Reuse-based Analytical Models for Caches

Rathijit Sen David A. Wood
Department of Computer Sciences
University of Wisconsin-Madison
{rathijit,david}@cs.wisc.edu

ABSTRACT

We develop a reuse distance/stack distance based analytical modeling framework for efficient, online prediction of cache performance for a range of cache configurations and replacement policies LRU, PLRU, RANDOM, NMRU. Such a predictive framework can be extremely useful in selecting the optimal parameters in a dynamic reconfiguration environment that performs power-shifting or resource reallocation through cache partitioning.

Our framework unifies existing cache miss-rate prediction techniques such as Smith's associativity model, Poisson variants, and hardware way-counter based schemes. We also show how to adapt way-counters to work when the number of sets in the cache changes.

We propose a novel low-overhead hardware mechanism to estimate reuse distance/stack distance distributions using a combination of set-sampling and time-sampling. This can be used even in cases where using way-counters is not possible, e.g. RANDOM/NMRU replacement policies.

Categories and Subject Descriptors

C.0 [**General**]: Modeling of computer architecture ; C.4 [**Performance of Systems**]: Modeling techniques

General Terms

Performance

Keywords

Cache, Stack Distance, Reuse Distance, Replacement policies, LRU, PLRU, RANDOM

1. INTRODUCTION

Processor caches are critical components of the memory hierarchy since they significantly boost performance by exploiting locality to keep frequently-accessed data on-chip. Large caches reduce miss-rates but are costly and burn power. Cache performance is workload dependent and phase dependent even within the same workload. A number of studies have considered power-efficient performance by dynamically placing caches in low-power mode [14, 3, 12] or improved performance by better resource allocation through dynamic partitioning [40, 31]. In this work we study the problem of developing efficient online techniques for predicting cache miss-rates for possible target configurations. We are particularly interested in last-level caches which, being large in size, offer a lot of good/missed opportunities for reconfiguration.

Our analysis is motivated by two foundational works: Mattson's stack distance characterization [25] (also used later as reuse distance [11, 6]) and Smith's associativity model [35, 16] for LRU caches. We develop an alternate workload characterization (Section 3) that conceptualizes traces as functions and allows succinct definitions for two locality metrics: unique reuse distance (numerically same as stack distance) and absolute reuse distance. We then develop a general framework that has two components: a cache-hit function (Section 4) and average per-set locality metrics (Section 5). Using cache-hit functions we demonstrate how our framework can be used to predict cache miss-rates for four popular replacement policies: LRU, RANDOM, NMRU, PLRU.

A hardware technique for predicting cache miss-rates for changed associativity uses way-counters [39]. We show that their computational efficiency is due to the number of sets being fixed (Section 6.2). However, this is also their limitation since they can predict only for changed associativities not changed number of sets. Moreover they work only for LRU,PLRU [17] and are not applicable to other important policies like RANDOM,NMRU that can also be predicted well using reuse information (Sections 7, 8).

Allowing for changes in the number of cache sets requires re-computing per-set locality metrics. The reuse distance/stack distance characterization of a trace is lossy (Section 3). Since, in general, the original trace cannot be retrieved from the distribution, we have no choice other than to convert the existing distribution directly to one for a cache with a different number of sets. This holds even if an offline analysis is considered. However, a binomial transformation can be used to achieve the conversion provided a sufficient amount of the tail of the distribution is available (Section 6.3).

The reuse distance/stack distance distribution is very long tailed and truncating it can cause significant errors (Section 5). This poses problems in online estimation, as for example, with way counters (Section 5). We overcome this problem in our proposed hardware-based estimation technique (Section 10) using set-sampling [15, 29, 18, 42] to reduce the

size \ A	2	4	8	16	32
2MB	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}
4MB	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}
8MB	2^{16}	2^{15}	2^{14}	2^{13}	2^{12}
16MB	2^{17}	2^{16}	2^{15}	2^{14}	2^{13}

Table 1: Relation between number of sets (S) and associativity (A) for different cache configurations. Way-counters can predict for *at most* 3 of 19 possible target configurations at any time.

maximum distance that needs to be tracked.

The major contributions of our work are:

1. We describe a unified framework for analytically predicting cache-miss rates using reuse distance/stack distance distributions for 4 different replacement policies. At a high level, our framework consists of the following steps:

   (a) Measure or estimate unique reuse distance/stack distance distribution from the access stream (Section 10).

   (b) Compute per-set reuse distance distribution (Section 5).

   (c) If not LRU, compute absolute distance distribution (Section 3).

   (d) Compute cache hit-functions and miss-rates for the particular replacement policy (Sections 4,6,7,8,9).

2. We show that Smith's associativity model [35, 16] for LRU caches is a special instance of our framework (Section 6).

3. We show that Cypher's Poisson model [9, 10] for LRU caches is a special instance of our framework and further extend it to piecewise-linear approximations (Section 6.1).

4. We show that the traditional hardware way-counter based prediction [39] for varying associativity is a special instance of our unified framework (Section 6.2).

5. We show how way-counter data may be transformed to apply to caches with a different number of sets. (Section 6.3)

6. We propose a novel hardware scheme for efficient online estimation of reuse distance/stack distance distributions that can be used even in cases where way-counters are not applicable, such as changes in the number of sets (Section 10).

We evaluate our analyses using address traces for commercial workloads [2] obtained from a full-system simulator built using GEMS [24]. Our simulated system has 8 cores and a 16MB 32-way last-level cache. Address traces as seen by the last-level cache are collected. We predict miss-rates for a range of cache organizations spanning sizes 2MB-16MB and 2-way to 32-way set-associativity. Table 1 shows the studied configurations. For evaluating our miss-rate predictions we run the traces through a standalone cache simulator (that does not model timing) and compare measured against predicted metrics.

We assume uniformity in distribution of accesses to cache sets. While the conventional index mapping through bit selection may not result in uniformity, simple XOR-based hashed functions such as those in modern processor caches [21] can enhance this. The uniformity assumption allows application of the Binomial model for our analysis and use of uniform set-sampling techniques.

2. RELATED WORK

Characterizing cache behavior using stack distances (also known as reuse distances or gap) is well known [11, 6, 35, 16, 9, 10, 23, 34, 44]. Analyses can be viewed as being in one of 3 categories: offline, online, and mixed.

In offline algorithms, stack distances are computed offline from an available trace. Computation time required to determine the distribution can be reduced through efficient algorithms [4] or by approximate analysis [44]. Shi, et al. [34] perform single-pass stack simulation to project CMP cache performance and to study the impact of data replication for various L2 cache configurations. Online determination of the stack distance distribution cannot directly reuse techniques from offline methods due to tight constraints on computational state and complexity.

Mixed algorithms deal with efficient online trace collection and offline processing. Tam, et al. [41] use hardware mechanisms for address sampling and post-processing software for computing stack distance distributions. Since distribution estimation and hit-rate computation is offline, it cannot react to workload changes in real time.

Online algorithms deal with all the steps of trace collection, distribution generation and hit/miss-rate computation online. These steps may be combined together to compute the hit/miss-rate. Agarwal, et. al [1] propose an analytical cache model that uses a binomial model and metrics for time-average of total unique number of accesses but predicting miss-rates for set-associative caches is difficult.

Suh, et al. [40], Qureshi, et al. [31] propose mechanisms for partitioning of shared caches (L2) among competing processes using way counters. LRU caches maintain the stack property per-set and can be viewed as having S stacks, each of depth A [25, 26]. Way-counters maintain a counter per *logical stack position*, not per physical way. For every access that hits in some set with physical way number a, the logical position l corresponding to a is determined. This essentially involves determining the position in a sorted list of length A ordered by access time. Suh, et al. [39] show how to determine the sorted position in hardware using multiplexers. This paper also proposes set counters in LRU order, with each set tracking accesses to a group of sets. We discuss way counter and set counter limitations in Sections 6.2 and 5 respectively.

Cypher proposes methods [9, 10] for online estimation of stack distances using hash tables. In that work, the effective distance to be tracked is reduced using filter fraction metrics which are then applied to a Poisson prediction model. However, computing filter fractions themselves are difficult, require additional logic and could be subject to approximations depending on available hardware state.

We propose an additional online scheme that can be used for efficient prediction of cache performance. Throughout the paper we compare and contrast this new approach with that using way counters. In contrast to Cypher's approach, the new scheme uses sampling of (possibly non-contiguous) cache sets to reduce the effective distance, does not require any complex filter logic, and uses Bloom filters [7] for compact representation of sets.

The Binomial model has been successfully used to analyze cache behavior for other applications. Stone, et al. [37], Falsafi, et al. [13] use binomial probability models to model cache reload transients due to context switches based on the footprints of the competing programs and cache size. Other models, e.g. Markov models have also been used to analyze the behavior of context switch misses [22].

A different classification of analyses can be based on the nature of metrics studied. Reineke, et al. [32] prove relations on best and worst-case bounds of cache performance for a number of replacement policies. Our work, in contrast, studies average case behavior.

3. MEASURES OF LOCALITY

A central contribution of this work is to conceptualize a trace as a function, rather than as a sequence of addresses. This allows succinct representations for other concepts described here . Let $\mathbb{B}$ denote the domain of addresses. A trace is a bijective function $T : \mathbb{N} \mapsto \mathbb{B} \times \mathbb{N} \cup \{0\}$ with the following restrictions:

1. $T(i) = (x, m), T(j) = (x, m'), i < j \implies m < m'$.
2. $T(i) = (x, m), m > 0 \implies \exists j < i, T(j) = (x, m-1)$.

Intuitively, it can be thought of as mapping natural numbers to tuples (x, m) where x identifies the addresses and m identifies its repetition number. T can be constructed from a sequence of addresses by mapping positions in the sequence to *consecutive* natural numbers and addresses to tuples. The first occurrence of address x in the sequence is represented by $(x, 0)$.

Let $t = T^{-1}$ denote the inverse function. $t(x, m)$ denotes the position of the m^{th} occurrence of address x in the trace. We now introduce a few more definitions.

Reuse Interval: The **reuse interval** (RI) is defined only when $m > 0$ and denotes the portion of the trace enclosed between the m^{th} and $(m-1)^{th}$ occurrence of x. Formally, $RI(x, m) =$

$$\begin{cases} \{(z, m') | t(x, m-1) < t(z, m') < t(x, m)\} & \text{if } m > 0 \\ \text{undefined} & \text{otherwise} \end{cases}$$

Unique Reuse Distance: This denotes the total number of addresses between two occurrences of the same address in the trace. Thus,

$$URD(x, m) = \begin{cases} \left|\{z | (z, m') \in RI(x, m)\}\right| & \text{if } m > 0 \\ \infty & \text{otherwise} \end{cases}$$

Numerically, this is 1 less than Mattson's much earlier stack distance [25]. But we prefer to use this definition as it is not constrained by the notion of stack.

Absolute Reuse Distance: This denotes the total number of positions between two occurrences of the same address in the trace. Thus, $ARD(x, m) =$

$$\begin{cases} \left|RI(x, m)\right| = t(x, m) - t(x, m-1) - 1 & \text{if } m > 0 \\ \infty & \text{otherwise} \end{cases}$$

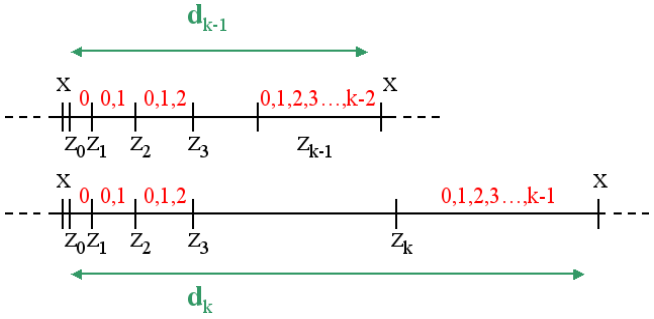


Figure 1: Schematic diagram showing relation between $\boldsymbol{d}_k(t)$ and $\boldsymbol{d}_{k-1}(T)$. Possible URDs for accesses in each sub-interval are shown in red.

3.1 Reuse Distance Distributions

Our study is concerned with average-case behavior. So instead of focusing on each individual point in T, we characterize it using probability vectors that reflect average/expected distributions.

The **unique reuse distance distribution** of trace T is a probability distribution that we denote by row vector $\boldsymbol{r}(T)$ such that the k^{th} component,

$\boldsymbol{r}_k(T) = P(URD(x, m) = k), \forall (x, m) \in range(T)$

The **expected absolute distance distribution** of trace T is a row vector that we denote by $\boldsymbol{d}(T)$ such that the k^{th} component,

$\boldsymbol{d}_k(T) = E(ARD(x, m) | URD(x, m) = k), \forall (x, m) \in range(T)$

In later analysis it will be useful to know the average absolute reuse distance if the unique reuse distance is known.

It is obvious that $ARD(x, m) \geq URD(x, m), \forall x, m$. It then follows that $\boldsymbol{d}_k(T) \geq k, \forall k$ such that $\boldsymbol{r}_k(T) > 0$. Also, $\boldsymbol{d}_0(T) = 0$. We now show how $\boldsymbol{d}(T)$ can be approximately computed given $\boldsymbol{r}(T)$.

Figure 1 shows a schematic of a trace and organization of URDs within a reuse interval for some address x. z_0, z_1,...z_k denote distinct addresses. We want to emphasize that this is just a conceptual tool to compute the estimated distance and does not constrain the actual permutation of addresses in a particular reuse interval. The immediate next access after reference address x must be something other than x (otherwise the reuse interval would immediately terminate with $k = 0$). Between this first address z_0 and the next different address z_1, the only possible URDs of accesses must be 0. Between z_1 and z_2, the only possible URDs can be 0 and 1. Extending this reasoning till z_{k-1} and z_k we realize that $\boldsymbol{d}_k(T)$ and $\boldsymbol{d}_{k-1}(T)$ differ only in the last sub-sequence which consists of a run of accesses with URDs in $\{0, 1, .., k-1\}$. The length of this run can be estimated as the expected number of trials to success in a geometric distribution with success probability $\sum_{i=k}^{\infty} \boldsymbol{r}_i(T)$.

We thus arrive at the following recurrence:

$$\begin{aligned} \boldsymbol{d}_0(T) &= 0 \\ \boldsymbol{d}_k(T) &= \boldsymbol{d}_{k-1}(T) + \frac{1}{\sum_{i=k}^{\infty} \boldsymbol{r}_i(T)} \end{aligned} \quad (1)$$

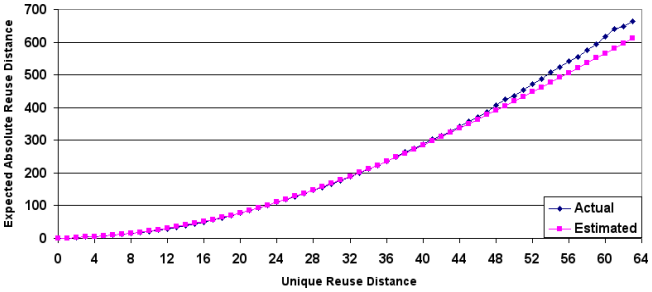


Figure 2: Actual vs estimated $\boldsymbol{d}(T)$ per set for oltp with a 4MB 32-way cache. $\boldsymbol{d}(T)$ depends on S but is independent of A.

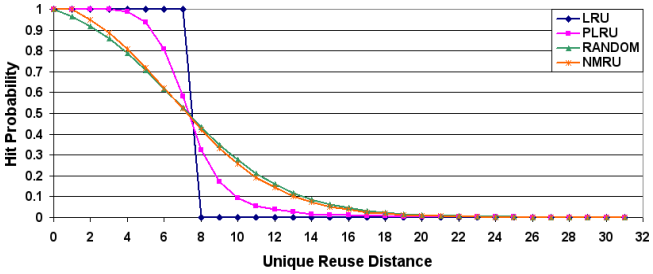


Figure 3: Representative hit-rate functions for an 8-way cache with different replacement policies but with the same trace and number of sets.

In the context of cache performance, the vectors $\boldsymbol{r}$ and $\boldsymbol{d}$ capture information about temporal locality. They are workload characteristics and independent of the cache configuration or other properties of the hardware platform on which the workloads are run.

Figure 2 shows actual vs estimated values for $\boldsymbol{d}(T)$ using recurrence 1 with an estimate of $\boldsymbol{r}(T)$ on a per-set basis (see Section 5).

The characterization $\boldsymbol{r}(T)$ of T is lossy in the sense that in general, T cannot be recovered from $\boldsymbol{r}(T)$ even upto permutation of entity identifiers. The proof is simple: consider two traces T_A and T_B such that they have disjoint sets of entities and different values of reuse metrics. Let T_{AB} denote a new trace formed from concatenating, in order, sequences represented by T_A and T_B. This operation is not commutative, that is, T_{AB} and T_{BA} are distinct, yet have the same values for the reuse metrics. So the reverse mapping from $\boldsymbol{r}(T)$ to T is not unique. The argument can be extended to show that any trace characterization using position-agnostic metrics must be lossy.

4. CACHE HIT FUNCTIONS

Traditionally, cache studies focus on the miss-rate because of its huge impact on performance. Although our analyses compute the hit-rate h, it can be trivially converted to the miss-rate θ using the relation $\theta = 1 - h$.

In our study, caches are characterized by the number of sets S, associativity A, and replacement policy. We assume a fixed line size of 64 bytes. Table 1 shows the relation between S, A and cache size for the configurations we study. Given a cache organization $(S, A, policy)$ and a trace T, our goal is to determine a vector $\boldsymbol{\phi}(\boldsymbol{r}(T), S, A, policy)$ such that the expected hit-rate for the trace, $h = \boldsymbol{r} \cdot \boldsymbol{\phi}$. The idea is to characterize workload traces by $\boldsymbol{r}$ and caches by $\boldsymbol{\phi}$ so that the effect on hit-rate for changes in traces or cache configurations can be readily estimated.

We call $\boldsymbol{\phi}$ the cache hit function. The value of the k^{th} component, $\boldsymbol{\phi}_k$ is the conditional probability of a hit for accesses x such that $URD(x, m) = k$ where m is the repetition count for x at that point in the trace when the access happens. $\boldsymbol{\phi}_k$ monotonically decreases with k. This is because for the reference address not to have been evicted after accesses involving k unique elements, it must not have been evicted after accesses involving $k - 1$ unique elements *and* anything following that. If there are no intervening accesses $(k = 0)$, the access must be a hit. Accesses hitherto never seen $(k = \infty)$ must miss. So,

$$\boldsymbol{\phi}_k = \begin{cases} 1 & \text{if } k = 0 \\ \leq \boldsymbol{\phi}_{k-1} & \text{if } k \geq 1 \\ 0 & \text{at } k = \infty \end{cases} \quad (2)$$

For a set-associative LRU cache with associativity A, it is well known that all accesses with addresses re-appearing with less than A unique intervening elements must hit and all other accesses must miss. This leads us to the following characterization of the LRU hit-rate function.

$$\boldsymbol{\phi}_k(\boldsymbol{r}(T), S, A, LRU) = \begin{cases} 1 & \text{if } 0 \leq k < A \\ 0 & \text{if } k \geq A \end{cases} \quad (3)$$

Cache hit functions for other replacement policies are not so straight-forward to determine. LRU is typically not implemented in real caches for $A > 4$ due to hardware complexity. Figure 3 shows representative characteristic functions for commonly implemented replacement policies.

Two points may be noted here: first, that $\boldsymbol{\phi}$ may not be independent of $\boldsymbol{r}$ for different replacement policies. Caches that employ LRU replacement policies have $\boldsymbol{\phi}$ that are independent of $\boldsymbol{r}$. As we shall show later, caches employing RANDOM replacement policy depend on $\boldsymbol{d}$, while caches employing PLRU policy may need even more information. The goal then is to identify aspects of $\boldsymbol{\phi}$ that are trace-dependent so that the particular metrics can be computed and plugged into the equations such that hit-rate predictions can be readily made.

The second point is that while replacement policies for set-associative caches are based on per-set behavior, the earlier definitions of locality metrics do not make this explicit to prevent limiting their scope of applicability. Changing the number of sets in the cache will change the access trace each cache set will observe and will hence change the values of locality metrics. So we extend these definitions to include the number of sets S. $\boldsymbol{r}(T, S)$ denotes the unique reuse distribution of the sub-sequence of T that a single set in the cache observes *on average*. $\boldsymbol{d}(T)$ is adapted to $\boldsymbol{d}(T, S)$ similarly.

In view of these considerations we write

$$h(T, S, A, policy) = \boldsymbol{r}(T, S) \cdot \boldsymbol{\phi}(\boldsymbol{r}(T), S, A, policy) \quad (4)$$

Henceforth, for brevity of notation, we will omit specifying one or more parameters when their values are clear from the context.

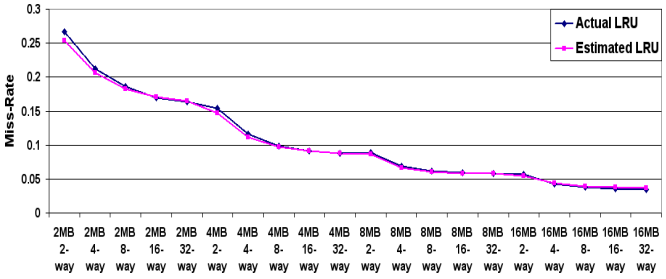


Figure 4: Actual vs estimated hit-rate function for oltp with LRU replacement policy

5. ESTIMATING PER-SET REUSE METRICS

When reconfiguring caches, it is sometimes better to change the number of sets (S) and leave associativity fixed. But since cache replacements work on a per-set basis, changing S changes per-set locality. As discussed in Section 4, this was the motivation for extending $\boldsymbol{r}(T)$ and $\boldsymbol{d}(T)$ to $\boldsymbol{r}(T,S)$ and $\boldsymbol{d}(T,S)$. The number of sets, S, and reuse interval, RI, are negatively correlated. Decreasing S increases RI since accesses that hitherto mapped to other sets now start getting mapped to the reference set. Similarly, increasing S decreases RI. Since set counters [39] can only correct for changes in the number of accesses per set *but not changes in the locality per set*, they are not expected to be useful for workloads with large working sets.

Since *the target cache configuration for which we want to estimate the hit-rate may have a different number of sets than the one on which the reuse metrics were measured*, the goal is to be able to estimate $\boldsymbol{r}(S')$ given $\boldsymbol{r}(S)$ for $S' \neq S$. $\boldsymbol{d}(S')$ can be estimated from $\boldsymbol{r}(S')$ for any S' (see Equation 1). Estimating the metrics for all accesses in the trace, that is, without any filtering on the trace is analogous to the case when $S = 1$. We will start with this case first in the following analysis.

Similar to Smith's model, we make the simplifying assumption that the mapping of unique lines to cache sets are independent of each other [35, 16]. Thus, accesses to a given set can be modeled as successive Bernoulli trials with the success of each trial having probability $\frac{1}{S'}$. Viewing this through the lens of matrix multiplication, we recognize that the required transformer is a generalized stochastic Binomial Matrix [38], $\boldsymbol{B}(1-\frac{1}{S'},\frac{1}{S'})$. Thus,

$$\boldsymbol{r}(S') = \boldsymbol{r}(1)\cdot\boldsymbol{B}(1-\frac{1}{S'},\frac{1}{S'}) \tag{5}$$

It is trivial to show that the above transformation respects $\sum_{i=0}^{\infty} r_i(S') = \sum_{i=0}^{\infty} r_i(1) = 1$. Qualitatively, this transformation results in a re-distribution of mass with $\boldsymbol{r}(S')$ getting compressed as S' is increased and dilated as S' is decreased.

Figure 4 shows actual vs estimated values for LRU with the estimates computed using equations 5, 3 and 4.

So far, we have tacitly avoided specifying the dimensions of the matrices involved. The size is determined by the maximum unique reuse distance, n, we are required to maintain. For error rate ϵ, we need the minimum n such that

$$\sum_{i=0}^{n} \boldsymbol{r}_i(1) + \boldsymbol{r}_\infty(1) \geq 1-\epsilon \tag{6}$$

Good predictions for the commercial workloads we study require n to be large, of the order of the number of lines in the target cache. To put this in perspective, a 16MB cache with 64B line size has 256K lines. *The $\boldsymbol{r}(1)$ distribution can be very long-tailed.* Ignoring/truncating the tail usually result in significant errors. The reason is simple: workloads with large working sets must have accesses with large reuse intervals and ignoring accesses with large URD means counting them as misses when large caches can accommodate them resulting in hits. Thus, estimation errors due to tail truncation are pronounced for large caches. In Section 10 we will discuss how the problem may be addressed using set-sampling.

We will now show how to compute $\boldsymbol{r}(S')$ from any starting cache configuration S. This has no implication for the new hardware mechanism we propose (Section 10) that is decoupled from the current cache configuration. However it will be useful in reasoning about way-counters (Section 6.2). It also shows how computations can be reused instead of always needing to start from the ground configuration ($S = 1$).

Binomial Matrices are invertible (when the second parameter is non-zero) and are closed under multiplication within the same dimension [38]. We use these properties and associated formulae in the following analysis.

$$\begin{aligned}
\boldsymbol{r}(S') &= \boldsymbol{r}(1)\cdot\boldsymbol{B}(1-\frac{1}{S'},\frac{1}{S'}) \\
&= \boldsymbol{r}(S)\cdot(\boldsymbol{B}(1-\frac{1}{S},\frac{1}{S}))^{-1}\cdot\boldsymbol{B}(1-\frac{1}{S'},\frac{1}{S'}) \\
&= \boldsymbol{r}(S)\cdot\boldsymbol{B}(1-S,S)\cdot\boldsymbol{B}(1-\frac{1}{S'},\frac{1}{S'}) \\
&= \boldsymbol{r}(S)\cdot\boldsymbol{B}(1-\frac{S}{S'},\frac{S}{S'})
\end{aligned} \tag{7}$$

Equation 7 is a general form of equation 5. The transformer depends only on the ratio of the number of the sets in the current cache to the target cache. There are two cases to consider depending on the value of this ratio:

Case 1, $\mathbf{S' \geq S}$: The transformation is always safe in the sense that the computed probabilities are valid ($\in [0,1]$) *even if $\boldsymbol{r}(S)$ has not been computed binomially.* This is just (re-)applying the Bernoulli model with probability of success of each trial $0 \leq \frac{S}{S'} \leq 1$.

Case 2, $\mathbf{S' < S}$: The transformation is valid if $\boldsymbol{r}(S)$ has been computed binomially *and* the vector dimensions have not changed. For practical reasons it may be necessary to maintain $\boldsymbol{r}(S)$ only upto $n_s < n$. n_s can be computed analogously to equation 6 using $\boldsymbol{r}(S)$ instead of $\boldsymbol{r}(1)$. The derivation of Equation 7 for $S' < S$ is basically equivalent to solving a system of equations with n unknowns but using n_s, the number of equations available is $n_s < n$. In this case the computed results are often not valid ($\notin [0,1]$). However, once $\boldsymbol{r}(S)$ has been computed (and truncated to n_s), we can always convert to $S'' \geq S$ (Case 1) and then to any S' with $S'' \geq S' \geq S$ (Case 2).

6. OPTIMIZATIONS FOR LRU

A naive combination of equations 4, 3 and 5 results in $\sum_{i=0}^{A'-1}(n-i) = nA' - \frac{A'(A'-1)}{2}$ multiplications with binomial computations to estimate the hit-rate for a cache with S' sets and associativity A'. The number of multiplications can be reduced by recognizing that due to the step-function

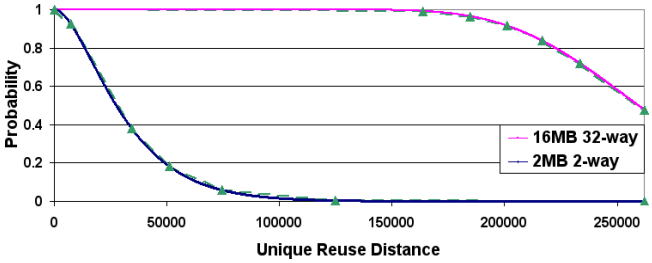


Figure 5: cumulative Poisson and piece-wise linear approximations for two cache configurations.

nature of $\boldsymbol{\phi}(LRU)$, some of the coefficients will sum to 1. Symbolically expanding the computation and simplifying, we get

$$h(S') = \sum_{i=0}^{A'-1} \boldsymbol{r}_i(1) + \sum_{i=A'}^{n} \boldsymbol{r}_i(1) \cdot \sum_{k=0}^{A'-1} {}^{i}C_k \cdot \left(\frac{1}{S'}\right)^k \cdot \left(1 - \frac{1}{S'}\right)^{(i-k)} \tag{8}$$

where ${}^{i}C_k$ denotes the k^{th} binomial coefficient. Equation 8 is an optimized version of Smith's associativity model [35, 16].

The above requires $nA' - A'(A'-1)$ multiplications which is $\frac{A'(A'-1)}{2}$ less than earlier. But computing binomial terms is costly and the above optimization hardly makes a dent when n is large compared with A'. Two approaches can be used to further reduce computational costs. The following subsections discuss each of these approaches.

6.1 Approximations to Binomial

Cypher [9, 10] uses a Poisson approximation to binomial for reducing computational costs.

The Poisson distribution is a discrete probability distribution that expresses the probability of a number of events occurring in a fixed period of time if these events occur with a known average rate and independently of the time since the last event [28]. The Poisson distribution is computationally more efficient and approximates the Binomial distribution when the number of trials is large and the probability of success of any particular event is small such that the product of these two terms is of intermediate magnitude. It is particularly useful in modeling small global miss-rates for large caches, such as the LLC.

The term $\sum_{k=0}^{A'-1} {}^{i}C_k \cdot \left(\frac{1}{S'}\right)^k \cdot \left(1 - \frac{1}{S'}\right)^{(i-k)}$ is the cumulative binomial sum up to $A'-1$ with parameters i and $\frac{1}{S'}$. When i is large and $\frac{1}{S'}$ is small, the binomial distribution can be approximated by a Poisson distribution with parameter $\lambda = \frac{i}{S'}$. Thus, we have

$$h(S') = \sum_{i=0}^{A'-1} \boldsymbol{r}_i(1) + \sum_{i=A'}^{n} \boldsymbol{r}_i(1) \cdot cumulative\_Poisson(A'-1, \lambda) \tag{9}$$

which is easier to compute than with the binomial coefficients.

Figure 5 shows the cumulative Poisson transformer for two very different cache organizations. Since the transformer function is slowly-varying, we can carry Cypher's idea forward and substitute the transformer with *piecewise linear functions* which are even simpler to compute. We use a simple algorithm to automatically select a small number of points on each curve such that the distribution can be approximated reasonably well with a linear interpolation between the chosen points. The triangles and the dashed lines show the selected points for piece-wise linear approximation. The approximation showed uses 7 or fewer points per transformer (with a common point (0,1)). The distribution is precomputed for each chosen point and stored. At run-time, these small number of constants are retrieved and used in the computation for the miss-rate. Values of intermediate points are approximated using linear interpolation. This method provides very good approximations with reduced computational cost but a moderate storage overhead.

6.2 Way Counters and Shadow Tags

As described earlier (Section 2), way counters work by incrementing the counter associated with the logical stack position (ordered by access times) on every cache hit. Since tags do not physically move about in the the cache, the sorted position must be determined for every access. With this arrangement, the number of hits for associativity A' is the sum of the counter values from 0 to A'.

Notice that the above assumes $A' \leq A$. In dynamic reconfiguration situations this is problematic since the cache may need to be sized up, not only sized down. To circumvent this difficulty, shadow tags [29] (or auxiliary tag directories [31]) can be used. Shadow tags maintain a copy of the tags and this copy is not deactivated during reconfigurations. This always maintains a stack depth to the maximum desired value and facilitates simulating the effect of hits and misses on a cache with associativity larger than that of the current configuration. Qureshi, et al. [30] used dynamic set sampling to reduce storage and power costs of the shadow copy.

Way-counter values, when converted to probabilities, estimate $\boldsymbol{r}(S)$ truncated to length A. The estimation is exact in case of LRU. Way counters solve the costly multiplication problem discussed earlier by not multiplying at all. To understand their operation, consider equation 8 but deriving it from equation 7 instead of from 5. We have

$$h(S') = \sum_{i=0}^{A'-1} \boldsymbol{r}_i(S) + \sum_{i=A'}^{n_s} \boldsymbol{r}_i(S) \cdot \sum_{k=0}^{A'-1} {}^{i}C_k \cdot \left(\frac{S}{S'}\right)^k \cdot \left(1 - \frac{S}{S'}\right)^{(i-k)} \tag{10}$$

Under the assumption $S' = S$ equation 10 simply reduces to

$$h(S') = \sum_{i=0}^{A'-1} \boldsymbol{r}_i(S) \tag{11}$$

which is computationally extremely efficient.

But there is no free lunch. The very relation ($S' = S$) that makes it so efficient is also the reason for its most fundamental limitation: the number of sets must be fixed. As can be observed from Table 1 only 3 of 19 configurations can be predicted at any time; other predictions must be preceded by (time-consuming) re-training for the changed S'.

The second limitation of way counters is that since they are tightly coupled with the implementation of replacement policies that track stack positions (e.g. LRU), they are unusable with other policies such as RANDOM that can also be predicted remarkably well using reuse information.

Way counters work depend on the replacement policy main-

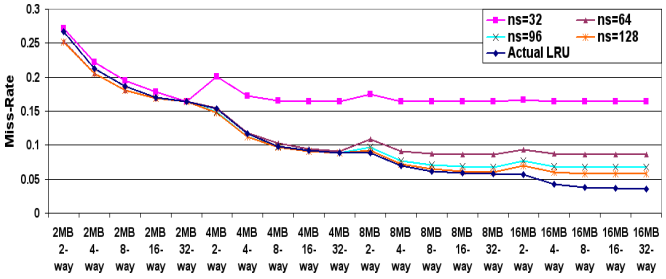


Figure 6: Transformation accuracy with information limited by n_s. Base state has $S = 2^{10}, A = 32$.

taining/mimicking stack operation, so they run into trouble when the stack is absent (PLRU,RANDOM,NMRU) or reconfigured ($S' \neq S$).

6.3 Transforming way-counters

As mentioned earlier, way-counters for LRU estimate $\boldsymbol{r}(S)$ (upto a certain limit) exactly. This allows us to use equation 10 to transform the values when $S' \neq S$. However, in accordance with the discussion in Section 5, Cases 1 and 2, we allow this conversion only when $S' \geq S$. With reference to Table 1, maintaining shadow tags corresponding to $S = 2^{10}$ allows conversion of values for any $S' \neq S$. Figure 6 shows the effect of this transformation for different values of n_s. This shows that to use way counter values for a cache with a larger number of sets, the shadow tags and counter values must be maintained for more than A positions. These extra counters provide information required since these stack positions will now map to positions $< A$.

7. PREDICTING FOR RANDOM

The RAND replacement algorithm [5] (also popularly called RANDOM) chooses a line (uniformly) randomly from the lines in the set for eviction on a miss.

For an A-way set-associative cache, the probability of replacement of a given line on a miss is $\frac{1}{A}$. Accounting for the number of misses in between successive reuses of an address is therefore needed. For expected miss rate θ, the expected number of misses for a sequence of α accesses is $\alpha \cdot \theta$. This is why $\boldsymbol{d}$ is important for RANDOM whereas LRU works independent of such information.

We make the simplifying assumption that miss occurrences (not specific addresses) are not correlated and hence amenable to be modeled as a Bernoulli process. While this may not be accurate, it allows us to make reasonably good predictions without tracking additional state.

Let $\boldsymbol{d}_k = \alpha$. The probability of i misses is estimated by ${}^{\alpha}C_i \cdot \theta^i \cdot (1-\theta)^{(\alpha-i)}$. The probability that a specific line is not replaced after i misses is $\left(1 - \frac{1}{A}\right)^i$. We thus have

$$\begin{aligned} h(RANDOM) &= \boldsymbol{r} \cdot \boldsymbol{\phi}(RANDOM) \\ \boldsymbol{\phi}_k(RANDOM) &= \sum_{i=0}^{\alpha|\boldsymbol{d}_k=\alpha} {}^{\alpha}C_i \cdot \theta^i \cdot (1-\theta)^{(\alpha-i)} \cdot \left(1 - \frac{1}{A}\right)^i \\ \theta &= 1 - h(RANDOM) \end{aligned} \quad (12)$$

To simplify the computation, we use the earlier trick of approximating Binomial(α,θ) by Poisson($\lambda = \alpha \cdot \theta$). Let $q = \left(1 - \frac{1}{A}\right)$. This gives

$$\begin{aligned} \boldsymbol{\phi}_k(RANDOM) &= \sum_{i=0}^{\alpha|\boldsymbol{d}_k=\alpha} {}^{\alpha}C_i \cdot \theta^i \cdot (1-\theta)^{(\alpha-i)} \cdot q^i \\ &= \sum_{i=0}^{\infty} {}^{\alpha}C_i \cdot \theta^i \cdot (1-\theta)^{(\alpha-i)} \cdot q^i \\ &\approx \sum_{i=0}^{\infty} e^{-\lambda} \cdot \frac{\lambda^i}{i!} \cdot q^i \\ &= e^{-\lambda(1-q)} \sum_{i=0}^{\infty} e^{-\lambda q} \cdot \frac{(\lambda q)^i}{i!} \\ &= e^{\frac{-\alpha\theta}{A}} \end{aligned} \quad (13)$$

The system of equations in 12 can now be approximated by the following system.

$$\begin{aligned} h(RANDOM) &= \boldsymbol{r} \cdot \boldsymbol{\phi}(RANDOM) \\ \boldsymbol{\phi}_k(RANDOM) &= e^{\frac{-\boldsymbol{d}_k\theta}{A}} \\ \theta &= 1 - h(RANDOM) \end{aligned} \quad (14)$$

$\boldsymbol{d}$ is estimated using equation 1.

7.1 Iterative solution and convergence

We solve the system of equations in 14 with the initial value $h = \boldsymbol{r}_0$. Usually 5 or fewer iterations suffice to reach reasonably close to a fix-point. But why do the iterations converge to a fix-point?

First note that if a fix-point exists, the solution satisfies the general conditions of $\boldsymbol{\phi}$ (equation 2). This is because $\boldsymbol{d}_0 = 0$ (equation 1) and from equation 14,

$$\begin{aligned} \frac{\boldsymbol{\phi}_k}{\boldsymbol{\phi}_{k-1}} &= e^{\frac{-(\boldsymbol{d}_k - \boldsymbol{d}_{k-1})\theta}{A}} = e^{-\left(\frac{\theta/A}{\sum_{i=k}^{\infty} \boldsymbol{r}_i(T)}\right)} \\ &\leq 1 \end{aligned} \quad (15)$$

Let H denote a fix-point and h^0, h^1, h^2, ... denote successive approximations. By re-arranging the system of equations in 14 we have

$$h^{j+1} = \boldsymbol{r}_0 + \sum_{i=0}^{n} \boldsymbol{r}_i e^{\frac{\boldsymbol{d}_i(-1+h^j)}{A}} \quad (16)$$

Since the exponential function is monotonic, H must be unique. Since $0 \leq \boldsymbol{r}_i \leq 1, \forall i$, $\boldsymbol{r}_0 \leq H \leq 1$.

Also, it is easy to show that $h^j \leq h^{j-1} \implies h^{j+1} \leq h^j$. Thus, successive iterations produce a chain of values $\boldsymbol{r}_0 = h^0 \leq h^1 \leq h^2$....

We will now prove that $h^j \leq H, \forall j$. This is certainly true at $j = 0$. For induction, let $h^j = H - \epsilon$ with $\epsilon \geq 0$. Then,

$$\begin{aligned} h^{j+1} &= \boldsymbol{r}_0 + \sum_{i=0}^{n} \boldsymbol{r}_i e^{\frac{\boldsymbol{d}_i(-1+h^j)}{A}} \\ &= \boldsymbol{r}_0 + \sum_{i=0}^{n} \boldsymbol{r}_i e^{\frac{\boldsymbol{d}_i(-1+H)}{A}} \cdot e^{-\frac{\boldsymbol{d}_i\epsilon}{A}} \\ &\leq \boldsymbol{r}_0 + \sum_{i=0}^{n} \boldsymbol{r}_i e^{\frac{\boldsymbol{d}_i(-1+H)}{A}} = H \end{aligned} \quad (17)$$

This shows a convergence chain $\boldsymbol{r}_0 = h^0 \leq h^1 \leq h^2 ... \leq H$.

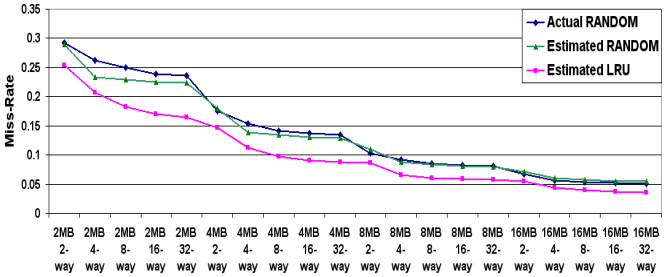


Figure 7: Actual vs estimated hit-rate function for oltp with RANDOM replacement policy. LRU estimates are shown as reference.

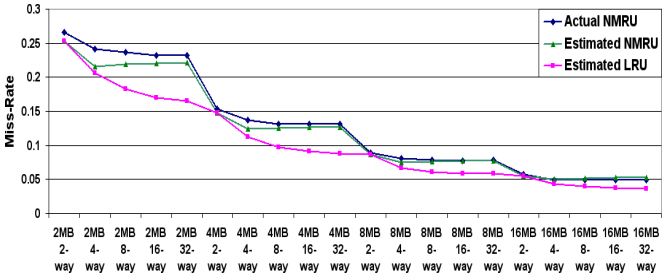


Figure 8: Actual vs estimated hit-rate function for oltp with NMRU replacement policy. LRU estimates are shown as reference.

7.2 Optimization

A better approximation for $A = 2$ can be obtained by using the fact that for the reference element not to be evicted at $URD \geq 2$, the previous element must be evicted (since the set can hold only 2 elements). The probability of the previous element to be evicted is $1 - \boldsymbol{\phi}_1$. For the reference element to hit at $URD = k$, it must hit at $URD = k - 1$ and the above condition must hold. This leads us to the following approximation.

$$\boldsymbol{\phi}_k \quad = \quad \boldsymbol{\phi}_{k-1} \cdot (1 - \boldsymbol{\phi}_1), \quad k \geq 2 \tag{18}$$

This approximation is possible at $A = 2$ since the model can exactly determine the set contents for $URD >= 2$. For higher associativities exact determination of set contents is difficult.

Figure 7 shows actual vs estimated values of hit-rates for RANDOM with the estimates computed using equations 5, 14, 18 and 4.

8. PREDICTING FOR NMRU

The NMRU (or non-MRU) replacement algorithm differentiates the most recently accessed (MRU) line from other lines in the set [36]. On a miss, a line is chosen (uniformly) randomly from among the $A - 1$ non-MRU lines.

At $A = 2$, $\boldsymbol{\phi}(NMRU) = \boldsymbol{\phi}(LRU)$. For the rest of the cases, the framework is similar to that of RANDOM except that accesses at $URD \leq 1$ are guaranteed to hit. Moreover, the replacement logic has $A - 1$ possible choices for an eviction in case of a miss. This leads to a few simple modifications to the system of equations in 14. The modified system is shown below:

$$\begin{aligned} \boldsymbol{\phi}_1(NMRU) &= 1 \\ h(NMRU) &= \boldsymbol{r} \cdot \boldsymbol{\phi}(NMRU) \\ \boldsymbol{\phi}_k(NMRU) &= e^{\frac{-(\boldsymbol{d}_k - \boldsymbol{d}_1)\theta}{A-1}} \\ \theta &= 1 - h(NMRU) \end{aligned} \tag{19}$$

Figure 8 shows actual vs estimated values of hit-rates for NMRU with the estimates computed using equations 5, 19 and 4.

9. PREDICTING FOR PLRU

Partitioned LRU [36] (also popularly called pseudo-LRU) maintains a balanced binary tree that, at each level of the tree, differentiates between the two sub-trees based on access recency. Every internal node is represented by a single bit whose value decides which of the two subtrees was accessed more recently. The cache lines are represented by the leaves of the tree. Whenever a line is accessed, the nodes on the path from the root to the leaf flip their bit values, thus pointing to the other subtree at each level. On a miss, the subtree pointed to is chosen, recursively starting from the root. The line corresponding to the leaf reached in this way is chosen for eviction. The bit-values along this path are then flipped.

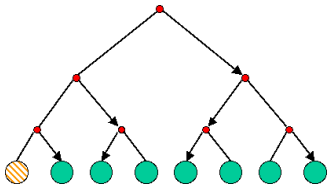


Figure 9: Schematic of an 8-way PLRU tree. The tree bits are in a plausible setting immediately after the reference element (striped) is accessed.

In the PLRU scheme, the most recently accessed element is always known but the least recently accessed one is not. In contrast to the LRU scheme, that maintains a total access order between the lines, PLRU maintains only a partial order. Since there is no difference between partial and total orders involving 2 elements, PLRU is LRU when $A = 2$. In contrast to LRU that guarantees exactly $A - 1$ unique accesses before eviction, PLRU guarantees at least $log_2(A)$ (=number of tree levels) unique accesses before the reference address is evicted.

On a miss, the reference line will be evicted if and only if the immediately preceding sequence of accesses follows a particular pattern. These patterns can be described using regular expressions (see Appendix A). In contrast to RANDOM, not only the number of misses in the reuse interval, but also the pattern of accesses determines eviction probability. In theory, it may be possible to estimate $\boldsymbol{\phi}(PLRU)$ by computing probabilities of the above regular expressions. But there are several difficulties: the distance to misses within the reuse interval is unknown and the ways occupied by the intervening elements are also not known. Instead, we use a different approach.

First, we compute $\boldsymbol{\phi}(A = 4, PLRU)$ then compute $\boldsymbol{\phi}(A = 8, PLRU)$ by dividing traffic using a binomial distribution and applying $\boldsymbol{\phi}(A = 4, PLRU)$ on the divided traffic. Refer-

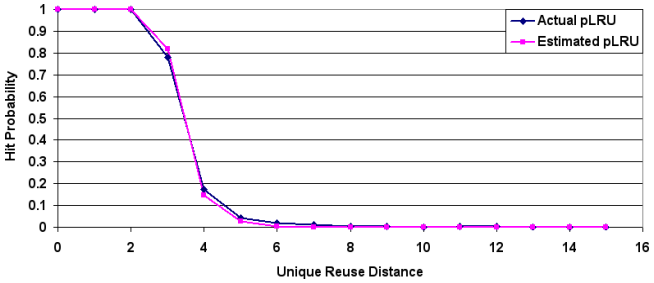


Figure 10: Representative $\boldsymbol{\phi}(A = 4, PLRU)$, actual and estimated.

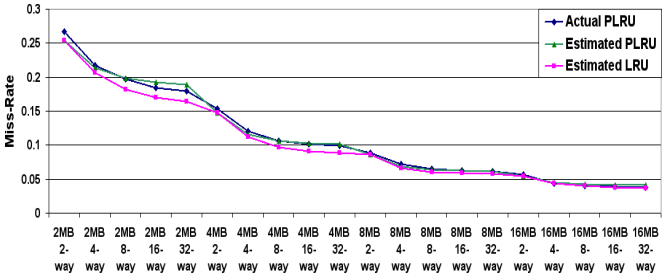


Figure 11: Actual vs estimated hit-rate function for oltp with PLRU replacement policy. LRU estimates are shown as reference.

ring to Figure 9 we claim that an 8-way tree can be viewed as a composition of two 4-way trees with the top-node dividing traffic between the two subtrees. Similar observations hold between 8-way and 16-way trees and so on. This helps us to estimate $\boldsymbol{\phi}(PLRU)$ for successively higher associativities. For ease of computation we assume that the top node divides traffic evenly between its two constituent sub-trees.

9.1 Base case: $A = 4$

Since $log_2(4) = 2$, $\boldsymbol{\phi}_k$ is 1 when $k \leq 2$. When $k \geq 4$, there are 3 elements in the set other than the reference element, but 2 of those cannot be replaced as they have less than 2 intervening elements for themselves. The only eviction candidate other than the reference element is the 3^{rd} element with URD 3. Thus, $\boldsymbol{\phi}_k = \boldsymbol{\phi}_{k-1} \cdot (1 - \boldsymbol{\phi}_3)$. Figure 10 shows a sharp fall at $k = 4$. This happens since $\boldsymbol{\phi}_4 = \boldsymbol{\phi}_3 \cdot (1 - \boldsymbol{\phi}_3)$. The length of the discontinuity is thus $\boldsymbol{\phi}_3 - \boldsymbol{\phi}_4 = \boldsymbol{\phi}_3^2$.

The only case that remains is when $k = 3$. Consider the k^{th} element. There are two subcases here:

1. It maps to the other subtree on either being present there (hit) or being allocated there on a miss. If it is present in the tree, it may occupy either subtree with probability (almost) $\frac{1}{2}$. If it is not present, it will be allocated to the subtree depending on how the top node switched on the last access. Since we assumed that the top node switches evenly between the two subtrees, this element will be allocated to the other subtree with probability $\frac{1}{2}$. Putting both subcases together, we estimate the probability that the k^{th} element will not be in the same subtree as the reference element as $\frac{1}{2}$. In this case $\boldsymbol{\phi}_k = \boldsymbol{\phi}_{k-1} = 1$.

2. It maps to the same subtree as the reference element. The reference element will be evicted only if the k_{th} element is not present (miss) and PLRU estimates the wrong stack. But in 4-way PLRU, there are only 3 admissible total orders of which 2 share the same last element. So the probability of a correct selection is $\frac{2}{3}$. The k_{th} element has URD> 2. Using $h(LRU)$ as an approximation, the probability of a miss for the k_{th} element is $\sum_{i=0}^{3} \boldsymbol{r}_i - \sum_{i=0}^{2} \boldsymbol{r}_i = \boldsymbol{r}_3$. So the hit-probability for the reference element in this case is $\frac{2}{3} \cdot (1 - \boldsymbol{r}_3)$.

Putting everything together,

$$\boldsymbol{\phi}_k = \begin{cases} 1 & \text{if } 0 \leq k \leq 2 \\ \frac{1}{2} + \frac{(1 - \boldsymbol{r}_3)}{3} & \text{if } k = 3 \\ \boldsymbol{\phi}_{k-1} \cdot (1 - \boldsymbol{\phi}_3) & \text{if } k \geq 4 \end{cases} \tag{20}$$

9.2 Recurrence: $A \geq 8$

Let $L = log_2(A)$ and $\boldsymbol{\psi} = \boldsymbol{\phi}(A/2)$. For the first case, when $k \leq L$, $\boldsymbol{\phi}_k$ must be 1. For $k > L$, consider the k^{th} element. There are two subcases here:

1. It maps to the other subtree. The argument is the same as given earlier and $\boldsymbol{\phi}_k = \boldsymbol{\phi}_{k-1}$.

2. It maps to the same subtree as the reference element. If $k \leq \frac{A}{2} + 1$, the $\frac{A}{2}$ other elements can all occupy the other subtree. So, the binomial sum starts with $\boldsymbol{\psi}_{1+i}$. If $k \geq \frac{A}{2} + 2$, there must be at least one other element in the reference subtree apart from the reference element and the k^{th} element. So, the binomial sum starts with $\boldsymbol{\psi}_{2+i}$.

Putting everything together,

$$\boldsymbol{\phi}_k = \begin{cases} 1 & \text{if } 0 \leq k \leq L \\ \frac{\boldsymbol{\phi}_{k-1}}{2} + \frac{1}{2} \sum_{i=0}^{k-2} {}^{k-2}C_i \left(\frac{1}{2}\right)^{(k-2)} \cdot \boldsymbol{\psi}_{1+i} & \text{if } L+1 \leq k \leq \frac{A}{2} + 1 \\ \frac{\boldsymbol{\phi}_{k-1}}{2} + \frac{1}{2} \sum_{i=0}^{k-3} {}^{k-3}C_i \left(\frac{1}{2}\right)^{(k-3)} \cdot \boldsymbol{\psi}_{2+i} & \text{if } k \geq \frac{A}{2} + 2 \end{cases} \tag{21}$$

Figure 11 shows actual vs estimated values of hit-rates for PLRU with the estimates computed using equations 5, 20, 21 and 4.

9.3 Way-counters for PLRU

In PLRU, the MRU line is known with certainty but the rest of the logical ordering is not precisely known. This prevents direct construction of way counters. Kedzierski, et al. proposed a heuristic for approximating logical stack positions for PLRU caches to enable way-counter based prediction [17]. Let *waynum* be the way number of the accessed line and *pathbits* denote the bit-values of the tree nodes along the path from the root to the leaf with root bit in MSB position. Let the function $reverse(b)$ reverse bit positions in the binary representation of b. The following heuristic is used to approximate $URD(x, m)$:

State	Action

idle	x = rand() % 10;
	if(x) state = idle
	else state = start-sample

start-sample	ref. addr = addr state = process-sample

process-sample	if(addr == ref. addr)
	state = end-sample-a
	else {
	If (hit in Bloom Filter)
	state = process-sample
	else {
	if (ctr>256) state = end-sample-b
	else {
	ctr++;
	Insert addr into Bloom Filter
	state = process-sample; }}}

end-sample-a	Emit URD = ctr value reset state = idle
end-sample-b	Emit URD = ∞ reset state = idle

Figure 12: Sampling Control

$\hat{URD}(x,m) = A - 1 - (reverse(waynum) \oplus pathbits)$
This approach aims to compute $\boldsymbol{r} \cdot \boldsymbol{\phi}(LRU)$ with $\boldsymbol{r}$ approximately measured using the above mechanism. The approximation for $\boldsymbol{r}$ is quite good (data not shown). However, apart from suffering from the traditional limitations of way counters (Section 6.2), it also ignores the fact that $\boldsymbol{\phi}(PLRU) \neq \boldsymbol{\phi}(LRU)$ (see Figure 11, 3). In contrast, our framework overcomes both limitations.

10. ONLINE ESTIMATION

We now propose an efficient hardware technique that can be used for online estimation of $\boldsymbol{r}$. We make two key observations:

1. The definition of URD (Section 3) depends *only on the cardinality of the reuse interval (RI) and not on the contents of the set.* This suggests applicability of hardware signatures, such as Bloom filters [7], that can be used to construct compact representations of sets. Whereas shadow tags store entire tag addresses, a Bloom filter uses only one bit to represent each address.

2. Although, $\boldsymbol{r}$ is a long-tailed distribution, *the limit to which $\boldsymbol{r}$ need to be tracked decreases with the size of the cache.* This suggests applicability of set-sampling techniques [15, 29, 18, 42] to conceptually reduce the size of the cache considered in the sample.

Our proposed hardware makes use of these two insights. It uses a Bloom filter (to represent RI), a counter to determine $|RI|$, and set-sampling logic. A schematic diagram of the hardware support is shown in Figure 13. We employ a 512-bit parallel Bloom filter [33] with two H_3 hash functions [8] to determine $|RI|$ between access to the reference line being sampled.

When a set-sample is chosen, only accesses with addresses mapping to those sets are considered. Once a set-sample is

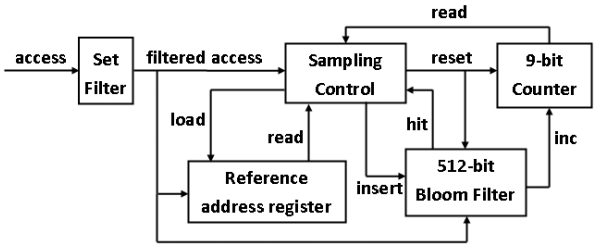


Figure 13: Schematic of Hardware support

fixed, not every access needs to be considered to compute the distribution. We employ *time-sampling* [20, 19, 18, 42] to prune this set. The state machine for the sampling control is shown in Figure 12. In effect, our proposed hardware estimates $\boldsymbol{r}(1)$, but limited in length for a (conceptually) smaller cache.

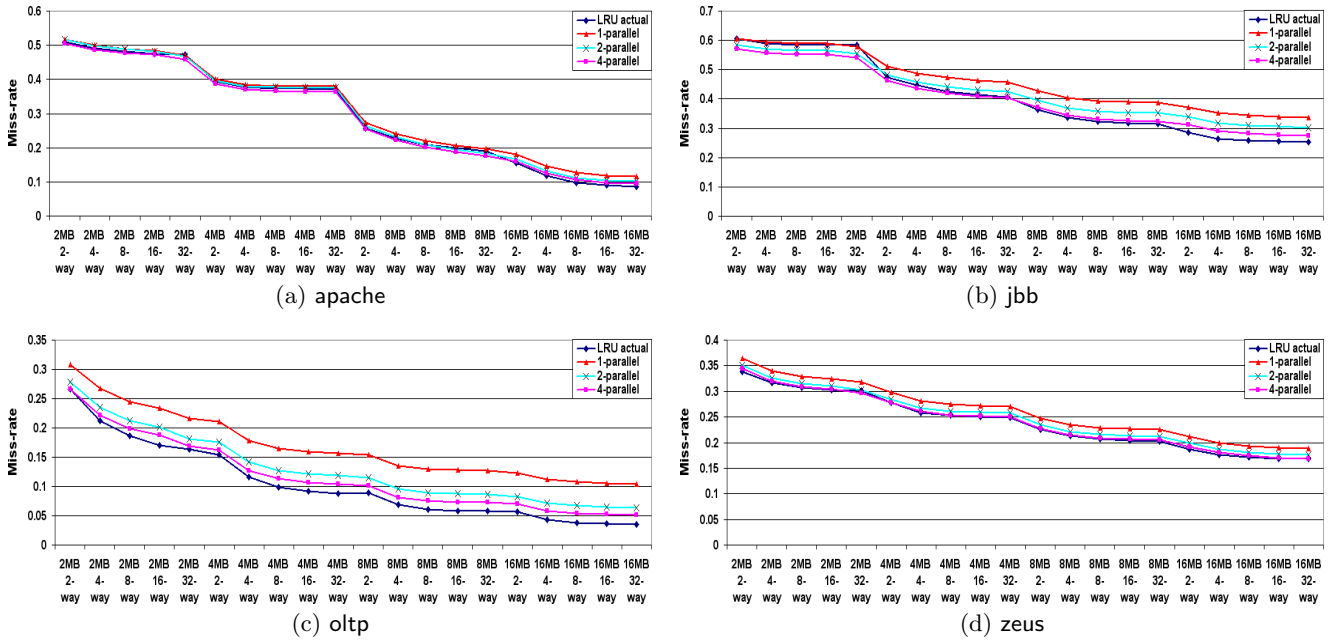
Figure 14 shows estimates computed using a simulated model of the hardware compared to actual values. We show average estimates with 1, 2 and 4 copies of the hardware in parallel. Having parallel copies shortens the length of the run required to get good approximations and the estimated curves asymptotically approach the actual curves. Increasing the sample length is equivalent to having parallel hardware. The errors between estimated and actual values can be reduced by using other efficient hashing techniques, e.g. PBX hashing [43], and/or by correcting for aliasing errors.

This analysis suggests that way counters and our proposed hardware have complimentary strengths. Way counters have more parallelism and hence shorter reaction times, but require retraining if S changes. They are more suitable for operating conditions where reconfiguration cost is not very high, such as data-preserving cache optimizations [14, 3]. Our new scheme would be more useful where reconfiguration is costly, for example, power-gating optimizations that lose data [27], where non-optimal decisions must be avoided and which inherently provide longer training intervals to amortize reconfiguration cost.

11. CONCLUSIONS

The central theme of this paper is an unified modeling framework to analyze cache performance at runtime for a wide range of organizations and replacement policies. It uses the concept of stack distances and transformations of probability vectors with Binomial matrices. The framework unifies previous analytical models such as Smith's associativity model, Cypher's Poisson model, and hardware techniques such as way counters. We have discussed limitations of set and way counters, given a method to convert way counter values for caches with a different number of sets and shown that this requires maintaining shadow tags for more than the maximum associativity. We have also proposed an alternate scheme that is decoupled from the cache configuration, uses hardware signatures for compact representation of reuse intervals and can be used as an alternative to way counters.

Although we have demonstrated how several popular replacement algorithms can be modeled, we have not discussed what cannot be modeled with our framework. We hypothesize that replacement policies based on access frequency may not be analyzable within this framework or may need addi-



(a) apache (b) jbb

(c) oltp (d) zeus

Figure 14: Actual vs estimated miss-rates for LRU with r computed using the proposed hardware. k-parallel indicates k units used in parallel. Each set-sample is 1/1024 of the cache. Sampling estimates shown are means over all set-samples.

tional information. Formulating membership criteria and modeling other replacement policies remain the focus of our future work.

12. REFERENCES

[1] A. Agarwal, J. Hennessy, and M. Horowitz. An analytical cache model. *ACM Transactions on Computer Systems*, 7:184–215, May 1989.

[2] A. R. Alameldeen, M. M. K. Martin, C. J. Mauer, K. E. Moore, M. Xu, D. J. Sorin, M. D. Hill, and D. A. Wood. Simulating a $2m commercial server on a $2k pc. *IEEE Computer*, 36(2):50–57, Feb. 2003.

[3] D. H. Albonesi. Selective cache ways: on-demand cache resource allocation. In *Proc. of the 32nd Annual IEEE/ACM International Symp. on Microarchitecture*, pages 248–259, Nov. 1999.

[4] G. Almási, C. Caşcaval, and D. A. Padua. Calculating stack distances efficiently. *ACM SIGPLAN Notices*, 38:37–43, Feb. 2003.

[5] L. A. Belady. A study of replacement algorithms for virtual-storage computer. *IBM Systems Journal*, 5(2):78–101, 1966.

[6] K. Beyls and E. D'Hollander. Reuse distance as a metric for cache behavior. In *Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems*, pages 617–622, Aug. 2001.

[7] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, July 1970.

[8] J. L. Carter and M. N. Wegman. Universal classes of hash functions (extended abstract). In *Proceedings of the 9th Annual ACM Symposium on Theory of Computing*, pages 106–112, 1977.

[9] R. Cypher. Apparatus and method for determining stack distance including spatial locality of running software for estimating cache miss rates based upon contents of a hash table. *US7366871*, Apr. 2008.

[10] R. Cypher. Apparatus and method for determining stack distance of running software for estimating cache miss rates based upon contents of a hash table. *US7373480*, May 2008.

[11] C. Ding and Y. Zhong. Reuse distance analysis. Technical report, University of Rochester, Rochester, NY, USA, 2001.

[12] S. Dropsho, A. Buyuktosunoglu, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas, G. Semeraro, G. Magklis, and M. L. Scott. Integrating adaptive on-chip storage structures for reduced dynamic power. In *Proc. of the Intnl. Conf. on Parallel Architectures and Compilation Techniques*, pages 141–152, Sept. 2002.

[13] B. Falsafi and D. A. Wood. Modeling cost/performance of a parallel computer simulator. *ACM Transactions on Modeling and Computer Simulation*, 7(1):104–130, Jan. 1997.

[14] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy caches: simple techniques for reducing leakage power. In *Proc. of the 29th Annual Intnl. Symp. on Computer Architecture*, May 2002.

[15] P. Heidelberger and H. S. Stone. Parallel trace-driven cache simulation by time partitioning. Technical Report IBM Technical Report RC 15500 , IBM, February, 1990.

[16] M. D. Hill and A. J. Smith. Evaluating associativity in cpu caches. *IEEE Transactions on Computers*,

38(12):1612–1630, Dec. 1989.

[17] K. Kedzierski, M. Moreto, F. Cazorla, and M. Valero. Adapting cache partitioning algorithms to pseudo-lru replacement policies. In *Proc. of the 24th IEEE Intnl. Parallel and Distributed Processing Symposium*, pages 1–12, Apr. 2010.

[18] R. E. Kessler, M. D. Hill, and D. A. Wood. A comparison of trace-sampling techniques for multi-megabyte caches. *IEEE Transactions on Computers*, 43(6):664–675, 1994.

[19] S. Laha. *Accurate low-cost methods for performance evaluation of cache memory systems*. PhD thesis, University of Illinois, Department of Computer Science, Urbana, IL, USA, 1988.

[20] S. Laha, J. H. Patel, and R. K. Iyer. Accurate low-cost methods for performance evaluation of cache memory systems. *IEEE Transactions on Computers*, 37(11):1325–1336, 1988.

[21] H. Le, W. Starke, J. Fields, F. O'Connell, D. Nguyen, B. Ronchetti, W. Sauer, E. Schwarz, and M. Vaden. Ibm power6 microarchitecture. *IBM Journal of Research and Development*, 51(6), 2007.

[22] F. Liu, F. Guo, Y. Solihin, S. Kim, and A. Eker. Characterizing and modeling the behavior of context switch misses. In *Proc. of the Intnl. Conf. on Parallel Architectures and Compilation Techniques*, pages 91–101, Oct. 2008.

[23] Y. Liu and W. Zhang. Exploiting stack distance to estimate worst-case data cache performance. In *Proceedings of the 2009 ACM symposium on Applied Computing*, pages 1979–1983, Mar. 2009.

[24] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood. Multifacet's general execution-driven multiprocessor simulator (gems) toolset. *Computer Architecture News*, pages 92–99, Sept. 2005.

[25] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger. Evaluation techniques for storage hierarchies. *IBM Systems Journal*, 9(2):78–117, 1970.

[26] M. Moreto, F. J. Cazorla, A. Ramirez, and M. Valero. Dynamic cache partitioning based on the mlp of cache misses. In P. Stenström, editor, *Transactions on high-performance embedded architectures and compilers III*, pages 3–23. Springer-Verlag, Berlin, Heidelberg, 2011.

[27] S. G. Narendra and A. Chandrakasan. *Leakage in Nanometer CMOS Technologies (Series on Integrated Circuits and Systems)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[28] Poisson distribution. http://en.wikipedia.org/wiki/Poisson_distribution.

[29] T. R. Puzak. *Analysis of Cache Replacement Algorithms*. PhD thesis, Dept. of Electrical and Computer Engineering, University of Massachusetts, 1985.

[30] M. K. Qureshi, D. N. Lynch, O. Mutlu, and Y. N. Patt. A case for mlp-aware cache replacement. In *Proc. of the 33nd Annual Intnl. Symp. on Computer Architecture*, ISCA '06, pages 167–178, Washington, DC, USA, June 2006. IEEE Computer Society.

[31] M. K. Qureshi and Y. N. Patt. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In *Proc. of the 39th Annual IEEE/ACM International Symp. on Microarchitecture*, pages 423–432, Dec. 2006.

[32] J. Reineke and D. Grund. Relative competitive analysis of cache replacement policies. In *Proceedings of the 2008 ACM SIGPLAN-SIGBED conference on Languages, compilers, and tools for embedded systems*, LCTES '08, pages 51–60, New York, NY, USA, 2008. ACM.

[33] D. Sanchez, L. Yen, M. D. Hill, and K. Sankaralingam. Implementing signatures for transactional memory. In *Proc. of the 40th Annual IEEE/ACM International Symp. on Microarchitecture*, Dec. 2007.

[34] X. Shi, F. Su, J.-K. Peir, and Z. Yang. Modeling and stack simulation of cmp cache capacity and accessibility. *IEEE Transactions on Parallel and Distributed Systems*, 20(12):1752–1763, Dec. 2009.

[35] A. J. Smith. A comparative study of set associative memory mapping algorithms and their use for cache and main memory. *IEEE Trans. on Software Engineering*, SE-4(2):121–130, March 1978.

[36] K. So and R. N. Rechtschaffen. Cache operations by mru change. *IEEE Transactions on Computers*, 37(6):700–709, June 1988.

[37] H. S. Stone and D. Thibaut. Footprints in the cache. In *ACM SIGMETRICS Performance Evaluation Review*, pages 4–8, May 1986.

[38] J. E. Strum. Binomial matrices. *The Two-Year College Mathematics Journal*, 8(5):260–266, Nov. 1977.

[39] G. E. Suh, S. Devadas, and L. Rudolph. A new memory monitoring scheme for memory-aware scheduling and partitioning. In *Proceedings of the Eighth IEEE Symposium on High-Performance Computer Architecture*, Feb. 2002.

[40] G. E. Suh, L. Rudolph, and S. Devadas. Dynamic cache partitioning for cmp/smt systems. *Journal of Supercomputing*, pages 7–26, 2004.

[41] D. K. Tam, R. Azimi, L. B. Soares, and M. Stumm. Rapidmrc: approximating l2 miss rate curves on commodity systems for online optimizations. In *Proc. of the 14th Intnl. Conf. on Architectural Support for Programming Languages and Operating Systems*, pages 121–132, Mar. 2009.

[42] N. C. Thornock and J. K. Flanagan. Facilitating level three cache studies using set sampling. In *Proceedings of the 32nd conference on Winter simulation*, pages 471–479, Dec. 2000.

[43] L. Yen, S. C. Draper, and M. D. Hill. Notary: Hardware techniques to enhance signatures. In *Proc. of the 41st Annual IEEE/ACM International Symp. on Microarchitecture*, Nov. 2008.

[44] Y. Zhong, X. Shen, and C. Ding. Program locality analysis using reuse distance. *ACM Transactions on Programming Languages and Systems*, 31(6):1–39, Aug. 2009.

APPENDIX

A. PLRU REGULAR EXPRESSIONS

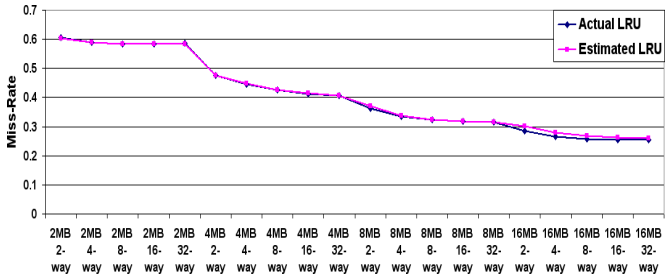
Since the PLRU tree is symmetric, we can fix any way as reference without loss of generality. Let the immediate neighbor be denoted by Q_0, the next two neighbors be collectively denoted by Q_1 and so on with the most distant group of $A/2$ neighbors denoted by $Q_{log_2(A)-1}$.

To calculate the probability that the reference line will be evicted on a particular miss we need to consider the immediate past sequence of accesses to that set. A necessary and sufficient condition for the reference line to be evicted is for the suffix of the trace to match the particular regular expression described below.

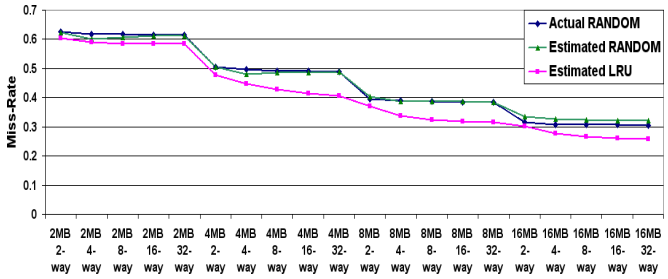
$$
\begin{aligned}
A = 2 &: Q_0^+ \\
A = 4 &: Q_0 Q_1^+ \\
A = 8 &: Q_0 (Q_1 + Q_2)^* Q_1 Q_2^+ \\
A = 16 &: Q_0 (Q_1 + Q_2 + Q_3)^* Q_1 (Q_2 + Q_3)^* Q_2 Q_3^+ \\
A = 32 &: Q_0 (Q_1 + Q_2 + Q_3 + Q_4)^* Q_1 (Q_2 + Q_3 + Q_4)^* \\
&\quad Q_2 (Q_3 + Q_4)^* Q_3 Q_4^+
\end{aligned}
$$

B. MODEL ESTIMATES

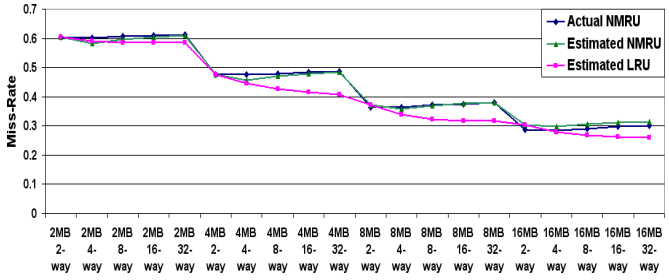
Figures 15, 16 and 17 show actual vs estimated miss-rates for the other commercial workloads: jbb, apache and zeus. While $h(LRU)$ may be used to approximate $h(PLRU)$ for these workloads, it cannot be used for $h(RANDOM)$ or $h(NMRU)$ without incurring significant errors.



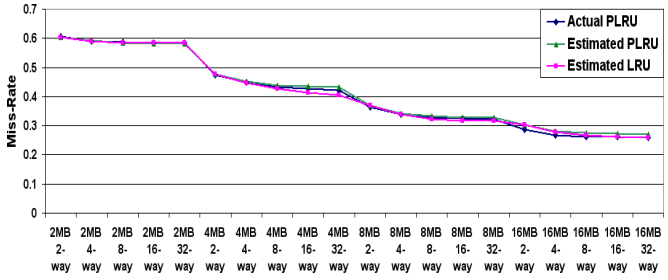
(a) LRU



(b) RANDOM, with LRU estimates shown as reference

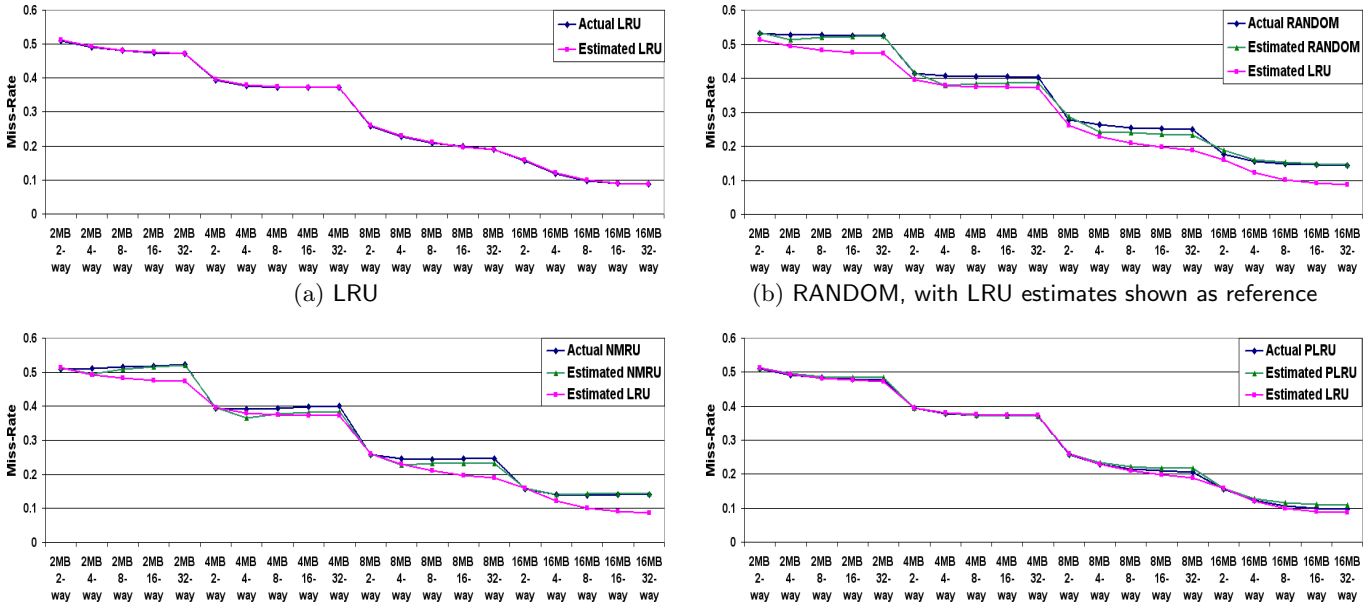


(c) NMRU, with LRU estimates shown as reference



(d) PLRU, with LRU estimates shown as reference

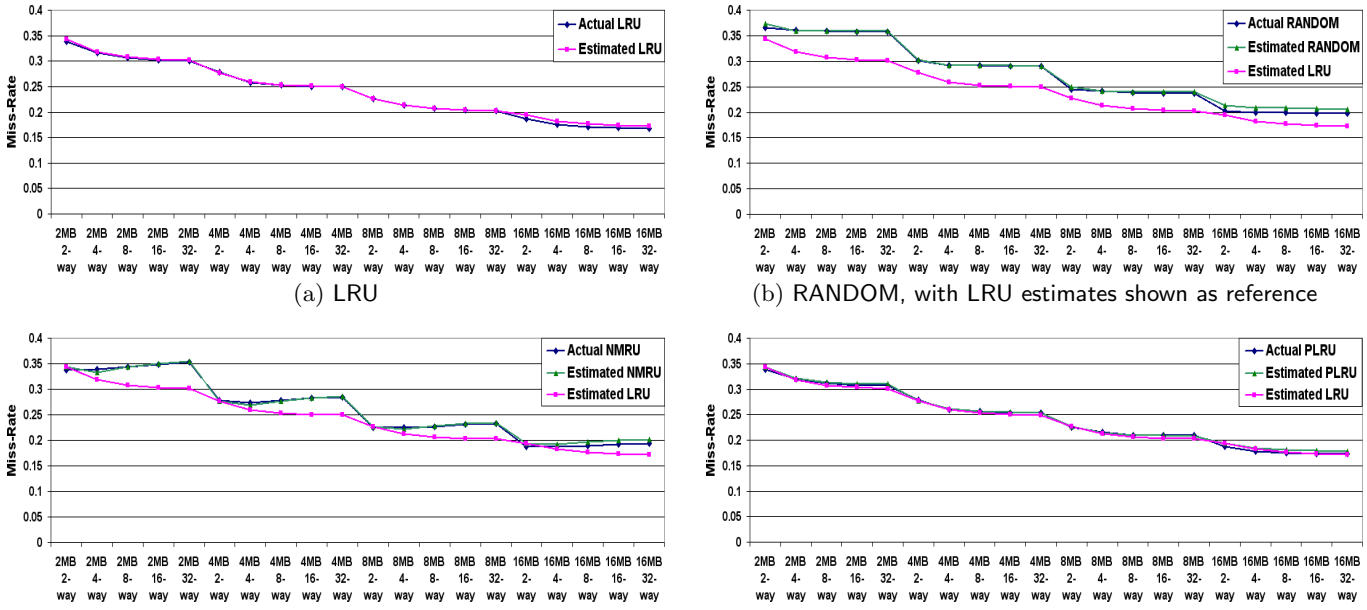
Figure 15: Actual vs estimated miss-rates for jbb



(a) LRU (b) RANDOM, with LRU estimates shown as reference

(c) NMRU, with LRU estimates shown as reference (d) PLRU, with LRU estimates shown as reference

Figure 16: Actual vs estimated miss-rates for apache



(a) LRU (b) RANDOM, with LRU estimates shown as reference

(c) NMRU, with LRU estimates shown as reference (d) PLRU, with LRU estimates shown as reference

Figure 17: Actual vs estimated miss-rates for zeus