Appears in the proceedings of the 12th USENIX Workshop on Hot Topics in Storage (HotStorage '20)

Desperately Seeking ... Optimal Multi-Tier Cache Configurations

Tyler Estro,[1] Pranav Bhandari,[2] Avani Wildani,[2] and Erez Zadok[1]
[1]*Stony Brook University* *and* [2]*Emory University*

Abstract

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.

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 [15, 16, 21], 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 [4, 14, 46, 48, 58].

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 [63], dynamically switch between replacement algorithms [45, 46, 53], or enhance existing eviction policies [2]. I/O classification has been used to enforce caching policies and improve file system performance [39]. Multi-tier cache eviction algorithms that are aware of some or all layers in the hierarchy at any given time are being developed [14]. The challenges of cache resource allocation and provisioning are being investigated as well [5, 31]. Zhang *et al.* introduced CHOPT, a choice-aware, optimal, offline algorithm for data placement in multi-tier systems [65]. 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 [7, 25, 26, 54]. 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. [20].

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 [18, 27, 38, 43, 56], 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 [1, 23, 59]. PyMimircache [62] 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.

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 [10, 13–15, 20, 44, 57].

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 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 [49, 61] and web [5, 17, 24, 28] services, as well as at the hardware level [8, 19, 32, 36]. Inter-cache traffic analysis has been used to design more efficient cache hierarchies in modern microprocessors [42]. Reducing the energy consumption of storage systems is beneficial for the environment, lowers operation costs, and promotes advancements in hardware design [9, 34, 47, 52]. 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) [33, 35].

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.

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

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

Table 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 [3], Tom's Hardware [50], and UserBenchmark [51].

for single-layer or global policies, as well as the ability to easily add new policies. **(4)** *Trace sampling* techniques (*e.g.*, Miniature Simulations [55]) 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 [64] 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 [22]. Moving between configurations in any caching system has a *temporarily* negative impact on performance, until the new caches are fully warmed [12, 66]. 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 [62], 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 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.* [40]. 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) [52] and 106 traces from CloudPhysics [54], 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 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).

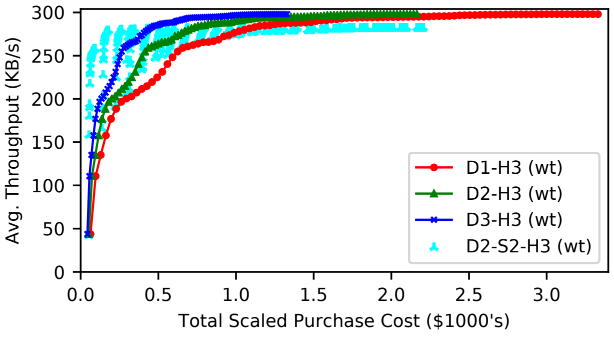

Figure 1: Effects of an intermediate SSD tier (Workload MSR hm-1)

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 3) and then calculated cost and performance metrics using the device specifications described in Table 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. We call this technique *reverse-mini-sim*: the reverse of the miniature simulations technique for down-scaling traces introduced by Waldspurger *et al.* [55]. 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 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 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-H3, blue) for the same cost of a more expensive DRAM (D1-H3) yields overall better performance. Therefore,

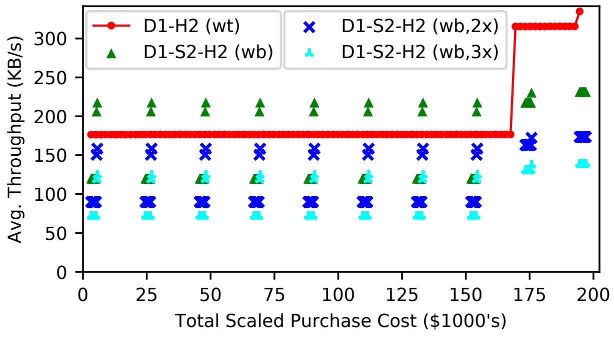

Figure 2: SSD Aging Effects (Workload MSR src2-1). 2× and 3× indicate configurations where S2 has 2–3× increased latency due to the potential effects of SSD aging

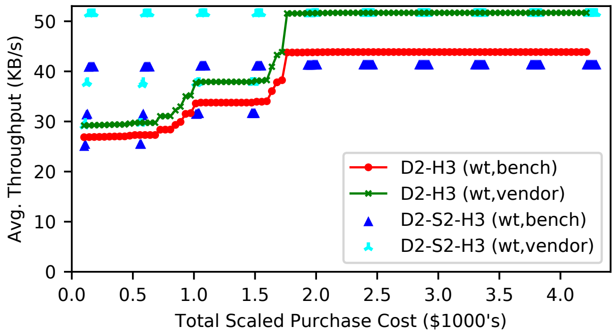

Figure 3: Variation between vendor-reported specs and independently operated benchmarks (Workload MSR web-3)

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 [29,37,41]. 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× [30]. To simulate this effect, we multiplied the latency specifications of device S2 and analyzed the results alongside simulations using its original specifications. Figure 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 money, but also yield a similar or better average throughput over time.

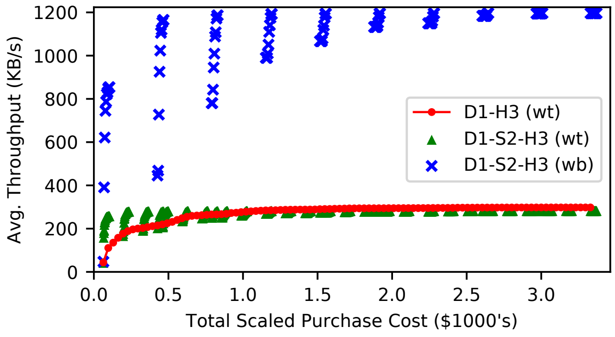

Figure 4: Write-through vs. Write-back policy effects (Workload MSR hm-1)

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 [3], Tom's Hardware [50] and UserBenchmark [51].

In this experiment we show the difference between numbers reported by vendors and others. Figure 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 [3] (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 compares write-through and write-back policies (policy implementations described in Section 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.

5 Related Work

Modern processors are designed with multiple cores, each containing multiple tiers of cache, as well as a shared cache. Several architecture simulators have been developed and are widely used by industry and academia to facilitate engineering, research, and education [18, 27, 38, 43, 56]. One example of this is gem5, a popular architecture simulator that has been actively developed for nearly two decades [6]. It supports multiple ISAs and can accurately model complex multi-level non-uniform cache hierarchies with heterogeneous memories. Architecture simulators such as gem5 have great value, but are fundamentally different than storage simulators. Complex cache replacement algorithms that are designed specifically for storage devices (*e.g.*, SAC [11] and GCaR [60]) could not be reasonably implemented in an architecture simulator. Architecture simulators are typically driven by binaries or instruction-level traces, and could not operate on traces captured at the block, network, or system call layers.

Conversely, storage cache simulators are scarce and lacking in features. Accusim was developed to evaluate the performance impact of kernel prefetching [1]. It was designed specifically for file system caching and can not to model *n* tiers. SimIdeal is a multi-tier simulator that implements several cache replacement and write policies [23]. It hard-codes the number of tiers to four and forces evictions to the immediate lower layer, and thus cannot support inclusive caching. There are also a handful of outdated simulators such as Pantheon [59]. Unfortunately, there are few storage cache simulators available, and caching research is commonly done using proprietary simulators that are not available publicly.

6 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.

7 Acknowledgements

We thank the anonymous HotStorage reviewers, our shepherd Michael Mesnier, and Carl Waldspurger for their valuable feedback. This work was made possible in part thanks to Dell-EMC, NetApp, and IBM support; and NSF awards CCF-1918225, CNS-1900706, CNS-1729939, CNS-1755958, and CNS-1730726.

8 Discussion Topics

Our proposal includes redefining best practices in cache research and the development of a sophisticated simulator with many features. We find the following discussion topics to be of interest and valuable to this line of work:

1. What features should be included in a multi-tier cache simulator? We give a high-level view of the features we believe to be necessary in Section 3, but are we missing anything important?

2. How a feature is implemented determines how useful it is for the research community. For example, eviction policies can include a single layer, global awareness, or machine-learning algorithms; it can also have support for users to easily define their own. What is important within each feature we implement?

3. We consider monetary cost to be the primary metric when designing any caching system. How important is monetary cost? Are there more important metrics to consider?

4. The ability to reconfigure a cache hierarchy in real-time is considerably valuable, especially in multi-tenant scenarios where resizing involves repartitioning across tenants. What is a good time-frame for a simulator to produce valuable results that are not already stale? What policy parameters could be changed dynamically?

5. We propose best practices in cache research that should include the evaluation of multi-tier hierarchies, analyzed using a diverse set of metrics. What else should be a part of best practice?

6. Under what circumstances would evaluating caching (system performance, design, algorithms, etc.) not benefit from a multi-tier analysis?

7. Are there any issues we are not taking into consideration (*e.g.*, missing features, future technologies, algorithmic solutions)?

8. We are currently developing a general, *n*-level I/O cache simulator by building on top of PyMimircache. Is there a better simulator to use as a foundation, and if so, why is it better?

References

[1] Accusim: Accurate simulation of cache replacement algorithms, March 2020. *https://engineering.purdue.edu/~ychu/accusim/*.

[2] 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.

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

[4] Dulcardo Arteaga, Jorge Cabrera-Gámez, Jing Xu, Swaminathan Sundararaman, and Ming Zhao. Cloudcache: On-demand flash cache management for cloud computing. In *FAST*, 2016.

[5] 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 *OSDI*, 2018.

[6] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 simulator. *SIGARCH Computer Architecture News*, 39(2):1—7, August 2011.

[7] Daniel Byrne, Nilufer Onder, and Zhenlin Wang. mpart: Miss-ratio curve guided partitioning in key-value stores. In *ISMM*, 2018.

[8] 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.

[9] X. Chen, N. Khoshavi, J. Zhou, D. Huang, R. F. DeMara, J. Wang, W. Wen, and Y. 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.

[10] 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.

[11] Zhiguang Chen, Nong Xiao, and Fang Liu. Sac: Rethinking the cache replacement policy for ssd-based storage systems. In *Proceedings of the 5th Annual International Systems and Storage Conference*, SYSTOR '12, New York, NY, USA, 2012. Association for Computing Machinery.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

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

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

[19] 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.

[20] 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.

[21] U. U. Hafeez, M. Wajahat, and A. 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.

[22] U. U. Hafeez, M. Wajahat, and A. Gandhi. Elmem: Towards an elastic memcached system. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 278–289, 2018.

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

[24] 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.

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

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

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

[28] 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.

[29] N. Jeremic, G. Ḿuhl, 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.

[30] 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.

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

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] 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.

[42] A. V. Nori, J. Gaur, S. Rai, S. Subramoney, and H. Wang. Criticality aware tiered cache hierarchy: A fundamental relook at multi-level cache hierarchies. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 96–109, June 2018.

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

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

[45] 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.

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

[47] 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)*, pages 253–266, San Jose, CA, February 2010. USENIX Association.

[48] 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.

[49] 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.

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

[51] Userbenchmark. *www.userbenchmark.com*.

[52] 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.

[53] 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 *HotStorage*, 2018.

[54] Carl A. Waldspurger, Nohhyun Park, Alex Garthwaite, and Irfan Ahmad. Efficient mrc construction with shards. In *FAST*, 2015.

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

[56] Han Wan, Xiaopeng Gao, Xiang Long, and Zhiqiang Wang. Gcsim: A gpu-based trace-driven simulator for multi-level cache. In Yong Dou, Ralf Gruber, and Josef M. Joller, editors, *Advanced Parallel Processing Technologies*, pages 177–190, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

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

[58] 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.

[59] John Wilkes. The pantheon storage-system simulator. 1996.

[60] Suzhen Wu, Yanping Lin, Bo Mao, and Hong Jiang. Gcar: Garbage collection aware cache management with improved performance for flash-based ssds. In *Proceedings of the 2016 International Conference on Supercomputing*, ICS '16, New York, NY, USA, 2016. Association for Computing Machinery.

[61] 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.

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

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

[64] 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.

[65] 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. To appear.

[66] 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.