2023 31st International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS) | 979-8-3503-1948-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/MASCOTS59514.2023.10387545

Guiding Simulations of Multi-Tier Storage Caches Using Knee Detection

Tyler Estro,* Mário Antunes,† Pranav Bhandari,‡ Anshul Gandhi,* Geoff Kuenning,§
Yifei Liu,* Carl Waldspurger,¶ Avani Wildani,‡ and Erez Zadok*
*Stony Brook University, Stony Brook, NY, USA. Email: {testro, anshul, yifeliu, ezk}@cs.stonybrook.edu
†Instituto de Telecomunicações, Universidade de Aveiro,
Campus Universitário de Santiago, 3810-193 Aveiro, Portugal. Email: mario.antunes@av.it.pt
‡Emory University, Atlanta, GA, USA. Email: pranav.bhandari@emory.edu, agadani@gmail.com
§Harvey Mudd College, Claremont, CA, USA. Email: geoff@cs.hmc.edu
¶Carl Waldspurger Consulting, Palo Alto, CA, USA. Email: carl@waldspurger.org

***Abstract*—Simulating storage cache hierarchies enables efficient exploration of their configuration space, including diverse topologies, parameters and policies, and devices with varied performance characteristics, while avoiding expensive physical experiments. Miss Ratio Curves (MRCs) efficiently characterize the performance of a cache over a range of cache sizes. These useful tools reveal "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 evaluate our framework against seven other knee-detection algorithms, using both ARC and LRU MRCs from 106 diverse real-world workloads, and apply it to identify key points in multi-tier MRCs. Compared to naïve approaches, our framework reduces 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.

***Index Terms*—multi-tier caching, miss ratio curve, knee detection**

I. 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 [7], [28]. MRCs can also inform offline evaluations such as comparing caching algorithms or analyzing monetary cost vs. storage-system performance [13].

There are many efficient techniques for generating MRCs [14], [18], [23], [25]. The MRC's reported miss ratios are good indicators of expected performance (*e.g.*, throughput), but real system performance can vary due to device characteristics, write policies, and admission policies [13]. 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 [27], [28]. Many storage-cache simulators have been developed that replay traces while attempting to faithfully reproduce real system behavior [1], [17], [29]. 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-tier would need 100^3, and so on. Thus, it is essential to 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 that will be evaluated at each level.

Intuitively, the most promising candidates are points where a little extra cache space produces a relatively large drop in the

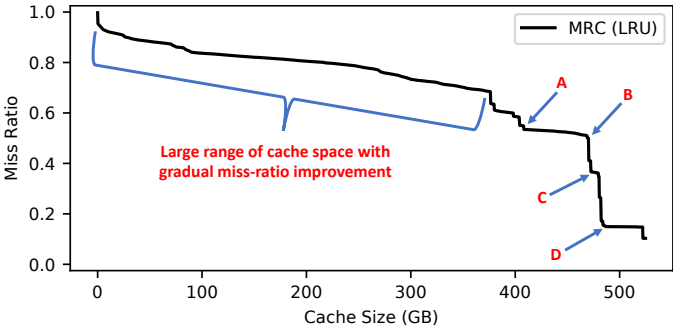

Fig. 1: MRC for trace w10, annotated to illustrate several key points: useful "knees" (points A, C, and D), a useless "cliff" (B), and a large range of cache sizes with relatively gradual miss-ratio improvement.

miss ratio; such points are often visible as "knees" in MRCs—*e.g.*, points A, C, and D in Figure 1. (Note that although B has sharp curvature, it is not of interest since C provides a much lower miss rate.) Given enough computational time, 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 paper 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 [27], [28]; **(2)** Next, we use the Ramer-Douglas-Peucker (RDP) line simplification algorithm [20] 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 [4], [21], [22], [26]; 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.

Our framework substantially reduces the number of simulated configurations needed to evaluate multi-tier caches without compromising accurate identification of the best configurations. This 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 release the code library containing all techniques used in this work;[1] and
5) 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.

[1]https://github.com/mariolpantunes/knee

II. Background

A. 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 [19] 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 [28]. Thus, non-stack-based MRCs need not be strictly decreasing.

B. Knee-Detection Algorithms

Many heuristic algorithms have been developed that find a single knee in a curve, although the definition of a knee can vary for any given algorithm. For example, one can define a knee as the point with the maximum curvature in a function [26]. However, a formal definition of a "good knee" is elusive, because whether a point is "good enough" to be classified as a knee depends on the application.

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 [22]. 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 this 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 [30]. 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 [3]–[5], [21], [26] 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 III) by enabling these algorithms to find multiple knee points.

C. 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 [6] 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 [28] technique for transforming sophisticated non-LRU replacement policies used in software caches. CliffHanger [12] applied a similar idea to key-value web caches, but instead estimated the MRC gradient without explicitly constructing one.

The recent eMRC [16] 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.

III. 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 III-A). Next, we made substantial modifications to each point-selection technique, enabling them to output a set of multiple knees instead of just one (Section III-B). Finally, we added a post-processing stage (Section III-C) that filters and ranks knees based on an appropriate definition.

A. 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 [28]; 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 [20]. 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.

B. Methods

Except for Kneedle, the algorithms we evaluate in this work (see Section II-B) 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 post- processing methods to work properly on MRCs.

C. 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 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 removes cliff points, located after a smooth, near-horizontal area that precedes a sharp descent. Point B in Figure 1 is a good example of a cliff point. Relevant knee points are generally located around the last point (*i.e.*, at the bottom) of the descent, such as points C and D, since those are the points most worth evaluating.

The third and final filter 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 (1):

$$S(K_i, \vec{L}) = |K_h - K_i| \cdot R^2(\vec{L}), \quad (1)$$

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.

IV. Z-Method

Our design for Z-Method was inspired by the DFDT [4] and DSDT [5] 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 example, 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 III-A and III-C. As such, it works for curves that are non-monotonic, with both convex and concave regions (see Section II-A).

As shown in Algorithm 1, Z-Method takes as input a discrete curve *D* consisting 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 7–22).

We first convert dx and dy, specified as percentages, into absolute values Δx and Δy 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 [10]; 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 ≥ 3 is a widely accepted value for outliers [2] (lines 5–6).

We then enter the main selection loop (lines 7–22), 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 Δx and Δy 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 11–14) 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 Δx apart, based on the dx parameter constraint (lines 15–16). This takes $O(|C|)$ time, effectively forming groups of points such that there is at least Δx distance between every group. We then sort the groups in G in descending order by the maximum z-score of each group (line 17). 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 19); we then check that the selected point is not within a minimum Δy distance from other points that have already been selected, enforcing the dy parameter (line 20). 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 21). We then decrement the $zLimit$ by dz and continue with the next loop iteration (line 22).

This loop terminates only after we have reached a $zLimit \leq 0$ and there are no remaining points that can be selected

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

1 $\Delta x \leftarrow \text{length}(D) \cdot (dx/100)$
2 $\Delta y \leftarrow (\max(y) - \min(y)) \cdot (dy$ / 100)
3 $D'' \leftarrow$ calculate second derivative of D
4 $Z \leftarrow$ calculate z-scores for D''
5 $K \leftarrow$ empty list
6 $zLimit \leftarrow 3$ # standard outlier threshold [2]
7 **while** $TRUE$ **do**
8 $C \leftarrow$ points in Z with z-score $\geq zLimit$
9 **and** at least Δx and Δy apart from all points in K
10 $Z \leftarrow Z$ - C
11 **if** $zLimit \leq 0$ ***and*** $\text{length}(C) == 0$ **then**
12 Remove points from K to ensure that
13 y always decreases as x increases
14 **return** K
15 $G \leftarrow$ group all points in C such that
16 all adjacent points in each group are $< \Delta x$ apart
17 sort G in descending order by max z-score of each group
18 **foreach** *group in* G **do**
19 $p \leftarrow$ point in $group$ with the lowest y value
20 **if** p is at least Δy from all points in K **then**
21 $K.append(p)$
22 $zLimit \leftarrow zLimit - dz$

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 ≥ 3 and iteratively approaching 0, we select the largest knees first and gradually lower our threshold for how big a knee should be.

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 (lines 12–13); 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.

V. Evaluation

We first compared the accuracy of 8 different knee-detection algorithms, including Z-Method, then evaluated our framework's ability to quickly find optimal multi-tier configurations.

A. Experimental Setup

We evaluated our techniques on 106 real-world block traces collected by CloudPhysics [27], each representing a week of virtual disk activity from production VMware environments. We used hash-based spatial sampling [27], [28], with a sized-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 [16].

To evaluate multi-tier systems, we extended PyMimircache [29], 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 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 [16]. 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.

B. 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 with peak detection methods, and can be used for single-knee detection by selecting only the highest possible peak; we call that 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.

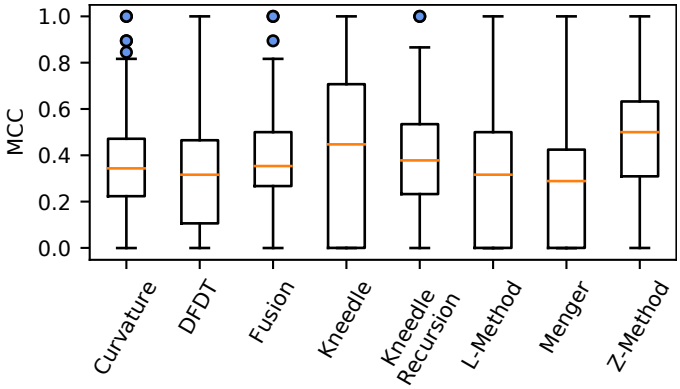

Fig. 2: An MCC evaluation of 8 knee detection algorithms using our optimized hyper-parameters for accurately identifying knees that were manually selected by experts. Higher MCC values and lower standard deviations are better. Kneedle and Z-Method have the highest median MCC values of 0.45 and 0.5, respectively, with Z-Method achieving much tighter bounds.

Most of our techniques have one or more hyper-parameters that can influence which points are selected. While the default parameters offered acceptable performance, a more complete evaluation requires optimized parameters [9]. Therefore, we developed a cost function and ran an optimization algorithm for each knee-detection method using all 212 MRCs.

Although the knee-detection problem is better modeled as a regression, we based our cost function 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). The Matthews correlation coefficient (MCC) [11] measures classification quality by considering the balance ratios of the four confusion matrix categories: true positives and negatives, and false positives and negatives.

Figure 2 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. 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.

C. Multi-Tier MRCs

Miss-ratio curves are typically used to find configurations that minimize both miss ratio and cache size(s). We seek mul-

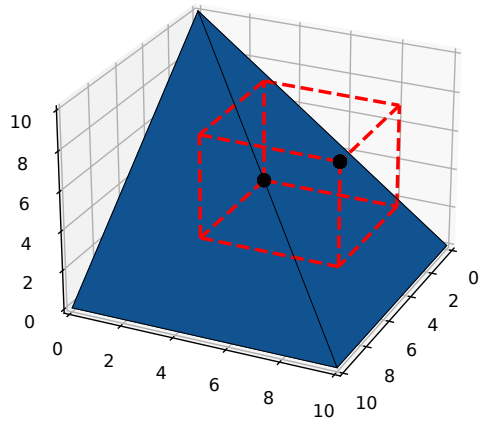

Fig. 3: An example of how the *HyperVolume Indicator (HVI)* is calculated for a single point of a 3-dimensional data series. A cube is drawn from the reference point (10,10,10) to the data point (5,5,5) creating a $5 \times 5 \times 5$ cube with a hypervolume of 125.

tiple 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 [15]; the most commonly used is the *HyperVolume Indicator (HVI)* [8], 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 3 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 2-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*[2] 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 $6 \times 6 \times 6$ cube with a hypervolume of 216. Thus, configurations with lower cache size and miss ratio result in a larger hypervolume. 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.

[2]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.

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 I: 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).

Another metric highly relevant to our problem is the Ratio of Non-Dominated Individuals (RNI) [24], 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 13.

In Table I, 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 2-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

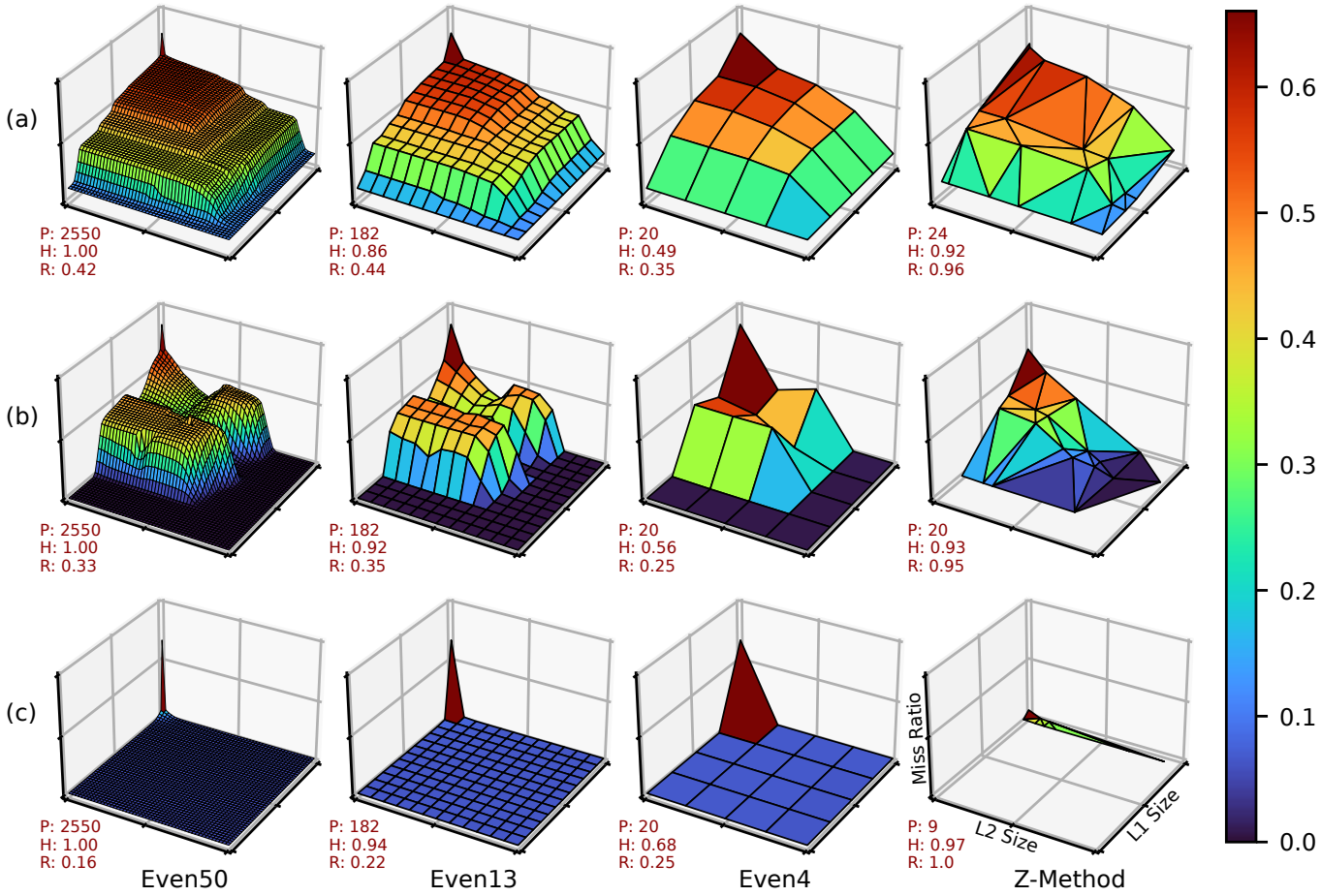

Fig. 4: Examples of point selection on 2-tier MRCs that highlight three different commonly observed scenarios. Each point represents the total miss ratio of a configuration of some L1 and L2 sizes. The x and y axes are the normalized L1 and L2 sizes, respectively, while the z-axis is the miss ratio. 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).

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 4, we show visualizations of the points selected by each method for a few selected 2-tier MRCs with fairly different characteristics. Figure 4a (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 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 4b (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 4c (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.

VI. 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.

Acknowledgment

We thank the anonymous reviewers for their constructive feedback. This work was made possible in part thanks to Dell-EMC, NetApp, Facebook, and IBM support; a SUNY/IBM Alliance award; and NSF awards CCF-1918225, CNS-1750109, CNS-1900706, CNS-1951880, CNS-2106263, CNS-2106434, and CNS-2214980. This work is also partially funded by FCT/MCTES through national funds and when applicable co-funded EU funds under the project UIDB/50008/2020-UIDP/50008/2020.

References

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

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

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

[21] 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, 2004.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

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

[27] 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.

[28] 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.

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

[30] 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.