Reuse-based Online Models for Caches

Rathijit Sen
rathijit@cs.wisc.edu

David A. Wood
david@cs.wisc.edu

Department of Computer Sciences
University of Wisconsin-Madison

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. 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 LRU way-counters to work when the number of sets in the cache changes. As an example application, we demonstrate how results from our models can be used to select, based on workload access characteristics, last-level cache configurations that aim to minimize energy-delay product.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Modeling techniques ; B.3.2 [**Memory Structures**]: Cache memories

General Terms

Performance

Keywords

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

1. INTRODUCTION

Processor caches are critical components of the memory hierarchy that exploit locality to keep frequently-accessed data on chip. Caches can significantly boost performance and reduce energy usage, but their benefit is highly workload dependent. Figure 1 illustrates the miss rates for 16 multithreaded workloads over 5 different sizes of last-level cache (LLC). Some workloads (e.g., `apache`) benefit substantially from larger caches, while others (e.g., `equake`) are largely indifferent to the cache size.

In modern power and energy constrained computer systems, understanding a workload's dynamic cache behavior is

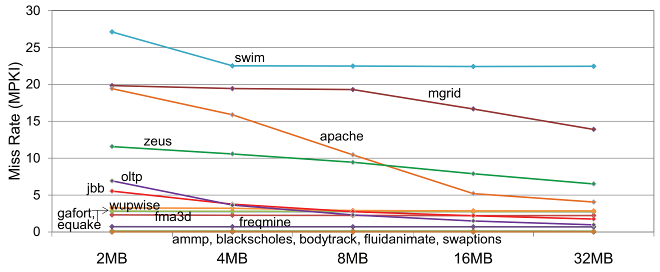


Figure 1: LLC Misses-per-thousand-instructions (MPKI) for our workloads (Section 6). There is significant miss rate variation depending on the workload and LLC size.

important for making critical resource allocation and scheduling decisions. For example, allocating excess cache capacity to a workload wastes power, as large caches dissipate significant leakage power, while allocating insufficient cache capacity hurts performance and increases main memory power. Previous research has explored placing some or all of a cache in low-power mode [4,16,18] or dynamically partitioning the cache to eliminate resource contention [35,45]. A recent Intel processor [22] can dynamically reduce its LLC's capacity to save power.

Because a workload's cache performance can vary with different execution phases, cache resource decisions must be made online and depend upon predicting the cache performance for configurations different than the current settings. For example, set-associative caches map each address to a set, each of which contains a number of ways (lines); since both sets and ways can be dynamically configured [48], we need to predict cache performance for many configurations (see Table 1). Suh, et al. [44] use hardware way-counters to predict cache miss ratios, but their technique cannot be used for configurations with a different number of sets. Gordon-Ross, et al. [20] use a hardware TCAM to track reuse distances for determining miss ratios for an 8KB L1 cache, with 12% area overhead. However, while 12% overhead may be acceptable for small L1 caches, it is prohibitively high for the multi-megabyte LLCs of modern multicore processors.

In this work we study the problem of developing efficient online techniques for predicting cache miss rates for large caches that vary in both associativity and number of sets, and have practical replacement policies such as RANDOM, NMRU, and PLRU, unlike prior work that has largely focused on LRU. Our analysis framework is inspired by two foundational works: Mattson's stack distance characteriza-

size \ assoc.	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}
32MB	2^{18}	2^{17}	2^{16}	2^{15}	2^{14}

Table 1: Relation between number of sets and associativity for different cache sizes. Assuming some cache configuration is the current configuration, there are a total of 25-1=24 possible other target configurations. An inspection of the table reveals that *at most* 4 of these possible 24 configurations can have the same number of sets as the current configuration. For example, with 32MB 32-way as the current configuration, target configurations with the same number of sets (2^{14}) are: 2MB 2-way, 4MB 4-way, 8MB 8-way and 16MB-16-way. Thus, way-counters (Section 5.3) can predict for *at most* 4 of 24 possible target configurations at any time.

tion [32] (also used later as reuse distance [8,15]) and Smith's associativity model [21, 40] for LRU caches.

We show how simple hardware, requiring approximately 2KB of state, can provide the dynamic information needed to drive the model. As an example application, we demonstrate that it can be used to select an LLC configuration to minimize energy-delay product (EDP). The LLC configuration can cause EDP to vary by as much as a factor of 3; using our model selects a configuration within 7% of optimal.

The major contributions of our work are:

1. We formulate an analytical framework based on generalized stochastic Binomial Matrices [43] for transforming reuse distance distributions (Sections 3, 4).
2. We formulate new miss ratio prediction models for RANDOM (Section 4.2), NMRU (Section 4.3), PLRU (Section 4.4) replacement policies.
3. We show that the traditional hardware way-counter based prediction [44] for varying associativity is a special instance of our unified framework (Section 5.3). Further, we show how way-counter data for LRU may be transformed to apply to caches with a different number of sets. (Section 5.3.2)
4. We propose a novel hardware scheme for efficient online estimation of reuse distance/stack distance distributions (Section 5.1).
5. We demonstrate one application of the model in finding the minimum EDP (energy-delay product, [19]) configuration (Section 6). The results are within 7% of the optimum.

1.1 Model overview and Paper Organization

The central theme of our predictive framework is to decouple temporal characteristics in the cache access stream from characteristics of the replacement policy. The rest of this paper is divided into five major portions:

Characterizing temporal locality: Section 2 defines reuse distributions that capture the temporal locality of address streams. Section 3 shows how to modify these to apply for a cache with a different number of sets.

Characterizing replacement policies: Section 4 introduces the notion of cache hit-functions that, when multiplied with the per-set reuse distribution, produce expected cache hit ratios. Sections 4.1.1 and 4.1.2 consider optimizations for LRU hit ratio prediction. Sections 4.2, 4.3 and 4.4 develops new prediction models for RANDOM, NMRU, PLRU respectively. Section 4.5 discusses prediction accuracy and computation overheads.

Hardware Support: Section 5.1 presents the novel, low-cost hardware for estimating reuse distributions. It also discusses two traditional hardware mechanisms – set-counters (Section 5.2) and way-counters (Section 5.3).

Example Application: Section 6 shows how our model can be used to find the minimum EDP configuration.

Epilogue: Section 7 discusses related work and Section 8 concludes the paper.

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.

Our models estimate hit ratio (hit/access). This can be easily converted into other measures: miss ratio=1-hit ratio; miss rate=miss ratio*access/instruction. For evaluating prediction quality, we obtain address traces of accesses to a 32MB 32-way LLC in a simulated system (Table 2, Section 6) for our workloads (Section 6), run the traces through a standalone cache simulator (that does not model timing) and compare measured against predicted metrics.

2. MEASURES OF TEMPORAL LOCALITY

In this section we develop metrics of temporal locality in the address stream that are independent of the cache configuration. These metrics will be used for estimating the miss ratios for arbitrary cache configurations. For our study, all addresses are line addresses of cache accesses.

Consider an address trace T as a mapping of consecutive integers in increasing order, representing successive positions in the trace, to tuples (x, m) where x identifies the address and m identifies its repetition number. The first occurrence of address x in the trace 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 unique 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 [32].

Absolute Reuse Distance: This denotes the total number of positions between two occurrences of the same address

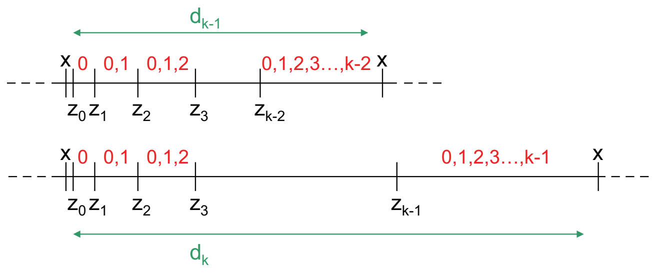


Figure 2: Schematic diagram showing relation between $\boldsymbol{d}_k(t)$ and $\boldsymbol{d}_{k-1}(T)$. Possible URDs for accesses in each sub-interval are shown in red.

in the trace. Thus, $ARD(x, m) =$

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

As an example, in the access sequence $a\ b\ b\ c\ d\ b\ a$, $URD(a, 1) = 3$ and $ARD(a, 1) = 5$.

2.1 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, $\boldsymbol{r}_k(T) = P(URD(x,m) = k), \forall(x,m) \in image(T)$. The characterization is lossy in the sense that in general, T cannot be recovered from $\boldsymbol{r}(T)$ even upto permutation of entity identifiers (See Appendix A).

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

2.2 $\boldsymbol{d}$(T) Estimation

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 to compute (an approximation to) $\boldsymbol{d}(T)$ given $\boldsymbol{r}(T)$.

Figure 2 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. This is just a conceptual tool 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 observe 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 $\{0, 1, .., k-1\}$. We approximate the length of this run with the expected number of trials to success in a geometric distribution with success probability $\sum_{i=k}^{\infty} \boldsymbol{r}_i(T)$.

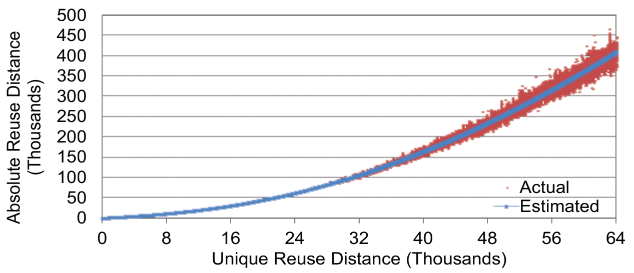


Figure 3: Actual vs estimated $\boldsymbol{d}(T)$ for oltp.

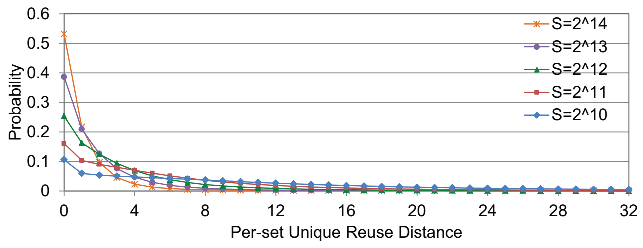


Figure 4: Effect of the number of sets (S) on per-set locality for oltp.

We thus arrive at the following recurrence:

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

Expanding the recurrence gives us

$$\boldsymbol{d}_k(T) = \sum_{j=1}^{k} \frac{1}{\sum_{i=j}^{\infty} \boldsymbol{r}_i(T)} = \sum_{j=1}^{k} \frac{1}{1 - \sum_{i=0}^{j-1} \boldsymbol{r}_i(T)}$$

This is similar to known approximations for the coupon collector's problem assuming a given order of coupons [11]. We have seen good agreement in trends between observed and estimated values of $\boldsymbol{d}(T)$ as illustrated in Figure 3. More accurate approximations for $\boldsymbol{d}(T)$ can be considered if needed at the expense of more computation time.

3. PER-SET LOCALITY

Replacement policy decisions (determining which cache line to evict) in traditional caches happen for each individual set. This in turn influences the miss ratio. So it is essential to determine the locality in the address stream that each individual set sees on average. We refer to the temporal locality in the per-set address stream as the per-set locality.

Per-set locality is strongly influenced by the number of sets (S) in the cache. The set of *unique* addresses in the address stream is split among the sets based on the index mapping function. The address stream that any individual set sees is the subset of the original address stream consisting of *all* accesses to the addresses mapping to that set. S thus determines the degree to which the address stream is split. Decreasing S increases URDs of the per-set address streams since addresses that hitherto mapped to other sets now get mapped to the reference set, and vice versa.

Accordingly, we extend our previous notations of locality metrics to additionally include S as a parameter. Thus

$\boldsymbol{r}(T, S)$ denotes the unique reuse distribution of the subsequence of T that a single set in the cache observes on average. $\boldsymbol{r}(T, 1)$ is the temporal locality of the original address stream, which is also the per-set locality of a fully-associative cache ($S = 1$). Figure 4 illustrates how $\boldsymbol{r}(T, S)$ changes with S for `oltp`. $\boldsymbol{d}(T)$ is adapted to $\boldsymbol{d}(T, S)$ similarly and can be estimated from $\boldsymbol{r}(T, S)$ using Equation 1. For brevity of notation, we will omit specifying one or more parameters when their values are clear from the context.

As Table 1 shows, cache configurations in our study have a range of number of sets (2^{10} to 2^{18}). For efficiently predicting miss ratios it is essential to be able to determine how $\boldsymbol{r}(S)$ can be transformed to $\boldsymbol{r}(S')$ for $S' \neq S$. The rest of this section develops a (new) methodology for this.

3.1 $\boldsymbol{r}(\mathbf{S'})$ Estimation

For set-associative caches with $S' > 1$ we make the simplifying assumption, similar to Smith's model [21, 40], that the mapping of unique lines to cache sets are independent of each other. While this assumption does not always hold with the traditional bit selection index function, some processors use simple XOR hashing functions that increase uniformity [27]. The uniformity assumption enables both the following model and the use of uniform set-sampling techniques.

Accesses to a given set can thus be modeled as successive Bernoulli trials with the success of each trial having probability $\frac{1}{S'}$. While computing $\boldsymbol{r}(S')$ from $\boldsymbol{r}(1)$, we note that $\boldsymbol{r}_j(S')$ is the sum of the probability of exactly j successes (j addresses mapping to the reference set) from $\boldsymbol{r}_k(1)$, $\forall k$. The generalized stochastic Binomial Matrix [43] $\boldsymbol{B}(x, y)$ has the value ${}^kC_j y^j x^{k-j}$ in row k, column j, where ${}^kC_j$ denotes the j^{th} binomial coefficient and $x + y = 1$. This is the same as the probability of exactly j successes in k Bernoulli trials with probability of each success being y. Viewing the computation of $\boldsymbol{r}(S')$ from $\boldsymbol{r}(1)$ through the lens of matrix multiplication, we recognize that the transformer is a generalized stochastic Binomial Matrix, $\boldsymbol{B}(1 - \frac{1}{S'}, \frac{1}{S'})$. Thus,

$$\boldsymbol{r}(S') = \boldsymbol{r}(1) \cdot \boldsymbol{B}(1 - \frac{1}{S'}, \frac{1}{S'}) \tag{2}$$

It is straight-forward to show that the above transformation respects $\sum_{i=0}^{\infty} \boldsymbol{r}_i(S') = \sum_{i=0}^{\infty} \boldsymbol{r}_i(1) = 1$. 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 (see Figure 4).

We will now show how to compute $\boldsymbol{r}(S')$ from any starting cache configuration S. This shows how computations can be reused instead of always needing to start from the ground configuration ($S = 1$) and will also be useful in reasoning about way-counters (Section 5.3).

Binomial Matrices are invertible (when the second parameter is non-zero) and closed under multiplication within the same dimension [43]. Using identities $\boldsymbol{B}(x, y)\boldsymbol{B}(w, z) = \boldsymbol{B}(x + yw, yz)$ and $\boldsymbol{B}(x, y)^{-1} = \boldsymbol{B}(-xy^{-1}, y^{-1})$, [43], we get

$$\begin{aligned} \boldsymbol{r}(S') &= \boldsymbol{r}(1) \cdot \boldsymbol{B}(1 - \frac{1}{S'}, \frac{1}{S'}) \\ &= \boldsymbol{r}(S) \cdot (\boldsymbol{B}(1 - \frac{1}{S}, \frac{1}{S}))^{-1} \cdot \boldsymbol{B}(1 - \frac{1}{S'}, \frac{1}{S'}) \\ &= \boldsymbol{r}(S) \cdot \boldsymbol{B}(1 - \frac{S}{S'}, \frac{S}{S'}) \end{aligned} \tag{3}$$

Equation 3 is a general form of Equation 2. The transformer depends only on the ratio of the number of the sets in the current cache to that in 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 that the computed probabilities are valid ($\in [0, 1]$) *even if* $\boldsymbol{r}(S)$ *has not been computed binomially.* Moreover, this allows intermediate steps; for example, computing $\boldsymbol{r}(2^{14})$ from $\boldsymbol{r}(1)$ is equivalent to first computing $\boldsymbol{r}(2^{10})$ from $\boldsymbol{r}(1)$ and then computing $\boldsymbol{r}(2^{14})$ from $\boldsymbol{r}(2^{10})$. This provides an opportunity to *reuse intermediate computations.* So, $\boldsymbol{r}(S)$ can be computed once from $\boldsymbol{r}(1)$ for the smallest S (2^{10} in our study, see Table 1) and used for all other target configurations.

Case 2, $\mathbf{S' < S}$: Since $\boldsymbol{B}(1 - \frac{S}{S'}, \frac{S}{S'}) = (\boldsymbol{B}(1 - \frac{S'}{S}, \frac{S'}{S}))^{-1}$, Case 2 transforms can invert Case 1 transforms provided Case 1 results have not been truncated (see below). Otherwise, the computed probabilities may not be valid ($\notin [0, 1]$).

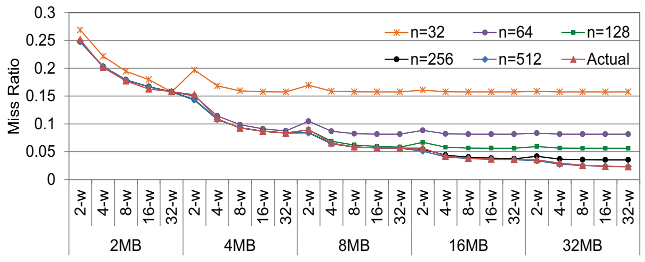


Figure 5: LRU prediction with reuse information limited to n at $\boldsymbol{r}(2^{10})$ which is first computed from $\boldsymbol{r}(1)$ (Equation 2).

3.2 Matrix dimension and Truncation of $\boldsymbol{r}$

The dimension of $\boldsymbol{B}$ is determined by the maximum (per-set) URD, n, we are interested to maintain to avoid large computational costs. If $\boldsymbol{r}$ is limited to position n, $\boldsymbol{r}_{\infty}(S)$ must be adjusted so that $\boldsymbol{r}_{\infty}(S) = 1 - \sum_{i=0}^{n} \boldsymbol{r}(S)$.

Assume $\boldsymbol{r}(2^{10})$ is available, computed from $\boldsymbol{r}(1)$ using Equation 2. Figure 5 shows predicted miss ratios for `oltp` with $\boldsymbol{r}(2^{10})$ maintained for various values of n. Section 4.1 explains LRU prediction. Although the maximum associativity that we consider is 32, Figure 5 shows that n has to be much larger than that (≥ 512) for good predictions for larger caches with $S' > 2^{10}$, such as 32MB caches (see Table 1).

While $n = 512$ is good for $\boldsymbol{r}(2^{10})$, the equivalent value for $\boldsymbol{r}(1)$ is very large, potentially upto $512 \cdot 2^{10}$. To appreciate this, consider the $\boldsymbol{r}(1)$ address stream as a merger of the 2^{10} mutually exclusive per-set address streams, each of which has reuse intervals of upto 512. Determining the long-tailed $\boldsymbol{r}(1)$ distribution or using large matrices to compute $\boldsymbol{r}(2^{10})$ from $\boldsymbol{r}(1)$ in software is time-consuming. Section 5.1 proposes low-cost hardware support to approximately estimate $\boldsymbol{r}(2^{10})$ with $n = 512$.

4. CACHE HIT FUNCTIONS

Given a target cache organization ($S', A', policy$) and a trace T, our goal is to determine a vector $\boldsymbol{\phi}(\boldsymbol{r}(S'), S', A', policy)$ such that the expected hit ratio for the trace is

$$h = \boldsymbol{r} \cdot \boldsymbol{\phi} \tag{4}$$

The idea is to characterize workload traces by $\boldsymbol{r}$ and caches by $\boldsymbol{\phi}$ so that the effect on hit ratio for changes in traces or cache configurations can be readily estimated.

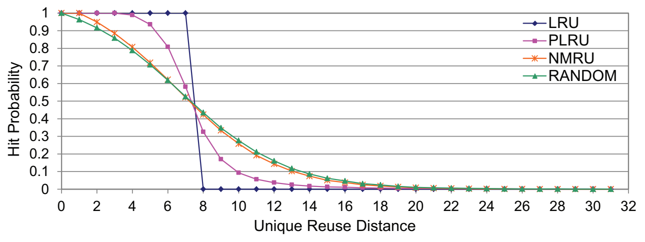


Figure 6: Representative hit ratio functions for an 8-way cache with different replacement policies but with the same trace and number of sets.

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 in this model. This is because non-eviction of a cache-resident address after accesses involving k other unique addresses implies non-eviction after accesses involving $k-1$ unique addresses *and* the remaining accesses. If there are no intervening accesses ($k = 0$), the access must be a hit. Accesses hitherto never seen ($k = \infty$) must miss. So,

$$\boldsymbol{\phi}_k = \begin{cases} 1 & \text{if } k = 0 \\ \leq \boldsymbol{\phi}_{k-1} & \text{if } k \geq 1 \\ 0 & \text{at } k = \infty \end{cases} \tag{5}$$

Figure 6 shows representative characteristic functions for common replacement policies. We consider the well-known, but rarely-implemented[1] LRU policy as well as the practical RANDOM, NMRU, and PLRU policies. Note that RANDOM, NMRU, PLRU have hit functions that are non-zero beyond A'(=8 in the figure). So, computing $\boldsymbol{r} \cdot \boldsymbol{\phi}$ upto A' is not sufficient for these replacement policies. For our evaluations, we compute the dot-product for $2A'$ terms; longer than that has diminishing returns for our workloads.

Apart from LRU, $\boldsymbol{\phi}$ is not independent of $\boldsymbol{r}$ for different replacement policies. As we shall show later, $\boldsymbol{\phi}(RANDOM)$ depends on $\boldsymbol{d}$, while $\boldsymbol{\phi}(PLRU)$ may need more information.

4.1 Estimating $\boldsymbol{\phi}$(LRU)

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 ratio function.

$$\boldsymbol{\phi}_k(LRU) = \begin{cases} 1 & \text{if } 0 \leq k < A' \\ 0 & \text{if } k \geq A' \end{cases} \tag{6}$$

Figure 5 shows actual vs estimated ($n = 512$) miss ratios for `oltp` with LRU using Equations 3, 6 and 4. Figure 16 (Appendix F) shows more examples. As observed earlier by Hill and Smith [21], increasing A' yields diminishing returns.

4.1.1 *Optimization (Smith's Model)*

A naive combination of Equations 4, 6 and 3 results in $\sum_{i=0}^{A'-1}(n-i) = nA' - \frac{A'(A'-1)}{2}$ multiplications with binomial computations to estimate the hit ratio for a cache with S' sets and associativity A'. The number of multiplications can be reduced by observing that due to the step-function nature of $\boldsymbol{\phi}(LRU)$, some of the coefficients will sum to 1. Expanding the computation and simplifying, we get

[1]LRU is typically not implemented in real caches for associativity larger than 4 due to hardware complexity.

$$h(S') = \sum_{i=0}^{A'-1} \boldsymbol{r}_i(1) + \sum_{i=A'}^{n} \boldsymbol{r}_i(1) \cdot \sum_{k=0}^{A'-1} {}^{i}C_k \cdot \left(\frac{1}{S'}\right)^k \cdot \left(1 - \frac{1}{S'}\right)^{(i-k)} \tag{7}$$

Equation 7 is an optimized version of Smith's associativity model [21, 40]. It requires $nA' - A'(A'-1)$ multiplications which is $\frac{A'(A'-1)}{2}$ less than the naive combination. But computing binomial terms is costly and n is usually much larger than A'.

4.1.2 *Poisson approximation to Binomial*

The last sum in Equation 7 is the cumulative binomial sum up to $A'-1$ with parameters i and $\frac{1}{S'}$. Cypher [13, 14] uses a Poisson approximation to binomial for reducing computational costs – when i is large and $\frac{1}{S'}$ is small, the binomial distribution can be approximated by a Poisson distribution with parameter $\lambda = \frac{i}{S'}$. This is easier to compute than with the binomial coefficients.

We can further optimize Cypher's approach by substituting the cumulative Poisson sum with piecewise linear transformations. This involves precomputing the distribution at a small number of points and storing the values which can then be used at run-time to compute the miss ratio. Values of intermediate points can be approximated using linear interpolation. This method provides good approximations with reduced computational cost but a moderate storage overhead. Appendix C shows an example approximation using ∼7 precomputed points per cache configuration.

4.2 Estimating $\boldsymbol{\phi}$(RANDOM)

The RAND replacement algorithm [7] (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 independent 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

$$\begin{aligned} h(RANDOM) &= \boldsymbol{r} \cdot \boldsymbol{\phi}(RANDOM) \\ \boldsymbol{\phi}_k(RANDOM) &= \sum_{i=0}^{\alpha | \boldsymbol{d}_k = \alpha} {}^{\alpha}C_i \cdot \theta^i \cdot (1-\theta)^{(\alpha-i)} \cdot \left(1 - \frac{1}{A'}\right)^i \\ \theta &= 1 - h(RANDOM) \end{aligned} \tag{8}$$

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

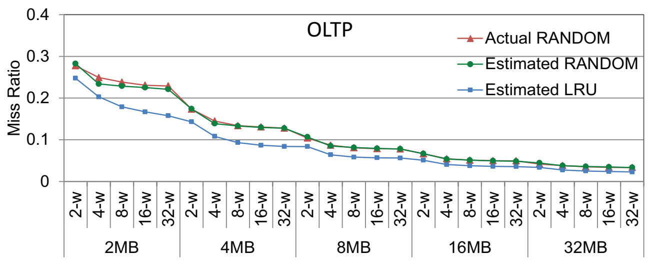


Figure 7: Actual vs estimated miss ratios for oltp with RANDOM replacement policy. LRU estimates are shown as reference. $\boldsymbol{r}(2^{10})$ is first computed from $\boldsymbol{r}(1)$ (Equation 2).

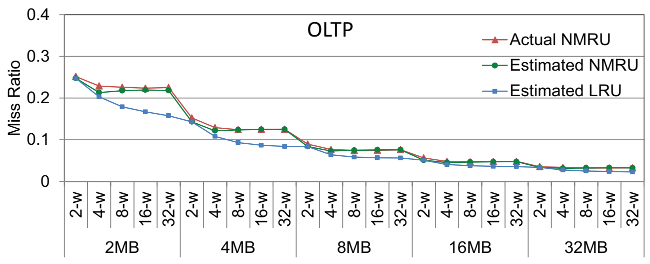


Figure 8: Actual vs estimated miss ratios for oltp with NMRU replacement policy. LRU estimates are shown as reference. $\boldsymbol{r}(2^{10})$ is first computed from $\boldsymbol{r}(1)$ (Equation 2).

$$\begin{aligned}
\boldsymbol{\phi}_k(RANDOM) &= \sum_{i=0}^{\alpha|\boldsymbol{d}_k=\alpha} {}^{\alpha}C_i \cdot \theta^i \cdot (1-\theta)^{(\alpha-i)} \cdot q^i \\
&= \sum_{i=0}^{\infty} {}^{\alpha}C_i \cdot \theta^i \cdot (1-\theta)^{(\alpha-i)} \cdot q^i \\
&\approx \sum_{i=0}^{\infty} e^{-\lambda} \cdot \frac{\lambda^i}{i!} \cdot q^i \\
&= e^{-\lambda(1-q)} \sum_{i=0}^{\infty} e^{-\lambda q} \cdot \frac{(\lambda q)^i}{i!} \\
&= e^{\frac{-\alpha\theta}{A'}} \qquad (9)
\end{aligned}$$

The system of equations in 8 can now be approximated by the following system.

$$\begin{aligned}
h(RANDOM) &= \boldsymbol{r} \cdot \boldsymbol{\phi}(RANDOM) \\
\boldsymbol{\phi}_k(RANDOM) &= e^{\frac{-\boldsymbol{d}_k\theta}{A'}} \\
\theta &= 1 - h(RANDOM) \qquad (10)
\end{aligned}$$

We solve the system of equations in 10 with the initial value $h = \boldsymbol{r}_0$. $\boldsymbol{d}$ is estimated using equation 1. Usually 5 or fewer iterations suffice to reach within 1% of a fix-point. Proof of convergence is presented in Appendix D.

4.2.1 *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.

$$\boldsymbol{\phi}_k = \boldsymbol{\phi}_{k-1} \cdot (1 - \boldsymbol{\phi}_1), \quad k \geq 2 \qquad (11)$$

This approximation is possible 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 ($n = 512$) values of miss ratios for RANDOM with the estimates computed using Equations 3, 10, 11 and 4.

4.3 Estimating $\boldsymbol{\phi}$(NMRU)

The NMRU (or non-MRU) replacement algorithm differentiates the most recently accessed (MRU) line from other lines in the set [41]. 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 10. The modified system is shown below:

$$\begin{aligned}
\boldsymbol{\phi}_1(NMRU) &= 1 \\
h(NMRU) &= \boldsymbol{r} \cdot \boldsymbol{\phi}(NMRU) \\
\boldsymbol{\phi}_k(NMRU) &= e^{\frac{-(\boldsymbol{d}_k - \boldsymbol{d}_1)\theta}{A'-1}} \\
\theta &= 1 - h(NMRU) \qquad (12)
\end{aligned}$$

Figure 8 shows actual vs estimated ($n = 512$) values of miss ratios for NMRU with the estimates computed using Equations 3, 12 and 4.

4.4 Estimating $\boldsymbol{\phi}$(PLRU)

Partitioned LRU [41] (also popularly called pseudo-LRU) maintains a balanced binary tree that, at each level, 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.

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 E). In contrast to RANDOM, not only the number of misses in the reuse interval, but also the pattern of accesses determines eviction probability. It is difficult to estimate $\boldsymbol{\phi}(PLRU)$ by computing probabilities of the regular expressions since the distance to misses within the reuse interval as well as the ways occupied by the intervening elements are 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. 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. (See Appendix E, Figure 13 for visualization.) 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.

4.4.1 *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 tree 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, $\boldsymbol{\phi}_k = \boldsymbol{\phi}_{k-1} \cdot (1 - \boldsymbol{\phi}_3)$.

The case that remains is when $k = 3$. Consider the k^{th} element. There are two subcases here, each with probability $\frac{1}{2}$ assuming that the top node switches evenly between the two subtrees:

1. It maps to the other subtree on either being present there (hit) or being allocated there on a miss. So it cannot affect the reference element. Thus, $\boldsymbol{\phi}_k = \boldsymbol{\phi}_{k-1} = 1$.

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 approximated as $\frac{2}{3} \cdot (1 - \boldsymbol{r}_3)$.

Putting the two subcases together,

$$\boldsymbol{\phi}_k = \begin{cases} 1 & \text{if } 0 \leq k \leq 2 \\ \frac{1}{2} + \frac{(1 - \boldsymbol{r}_3)}{3} & \text{if } k = 3 \\ \boldsymbol{\phi}_{k-1} \cdot (1 - \boldsymbol{\phi}_3) & \text{if } k \geq 4 \end{cases} \qquad (13)$$

4.4.2 *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 $k \geq \frac{A'}{2} + 2$, it is likely that there is 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 $\boldsymbol{\psi}_{2+i}$. 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 $\boldsymbol{\psi}_{1+i}$.

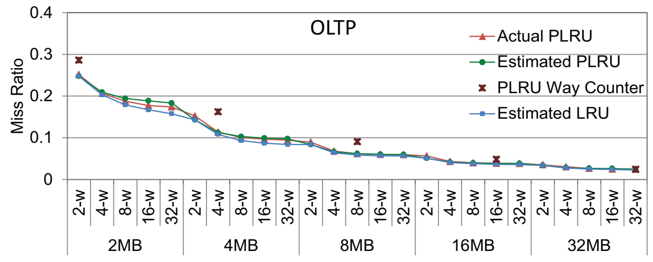


Figure 9: Actual vs estimated miss ratios for oltp with PLRU replacement policy. LRU estimates are shown as reference. $\boldsymbol{r}(2^{10})$ is first computed from $\boldsymbol{r}(1)$ (Equation 2). Section 5.3.1 describes PLRU Way-Counters.

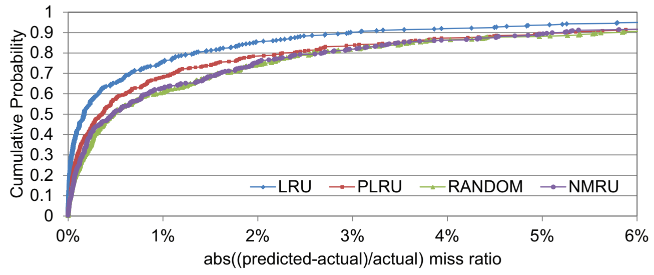


Figure 10: Relative-error density plot combined over all (16) workloads.

Putting everything together,

$$\boldsymbol{\phi}_k = \begin{cases} 1 & \text{if } 0 \leq k \leq L \\ \frac{\boldsymbol{\phi}_{k-1}}{2} + \frac{1}{2} \sum_{i=0}^{k-3} {}^{k-3}C_i \left(\frac{1}{2}\right)^{(k-3)} \cdot \boldsymbol{\psi}_{2+i} & \text{if } k \geq \frac{A'}{2} + 2 \\ \frac{\boldsymbol{\phi}_{k-1}}{2} + \frac{1}{2} \sum_{i=0}^{k-2} {}^{k-2}C_i \left(\frac{1}{2}\right)^{(k-2)} \cdot \boldsymbol{\psi}_{1+i} & \text{otherwise} \end{cases} \qquad (14)$$

Figure 9 shows actual vs estimated ($n = 512$) values of miss ratios for PLRU with the estimates computed using Equations 3, 13, 14 and 4.

4.5 Estimation Accuracy and Compute Time

Figure 10 shows the cumulative distribution of relative errors in miss ratio prediction for all workloads. More than 90% of predictions have relative errors within 6%. Prediction for LRU is the most accurate.

A major contributor to hit ratio computation time is the determination of $\boldsymbol{r}$. Subsection 5.1 proposes low-cost hardware to approximate $\boldsymbol{r}(2^{10}), n = 512$ online. Assuming this is available, the average hit ratio computation time per cache configuration on a Nehalem 2.26 GHz machine were – LRU: 0.020 msec; PLRU: 0.022 msec; RANDOM, NMRU: 0.030 msec. A large fraction of this is due to computation of Equation 3 (Appendix B shows pseudocode) which takes ~0.080 msec, but gets amortized for caches with the same S'.

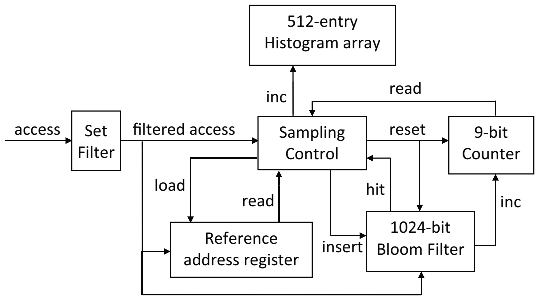


Figure 11: Schematic of new hardware support

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 (counter value == 511)
	      state = `end-sample-b`
	    else {
	      Increment counter
	      Insert addr into Bloom Filter
	      state = `process-sample` }}}

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

Figure 12: Sampling Control for new hardware

5. HARDWARE SUPPORT

Section 3.2 discussed that to avoid expensive computation to determine $\boldsymbol{r}(1)$ or compute $\boldsymbol{r}(2^{10})$ from $\boldsymbol{r}(1)$, we need hardware support to directly estimate $\boldsymbol{r}(2^{10})$. Subsection 5.1 presents our proposed hardware technique to do this.

Subsections 5.2 and 5.3 discuss two traditional hardware mechanisms that help in cache miss ratio estimation – set-counters and way-counters.

5.1 New hardware support to estimate reuse distributions ($\boldsymbol{r}(2^{10}), \mathrm{n} = 512$)

The definition of unique reuse distance (URD) 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 [10], that can construct compact representations of sets. Whereas shadow tags store entire tag addresses, a Bloom filter uses only one bit per hash function to represent each address.

Our proposed hardware, shown in Figure 11, uses a Bloom filter (to summarize RI), a counter to determine $|RI|$, and set-sampling logic. We use a 1024-bit parallel Bloom filter [37] with two H_3 hash functions [12] and a 9-bit counter. The Bloom filter can be at most half-full (512 elements) before being reset. Larger Bloom filters can be used to reduce aliasing errors at the cost of more area/power overhead.

The hardware uses a combination of set sampling and time sampling techniques [24–26, 33, 47]. The set sampling hardware restricts the Bloom filter to addresses matching a subset of the cache index bits. This allows to focus on addresses mapping to one set of a cache with $S = 2^{10}$, and hence track $\boldsymbol{r}(2^{10})$. Once a set-sample is fixed, the hardware randomly chooses an address mapping to the set-sample as the reference address. It then estimates the length of the reuse interval to the next access with the reference address and records it in the histogram. This process is repeated. Figure 12 describes the sampling control state machine.

Assuming 4-byte counters, the size of the histogram array is $512 \times 4 = 2KB$. We model the histogram and Bloom filter arrays in CACTI [39] as direct-mapped SRAM caches (ITRS-LOP, 32nm technology) with 8-byte line sizes. The static power is ∼1.2 mW and the dynamic energy per access is ∼10.8 pJ leading to less than 0.03% of power overhead for the system simulated (Section 6, Table 2). CACTI numbers suggest less than 1 sq mm area overhead for the arrays.

The technique can be generalized to estimate $\boldsymbol{r}(2^x)$ by sizing the Bloom filter, histogram, and set filter appropriately.

5.2 Set-Counters

Set-counters [44] use counters that track the number of accesses per set or a group of sets. However, since they can only track changes in the number of accesses per set *but not changes in per-set locality*, they would be unable to model the behavior shown in Figure 4.

5.3 Way-Counters and Shadow Tags

Way-counters [44] increment a counter associated with each logical stack position (ordered by access recency) on every cache hit. The number of hits for associativity A' is the sum of the counter values from 0 to $A' - 1$.

The above assumes $A' \leq A$ where A is the associativity of the current/predicting cache (32MB 32-way in this study). In applications such as dynamic reconfiguration situations, this is problematic since the cache may need to be sized up, not only sized down. Shadow tags [33] (or auxiliary tag directories [35]) circumvent this difficulty by maintaining a copy of the tags that 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 cache. Qureshi, et al. [34] used dynamic set sampling to reduce storage and power costs of the shadow copy.

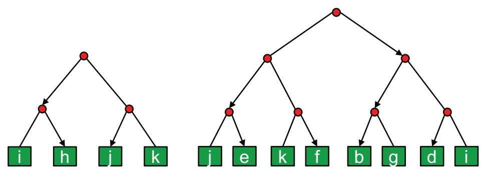
Way-counter values, converted to probabilities, estimate $\boldsymbol{r}(S)$ upto length A. The estimation is exact for LRU caches. Their operation can be understood by deriving Equation 7 from Equation 3 instead of from Equation 2. We get

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

Under the assumption $S' = S$, $h(S') = \sum_{i=0}^{A'-1} \boldsymbol{r}_i(S)$ which is computationally extremely efficient.

5.3.1 Way-counters for PLRU

In PLRU, the MRU line is known with certainty but the rest of the logical ordering is not precisely known. Kedzierski, et al. [23] proposed a heuristic for approximating logical stack positions for PLRU caches to enable way-counter based prediction . Let *waynum* be the way number of the accessed line and *pathbits* denote the bit-values of the tree



(a) **4-way** (b) **8-way**

Figure 13: PLRU trees demonstrating non-inclusion. The 8-way tree does not include element `h` of the 4-way 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 $URD(x, m)$:

$$U\hat{R}D(x, m) = A - 1 - (reverse(waynum) \oplus pathbits)$$

This approach aims to compute $\boldsymbol{r} \cdot \boldsymbol{\phi}(LRU)$ with $\boldsymbol{r}$ approximately measured using the above mechanism. However, apart from the traditional limitations of way-counters (Section 5.3), it also ignores the fact that $\boldsymbol{\phi}(PLRU) \neq \boldsymbol{\phi}(LRU)$ for $A \neq 2$. Interestingly, it fails to accurately estimate the hit ratio even for a 2-way cache where $\boldsymbol{\phi}(PLRU) = \boldsymbol{\phi}(LRU)$ *when the current configuration that does the estimation has* $A \neq 2$ (see, for example, Figure 9 where the current/predicting configuration has A=32). In contrast, our framework overcomes this by *decoupling hit ratio estimation from the organization of the current cache.*

5.3.2 *Way-Counter Limitations*

Way-counters (+shadow tags) have the following fundamental limitations:

Fixed number of sets: The relation ($S' = S$) that makes way-counters efficient also implies the restriction that the number of sets must be fixed. As can be observed from Table 1, miss ratios for only 4 of 24 configurations can be predicted at any time; other predictions must be preceded by (time-consuming) re-training for the changed S'.

However, our framework reveals that Equation 5.3 may be used to transform way-counter values when $S' \geq S$ (also see discussion for Case 1 in Section 3). With reference to Table 1, maintaining shadow tags corresponding to $S = 2^{10}$ allows conversion of values for any $S' \neq S$. But, Figure 5 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 $n(> A)$ positions.

Replacement policies with stack inclusion: Way-counters exploit the stack inclusion property [32] of LRU to predict miss ratios $\forall A' \leq A$. For replacement policies that do not guarantee stack inclusion (PLRU/RANDOM/NMRU), this is no longer true.

For example, consider the access sequence: `a b c d e d f e g h f i j i k` simultaneously to an 8-way PLRU set and a 4-way PLRU set. Figure 13 shows the two sets and associated PLRU trees after the sequence. The arrows in the figures point to the less recently used subtree. Initially, both sets were empty and the eviction bits in each tree were pointing to the "left" subtrees. At the end of the sequence, the two sets together contain 9 distinct elements (`i h j k e f b g d`) whereas a policy satisfying inclusion would have exactly 8 elements. Thus, maintaining information for 8 ways is not sufficient to accurately predict miss ratios for both a 4-way and an 8-way cache *even if* $S' = S$.

Tight coupling with replacement policy implementation: Since way-counters 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 well using reuse information. Way-counters depend on the replacement policy mimicking stack operation, so they run into trouble when the stack is absent (PLRU/RANDOM/NMRU) (see Section 5.3.1 for a discussion on PLRU) or reconfigured ($S' \neq S$).

Shadow Tag overhead: For very large caches, tag area and power are significant. Loh, et al. [30] propose novel tag management schemes for such caches. Maintaining additional shadow tags in those systems seem difficult.

6. MINIMUM EDP CONFIGURATION

This section describes an application of our technique in finding the LLC configuration that results in the minimum full-system EDP.

Table 2 describes the 8-core CMP we use in this study. We assume an 8-banked L3 cache that is dynamically reconfigurable for a total of 25 configurations (see Table 1). The access latency is conservatively assumed to be constant for all configurations. We use 2 copies of our sampling hardware (Section 5.1), but a common histogram array. To evaluate power and performance, we perform *full-system* simulation using GEMS [31] augmented with a detailed timing and power model. We use CACTI 5.3 [39] to determine the static power and dynamic activation energy per component.

We use 7 SPEComp [6] benchmarks (**`ammp`**, **`equake`**, **`fma3d`**, **`gafort`**, **`mgrid`**, **`swim`**, **`wupwise`**) with "ref" inputs, 5 PARSEC [9] benchmarks (**`blackscholes`**, **`bodytrack`**, **`fluidanimate`**, **`freqmine`**, **`swaptions`**) with "simlarge" inputs, and 4 Wisconsin commercial workloads [2] (**`apache`**, **`jbb`**, **`oltp`**, **`zeus`**). Each workload uses 8 threads and runs for a fixed amount of work (e.g. #transactions or loop iterations [3]) that corresponds to ~500M instructions per workload. Each simulation run starts from a mid-execution checkpoint that includes cache warmup.

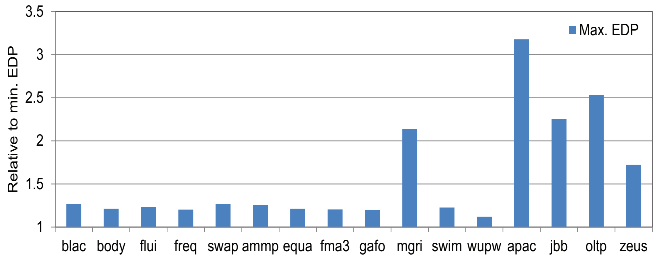
Figure 14a shows, for each workload, the maximum EDP relative to the minimum EDP for that workload resulting from the different LLC configurations. The chart below the figure lists the LLC configurations that resulted in the extremum points. There exists a significant range of system EDPs, particularly for the commercial workloads; `apache` for example has max EDP more than 3.2x times the min EDP; inaccurate predictions can thus cause the system to transition to a severely suboptimal state. The chart also shows that for 11 of 16 workloads, the min EDP configuration has a cache with a different number of sets than that of the current/predicting cache.

For this experiment, the configuration using a 32MB 32-way LLC performs miss ratio and min-EDP predictions. Figure 14b shows EDP of the configuration identified as the minimum one, relative to the actual minimum EDP, using both reuse-models and PLRU way-counters for predicting LLC miss rates, assuming that the workloads were rerun with the identified configuration. This also assumes simple activity counters to track energy consumption in cores, caches, memory, and performance counters to enable online linear regression to approximate the relation between LLC misses and cycles-per-instruction.

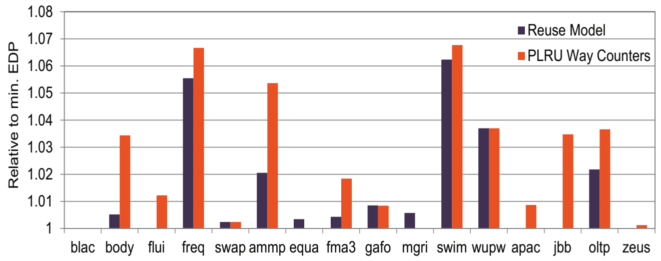
For this evaluation, we use the average sampled reuse distance distribution obtained over all 2^{10} sets as a proxy for the

Core configuration		4-wide out-of-order, 128-entry window, 32-entry scheduler					
Functional Units		4 integer, 2 floating-point, 2 mem units					
Branch Prediction		YAGS 4K PHT 2K Exception Table, 2KB BTB, 16-entry RAS					
Disambiguation		NoSQ 1024-entry predictor, 1024-entry double-buffered SSBF					
Fetch		32-entry buffer, Min. 7 cycles fetch-dispatch time					
Inclusive	**L1I Cache**	private 32KB 4-way per core, 2 cycle hit latency, ITRS-HP					
	L1D Cache	private 32KB 4-way per core, 2 cycle hit latency, ITRS-HP					
	L2 Cache	private 256KB 8-way per core, 6 cycle access latency, PLRU, ITRS-LOP					
	L3 Cache	shared, configurable 2–32 MB 2–32 way, 8 banks, 14 cycle access latency, PLRU, ITRS-LOP, serial					
Coherence protocol		MESI (Modified, Exclusive, Shared, Invalid), directory					
On-Chip Interconnect		2D Mesh, 16B bidirectional links					
Number of cores	8	**On-chip frequency**	2132 MHz	**Technology generation**	32nm	**Temperature**	340K
Main Memory		4GB DDR3-1066, 75ns zero-load off-chip latency, 2 memory controllers, closed page, pre-stdby					

Table 2: System configuration



(a) Maximum EDP over 25 possible LLC configurations.



(b) EDP of predicted minimum configurations.

workload	**blac**	**body**	**flui**	**freq**	**swap**	**ammp**	**equa**	**fma3**	**gafo**	**mgri**	**swim**	**wupw**	**apac**	**jbb**	**oltp**	**zeus**
min. config	2M-2w*	4M-8w	2M-4w	4M-8w	2M-16w	4M-8w	2M-2w*	4M-4w*	4M-4w*	32M-32w*	4M-32w	8M-32w	32M-16w	32M-8w	16M-8w	32M-8w
max. config	32M-32w*	32M-32w*	32M-32w*	32M-2w	32M-32w*	32M-16w	32M-4w	32M-4w	32M-32w*	4M-16w	2M-32w	32M-32w*	2M-2w*	2M-2w*	2M-2w*	2M-2w*
Reuse config	2M-2w*	2M-4w	2M-4w	2M-8w	2M-2w*	2M-8w	2M-4w	2M-8w	2M-8w	32M-16w	4M-16w	16M-16w*	32M-16w	32M-8w	16M-16w*	32M-8w
Way-ctr config	2M-2w*	2M-2w*	2M-2w*	2M-2w*	2M-2w*	2M-2w*	2M-2w*	2M-2w*	2M-2w*	32M-32w*	4M-4w*	16M-16w*	32M-32w*	32M-32w*	32M-32w*	32M-32w*

Figure 14: Maximum EDP and that of model-predicted configurations, all relative to the minimum EDP for the same workload, and with PLRU replacement policy. In the above table, xM-yw denotes an x MB, y-way associative LLC. Caches with the same number of sets (2^{14}) as the current/predicting configuration (32MB 32-way) are marked with a *.

sampled distribution. A practical implementation would additionally experience inter-sample variation, but we expect it to be small for long runs. For way-counters, we assume that full shadow tags are present for comparison purposes. In practice, shadow tags would also be set-sampled [34].

Both reuse-models and way-counters predict configurations that are within 7% of the minimum, but reuse models outperform or are the same as way-counters on all except 3 workloads (`equake`, `gafort`, `mgrid`), both due to their ability to predict for a different number of sets and being generally more accurate. The chart below Figure 14b lists the configurations chosen by both models. Workloads such as `apache` require large caches whereas others such as `equake` should be allocated small caches for energy-efficient performance.

7. RELATED WORK

Characterizing cache behavior using stack distances (also known as reuse distances or gap) is well known [8, 13–15, 21, 29, 38, 40, 49]. 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 address trace. Computation to determine the distribution can be reduced with efficient algorithms [5] or by approximate analysis [49]. Shi, et al. [38] perform single-pass stack simulation to project cache performance and to study the impact of data replication for various L2 cache configurations. Online determination of the stack distance distribution cannot directly apply techniques from offline methods due to constraints on computational state and complexity.

Mixed algorithms deal with efficient online trace collection and offline processing. Tam, et al. [46] use hardware mechanisms for address sampling and post-processing software for computing stack distance distributions. Since distribution estimation and hit ratio 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. 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. [45], Qureshi, et al. [35] propose mechanisms for partitioning of shared caches (L2) among competing processes using way-counters. Suh, et al. [44] also proposes set-counters in LRU order, with each counter tracking accesses to a group of sets. We discuss set and way-counters in Sections 5.2 and 5.3 respectively. Gordon-Ross, et al. [20] use a hardware TCAM to track stack distances for LRU miss-ratio predictions.

Cypher proposes methods [13, 14] 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 are difficult, require additional logic and could be subject to approximations depending on available hardware state. In contrast, our scheme uses set sampling that does not require complex filter logic, and Bloom filters for compact representation of sets.

The Binomial model has been successfully used to analyze cache behavior for other applications. Stone, et al. [42], Falsafi, et al. [17] 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 [28].

Reineke, et al. [36] prove relations on best and worst-case bounds of cache performance for several replacement policies. Our work, in contrast, studies average case behavior.

8. CONCLUSIONS

The central theme of this paper is an online modeling framework, new analytical models, and efficient hardware support, to predict cache performance at runtime for a range of 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 a new predictor 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. Extending the models to other replacement policies is the focus of our ongoing work.

9. ACKNOWLEDGMENTS

This work is supported in part by the National Science Foundation (CNS-0916725, CCF-1017650, CNS-1117280, and CCF-1218323), Sandia/DOE (#MSN123960/DOE890426), and a University of Wisconsin Vilas award. The views expressed herein are not necessarily those of the NSF, Sandia or DOE. Wood has a significant financial interest in AMD. This work was inspired by Bob Cypher and improved by feedback from the anonymous reviewers and others.

10. REFERENCES

[1] A. Agarwal, J. Hennessy, and M. Horowitz. An analytical cache model. *ACM Transactions on Computer Systems*, 7(2):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] A. R. Alameldeen and D. A. Wood. IPC considered harmful for multiprocessor workloads. *IEEE Micro*, 26(4):8–17, Jul/Aug 2006.

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

[5] G. Almási, C. Caşcaval, and D. A. Padua. Calculating stack distances efficiently. In *Proceedings of the 2002 workshop on Memory system performance*, pages 37–43, June 2002.

[6] V. Aslot, M. Domeika, R. Eigenmann, G. Gaertner, W. Jones, and B. Parady. SPEComp: A new benchmark suite for measuring parallel computer performance. In *Workshop on OpenMP Applications and Tools*, pages 1–10, July 2001.

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

[8] 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.

[9] C. Bienia. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, Jan. 2011.

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

[11] A. Boneh and M. Hofri. The coupon-collector problem revisited – a survey of engineering problems and computational methods. *Communications in Statistics. Stochastic Models*, 13(1):39–66, 1997.

[12] 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, May 1977.

[13] 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.

[14] 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.

[15] C. Ding and Y. Zhong. Reuse distance analysis. Technical Report UR-CS-TR-741, University of Rochester, Feb. 2001.

[16] 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 *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, pages 141–152, Sept. 2002.

[17] 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.

[18] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy caches: simple techniques for reducing leakage power. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*, pages 148–157, May 2002.

[19] R. Gonzalez and M. Horowitz. Energy dissipation in general purpose microprocessors. In *IEEE Journal of Solid-State Circuits*, pages 1277–1284, Sept. 1996.

[20] A. Gordon-Ross, P. Viana, F. Vahid, W. Najjar, and E. Barros. A one-shot configurable-cache tuner for

improved energy and performance. In *Proceedings of the conference on Design, automation and test in Europe*, pages 755–760, Apr. 2007.

[21] M. D. Hill and A. J. Smith. Evaluating associativity in CPU caches. *IEEE Transactions on Computers*, 38(12):1612–1630, Dec. 1989.

[22] S. Jahagirdar, V. George, I. Sodhi, and R. Wells. Power management of the third generation Intel Core micro architecture formerly codenamed Ivy Bridge. In *Hot Chips 24*, Aug. 2012.

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

[24] 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.

[25] S. Laha. *Accurate low-cost methods for performance evaluation of cache memory systems*. PhD thesis, University of Illinois, Dept. of Computer Science, 1988.

[26] 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.

[27] 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.

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

[29] 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.

[30] G. H. Loh and M. D. Hill. Efficiently enabling conventional block sizes for very large die-stacked DRAM caches. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 454–464, Dec. 2011.

[31] 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.

[32] 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.

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

[34] M. K. Qureshi, D. N. Lynch, O. Mutlu, and Y. N. Patt. A case for MLP-aware cache replacement. In *Proceedings of the 33rd Annual International Symposium on Computer Architecture*, pages 167–178, June 2006.

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

[36] 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*, pages 51–60, June 2008.

[37] D. Sanchez, L. Yen, M. D. Hill, and K. Sankaralingam. Implementing signatures for transactional memory. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 123–133, Dec. 2007.

[38] 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.

[39] T. Shyamkumar, N. Muralimanohar, J. H. Ahn, and N. P. Jouppi. CACTI 5.1. Technical Report HPL-2008-20, Hewlett Packard Labs, 2008.

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

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

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

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

[44] 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.

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

[46] D. K. Tam, R. Azimi, L. B. Soares, and M. Stumm. RapidMRC: approximating L2 miss rate curves on commodity systems for online optimizations. In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 121–132, Mar. 2009.

[47] 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.

[48] C. Zhang, F. Vahid, and W. Najjar. A highly configurable cache architecture for embedded systems. In *Proceedings of the 30th Annual International Symposium on Computer architecture*, pages 136–146, June 2003.

[49] 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. LOSSY SUMMARIZATION

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.

B. EQUATION 3 PSEUDO-CODE WITH POISSON APPROXIMATION

```
void init() {
    int i;
    for(i=0;i<9;i++)
        precomputed_exp_inc[i]=exp(-1.0/(1<<i));
    for(i=1;i<64;i++)
        precomputed_v[i]=1.0/i;
}

void compute_per_set_r(int num_set_bits, int max_assoc) {
    const double *ptr=&r_histogram[0];
    double s3=precomputed_exp_inc[num_set_bits];
    double s2=1.0;
    double base_lambda=1.0/(1<<num_set_bits);
    double lambda=0;
    int i, rd;
    for(i=0;i<512;i++) {
        double s1=s2;
        for(rd=0;rd<2*max_assoc;rd++) {
            per_set_r[num_set_bits][rd]+=ptr[i]*s1;
            s1*=lambda*precomputed_v[rd+1];
        }
        s2*=s3;
        lambda+=base_lambda;
    }
}
```

`compute_per_set_r` computes Equation 3, using Poisson approximation to Binomial, assuming $\boldsymbol{r}(2^{10})$ upto $n = 512$ is available. $num\_set\_bits \in [1, 8] = log_2(\frac{S'}{S})$. The computation is done for $2A'$ terms (Section 4). Time taken depends on A': 0.006 msec for $A' = 2$ to 0.080 msec for $A' = 32$. Time is measured using the `gettimeofday()` library function and does not include time taken by `init()`.

C. PIECEWISE LINEAR APPROXIMATION FOR LRU

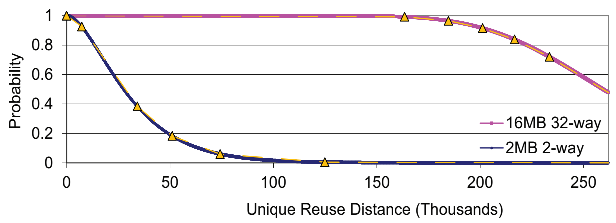


Figure 15: cumulative Poisson and piece-wise linear approximations for two cache configurations.

Figure 15 shows the cumulative Poisson transformer (Cypher's approach [13,14]) for two very different cache organizations. The triangles and the dashed lines show the selected points for a piece-wise linear approximation that uses 7 or fewer points per transformer (with a common point (0,1)).

D. CONVERGENCE FOR RANDOM

First note that if a fix-point exists, the solution satisfies the general conditions of $\boldsymbol{\phi}$ (Equation 5). This is because $\boldsymbol{d}_0 = 0$ (Equation 1) and from Equation 10,

$$\begin{aligned} \frac{\boldsymbol{\phi}_k}{\boldsymbol{\phi}_{k-1}} &= e^{\frac{-(\boldsymbol{d}_k - \boldsymbol{d}_{k-1})\theta}{A'}} = e^{-\left(\frac{\theta/A'}{\sum_{i=k}^{\infty} \boldsymbol{r}_i(T)}\right)} \\ &\leq 1 \end{aligned} \quad (15)$$

Let H denote a fix-point and h^0, h^1, h^2, ... denote successive approximations. By re-arranging the system of equations in 10 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 $h^j \geq h^{j-1} \implies h^{j+1} \geq h^j$. 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 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 $\boldsymbol{r}_0 = h^0 \leq h^1 \leq h^2 ... \leq H$.

E. 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 have accesses that match the particular regular expression described below.

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

F. MODEL ESTIMATES

Figure 16 show actual vs estimated miss ratios for the other commercial workloads: `apache`, `jbb` and `zeus`. For these workloads, $h(LRU)$ can well-approximate $h(PLRU)$ but not $h(RANDOM)$ or $h(NMRU)$. Figure 10 (Section 4.5) shows consolidated prediction errors for all workloads.

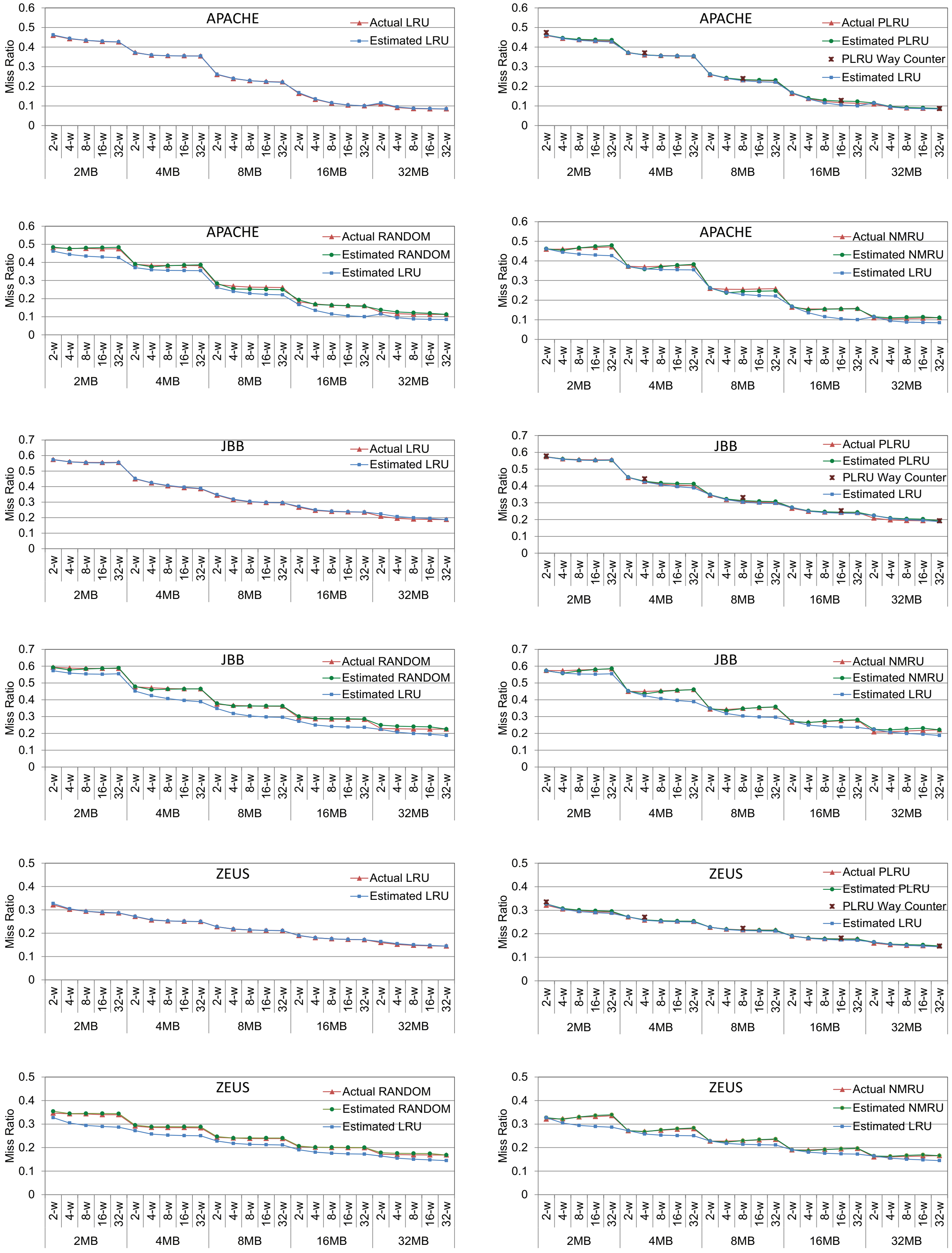


Figure 16: Actual vs estimated miss ratios for apache, jbb, zeus. $\boldsymbol{r}(2^{10})$ is first computed from $\boldsymbol{r}(1)$ (Equation 2). Section 5.3.1 describes PLRU Way-Counters.