track of the changes in the target variable over the course of the user interactions. D shows the information contained in each bar inside the Parameter Explorer and Aggregate View.

this, we contributed to the design and evaluation of the Interactive Configuration Explorer (ICE), a visual analytics tool developed to support performance-driven configuration exploration [132]. ICE was motivated by the observation that common techniques such as parallel sets, dimensionality reduction, or dashboards either lose information, obscure distributions, or do not scale as the number of parameters grows.

ICE's design is illustrated in Figure 6.1, which decomposes the exploration workflow into four coordinated views. Figure 6.1A shows the Parameter Explorer, where each categorical parameter is expanded into its individual levels, and each level is represented by a range-distribution bar encoding the full throughput distribution, key percentiles, extrema, and mean. This view allows direct comparison of how individual configuration choices affect both performance and variability, and supports interactive filtering by enabling analysts to deselect unpromising levels. Figure 6.1B presents the Aggregate View, which collapses all currently selected parameter levels into a single range-distribution bar, providing an immediate summary of the joint performance distribution induced by the current configuration. Figure 6.1C shows the Provenance Terminal, which records how the maximum and minimum achievable throughput evolve as the analyst iteratively filters the space, enabling backtracking and comparison between alternative exploration paths. Finally, Figure 6.1D annotates the internal structure of the range-distribution bars, clarifying how percentiles, extrema, and the underlying distribution are encoded. Together, these components allow analysts to reason about multi-objective trade-offs, such as peak throughput versus stability, without information loss or visual clutter, even in large, high-dimensional configuration spaces.

We used ICE to analyze storage system configuration spaces derived from multi-year performance measurements collected in our lab. The tool was developed around requirements we identified as systems researchers, including comparing full throughput distributions, reasoning about variability and stability, and tracking iterative configuration filtering decisions. ICE enabled systematic exploration of performance trade-offs across large categorical parameter spaces that were difficult to analyze using existing visualization techniques.


Figure 6.2: PC-Expo is a real-time all-in-one Parallel Coordinate Plot (PCP) axes reordering framework. PC-Expo detects local properties in high-dimensional data that can be used to reorder the PCP axes automatically or with human-in-the-loop (HIL) interactions A. We have implemented detectors for the 12 most common data properties used to reorder PCPs, shown on the properties panel B. Users can create their own optimization scheme using a weighted sum of these properties, by selecting respective properties and weights from (B), summarized as a donut chart E for automated axes reordering. PC-Expo also supports HIL axes reordering via a heatmap and local views D, C, and F. D summarizes the weighted sum of user-selected properties detected locally for each axis pair. C shows where these visualization properties were detected for a particular axis pair, with a linked scatterplot F for visualizing the 2D data points. Users can manually reorder the axes using these local views by clicking on D sequentially. The granularity slider in B lets users control the size of local regions used to detect the properties. Area charts next to PCP axes in A show the local regions where the properties selected on (B) are detected on the axis.

### 6.1.2 PC-Expo: A Metrics-Based Interactive Axes Reordering Method for Parallel Coordinate Displays

Parallel coordinate plots are widely used for analyzing high-dimensional systems data, but their effectiveness depends critically on axis ordering, which determines which patterns are visually apparent. Existing automated reordering techniques typically optimize for a single global metric and lack support for localized patterns or human-in-the-loop refinement. PC-Expo addresses these limitations by providing a unified framework for real-time, explainable, and localized axes reordering that supports both automated optimization and interactive exploration [133].

Figure 6.2 illustrates the PC-Expo interface and workflow. Figure 6.2A shows the main parallel coordinate plot augmented with area charts that summarize where selected properties are detected locally along each axis, supporting explainability of the final ordering. Figure 6.2B presents the properties panel, where users select from twelve common PCP properties and control both their relative weights and the localization granularity. Figures 6.2C and F provide local views and linked scatter plots that expose the specific data regions contributing to detected patterns, allowing users to inspect and validate local structure before reordering. Figure 6.2D shows the heatmap that aggregates the weighted property scores for each axis pair, enabling both automated reordering and human-in-the-loop selection. Finally, Figure 6.2E summarizes the global optimization objective via a donut chart that visualizes the contribution of each selected property.

We applied PC-Expo to systems datasets involving high-dimensional performance and configuration parameters. The framework directly addresses the need to identify local structure, balance multiple analytical objectives, and understand why a particular axes ordering exposes specific patterns. Through evaluation, PC-Expo enabled faster exploration and produced higher-quality axis orderings than prior PCP reordering techniques, particularly for systems workloads with complex, localized behavior.

### 6.1.3 Into the Void: Mapping the Unseen Gaps in High Dimensional Data

High-dimensional configuration spaces are often sparsely sampled, leaving large regions unexplored despite the potential for high-quality configurations. GapMiner addresses this problem by explicitly identifying and exploiting empty spaces in configuration datasets, treating them as opportunities rather than artifacts of

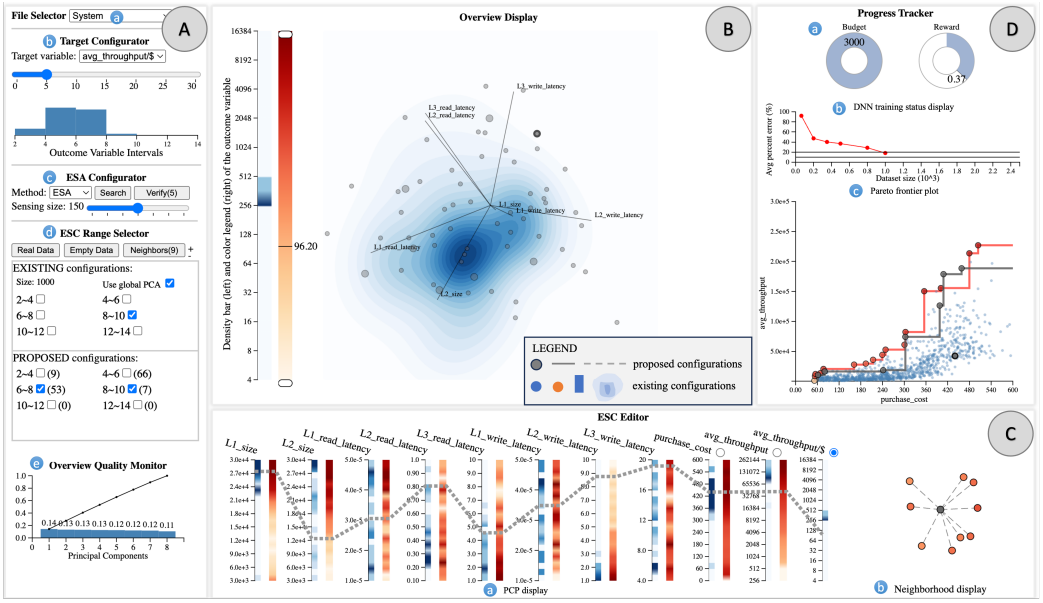



Figure 6.3: GapMiner visual interface where a selected ESC is reflected in all displays. (A) Control Panel. From top to bottom: (a) File Selector to load a dataset of initial verified configurations with values for all parameter variables. (b) Target Variable Configurator with an interface for breaking its value range into discrete intervals. (c) Empty-space Search Algorithm (ESA) Configurator to select the ESA and a slider to set the ESC batch size. (d) Empty-Space Configuration (ESC) Range Selector to control which target variable intervals are used for display and ESC proposals. (e) Overview Quality Monitor screeplot that shows the amount of data variance captured by the Overview (PCA) Display. (B) Overview (PCA) Display with data distribution contours, raw or modified ESCs rendered as points, and color legend. (C) Empty-space Configuration (ESC) Editor. From left to right: (a) Parallel Coordinate Plot Display where users can configure ESCs starting from a raw ESC or an existing configuration. (b) Neighbor Display of the selected ESC providing a local view of the distribution of its nearest existing configurations. (D) Progress Tracker. From top to bottom: (a) Budget/Reward Display that captures the aggregated evaluation cost and merit of the ESC exploration so far. (b) Training Status Display of the assistive DNN. (c) Pareto Frontier plot that shows the Pareto frontiers of existing configurations (red) and ESCs (gray) with respect to two user-chosen merit (target) variables.

sparsity [162]. The system combines a physics-inspired Empty-space Search Algorithm (ESA) with interactive visual analysis, enabling systematic discovery of novel configurations that are far from existing samples yet promising with respect to target objectives.

Figure 6.3 shows the GapMiner interface and its coordinated views. Figure 6.3A presents the Control Panel, which allows users to load verified configurations, define target variables and value intervals, configure the ESA, and monitor how well the PCA-based overview captures data variance. Figure 6.3B shows the Overview (PCA) Display, where density contours summarize existing configurations and proposed Empty-Space Configurations (ESCs) appear as points, providing global context for sparsity and clustering. Figure 6.3C is the ESC Editor, combining a parallel coordinate plot for refining configurations with a neighborhood display that reveals the local geometry and topology of the surrounding empty space. Figure 6.3D shows the Progress Tracker, which integrates budget tracking, DNN training status, and a Pareto frontier view to support multi-objective evaluation and guide verification effort.

We used GapMiner to explore multi-tier cache configuration spaces derived from real system workloads, where the number of possible cache size and latency combinations makes exhaustive evaluation infeasible. The interface supported reasoning about how candidate ESCs relate to existing cache designs, exposing trade-offs between average throughput and purchase cost and revealing which cache parameters mattered despite contributing little to global variance. In the caching case study, this process led to the discovery of previously unsampled cache configurations that expanded the throughput-cost Pareto frontier more effectively than random sampling or ESA-only search, while limiting the number of expensive system evaluations required.

## 6.2 Distribution Fitting

Accurately modeling storage system performance requires realistic representations of request inter-arrival and service-time behavior. Commonly used distributions, such as Exponential or Normal, offer analytical simplicity but fit real storage workloads poorly, while heavy-tailed alternatives often yield undefined moments or unstable models. We addressed this gap by systematically evaluating which distributions both fit real storage workloads well and remain suitable for practical performance modeling [140].

We analyzed over 250 block-layer storage traces spanning five workload fam-

ilies and evaluated twenty widely used distributions using multiple goodness-of-fit and complexity metrics, including ($R^2$), Jensen-Shannon divergence, log-likelihood, AIC, and BIC. Across workloads and metrics, a two-phase Hyper-exponential distribution consistently provided the most accurate and robust fit, outperforming commonly used alternatives while avoiding overfitting. Unlike many heavy-tailed distributions that fit individual traces well but fail to generalize, the Hyper-exponential captured both burstiness and long-tail behavior across diverse storage systems with a small number of parameters.

The structure of the Hyper-exponential distribution enabled exact Markov-chain–based performance models that are not feasible with most other fitted distributions. Using H2/H2/1 queuing models, we predicted mean response time for real mobile storage traces with a median error of 17.5%, compared to 48.8% for Exponential-based models. This modeling approach supported efficient what-if analyses of storage behavior, including workload variability, flash-induced delays, and transient load spikes, without requiring costly system experimentation.

# Chapter 7

# Proposed Work

We introduce the proposed work in this section to describe the specific research tasks and methodologies we plan to incorporate into the thesis. This section explains two main proposed works: CXL-based migration and tiered-memory migration.

**Proof-of-concept prototype** In this thesis proposal, we plan to extend QEMU to enable live VM migration using both CXL and tiered memory. As a proof-of-concept, we implemented a QEMU prototype that migrates over CXL using QEMU's native channel interface and transparently tiers guest memory in a CXL Type 3 device that is shared by both machines.

Our prototype requires two host machines to be sharing a CXL Type 3 device in DAX mode. It utilizes the standard pre-copy algorithm (QEMU's default option), which still includes dirty page tracking and the retransmission of dirtied pages. We designed a QEMU channel inspired by the SOSP 2025 paper Oasis, consisting of decoupled control and data path structures [165]. Both structures are stored on the shared CXL device: the control path consists of a fixed-size ring buffer that is stored at the beginning of the device and is immediately followed by the data buffers. Each control slot in the ring buffer contains the offset of the payload, along with a control bit to indicate that it is ready to be accessed by the destination machine (*i.e.*, the stores to both the control slot and corresponding data buffer are 100% complete).

Data transfers over the CXL device work as follows: For every chunk of data needed to be migrated, the source machine stores the payload in the next available data buffer and the pointer to that payload in the next available control slot of the ring buffer. After both stores are complete, the source machine flips the control

bit of the control slot it just accessed. Meanwhile, the destination machine has a dedicated polling thread that is separate from the main QEMU thread. This thread constantly polls on the ring buffer, checking for when the next slot is ready to be accessed. When it detects a control bit has been flipped, it notifies the main thread by incrementing an `eventfd`. The destination's main QEMU thread then accesses the payload and flips the corresponding control bit.

Since CXL 2.0 does not implement coherent sharing, which is introduced in CXL 3.0, both machines must carefully coordinate to manage coherency. For the source machine, store instructions may remain in local CPU caches rather than being stored in CXL memory. Therefore, they must issue a flush instruction for every cache line they modify. In addition, after all stores to the data buffer and corresponding control slot are complete, they must issue a fence instruction to ensure proper ordering before the control bit is flipped. Likewise, when the reader issues a load instruction, any cache lines that reside in their local CPU caches will be read rather than the relevant data stored on the CXL device. The reader must also issue flush and fence instructions to invalidate these stale cache lines, ensuring data is loaded from the CXL device. Managing coherency in this way adds a small amount of overhead.

We have gained access to servers with CXL hardware through a collaboration with Microsoft Azure. Simple benchmarks revealed that we could achieve 23.53GB/s throughput on a x8 CXL link in one direction. This was reduced by approximately 9.6% to 21.27 GB/s when we flushed and fenced every 64-byte cache line. These results are consistent with those in the literature (see Chapter 2.3).

While CXL bandwidth can technically be realized in both directions since PCIe supports bi-directional transfers, it is unlikely that we would achieve this performance in practice due to the coordination required between hosts when managing coherency. In the worst case, the destination machine would have to wait for the source machine to completely finish writing all of its data to the CXL device before it accessed it, effectively cutting the bandwidth in half. A more realistic approach involves transferring data in chunks so that both machines can be actively transferring data as much as possible.

While bandwidth may be somewhat reduced by the overhead of managing coherency, CXL can still achieve much greater bandwidth than RDMA via interleaving across multiple lanes and devices. CXL servers typically employ multiple CXL devices, providing up to 64 lanes in total on current production systems such as Intel Xeon 6 (Granite Rapids) [124, 70, 106, 107]. By default, the CPU interleaves memory accesses at a 256-byte granularity, effectively combining the band-

width of multiple CXL devices to achieve approximately 240GB/s [165]. In comparison, we measured approximately 11GB/s using RDMA write over 100Gbps between Mellanox ConnectX-5 and Mellanox ConnectX-6 Ex RNICs. This reflects a fair comparison between 100Gbps RDMA and PCIe 5.0 CXL configurations, demonstrating that CXL can enable significantly higher bandwidth, making it a stronger candidate for live VM migration.

**CXL-based migration** We first propose to implement a CXL-based migration method that improves on our prototype, eliminating the overhead of the native QEMU channel interface while removing the need for both dirty-page tracking and the retransmission of dirtied pages. In traditional migration, pages that are re-dirtied by the guest VM on the source machine after they are initially copied to the destination machine must be retransmitted in the next iteration of the pre-copy phase (see Chapter 2.1). The core reason for this is that the destination machine does not have direct access to the local memory of the source machine (*i.e.*, memory must be transferred over the network). We can overcome this limitation by exploiting the fact that both machines can access pages directly from the shared CXL device after they are transferred only once.

Our proposed migration method is inspired by Grapentin *et al.*'s work on POWER9-based disaggregated migration [46]. We propose to extend their work to CXL and explore ways to improve on their techniques. Our method of migration will consist of three phases:

**Phase 1**: The first phase involves the source machine transferring all of the VM's memory pages to the shared CXL device. During this phase, the VM is active and can freely access all pages. Traditional migration marks the entire guest VM's memory space as read-only, faulting on all writes so that it can track which pages have been dirtied, and then iterating numerous times while retransmitting dirtied pages. Instead, we iterate over every page a single time, issuing a `mprotect()`, a `memcpy()`, and then a `mmap()` with the `MAP_FIXED` argument. This effectively marks the page as read-only for the very brief duration of the memcpy. The call to `mmap()` with `MAP_FIXED` remaps the hypervisor's pointer for the guest's page to point to the CXL device rather than the source machine's local DRAM. The active guest VM can then write to this page since it is stored in CXL memory and has not been marked as read-only. In the unlikely event that the guest accesses that page in this brief window of time, it will trigger an interrupt. As a base method, we will implement an empty signal handler that captures the fault and does nothing, which simply signals the machine to try

the access again. We plan to explore if there are better ways to utilize the signal handler, such as tracking these pages to assist in tiering decisions.

**Phase 2**: The second phase begins once all pages have been transferred to CXL. At this point, the VM is paused and the remaining memory contents such as vCPU states and device states are transferred to CXL. After this is complete, the destination machine can immediately resume execution of the guest VM by directly accessing the shared CXL memory.

**Phase 3**: The final phase consists of a tiering decision by the destination machine. It can leave all of the pages in CXL, or launch background thread(s) that copy some or all pages from CXL to local DRAM. This phase also offers a unique opportunity to make more intelligent decisions on which pages to migrate.

Our migration strategy is designed to significantly reduce total migration time, limit blackout duration, and minimize data transfers between machines. In addition, we will implement custom RAM handlers that use direct load/store semantics, avoiding the overhead of QEMU's native channel interface. This design will enable scalable performance as the number of CXL devices and PCIe lanes increases, delivering substantially higher throughput than traditional network-based methods such as RDMA.

**Tiered-memory migration** Our second proposal is to implement tiered-memory migration in QEMU. Tiering memory with CXL has been shown to substantially increase memory capacity with minimal performance impact, improving resource utilization while maintaining near-DRAM performance (see Chapter 3.4). The benefits of tiered-memory migration are both straightforward and significant. By tiering some amount of guest memory in a CXL device that is shared by both the source and destination machines, this memory does not need to actually be transferred during live migration. Directly reducing the amount of data transferred leads to a proportional reduction in total migration time, which should roughly match the percentage of memory that is tiered.

We plan to implement tiering in a way that is entirely managed within the hypervisor and completely transparent to the guest VM. The guest will see both memory sources as a single device, unaware of whether pages are backed by local DRAM or remote CXL memory. QEMU currently supports backing guest system RAM entirely from a memory-backed file, but does not support tiering of different types of memory (*i.e.*, tiering local heap RAM and CXL memory). The implementation requires new data structures for tiered memory, mechanisms to migrate them without moving the data, and precise coordination of CXL memory

pointers between both machines.

Our base design will naively place some portion of the guest memory on the CXL device. We consider this approach to be adequate for demonstrating the benefits of tiered-memory migration. However, we believe there are many opportunities to more intelligently tier guest memory as well as migrate that memory (see Chapter 8). Ultimately, tiered-memory migration offers a simple yet powerful way to reduce migration time and memory movement without compromising guest performance.

**Evaluation plan** Our evaluation will include experiments run on Microsoft Azure's CXL hardware, consisting of two physical servers sharing a Type 3 memory expansion device. If possible, we will also experiment with their servers that interleave across multiple CXL devices to demonstrate CXL's superior bandwidth capabilities.

We will conduct both guest and hypervisor benchmarks. Guest benchmarks will measure performance metrics from applications running inside the VMs, such as the throughput of a database before, during, and after migration. We plan to test a variety of applications: a database (MySQL), a key-value store (LevelDB), and a web server (Apache). Hypervisor benchmarks will assess the time required to migrate a VM from one server to another, the duration of any pause in VM execution during migration, and the total amount of data transferred between servers.

We will compare CXL-based migration to both TCP- and RDMA-based migration, with and without tiering. Additionally, we will evaluate hybrid approaches with tiering, where tiered memory resides on CXL but local memory is migrated using TCP or RDMA. We believe there are still workloads and system configurations where this could be beneficial, such as when the available RDMA hardware is more advanced than the CXL hardware and when the guest workload is relatively idle, resulting in minimal dirty page activity and retransmission overhead. Together, these experiments will provide a comprehensive understanding of the trade-offs between migration strategies and highlight the practical benefits of CXL and tiered memory in real-world scenarios.

# Chapter 8

# Future Work

In this chapter, we discuss potential future work that extends beyond the scope of this thesis. These future directions focus on intelligent page placement for CXL-based migration and CXL-aware multi-VM placement to enhance performance, scalability, and energy efficiency.

**Migration with intelligent page placement** Page placement is the process of deciding which pages reside in each memory tier. In this work, we propose to implement tiered migration with a naïve page-placement baseline (see Chapter 7): CXL remote memory is simply used to expand the memory capacity of the guest VM, and no techniques are applied to actively manage placement across tiers. Because CXL memory has roughly twice the latency of local DRAM (see Chapter 3.4), minimizing accesses to it is critical for performance. Prior studies show that with effective placement, CXL can expand capacity while keeping latency-sensitive workloads within a few percent of DRAM performance [92, 164, 151, 86]. However, these studies focus on placement managed by the host operating system or hardware, whereas in our design the hypervisor is responsible for managing the guest's memory tiering.

Our implementation presents some unique opportunities for collecting useful placement information during migration. There is a very brief window during the one-time copy when a page fault can occur if the guest VM accesses a page that is currently being copied. While our CXL-based migration does not require dirty-page tracking to ensure a complete transfer, we can still record these faults as hints for future placement. Pages that trigger faults are likely to be accessed again and should be promoted to the fastest tier, local DRAM.

In the final migration phase, the destination host decides whether to copy

memory staged on the CXL device into DRAM. Using the hints gathered earlier, we can prioritize transferring hot pages into DRAM, reducing the likelihood of expensive future accesses to CXL memory. In general, it is worth exploring whether placement managed entirely within the hypervisor can be more effective than techniques that rely on the host operating system or hardware.

**Multi-VM placement** Multi-VM placement refers to deciding where groups of virtual machines should run so that service-level objectives (SLOs) are met while resources are used efficiently. Future work can examine how CXL might improve multi-VM placement in clusters that balance performance targets with operational goals. Prior studies emphasize availability as the primary objective, defined as the probability that a service remains operational and able to meet its SLOs, and show how cluster-level placement decisions influence a system's ability to sustain required uptime and meet service targets [85, 4].

Placement models can be extended to reason about hosts that share access to a CXL memory pool. When hosts share CXL memory, they can migrate VMs while leaving some portion of their memory resident in the pool, which significantly changes cost models for moving many VMs at once. These models could assign VMs to hosts that share a CXL memory pool while accounting for capacity limits, contention, and balancing pool usage with load distribution and availability targets.

Placement policies could also be adapted to co-locate VMs that share datasets residing on the same CXL memory to reduce data movement and improve efficiency. At the same time, these policies would need to manage the risks of over-concentrating workloads within a single memory domain.

Another direction is to study multi-VM placement with energy and thermal objectives. Energy- and thermal-aware strategies seek to reduce power consumption and cooling requirements while keeping applications within their service goals [84, 74]. With a CXL shared memory tier, placement decisions must weigh trade-offs between consolidating workloads onto fewer active hosts to reduce power usage, allocating each VM's memory between local DRAM and CXL in ways that optimize energy efficiency, and controlling the rate of VM rebalancing across hosts to reduce avoidable energy overhead. Understanding these tradeoffs requires models that capture how memory placement, host consolidation, and migration frequency interact to influence both power draw and thermal load. Such models could guide policies that maximize the energy-saving potential of CXL while preventing performance degradation in large-scale data centers.

# Chapter 9

# Conclusion

Tiered storage and memory systems combine diverse types of devices, each with different characteristics, with the objectives of maximizing performance while minimizing costs. As the configuration space expands and technology continues to evolve, identifying and exploiting the most effective configurations has become increasingly challenging. With CXL introducing a shared, byte-addressable memory tier between hosts, the tiering space grows more complex and requires novel approaches to realize its benefits. To address this issue, we developed techniques for evaluating tiered storage and memory systems and for efficiently identifying promising configurations. We now propose to develop CXL-based live VM migration techniques that leverage shared memory to minimize data transfer, shorten blackout periods, and lower overall migration overhead.

We first examined limitations in existing approaches for evaluating tiered storage and memory systems, finding that they often focused on single-tier performance and failed to analyze cost versus performance trade-offs. To address this, we developed a general $n$-level cache simulator capable of modeling arbitrary hierarchies and capturing both performance and cost, enabling analysis across diverse metrics. We then developed a framework for efficiently exploring large configuration spaces using miss-ratio curves, combining hash-based sampling, curve simplification, knee detection, and our novel Z-Method to identify promising configurations more quickly. To support analysis at scale, we also developed visual analytics and workload modeling techniques that enable systematic reasoning about high-dimensional design spaces and performance behavior derived from real workloads. These results make it possible to explore the design space more efficiently and effectively, reveal subtle trade-offs, and provide valuable insights.

We propose two methods for applying tiering to live virtual machine migration

with CXL. The first is a CXL-based migration method designed to transfer each memory page only once, thereby eliminating dirty-page tracking and retransmission. The second is a tiered-memory migration method that allocates guest memory across DRAM and shared CXL, transferring only the portion stored in DRAM during migration. Both approaches work to decrease migration time, minimize service disruption, and reduce overall data transfer requirements versus traditional techniques.

Our prototype demonstrates the feasibility of live VM migration over CXL while transparently tiering guest memory in a shared device. It uses QEMU's native channel interface and preserves existing migration mechanisms, demonstrating compatibility with established virtualization frameworks. The design works on currently available CXL 2.0 hardware by managing coherency in software with simple, well-defined instructions. Access to Azure CXL servers provides a production-grade testbed for end-to-end experiments and development, supporting the feasibility of our proposal.

It is our thesis that tiered storage and memory systems expose a vast configuration space with the potential for significant performance and cost optimizations. Fully realizing these benefits requires efficient techniques for exploring and exploiting this space, particularly as the introduction of CXL shared memory adds new and powerful opportunities for tiering. We plan to design and evaluate CXL-based migration and tiered-memory migration methods that reduce migration time, blackout duration, and total data movement.

# Chapter 10

# Acknowledgments

I thank the many collaborators from both industry and academia for their insights, discussions, and feedback throughout this work. I am also grateful to the undergraduate and graduate students who assisted with implementation, experimentation, data collection, and analysis.

This work was made possible in part through support from Microsoft Azure, Dell-EMC, NetApp, IBM, and Facebook, as well as a SUNY/IBM Alliance award and a Stony Brook OVPR Seed Grant. Computational resources were provided by the SeaWulf cluster at the Institute of Advanced Computational Science.

This research was supported by the National Science Foundation under awards IIS-1527200, CCF-1918225, IIS-1941613, CNS-1251137, CNS-1302246, CNS-1305360, CNS-1622832, CNS-1650499, CNS-1729939, CNS-1730726, CNS-1750109, CNS-1755958, CNS-1900589, CNS-1900706, CNS-1910327, CNS-1951880, CNS-2106263, CNS-2106434, and CNS-2214980, as well as NSF SBIR contract 1926949. Additional support was provided by the Office of Naval Research under award N00014-16-1-2264.

This work was also supported by FCT - Fundação para a Ciência e Tecnologia, I.P., through UIDB/50008/2020-UIDP/50008/2020 (DOI 10.54499/UIDB/50008), with national funds and, where applicable, co-funded European Union funds.

# Bibliography


[1] AccuSim: Accurate simulation of cache replacement algorithms, March 2020.

[2] Charu C. Aggarwal. *Outlier Analysis*. Springer Publishing Company, Incorporated, 2nd edition, 2016.

[3] Jeffrey O. Agushaka, Absalom El-Shamir Ezugwu, Laith Mohammad Abualigah, Samaher Khalaf Alharbi, and Hamiden Abd El-Wahed Khalifa. Efficient initialization methods for population-based metaheuristic algorithms: A comparative study. *Archives of Computational Methods in Engineering*, 2022.

[4] Yanal Alahmad and Anjali Agarwal. Multiple objectives dynamic vm placement for application service availability in cloud networks. *J. Cloud Comput.*, 13(1), February 2024.

[5] Waleed Ali, Sarina Sulaiman, and Norbahiah Ahmad. Performance improvement of least-recently-used policy in web proxy cache replacement using supervised machine learning. In *SOCO*, 2014.

[6] Anandtech: Hardware news and tech reviews since 1997. *www.anandtech.com*.

[7] Mário Antunes, Henrique Aguiar, and Diogo Gomes. AL and S methods: Two extensions for L-method. In *7th International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 371–376. IEEE, 2019.

[8] Mário Antunes, Diogo Gomes, and Rui L Aguiar. Knee/elbow estimation based on first derivative threshold. In *Fourth IEEE International Conference on Big Data Computing Service and Applications (BigDataService)*, pages 237–240, Bamberg, Germany, March 2018. IEEE.

[9] Mário Antunes, Joana Ribeiro, Diogo Gomes, and Rui L Aguiar. Knee/elbow point estimation through thresholding. In *6th IEEE International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 413–419, Barcelona, Spain, August 2018. IEEE.

[10] Dulcardo Arteaga, Jorge Cabrera-Gámez, Jing Xu, Swaminathan Sundararaman, and Ming Zhao. CloudCache: On-demand flash cache management for cloud computing. In *USENIX Conference on File and Storage Technologies (FAST)*, 2016.

[11] Adnan Ashraf, Sobia Pervaiz, Waqas Bangyal, Kashif Nisar, Ag Asri Ag Ibrahim, Joel Rodrigues, and Danda Rawat. Studying the impact of initialization for population-based algorithms with low-discrepancy sequences. *Applied Sciences*, 11:8190, 09 2021.

[12] Nathan Beckmann and Daniel Sanchez. Talus: A simple way to remove cliffs in cache performance. In *IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pages 64–75, 2015.

[13] Daniel S. Berger, Benjamin Berg, Timothy Zhu, Siddhartha Sen, and Mor Harchol-Balter. RobinHood: Tail latency aware caching — dynamic reallocation from cache-rich to cache-poor. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2018.

[14] Md Israfil Biswas, Gerard Parr, Sally McClean, Philip Morrow, and Bryan Scotney. A practical evaluation in openstack live migration of vms using 10gb/s interfaces. In *2016 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, pages 346–351, 2016.

[15] Daniel Byrne, Nilufer Onder, and Zhenlin Wang. mPart: Miss-ratio curve guided partitioning in key-value stores. In *Proceedings of the 2018 ACM SIGPLAN International Symposium on Memory Management (ISMM)*, pages 84–95, Philadelphia, PA, June 2018.

[16] Yongtao Cao, Byran J. Smucker, and Timothy J. Robinson. On using the hypervolume indicator to compare Pareto fronts: Applications to multicriteria optimal experimental design. *Journal of Statistical Planning and Inference*, 160:60–74, 2015.

[17] Zhen Cao, Vasily Tarasov, Sachin Tiwari, and Erez Zadok. Towards better understanding of black-box auto-tuning: A comparative analysis for

storage systems. In *USENIX Annual Technical Conference, (ATC)*, pages 893–907, Boston, MA, July 2018.

[18] Kevin K. Chang, Abhijith Kashyap, Hasan Hassan, Saugata Ghose, Kevin Hsieh, Donghyuk Lee, Tianshi Li, Gennady Pekhimenko, Samira Khan, and Onur Mutlu. Understanding latency variation in modern DRAM chips: Experimental characterization, analysis, and optimization. In *Proceedings of the 2016 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science*, SIGMETRICS'16, pages 323–336, New York, NY, USA, 2016. ACM.

[19] Xian Chen, Wenzhi Chen, Zhongyong Lu, Peng Long, Shuiqiao Yang, and Zonghiu Wang. A duplication-aware SSD-based cache architecture for primary storage in virtualization environment. *IEEE Systems Journal*, 11(4):2578–2589, December 2017.

[20] Xunchao Chen, Navid Khoshavi, Jian Zhou, Dan Huang, Ronald F. DeMara, Jun Wang, Wujie Wen, and Yiran Chen. AOS: Adaptive overwrite scheme for energy-efficient MLC STT-RAM cache. In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2016.

[21] Lerong Cheng, Jinjun Xiong, and Lei He. Non-Gaussian statistical timing analysis using second-order polynomial fitting. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(1):130–140, 2008.

[22] Yue Cheng, Aayush Gupta, Anna Povzner, and Ali R. Butt. High performance in-memory caching through flexible fine-grained services. In *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, New York, NY, USA, 2013. Association for Computing Machinery.

[23] Yuxia Cheng, Wenzhi Chen, Zonghui Wang, Xinjie Yu, and Yang Xiang. AMC: an adaptive multi-level cache algorithm in hybrid storage systems. *Concurrency and Computation: Practice and Experience*, 27(16):4230–4246, 2015.

[24] Yuxia Cheng, Yang Xiang, Wenzhi Chen, Houcine Hassan, and Abdulhameed Alelaiwi. Efficient cache resource aggregation using adaptive

multi-level exclusive caching policies. *Future Generation Computer Systems*, 86:964 – 974, 2018.

[25] Davide Chicco and Giuseppe Jurman. The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC Genomics*, 21, 2020.

[26] Anita Choudhary, Mahesh Chandra Govil, Girdhari Singh, Lalit K. Awasthi, Emmanuel S. Pilli, and Divya Kapil. A critical survey of live virtual machine migration techniques. *J. Cloud Comput.*, 6(1), December 2017.

[27] Asaf Cidon, Assaf Eisenman, Mohammad Alizadeh, and Sachin Katti. Dynacache: Dynamic cloud caching. In *7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 15)*, Santa Clara, CA, July 2015. USENIX Association.

[28] Asaf Cidon, Assaf Eisenman, Mohammad Alizadeh, and Sachin Katti. Cliffhanger: Scaling performance cliffs in web memory caches. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 379–392, Santa Clara, CA, March 2016. USENIX Association.

[29] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI'05, page 273–286, USA, 2005. USENIX Association.

[30] Compute Express Link Consortium. Compute express link (cxl). *https://computeexpresslink.org/*, 2025. Accessed: 2025-07-27.

[31] CXL Consortium Technical Task Force. Compute Express Link 2.0 Specification Now Available! *https://computeexpresslink.org/blog/compute-express-link-2-0-specification-now-available-2374/*, December 2020. Accessed: 2025-07-27.

[32] Debendra Das Sharma, Robert Blankenship, and Daniel Berger. An introduction to the compute express link (cxl) interconnect. *ACM Comput. Surv.*, 56(11), July 2024.

[33] Axel de Perthuis de Laillevault, Benjamin Doerr, and Carola Doerr. Money for nothing: Speeding up evolutionary algorithms through better initialization. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO '15. ACM, July 2015.

[34] Jeffrey Dean and Luiz André Barroso. The tail at scale. *Communications of the ACM*, 56(2):74–80, February 2013.

[35] Dinero IV trace-driven uniprocessor cache simulator. *http://pages.cs.wisc.edu/~markhill/DineroIV/*.

[36] Nosayba El-Sayed, Ioan A. Stefanovici, George Amvrosiadis, Andy A. Hwang, and Bianca Schroeder. Temperature management in data centers: Why some (might) like it hot. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS'12, pages 163–174, New York, NY, USA, 2012. ACM.

[37] Tyler Estro, Mário Antunes, Pranav Bhandari, Anshul Gandhi, Geoff Kuenning, Yifei Liu, Carl Waldspurger, Avani Wildani, and Erez Zadok. Guiding simulations of multi-tier storage caches using knee detection. In *31st Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS '23)*, Stony Brook, NY, October 2023. IEEE Computer Society.

[38] Tyler Estro, Pranav Bhandari, Avani Wildani, and Erez Zadok. Desperately seeking ... optimal multi-tier cache configurations. In *Proceedings of the 12th USENIX Workshop on Hot Topics in Storage (HotStorage '20)*, Boston, MA, July 2020. USENIX.

[39] Roja Eswaran, Mingjie Yan, and Kartik Gopalan. Template-aware live migration of virtual machines. In *Proceedings of the Eighth ACM/IEEE Symposium on Edge Computing*, SEC '23, page 336–340, New York, NY, USA, 2024. Association for Computing Machinery.

[40] Diogo Freitas, Luiz Guerreiro Lopes, and Fernando Morgado Dias. Particle swarm optimisation: A historical review up to the current developments. *Entropy*, 22, 2020.

[41] Jianyu Fu, Dulcardo Arteaga, and Ming Zhao. Locality-driven MRC construction and cache allocation. In *Proceedings of the 27th International*

*Symposium on High-Performance Parallel and Distributed Computing*, HPDC '18, pages 19–20, New York, NY, USA, 2018. ACM.

[42] M. García-Arnau, D. Manrique, J. Ríos, and A. Rodríguez-Patón. Initialization method for grammar-guided genetic programming. *Knowledge-Based Systems*, 20(2):127–133, 2007. AI 2006.

[43] Patrice Godefroid and Sarfraz Khurshid. Exploring very large state spaces using genetic algorithms. In Joost-Pieter Katoen and Perdita Stevens, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 266–280, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[44] Ron Goldman. Curvature formulas for implicit curves and surfaces. *Computer Aided Geometric Design*, 22(7):632–658, October 2005.

[45] 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, Carlsbad, CA, July 2022. USENIX Association.

[46] Andreas Grapentin, Felix Eberhardt, Tobias Zagorni, Andreas Polze, Michele Gazzetti, and Christian Pinto. Zero-Copy, Minimal-Blackout Virtual Machine Migrations Using Disaggregated Shared Memory. In João Bispo, Sotirios Xydis, Serena Curzel, and Luís Miguel Sousa, editors, *15th Workshop on Parallel Programming and Run-Time Management Techniques for Many-Core Architectures and 13th Workshop on Design Tools and Architectures for Multicore Embedded Computing Platforms (PARMA-DITAM 2024)*, volume 116 of *Open Access Series in Informatics (OASIcs)*, pages 3:1–3:13, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[47] Wei Lin Guay, Sven-Arne Reinemo, Bjørn Dag Johnsen, Chien-Hua Yen, Tor Skeie, Olav Lysne, and Ola Tørudbakken. Early experiences with live migration of sr-iov enabled infiniband. *Journal of Parallel and Distributed Computing*, 78:39–52, 2015.

[48] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. Rdma over commodity ethernet at scale. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16,

page 202–215, New York, NY, USA, 2016. Association for Computing Machinery.

[49] Ubaid Ullah Hafeez, Muhammad Wajahat, and Anshul Gandhi. ElMem: Towards an elastic memcached system. In *Proceedings of the 38th IEEE International Conference on Distributed Computing Systems*, pages 278–289, Vienna, Austria, 2018.

[50] Ubaid Ullah Hafeez, Muhammad Wajahat, and Anshul Gandhi. ElMem: Towards an elastic memcached system. In *Proceedings of the 38th IEEE International Conference on Distributed Computing Systems*, pages 278–289, Vienna, Austria, 2018.

[51] Alireza Haghdoost. Sim-ideal, Dec 2013. *https://github.com/arh/sim-ideal/tree/master*.

[52] Md E. Haque, Yong hun Eom, Yuxiong He, Sameh Elnikety, Ricardo Bianchini, and Kathryn S. McKinley. Few-to-many: Incremental parallelism for reducing tail latency in interactive services. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS'15, pages 161–175, New York, NY, USA, 2015. ACM.

[53] Raseena M. Haris, Mahmoud Barhamgi, Armstrong Nhlabatsi, and Khaled M. Khan. Optimizing pre-copy live virtual machine migration in cloud computing using machine learning-based prediction model. *Computing*, 106(9):3031–3062, July 2024.

[54] Lulu He, Zhibin Yu, and Hai Jin. FractalMRC: Online cache miss rate curve prediction on commodity systems. In *IEEE 26th International Parallel and Distributed Processing Symposium*, pages 1341–1351, 2012.

[55] Qinlu He, Pengze Gao, Fan Zhang, Genqing Bian, Weiqi Zhang, and Zhen Li. Design and optimization of a distributed file system based on rdma. *Applied Sciences*, 13(15), 2023.

[56] John H. Holland. *Adaptation in Natural and Artificial Systems*. The MIT Press, 1992.

[57] Xiameng Hu, Xiaolin Wang, Lan Zhou, Yingwei Luo, Chen Ding, and Zhenlin Wang. Kinetic modeling of data eviction in cache. In *2016 USENIX*

*Annual Technical Conference (USENIX ATC 16)*, pages 351–364, Denver, CO, June 2016. USENIX Association.

[58] Xiameng Hu, Xiaolin Wang, Lan Zhou, Yingwei Luo, Zhenlin Wang, Chen Ding, and Chencheng Ye. Fast miss ratio curve modeling for storage cache. *ACM Transactions on Storage (TOS)*, 14:12:1–12:34, 2018.

[59] Wei Huang, Qi Gao, Jiuxing Liu, and Dhabaleswar K. Panda. High performance virtual machine migration with rdma over modern interconnects. In *2007 IEEE International Conference on Cluster Computing*, pages 11–20, 2007.

[60] Wei Huang, Jiuxing Liu, Matthew Koop, Bulent Abali, and Dhabaleswar Panda. Nomad: migrating os-bypass networks in virtual machines. In *Proceedings of the 3rd International Conference on Virtual Execution Environments*, VEE '07, page 158–168, New York, NY, USA, 2007. Association for Computing Machinery.

[61] Khaled Z. Ibrahim, Steven Hofmeyr, Costin Iancu, and Eric Roman. Optimized pre-copy live migration for memory intensive applications. In *SC '11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–11, 2011.

[62] C. Isci, J. Liu, B. Abali, J. O. Kephart, and J. Kouloheris. Improving server utilization using fast virtual machine migration. *IBM Journal of Research and Development*, 55(6):4:1–4:12, 2011.

[63] Dr. Shaily Jain and Nitin Nitin. Memory map: A multiprocessor cache simulator. *Journal of Electrical and Computer Engineering*, 2012, 09 2012.

[64] Myeongjae Jeon, Saehoon Kim, Seung-won Hwang, Yuxiong He, Sameh Elnikety, Alan L. Cox, and Scott Rixner. Predictive parallelization: Taming tail latencies in web search. In *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval*, SIGIR'14, pages 253–262, New York, NY, USA, 2014. ACM.

[65] N. Jeremic, G. M´uhl, A. Busse, and J. Richling. The pitfalls of deploying solid-state drive RAIDs. In *Proceedings of the 4th Annual International Conference on Systems and Storage*, SYSTOR '11. ACM, 2011.

[66] Changyeon Jo, Hyunik Kim, and Bernhard Egger. Instant virtual machine live migration. In Karim Djemame, Jörn Altmann, José Ángel Bañares, Orna Agmon Ben-Yehuda, Vlado Stankovski, and Bruno Tuffin, editors, *Economics of Grids, Clouds, Systems, and Services*, pages 155–170, Cham, 2020. Springer International Publishing.

[67] M. Jung and M. Kandemir. Revisiting widely held SSD expectations and rethinking system-level implications. In *Proceedings of the ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '13, pages 203–216, New York, NY, USA, 2013. ACM.

[68] Borhan Kazimipour, Xiaodong Li, and A. K. Qin. A review of population initialization techniques for evolutionary algorithms. In *2014 IEEE Congress on Evolutionary Computation (CEC)*, pages 2585–2592, 2014.

[69] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948 vol.4, 1995.

[70] Patrick Kennedy. Lenovo has a cxl memory monster with 128× 128 gb ddr5 dimms. *ServeTheHome*, November 2024. Accessed: 2025-07-25.

[71] Ricardo Koller, Akshat Verma, and Raju Rangaswami. Generalized ERSS tree model: Revisiting working sets. *Performance Evaluation*, 67:1139–1154, 2010.

[72] Iwona Kotlarska, Andrzej Jackowski, Krzysztof Lichota, Michal Welnicki, Cezary Dubnicki, and Konrad Iwanicki. InftyDedup: Scalable and Cost-Effective cloud tiering with deduplication. In *21st USENIX Conference on File and Storage Technologies (FAST 23)*, pages 33–48, Santa Clara, CA, February 2023. USENIX Association.

[73] Saku Kukkonen and Jouni Lampinen. Gde3: The third evolution step of generalized differential evolution. In *2005 IEEE congress on evolutionary computation*, volume 1, pages 443–450. IEEE, 2005.

[74] Vaneet Kumar, Aleem Ali, Payal Mittal, Ibrahim Aqeel, Mohammed Shuaib, Shadab Alam, and Mohammed Y. Aalsalem. E2svm: Electricity-efficient sla-aware virtual machine consolidation approach in cloud data centers. *PLOS ONE*, 19(6):1–17, 06 2024.

[75] Nikita Lazarev, Varun Gohil, James Tsai, Andy Anderson, Bhushan Chitlur, Zhiru Zhang, and Christina Delimitrou. Sabre: Hardware-Accelerated snapshot compression for serverless MicroVMs. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 1–18, Santa Clara, CA, July 2024. USENIX Association.

[76] Scott Levy, Patrick Widener, Craig Ulmer, and Todd Kordenbrock. The case for explicit reuse semantics for rdma communication. In *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 879–888, 2020.

[77] Jialin Li, Naveen Kr. Sharma, Dan R. K. Ports, and Steven D. Gribble. Tales of the tail: Hardware, OS, and application-level sources of tail latency. In *Proceedings of the ACM Symposium on Cloud Computing*, SoCC'14, pages 9:1–9:14, New York, NY, USA, 2014. ACM.

[78] Miqing Li and Xin Yao. Quality evaluation of solution sets in multiobjective optimisation: A survey. *ACM Comput. Surv.*, 52(2), March 2019.

[79] Z. Li, M. Chen, A. Mukker, and E. Zadok. On the trade-offs among performance, energy, and endurance in a versatile hybrid drive. *ACM Transactions on Storage (TOS)*, 11(3), July 2015.

[80] Z. Li, M. Chen, and E. Zadok. Greendm: A versatile hybrid drive for energy and performance. Technical report, Stony Brook University, 2013. Paper under review.

[81] Z. Li, A. Mukker, and E. Zadok. On the importance of evaluating storage systems' $costs. In *Proceedings of the 6th USENIX Conference on Hot Topics in Storage and File Systems*, HotStorage'14, 2014.

[82] Chieh-Jan Mike Liang, Jie Liu, Liqian Luo, Andreas Terzis, and Feng Zhao. RACNet: A high-fidelity data center sensing network. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, SenSys'09, pages 15–28, New York, NY, USA, 2009. ACM.

[83] Hsu-Cheng Lin, Han-Chiang Chen, and Shun-Chieh Lin. A pre-registered buffering method for iser-based storage over ip san. In *INC2010: 6th International Conference on Networked Computing*, pages 1–4, 2010.

[84] Jianpeng Lin, Weiwei Lin, Wentai Wu, Wenjun Lin, and Keqin Li. Energy-aware virtual machine placement based on a holistic thermal model for cloud data centers. *Future Generation Computer Systems*, 161:302–314, 2024.

[85] Jiawei Liu, Gongming Zhao, Hongli Xu, Peng Yang, Baoqing Wang, and Chunming Qiao. Toward a service availability-guaranteed cloud through vm placement. *IEEE/ACM Trans. Netw.*, 32(5):3993–4008, May 2024.

[86] Jinshu Liu, Hamid Hadian, Hanchen Xu, and Huaicheng Li. Tiered memory management beyond hotness. In *19th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2025.

[87] Zhang Liu, Hee Won Lee, Yu Xiang, Dirk Grunwald, and Sangtae Ha. eMRC: Efficient miss rate approximation for multi-tier caching. In *19th USENIX Conference on File and Storage Technologies (FAST 21)*. USENIX Association, February 2021.

[88] Y. Lu, J. Shu, and W. Zheng. Extending the lifetime of flash-based storage through reducing write amplification from file systems. In *In Proceedings of the 11th USENIX Symposium on File and Storage Technologies (FAST '13)*, 2013.

[89] Edson Ramiro Lucas Filho, Lambros Odysseos, Yang Lun, Fu Kebo, and Herodotos Herodotou. DITIS: A distributed tiered storage simulator. *Infocommunications Journal*, XIV(4):18–25, December 2022.

[90] Zhongkun Ma and Guy A. E. Vandenbosch. Impact of random number generators on the performance of particle swarm optimization in antenna design. In *2012 6th European Conference on Antennas and Propagation (EUCAP)*, pages 925–929, 2012.

[91] Rano Mal and Yul Chu. A flexible multi-core functional cache simulator (FM-SIM). In *Proceedings of the Summer Simulation Multi-Conference*, SummerSim '17, San Diego, CA, USA, 2017. Society for Computer Simulation International.

[92] Hasan Al Maruf, Hao Wang, Abhishek Dhanotia, Johannes Weiner, Niket Agarwal, Pallab Bhattacharya, Chris Petersen, Mosharaf Chowdhury, Shobhit Kanaujia, and Prakash Chauhan. Tpp: Transparent page placement

for cxl-enabled tiered-memory. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ASPLOS 2023, page 742–755, New York, NY, USA, 2023. Association for Computing Machinery.

[93] Richard L. Mattson, Jan Gecsei, Donald R. Slutz, and Irving L. Traiger. Evaluation techniques for storage hierarchies. *IBM Systems Journal*, 9(2):78–117, 1970.

[94] Nimrod Megiddo and Dharmendra Modha. ARC: A self-tuning, low overhead replacement cache. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST '03)*, pages 115–130, San Francisco, CA, March 2003. USENIX Association.

[95] Michael Mesnier, Feng Chen, Tian Luo, and Jason B. Akers. Differentiated storage services. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 57–70, New York, NY, USA, 2011. ACM.

[96] Frank Mietke, Robert Rex, Robert Baumgartl, Torsten Mehlan, Torsten Hoefler, and Wolfgang Rehm. Analysis of the memory registration process in the mellanox infiniband software stack. In *Proceedings of the 12th International Conference on Parallel Processing*, Euro-Par'06, page 124–133, Berlin, Heidelberg, 2006. Springer-Verlag.

[97] Radhika Mittal, Alexander Shpiner, Aurojit Panda, Eitan Zahavi, Arvind Krishnamurthy, Sylvia Ratnasamy, and Scott Shenker. Revisiting network support for rdma. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, page 313–326, New York, NY, USA, 2018. Association for Computing Machinery.

[98] Seyed Jalaleddin Mousavirad, Azam Asilian Bidgoli, and Shahryar Rahnamayan. Tackling deceptive optimization problems using opposition-based DE with center-based Latin hypercube initialization. In *2019 14th International Conference on Computer Science & Education (ICCSE)*, pages 394–400, 2019.

[99] D. Narayanan, A. Donnelly, and A. Rowstron. Write off-loading: Practical power management for enterprise storage. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST 2008)*, 2008.

[100] Dushyanth Narayanan, Austin Donnelly, and Antony Rowstron. MSR Cambridge traces (SNIA IOTTA trace set 388). In Geoff Kuenning, editor, *SNIA IOTTA Trace Repository*. Storage Networking Industry Association, March 2007.

[101] Iyswarya Narayanan, Di Wang, Myeongjae Jeon, Bikash Sharma, Laura Caulfield, Anand Sivasubramaniam, Ben Cutler, Jie Liu, Badriddine Khessib, and Kushagra Vaid. SSD failures in datacenters: What? when? and why? In *Proceedings of the Ninth ACM Israeli Experimental Systems Conference (SYSTOR '16)*, pages 7:1–7:11, Haifa, Israel, May 2016. ACM.

[102] Anant Vithal Nori, Jayesh Gaur, Siddarth Rai, Sreenivas Subramoney, and Hong Wang. Criticality aware tiered cache hierarchy: A fundamental relook at multi-level cache hierarchies. In *45th ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*, pages 96–109, June 2018.

[103] Michael Oberg, Henry M. Tufo, Theron Voran, and Matthew Woitaszek. Evaluation of RDMA over ethernet technology for building cost effective linux clusters. In *Proceedings of the 7$^{th}$ LCI International Conference on Linux Clusters: The HPC Revolution*, pages 1–13, Norman, Oklahoma, USA, May 2006. Linux Clusters Institute.

[104] Massachusetts Institute of Technology. DynamoRIO: Dynamic instrumentation tool platform, February 2009. *http://www.dynamorio.org/*.

[105] Christian Pinto, Dimitris Syrivelis, Michele Gazzetti, Panos Koutsovasilis, Andrea Reale, Kostas Katrinis, and H. Peter Hofstee. Thymesisflow: A software-defined, hw/sw co-designed interconnect stack for rack-scale memory disaggregation. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 868–880, 2020.

[106] Lenovo Press. ThinkSystem SR860 V3 Server Product Guide (Form LP1606). *https://lenovopress.lenovo.com/servers/thinksystemâĂŚv3/sr860âĂŚv3*, 2025. Last updated: June 24 2025; Accessed: 2025-07-25.

[107] ASRock Rack. ASRock Rack GNRD8-2L2T CEB Server Motherboard (Intel Xeon 6700-series support). *https://www.asrockrack.com/general/productdetail.asp?Model=GNRD8-2L2T\#Specifications*, 2025. Accessed: 2025-07-25.

[108] Shahryar Rahnamayan, Hamid R. Tizhoosh, and Magdy M.A. Salama. A novel population initialization method for accelerating evolutionary algorithms. *Computers & Mathematics with Applications*, 53(10):1605–1614, 2007.

[109] Sundaresan Rajasekaran, Shaohua Duan, Wei Zhang, and Timothy Wood. Multi-cache: Dynamic, efficient partitioning for multi-tier caches in consolidated VM environments. In *IEEE International Conference on Cloud Engineering (IC2E)*, pages 182–191. IEEE, April 2016.

[110] Urs Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing*, 1(3):244–256, 1972.

[111] Mohammad J. Rashti and Ahmad Afsahi. Improving rdma-based mpi eager protocol for frequently-used buffers. In *2009 IEEE International Symposium on Parallel & Distributed Processing*, pages 1–8, 2009.

[112] Waleed Reda, Marco Canini, Dejan Kostić, and Simon Peter. RDMA is turing complete, we just did not know it yet! In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 71–85, Renton, WA, April 2022. USENIX Association.

[113] Adam Ruprecht, Danny Jones, Dmitry Shiraev, Greg Harmon, Maya Spivak, Michael Krebs, Miche Baker-Harvey, and Tyler Sanderson. Vm live migration at scale. In *Proceedings of the 14th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '18, page 45–56, New York, NY, USA, 2018. Association for Computing Machinery.

[114] R. Salkhordeh, S. Ebrahimi, and H. Asadi. Reca: An efficient reconfigurable cache architecture for storage systems with online workload characterization. *IEEE Transactions on Parallel and Distributed Systems*, 29(7):1605–1620, July 2018.

[115] Stan Salvador and Philip Chan. Determining the number of clusters/segments in hierarchical clustering/segmentation algorithms. In *16th IEEE International Conference on Tools With Artificial Intelligence*, pages 576–584. IEEE, 2004.

[116] Ricardo Santana, Steven Lyons, Ricardo Koller, Raju Rangaswami, and Jason Liu. To ARC or not to ARC. In *HotStorage*, 2015.

[117] Ville Satopaa, Jeannie Albrecht, David Irwin, and Barath Raghavan. Finding a "kneedle" in a haystack: Detecting knee points in system behavior. In *31st International Conference on Distributed Computing Systems Workshops*, pages 166–171, Minneapolis, MN, June 2011. IEEE.

[118] Priya Sehgal, Vasily Tarasov, and Erez Zadok. Evaluating performance and energy in file system server workloads. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST '10)*, pages 253–266, San Jose, CA, February 2010. USENIX Association.

[119] Debendra Das Sharma and Siamak Tavallaei. Compute Express Link 1.1 Specification: Now Available to Members. *https://computeexpresslink.org/blog/compute-express-link-1-1-specification-now-available-to-members-2339/*, March 2020. Accessed: 2025-07-27.

[120] Xiang Song, Jicheng Shi, Ran Liu, Jian Yang, and Haibo Chen. Parallelizing live migration of virtual machines. In *Proceedings of the 9th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '13, page 85–96, New York, NY, USA, 2013. Association for Computing Machinery.

[121] Carl Staelin and Hector Garcia-molina. Clustering active disk data to improve disk performance. Technical Report CS-TR-298-9, Princeton University, NJ, USA, 1990.

[122] Dragan Stancevic. nilmigration: Nearly instantaneous live migration of virtual machines over cxl. *https://nil-migration.org/*, September 2022. BoF presented at Linux Storage, Filesystem, Memory Management & BPF Summit, May 2023.

[123] R. Storn. On the usage of differential evolution for function optimization. In *Proceedings of North American Fuzzy Information Processing*, pages 519–523, 1996.

[124] Supermicro. Supermicro X14 Systems: Max-Performance Servers with Intel Xeon 6. *https://www.supermicro.com/en/products/x14*, 2025. Accessed: 2025-07-25.

[125] Lalith Suresh, Marco Canini, Stefan Schmid, and Anja Feldmann. C3: Cutting tail latency in cloud data stores via adaptive replica selection. In

*Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, NSDI'15, pages 513–527, Berkeley, CA, USA, 2015. USENIX Association.

[126] David K. Tam, Reza Azimi, Livio B. Soares, and Michael Stumm. RapidMRC: approximating L2 miss rate curves on commodity systems for online optimizations. *ACM Sigplan Notices*, 44(3):121–132, 2009.

[127] K.C. Tan, T.H. Lee, and E.F. Khor. Evolutionary algorithms for multi-objective optimization: performance assessments and comparisons. In *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546)*, volume 2, pages 979–986 vol. 2, 2001.

[128] Elvira Teran, Zhe Wang, and Daniel A. Jiménez. Perceptron learning for reuse prediction. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–12. IEEE, 2016.

[129] Xavier Tolsa. Principal values for the cauchy integral and rectifiability. *Proceedings of the American Mathematical Society*, 128(7):2111–2119, 2000.

[130] Thomas Tometzki and Sebastian Engell. Systematic initialization techniques for hybrid evolutionary algorithms for solving two-stage stochastic mixed-integer programs. *IEEE Transactions on Evolutionary Computation*, 15:196–214, 2011.

[131] Tom's hardware: For the hardcore pc enthusiast. *www.tomshardware.com*.

[132] Anjul Tyagi, Zhen Cao, Tyler Estro, Klaus Mueller, and Erez Zadok. ICE: Interactive configuration explorer for high dimensional categorical parameter spaces. In *IEEE Conference on Visual Analytics Science and Technology (VAST 2019)*, October 2019.

[133] Anjul Tyagi, Tyler Estro, Geoff Kuenning, Erez Zadok, and Klaus Mueller. PC-Expo: A metrics-based interactive axes reordering method for parallel coordinate displays. *IEEE Transactions on Visualization and Computer Graphics*, October 2022.

[134] Musa Unal, Vishal Gupta, Yueyang Pan, Yujie Ren, and Sanidhya Kashyap. Tolerate it if you cannot reduce it: Handling latency in tiered memory. In *Proceedings of the 2025 Workshop on Hot Topics in Operating Systems*,

HotOS '25, page 50–57, New York, NY, USA, 2025. Association for Computing Machinery.

[135] UserBenchmark. *www.userbenchmark.com*.

[136] Nguyen Quang Uy, Nguyen Xuan Hoai, RI McKay, and Pham Minh Tuan. Initialising PSO with randomised low-discrepancy sequences: the comparative results. In *2007 IEEE Congress on Evolutionary Computation*, pages 1985–1992, 2007.

[137] A. Verma, R. Koller, L. Useche, and R. Rangaswami. SRCMap: Energy proportional storage using dynamic consolidation. In *Proceedings of the 8th USENIX Conference on File and Storage Technologies*, FAST'10, 2010.

[138] Akshat Verma, Ricardo Koller, Luis Useche, and Raju Rangaswami. FIU traces (SNIA IOTTA trace set 390). In Geoff Kuenning, editor, *SNIA IOTTA Trace Repository*. Storage Networking Industry Association, March 2009.

[139] Giuseppe Vietri, Liana V. Rodriguez, Wendy A. Martinez, Steven Lyons, Jason Liu, Raju Rangaswami, Ming Zhao, and Giri Narasimhan. Driving cache replacement with ML-based LeCaR. In *Proceedings of the 10th USENIX Workshop on Hot Topics in Storage (HotStorage '18)*, Boston, MA, July 2018. USENIX.

[140] Muhammad Wajahat, Aditya Yele, Tyler Estro, Anshul Gandhi, and Erez Zadok. Analyzing the distribution fit for storage workload and internet traffic traces. *Performance Evaluation*, pages 102–121, 2020.

[141] Carl A. Waldspurger, Nohhyun Park, Alex Garthwaite, and Irfan Ahmad. Efficient MRC construction with SHARDS. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST '15)*, Santa Clara, CA, February 2015. USENIX Association.

[142] Carl A. Waldspurger, Trausti Saemundson, Irfan Ahmad, and Nohhyun Park. Cache modeling and optimization using miniature simulations. In *Proceedings of the 2017 USENIX Annual Technical Conference (ATC '17)*, pages 487–498, Berkeley, CA, USA, 2017. USENIX Association.

[143] Han Wan, Xiaopeng Gao, Xiang Long, and Zhiqiang Wang. *GCSim: A GPU-Based Trace-Driven Simulator for Multi-level Cache*, pages 177–190. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[144] Jiangtao Wang, Zhiliang Guo, and Xiaofeng Meng. An efficient design and implementation of multi-level cache for database systems. In *DASFAA*, 2015.

[145] Marcel Weisgut, Daniel Ritter, Pinar Tozun, Lawrence Benson, and Tilmann Rabl. Cxl memory performance for in-memory data processing. *Proceedings of the VLDB Endowment*, 18(9):3119–3133, 2025.

[146] Wikipedia contributors. Compute Express Link. *https://en.wikipedia.org/wiki/Compute_Express_Link*, 2025. Wikipedia, accessed 2025-07-27.

[147] A. Wildani, E. L. Miller, and L. Ward. Efficiently identifying working sets in block I/O streams. In *Proceedings of the 4th Annual International Conference on Systems and Storage*, SYSTOR '11, pages 5:1–5:12. ACM, 2011.

[148] John Wilkes. The Pantheon storage-system simulator, 1996.

[149] Jake Wires, Stephen Ingram, Zachary Drudi, Nicholas J. A. Harvey, and Andrew Warfield. Characterizing storage workloads with counter stacks. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2014)*, Broomfield, CO, October 2014. USENIX Association.

[150] Jiesheng Wu, P. Wyckoff, D. Panda, and R. Ross. Unifier: unifying cache management and communication buffer management for pvfs over infiniband. In *IEEE International Symposium on Cluster Computing and the Grid, 2004. CCGrid 2004.*, pages 523–530, 2004.

[151] 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 *Proceedings of the 18th USENIX Conference on Operating Systems Design and Implementation*, OSDI'24, USA, 2024. USENIX Association.

[152] Jun Xiao, Yaocheng Xiang, Xiaolin Wang, Yingwei Luo, Andy Pimentel, and Zhenlin Wang. Floria: A fast and featherlight approach for predicting cache performance. In *Proceedings of the 37th ACM International Conference on Supercomputing*, ICS '23, page 25–36, New York, NY, USA, 2023. Association for Computing Machinery.

[153] Yunjing Xu, Zachary Musgrave, Brian Noble, and Michael Bailey. Bobtail: Avoiding long tails in the cloud. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation (NSDI '13)*, pages 329–342, Berkeley, CA, USA, 2013. USENIX Association.

[154] Jian Yang, Joseph Izraelevitz, and Steven Swanson. FileMR: Rethinking RDMA networking for scalable persistent memory. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 111–125, Santa Clara, CA, February 2020. USENIX Association.

[155] Juncheng Yang. PyMimircache. *https://github.com/1a1a11a/PyMimircache*. Retrieved April 17, 2019.

[156] Juncheng Yang, Reza Karimi, Trausti Sæmundsson, Avani Wildani, and Ymir Vigfusson. MITHRIL: mining sporadic associations for cache prefetching. *CoRR*, abs/1705.07400, 2017.

[157] Juncheng Yang, Reza Karimi, Trausti Sæmundsson, Avani Wildani, and Ymir Vigfusson. Mithril: Mining sporadic associations for cache prefetching. In *Proceedings of the 2017 Symposium on Cloud Computing*, SoCC '17, pages 66–79, New York, NY, USA, 2017. ACM.

[158] Guo Yu, Lianbo Ma, Yaochu Jin, Wenli Du, Qiqi Liu, and Hengmin Zhang. A survey on knee-oriented multiobjective evolutionary optimization. *IEEE Transactions on Evolutionary Computation*, 26(6):1452–1472, December 2022.

[159] Guohui Zhang, Liang Gao, and Yang Shi. An effective genetic algorithm for the flexible job-shop scheduling problem. *Expert Systems with Applications*, 38(4):3563–3573, 2011.

[160] Lei Zhang, Reza Karimi, Irfan Ahmad, and Ymir Vigfusson. Optimal data placement for heterogeneous cache, memory, and storage systems. In *Proceedings of the ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '20, 2020.

[161] Weida Zhang, King Tin Lam, and Cho Li Wang. Adaptive live vm migration over a wan: Modeling and implementation. In *2014 IEEE 7th International Conference on Cloud Computing*, pages 368–375, 2014.

[162] Xinyu Zhang, Mário Antunes, Tyler Estro, Erez Zadok, and Klaus Mueller. Smart starts: Accelerating convergence through uncommon region exploration. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '25 Companion, pages 547–550, New York, NY, USA, 2025. Association for Computing Machinery.

[163] Xinyu Zhang, Tyler Estro, Geoff Kuenning, Erez Zadok, and Klaus Mueller. Into the Void: Mapping the Unseen Gaps in High Dimensional Data . *IEEE Transactions on Visualization & Computer Graphics*, (01):1–13, May 2025.

[164] Yuhong Zhong, Daniel S. Berger, Carl Waldspurger, Ryan Wee, Ishwar Agarwal, Rajat Agarwal, Frank Hady, Karthik Kumar, Mark D. Hill, Mosharaf Chowdhury, and Asaf Cidon. Managing memory tiers with CXL in virtualized environments. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 37–56, Santa Clara, CA, July 2024. USENIX Association.

[165] Yuhong Zhong, Daniel S. Berger, Pantea Zardoshti, Enrique Suarez, Jacob Nelson, Dan R. K. Ports, Antonis Psistakis, Joshua Fried, and Asaf Cidon. Oasis: Pooling pcie devices over cxl to boost utilization. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, October 2025. To appear.

[166] Timothy Zhu, Anshul Gandhi, Mor Harchol-Balter, and Michael A. Kozuch. Saving cash by using less cache. In *Proceedings of the 4th USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'12, page 3, USA, 2012. USENIX Association.