

# Relative Competitive Analysis of Cache Replacement Policies \*

Jan Reineke Daniel Grund

Saarland University  
Saarbrücken, Germany  
{reineke, grund}@cs.uni-sb.de

## Abstract

Caches are commonly employed to hide the latency gap between memory and the CPU by exploiting locality in memory accesses. On today's architectures a cache miss may cost several hundred CPU cycles.

In order to fulfill stringent performance requirements, caches are now also used in hard real-time systems. In such systems, upper and sometimes also lower bounds on the execution times of a task have to be computed. To obtain tight bounds, timing analyses *must* take into account the cache architecture. However, developing cache analyses – analyses that determine whether a memory access is a hit or a miss – is a difficult problem for some cache architectures.

In this paper, we present a tool to automatically compute relative competitive ratios for a large class of replacement policies, including LRU, FIFO, and PLRU. Relative competitive ratios bound the performance of one policy relative to the performance of another policy.

These performance relations allow us to use cache-performance predictions for one policy to compute predictions for another, including policies that could previously not be dealt with.

**Categories and Subject Descriptors** B.3.3 [Memory Structures]: Performance Analysis and Design Aids—Worst-case analysis; I.6.5 [Simulation and Modeling]: Model Development—Modeling Methodologies

**General Terms** Performance, Design, Algorithms, Theory

**Keywords** Cache performance, replacement policy, worst-case execution time, WCET analysis, predictability

## 1. Introduction

Caches are commonly employed to hide the latency gap between memory and the CPU by exploiting locality in memory accesses.

\* A two-page extended abstract of this work appears in SIGMETRICS 2008. This work has profited from discussions within the ARTIST2 Network of Excellence. It is supported by the German Research Foundation (DFG) as part of SFB/TR AVACS and the German-Israeli Foundation (GIF). The second author is supported by a scholarship in the DFG GK 623.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

LCIES'08, June 12–13, 2008, Tucson, Arizona, USA.  
Copyright © 2008 ACM 978-1-60558-104-0/08/06...\$5.00

On today's architectures a cache miss may take several hundred CPU cycles. Future architectures are expected to exhibit even larger cache miss penalties. Therefore, the cache performance has a strong and increasing influence on a system's overall performance.

In order to fulfill stringent timing requirements, caches are now also used in hard real-time systems. In such systems, guarantees have to be made concerning the best- and worst-case execution times (BCET and WCET) of tasks. To obtain tight bounds on the execution times of a task, timing analyses *must* take into account the cache architecture. However, developing cache analyses – analyses that statically determine whether a memory access associated with an instruction will always be a hit or a miss – is a difficult problem. Precise and efficient analyses have been developed for set-associative caches that employ the least-recently-used (LRU) replacement policy (Ferdinand et al. 1997; Ferdinand and Wilhelm 1999; White et al. 1997; Ghosh et al. 1998; Chatterjee et al. 2001). Other commonly used policies, like first-in-first-out (FIFO) or Pseudo-LRU (PLRU), are more difficult to analyze (Reineke et al. 2007). We are not aware of any published analysis that may safely predict cache misses in the presence of FIFO or PLRU replacement.

Relative competitive analyses yield upper (lower) bounds on the number of misses (hits) of a policy  $P$  relative to the number of misses (hits) of another policy  $Q$ . For example, a competitive analysis may find out that policy  $P$  will incur at most 30% more misses than policy  $Q$  and at most 20% less hits in the execution of any task. Note that  $P$  and  $Q$  may have different associativities.

We propose the following approach to determine safe bounds on the number of cache hits and misses by a task  $T$  under FIFO( $k$ ), PLRU( $l$ )<sup>1</sup>, or any another replacement policy:

1. Determine competitiveness of the desired policy  $P$  relative to a policy  $Q$  for which a cache analysis exists, like LRU.
2. Perform cache analysis of task  $T$  for policy  $Q$  to obtain a cache-performance prediction, i.e. upper (lower) bounds on the number of misses (hits) by  $Q$ .
3. Calculate upper (lower) bounds on the number of misses (hits) for  $P$  using the cache analysis results for  $Q$  and the competitiveness results of  $P$  relative to  $Q$ .

Note that step 1 has to be performed only once for each pair of replacement policies.

The approach is safe due to the monotonicity of the performance relations: They preserve upper (lower) bounds on the number of misses (hits).

A limitation of this approach is that it only produces upper (lower) bounds on the number of misses (hits) for the whole pro-

<sup>1</sup>  $k$  and  $l$  denote the respective associativities of FIFO( $k$ ) and PLRU( $l$ ).

gram execution. It does not reveal at which program points the misses (hits) will happen, something many timing analyses need. We will demonstrate that relative competitiveness results can also be used to obtain sound *may* and *must* cache analyses (Ferdinand and Wilhelm 1999), i.e. analyses that can classify individual accesses as hits or misses.

In this paper, we present a tool to automatically compute relative competitiveness results for a large class of replacement policies, including LRU, FIFO, and PLRU. We generalize some of the automatically computed results, which hold for fixed associativities, to arbitrary associativities. This is aided by the ability of our tool to generate example memory access sequences that exhibit the worst-case relative behavior. One of our results is that for any associativity  $k$  and any workload, FIFO( $k$ ) shows at least half the number of hits that LRU( $k$ ) would show.

## 1.1 Outline

The following section reviews some basics about caches. In Section 3 we formally introduce our notion of relative competitiveness and show how to use it to obtain cache-performance predictions. In Section 4, we describe how to compute competitive ratios automatically. Section 5 presents results obtained with our tool and a number of generalizations to arbitrary associativities. We delineate our work from previous work on competitive analysis and cache analysis in Section 6. Important consequences of our results and possibilities of future work are summarized in Section 7.

## 2. Caches

Caches are very fast but small memories that store a subset of the main memory’s contents to bridge the latency gap between main memory and the processor.

To reduce traffic and management overhead, the main memory is logically partitioned into a set of *memory blocks*  $B$  of size  $b$  bytes. Memory blocks are cached as a whole in cache lines of equal size. Usually,  $b$  is a power of two. This way the block number is determined by the most significant bits of a memory address.

When accessing a memory block one has to determine whether the memory block is stored in the cache (cache hit) or not (cache miss). To enable an efficient look-up, each memory block can be stored in a small number of cache lines only. For this purpose, caches are partitioned into equally-sized cache sets. The size of a cache set is called the *associativity*  $k$  of the cache. The number of such equally sized cache sets  $s$  is usually a power of two, such that the set number is determined by the least significant bits of the block number, the *index*. The remaining bits, known as the *tag* are stored along with the data to finally decide, whether and where a memory block is cached within a set.

Since the number of memory blocks that map to a set is usually far greater than the associativity of the cache, a so-called *replacement policy* must decide which memory block to replace upon a cache miss. To facilitate useful replacement decisions a number of status bits is maintained that store information about previous accesses. We only consider replacement policies that have independent status bits per cache set. Almost all known policies comply with this.

Let us briefly explain the three commonly used families of replacement policies under investigation in the course of the paper:

LRU (least-recently-used) replacement conceptually maintains a queue of length  $k$  for each cache set, where  $k$  is the associativity of the cache. If an element (a memory block) is accessed that is not in the cache (a miss), it is placed at the front of the queue. The last element of the queue is then removed if the set is full. It is the least-recently-used (LRU) element of those in the queue. At a cache hit, the element is moved from its position

in the queue to the front, in this respect treating hits and misses equally. LRU is used in the INTEL PENTIUM I and the MIPS 24K/34K.

FIFO (first-in-first-out) cache sets can also be seen as queues: new elements are inserted at the front evicting elements at the end of the queue. In contrast to LRU, hits do not change the queue. FIFO is used in the INTEL XSCALE, ARM9, ARM11.

PLRU (Pseudo-LRU) is a tree-based approximation of the LRU policy. It arranges the cache lines at the leaves of a tree with  $k - 1$  “tree bits” pointing to the line to be replaced next. For an in detail explanation of PLRU consider (Reineke et al. 2007; Al-Zoubi et al. 2004). PLRU is used in the POWERPC 75X and the INTEL PENTIUM II-IV.

## 3. Relative Competitiveness

In this section, we formally define our notion of relative competitiveness and show how to use it to obtain cache-performance predictions. For the following definitions consider the domains and notations introduced in Figure 1. The most important notions are  $m_P(q, s)$  and  $h_P(q, s)$ , which compute the number of misses and hits, respectively, of policy  $P$  starting in state  $q$  processing access sequence  $s$ .  $update_P(q, s)$  computes the cache-set state after accessing a sequence  $s$  in state  $q$  under policy  $P$ .  $C^P$  is the set of reachable cache-set states of policy  $P$  and  $S$  denotes the set of finite access sequences.

Before giving the central definitions we first need to introduce compatible cache-set states. We want to relate two policies in compatible state only. Intuitively, this ensures that no policy is given an undue advantage by the state it is starting in.

**DEFINITION 1 (Compatible States).** *Two cache-set states  $p \in C^P$  and  $q \in C^Q$  are called compatible, denoted  $p \sim q$ , iff there is some access sequence  $s \in S$ , s.t.  $p = update_P(i^P, s)$  and  $q = update_Q(i^Q, s)$ .*

Now we are ready to define our notion of relative competitiveness. It captures the worst-case performance of one policy relative to another policy:

**DEFINITION 2 (Relative Miss-Competitiveness).** *A policy  $P$  is  $k$ -miss-competitive relative to policy  $Q$  with additive constant  $c$ , if*

$$m_P(p, s) \leq k \cdot m_Q(q, s) + c$$

for all access sequences  $s \in S$  and compatible cache-set states  $p \in C^P, q \in C^Q$ .

In other words, policy  $P$  will incur at most  $k$  times the number of misses of policy  $Q$  plus a constant  $c$  on any access sequence. Observe that this relation holds for individual cache sets *not* for entire set-associative caches consisting of several cache sets. In the original definition of competitiveness (Sleator and Tarjan 1985), online policies were compared to the optimal offline policy MIN, i.e. instantiating  $Q$  in our definition with MIN yields the definition of (Sleator and Tarjan 1985) except for the treatment of compatible states.

We can define relative hit-competitiveness analogously to relative miss-competitiveness. The definition has no counterpart in the original competitiveness definition of (Sleator and Tarjan 1985). This is because no online policy is hit-competitive relative to the optimal offline policy MIN<sup>2</sup>. However, in our setting of comparing two online policies this is possible.

<sup>2</sup>For every online policy  $P$  one can incrementally construct an arbitrarily long access sequence that incurs no hits under  $P$ : Always access the element that was just evicted by the online policy. Due to its knowledge of future accesses, many of these accesses will be hits in MIN.

$a, b, c \in B$	the set of memory blocks
$\langle b, c, d \rangle, s \in S = B^*$	the set of finite access sequences
$P, Q \in Policy$	the class of replacement policies
$[b, e, c, f]_P, i^P, p, q \in C^P$	the set of reachable cache-set states of policy $P$ with $i^P$ the initial state of $P$ after starting up the hardware
$update_P(q, s) : C^P \times S \rightarrow C^P$	function computing the cache-set state $q$ after accessing a sequence $s$ under policy $P$
$m_P(q, s) : C^P \times S \rightarrow \mathbb{N}$	the number of misses incurred by policy $P$ on a given access sequence $s$ and state $q$ .
$h_P(q, s) : C^P \times S \rightarrow \mathbb{N}$	the number of hits incurred by policy $P$ on a given access sequence $s$ and state $q$ .

Figure 1. Domains and Notations

DEFINITION 3 (Relative Hit-Competitiveness).

A policy  $P$  is  $k$ -hit-competitive relative to policy  $Q$  with subtractive constant  $c$ , if

$$h_P(p, s) \geq k \cdot h_Q(q, s) - c$$

for all access sequences  $s \in S$  and compatible states  $p \in C^P, q \in C^Q$ .

Notice, that the two definitions are not redundant. If policy  $A$  is  $k$ -miss-competitive relative to policy  $B$ , with  $k > 1$ , this does not give us any clue regarding the hit-competitiveness of  $A$  relative to  $B$ : Rewriting Definition 2 in terms of hits ( $h_P(q, s) = |s| - m_P(q, s)$ ) yields  $|s| - h_P(q, s) \leq k \cdot (|s| - h_Q(q', s)) + c$ . For  $k > 1$  this inequality depends on  $|s|$ , the length of the access sequence.

We sometimes say that a policy is competitive relative to another policy without specifying an appropriate additive (subtractive) constant. In such cases, we implicitly demand that such a constant exists. The following definition is an example of such a case:

DEFINITION 4 (Competitive Ratio). The competitive miss and hit ratios  $c_{P,Q}^m$  and  $c_{P,Q}^h$  of  $P$  relative to  $Q$  are defined as

$$c_{P,Q}^m = \inf \{k \mid P \text{ is } k\text{-miss-competitive relative to } Q\}$$

$$\text{and } c_{P,Q}^h = \sup \{k \mid P \text{ is } k\text{-hit-competitive relative to } Q\}.$$

In our cases, there is always a smallest (greatest)  $k$ , such that  $P$  is  $k$ -miss-competitive ( $k$ -hit-competitive) relative to  $Q$ , which is not the case in general. Our focus will be on computing competitive ratios and appropriate additive (subtractive) constants. Why are we interested in competitive ratios? Consider a policy  $A$  that is  $k$ -miss-competitive relative to policy  $B$ .  $A$  is also  $l$ -miss-competitive relative to  $B$  for  $l > k$ . However, the former statement is clearly a better characterization of the policy's relative competitiveness. In this sense, the competitive ratio is the *best* characterization of the policy's relative competitiveness. In particular, there are access sequences, such that the ratio between the number of misses (hits) in policy  $A$  and the number of misses (hits) in policy  $B$  approaches the competitive ratio in the limit.

Every policy is by definition 0-hit-competitive relative to every other policy. However, a policy may not be  $k$ -miss-competitive relative to another policy for any  $k$ . In that case, we will call it  $\infty$ -miss-competitive. For a policy  $A$  that is  $\infty$ -miss-competitive relative to policy  $B$ , the number of misses incurred in  $A$  cannot be bounded by the number of misses in  $B$ .

As noted before the notions of relative competitiveness are defined for individual cache sets. However, they can easily be lifted to set-associative caches, which can be seen as the composition of a number of cache sets.

### 3.1 Computing Bounds on Cache Performance using Relative Competitiveness Results

Assume policy  $P$  is  $k$ -miss-competitive with additive constant  $c$  relative to policy  $Q$  for which we have a cache analysis, i.e. an analysis that can compute upper bounds on the number of cache misses incurred by a task  $T$ . How can we obtain a sound upper bound on the number of cache misses incurred by  $T$  under  $P$  using the bound computed for  $Q$ ?

To answer this question, we need to dig into what such an upper bound is. Let  $S(T) \subseteq S$  be the set of possible access sequences performed by task  $T$ . Depending on its input  $T$  may exhibit different access sequences. If  $T$ 's input is known  $S(T)$  may be a singleton set. Then  $\widehat{m_Q(T)}$  is an upper bound on the number of misses incurred by  $T$  under  $Q$ , if

$$\max_{s \in S(T)} \max_{q \in C^Q} m_Q(q, s) \leq \widehat{m_Q(T)}.$$

We quantify over all cache-set states  $q \in C^Q$  of the policy because we usually have no knowledge about the cache-set state in which the execution of  $T$  begins<sup>3</sup>. We claim that  $k \cdot \widehat{m_Q(T)} + c$  is an upper bound for  $P$ :

THEOREM 5. Let policy  $P$  be  $k$ -miss-competitive relative to policy  $Q$  with additive constant  $c$ , and let  $\widehat{m_Q(T)}$  be an upper bound on the number of misses incurred by policy  $Q$  on task  $T$ . Then,

$$\max_{s \in S(T)} \max_{p \in C^P} m_P(p, s) \leq k \cdot \widehat{m_Q(T)} + c,$$

i.e.  $k \cdot \widehat{m_Q(T)} + c$  is an upper bound on the number of misses incurred by  $P$  on task  $T$ .

PROOF 1. We have defined every cache-set state in  $C^Q$  to be reachable (see Figure 1), i.e. for every state  $q \in C^Q$  there is a sequence  $s \in S$ , s.t.  $q = update_Q(i^Q, s)$ . Observation (\*): This implies that every state  $q \in C^Q$  is compatible with at least one state  $p \in C^P$ , i.e.  $q \sim p$ . Then,

$$\begin{aligned} & \max_{s \in S(T)} \max_{p \in C^P} m_P(p, s) \\ \stackrel{\text{Def. 2}}{\leq} & \max_{s \in S(T)} \max_{p \in C^P} \max_{q \in C^Q, q \sim p} k \cdot \widehat{m_Q(q, s)} + c \\ = & k \cdot \left( \max_{s \in S(T)} \max_{p \in C^P} \max_{q \in C^Q, q \sim p} m_Q(q, s) \right) + c \\ \stackrel{\text{Observation (*)}}{=} & k \cdot \left( \max_{s \in S(T)} \max_{q \in C^Q} m_Q(q, s) \right) + c \\ \leq & k \cdot \widehat{m_Q(T)} + c \end{aligned}$$

<sup>3</sup>Do not confuse this with the initial state  $i^P$  of the policy. This is the state at startup of the hardware. When execution of  $T$  begins, other tasks have typically already been executed and thus modified the state.

Lower bounds on the number of hits can be computed similarly using hit-competitiveness results.

We have made the assumption that nothing is known about the state of the policy  $T$  is starting in. If we do have such knowledge, we can make use of it: Let  $I^P \subseteq C^P$  be the possible starting states. Then we can restrict the set of starting states for the cache analysis of  $Q$  to the subset  $\{q \in C^Q \mid q \sim p, p \in I^P\}$  of  $C^Q$ .

### 3.2 Obtaining May and Must Analyses using Relative Competitiveness

As mentioned in the introduction, we can also build sound *may* and *must* cache analyses (Ferdinand et al. 1997; Ferdinand and Wilhelm 1999) from competitiveness results. *may* cache analyses determine for each program point a superset of the set of memory blocks that may be in the cache when the control flow reaches this program point. Analogously, *must* cache analyses determine a subset of the set of memory blocks that are in the cache whenever the control flow reaches the program point. Results of a *must* analysis can be used to safely predict cache hits. Cache misses can be predicted using the results of a *may* analysis. To obtain *may* and *must* analyses we need 1-competitiveness:

As noted before, the notions of hit- and miss-competitiveness are in general not redundant. In the special case of 1-competitiveness, however, they are:

**THEOREM 6 (1-Competitiveness).** *Given two policies  $P$  and  $Q$ . The following two statements are equivalent:*

- (i)  $P$  is 1-miss-competitive relative to  $Q$  with additive constant  $c$ .
- (ii)  $P$  is 1-hit-competitive relative to  $Q$  with subtractive constant  $c$ .

**PROOF 2. Trivial:**  $m_P(q, s) \leq 1 \cdot m_Q(q', s) + c \Leftrightarrow |s| - m_P(q, s) \geq |s| - m_Q(q', s) - c \Leftrightarrow h_P(q, s) \geq 1 \cdot h_Q(q', s) - c$ . **QED**

One can obtain sound *may* and *must* analyses from competitiveness results in the special case of 1-competitiveness:

**THEOREM 7 (may and must analyses).** *If policy  $P$  is 1-miss-competitive relative to policy  $Q$  with additive constant 0, then*

- (i) A *must* analysis for  $Q$  is also a sound *must* analysis for  $P$ .
- (ii) A *may* analysis for  $P$  is also a sound *may* analysis for  $Q$ .

**PROOF 3.** For (i): As  $P$  is 1-miss-competitive relative to  $Q$  with additive constant 0, the contents of  $P$  always subsume the contents of  $Q$ . Memory blocks that are definitely in  $Q$  must also be in  $P$ . For (ii): Anything not contained in  $P$  cannot be in  $Q$  either. In other words, a cache miss in  $P$  must also be a cache miss in  $Q$ .

In general, any analysis that safely predicts hits (misses) for policy  $Q$  can be used as an analysis of policy  $P$ , if  $P$  ( $Q$ ) is 1-miss-competitive relative to  $Q$  ( $P$ ) with additive constant 0.

## 4. Computing Competitive Ratios

We have developed a tool that allows us to compute competitive ratios automatically. In the following we will describe our methodology.

The two policies  $P$  and  $Q$  under consideration induce a transition system that captures how the two policies act relative to each other, processing the same memory accesses. States of this transition system are pairs of cache-set states of policy  $P$  and policy  $Q$ . Figure 2 shows a small part of such a transition system. Competitive ratios and appropriate additive (subtractive) constants are

properties of this system. To obtain competitive ratios, we roughly have to determine the maximum (minimum) ratio of misses (hits) in policy  $P$  relative to the number of misses (hits) in policy  $Q$  on any path through it.

The main obstacle is that there are infinitely many cache-set states if one assumes the set of memory blocks to be infinite. Although the set of memory blocks is finite in practice, the state space of the transition system may still be prohibitively large. To overcome this problem we directly construct a finite quotient structure with respect to an equivalence relation on states that preserves competitiveness properties.

### 4.1 Induced Transition System

Let us formally define the transition system induced by a pair of policies  $P$  and  $Q$ .

**DEFINITION 8 (Induced Transition System).** *Two policies  $P$  and  $Q$  induce a transition system  $T_{P,Q} = (S_{P,Q}, R_{P,Q})$ , where*

$$S_{P,Q} = \{(p, q) \mid p \sim q, p \in C^P, q \in C^Q\},$$

*the states, are pairs of compatible cache-set states of policies  $P$  and  $Q$ .*

$$R_{P,Q} = \{((p, q), (m_p, m_q), (p', q')) \mid (p, q) \in S_{P,Q}, a \in B, (p', q') = \text{update}_{P,Q}((p, q), \langle a \rangle), (m_p, m_q) = m_{P,Q}((p, q), \langle a \rangle)\}$$

*is the transition relation, where  $\text{update}_{P,Q}$  and  $m_{P,Q}$  are lifted to pairs from the update and  $m$  functions. Transitions are labelled with the number of misses (0 or 1) incurred by the access in the two cache-set states, respectively.*

Competitiveness values depend on the number of misses (hits) on paths through the transition system:

**DEFINITION 9 (Paths).** *A path through a transition system  $T = (S, R)$ , where  $R \subseteq S \times L \times S$  and  $L$  is a set of labels, is a sequence of labels  $\pi = l_1 \dots l_n \in L^n$ , s.t.*

$$\exists s_1, \dots, s_{n+1} \in S. \forall i \in \{1, \dots, n\}. (s_i, l_i, s_{i+1}) \in R.$$

*The set of all paths of a transition system  $T$  is denoted by  $\Pi(T)$ .*

In our case, labels are pairs  $(m_p, m_q)$ . The definitions of hit- and miss-competitiveness translate directly to properties of paths of the induced transition system. For instance, a policy  $P$  is  $k$ -miss-competitive relative to policy  $Q$  with additive constant  $c$ , if

$$\sum_i \pi(i)_{|1} \leq k \cdot \sum_i \pi(i)_{|2} + c \text{ for all paths } \pi \in \Pi(T_{P,Q}),$$

where  $|1$  and  $|2$  select the first and second component of a tuple, respectively.

### 4.2 Cache-Set states

Up to now we have not made any assumptions about the internal structure of cache-set states. In order to ease the following presentation, we assume cache-set states to be representable as tuples of memory blocks and empty cache lines:

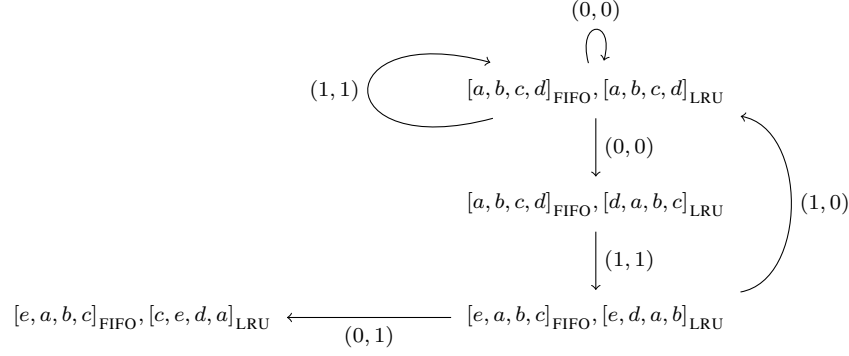
$$C_k = \{1, \dots, k\} \rightarrow B \cup \{\perp\},$$

where  $k$  is the associativity,  $B$  is the set of memory blocks and  $\perp$  represents empty cache lines. Note that we do not constrain the management of the policy in hardware, which may be very different.

Accesses can have two effects on such a states:

- The order of the elements in the tuple is changed, depending *only* on the *position* of the accessed memory block in the tuple. In LRU, for instance, the elements may be ordered from most to least recently used.

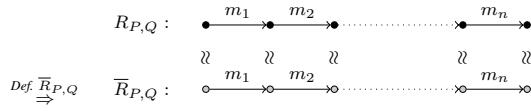




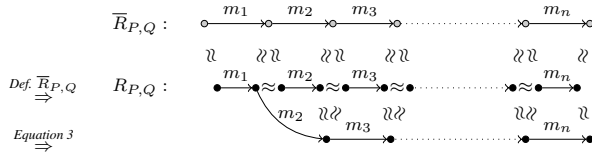
**Figure 4.** Running example revisited. States that are equivalent according to  $\approx$  are connected by dashed lines in Figure 2. “Merging” equivalent states yields the quotient structure depicted here.

PROOF 4. We need to show  $\pi = m_1 \dots m_n \in \Pi(T_{P,Q}) \Leftrightarrow \pi \in \Pi(\bar{T}_{P,Q})$ .

“ $\Rightarrow$ ” follows from the definition of  $\bar{R}_{P,Q}$  in Equation 4: Let  $s_1, \dots, s_{n+1}$  be such that  $(s_i, m_i, s_{i+1}) \in R_{P,Q}, \forall i \in \{1, \dots, n\}$ . Let  $\bar{s}_1, \dots, \bar{s}_{n+1}$  be the representants of  $s_1, \dots, s_{n+1}$  in  $\bar{S}_{P,Q}$ . Then  $(\bar{s}_i, m_i, \bar{s}_{i+1}) \in \bar{R}_{P,Q}, \forall i \in \{1, \dots, n\}$ :



“ $\Leftarrow$ ” is just a little more complicated. For each of the edges  $(\bar{s}_i, m_i, \bar{s}_{i+1})$  in  $\bar{R}_{P,Q}$  there is a corresponding edge in  $R_{P,Q}$  by definition. By Equation 3 we can construct a path through  $R_{P,Q}$  from these edges as illustrated below:



QED

OBSERVATION 11. The set of equivalence classes of  $\approx$  is finite, as it only depends on the relative positions of elements contained in both sets. In particular, the index of  $\approx$  is independent of the number of memory blocks.

Therefore, the quotient transition system  $\bar{T}_{P,Q}$  is finite. Note that we directly construct  $\bar{T}_{P,Q}$ , in particular we never construct the underlying transition system  $T_{P,Q}$ .

#### 4.4 Computation of Competitive Ratios

Once we have built up the quotient transition system, determining the minimal  $k$  such that  $P$  is hit- (miss-) competitive relative to  $Q$  amounts to computing the minimum (maximum) cycle ratio (Lawler 1966; Ahuja et al. 1993).

In the setting of miss-competitiveness, we wish to find a cycle through the quotient transition system that maximizes the ratio of misses in  $P$  relative to the misses in  $Q$ . Maximum ratio problems can easily be converted into minimum ratio problems by changing the sign of the numerator or the denominator.

The minimum cycle ratio problem, also known as the minimum cost-to-time ratio cycle problem, is the following: Given a directed

graph  $G$  with both a cost and a travel time associated with each edge, we wish to find a cycle in the graph with the smallest ratio of its cost to its travel time.

DEFINITION 12 (Minimum Cycle Ratio). The minimum cycle ratio  $\lambda^*$  of  $G$  is

$$\lambda^* = \min_{\text{Cycle } C \in G} \frac{\sum_{\text{Edge } (i,j) \in C} c_{ij}}{\sum_{\text{Edge } (i,j) \in C} \tau_{ij}}$$

where  $c_{ij}$  and  $\tau_{ij}$  are the cost and travel time associated with edge  $(i, j)$ .

(Lawler 1966; Ahuja et al. 1993) describe how to solve the minimum cycle ratio problem by repeated applications of the negative cycle detection algorithm. Their algorithm relies on the following observation: Let  $\lambda^*$  be the minimum cycle ratio of a graph  $G$ , then

$$\min_{\text{Cycle } C \in G} \sum_{\text{Edge } (i,j) \in C} c_{ij} - \lambda^* \tau_{ij} = 0.$$

Based on this observation, one can do a binary search for  $\lambda^*$ . If the graph  $G^\lambda$  with edge lengths  $l_{ij} = c_{ij} - \lambda \tau_{ij}$  contains negative cycles, then  $\lambda^* < \lambda$ . Otherwise, if all cycles have positive length,  $\lambda^* > \lambda$ . As one can bound the numerators and denominators of any cycle ratio in the system<sup>4</sup>, one can terminate the binary search if under these constraints only a single rational number may reside in the computed interval. This way the exact minimum cycle ratio is obtained.

As noted above, we need to compute the maximum cycle ratio of  $\bar{T}_{P,Q}$  to obtain the competitive miss ratio:

THEOREM 13 (Maximum Cycle Ratio). The maximum cycle ratio  $k$  of  $\bar{T}_{P,Q}$  is equal to the competitive miss ratio of  $P$  relative to  $Q$ .

PROOF 5. We need to show that

1.  $P$  is  $k$ -miss-competitive relative to  $Q$  with some additive constant  $c$ .
2.  $P$  is not  $k'$ -miss-competitive relative to  $Q$  with any additive constant  $c'$  for  $k' < k$ .

For 1. we need to show

$$\sum_i \pi(i)_{|1} \leq k \cdot \sum_i \pi(i)_{|2} + c \text{ for all paths } \pi \in \Pi(T_{P,Q}).$$

<sup>4</sup>By the number of states in the transition system.

Any path  $\pi$  can be split into three (possibly empty) parts  $\pi = \pi_0 \pi_1 \pi_2$ , s.t.  $\pi_0$  and  $\pi_2$  correspond to acyclic traversals of the state space  $\overline{S}_{P,Q}$  and  $\pi_1$  corresponds to a cycle in  $\overline{S}_{P,Q}$ . Since  $\overline{S}_{P,Q}$  is finite,  $|\pi_0| < |\overline{S}_{P,Q}|$  and  $|\pi_2| < |\overline{S}_{P,Q}|$ . For the cyclic part  $\pi_1$  we know that  $\sum_i \pi_1(i)_{|1} \leq k \cdot \sum_i \pi_1(i)_{|2}$ . The acyclic paths  $\pi_0$  and  $\pi_2$  can be “covered” by an appropriate constant  $c \leq 2 \cdot |\overline{S}_{P,Q}|$ .

For 2.: Choose a “cyclic” path  $\pi$  that corresponds to the maximal cycle ratio  $k$ . As  $k > k'$ ,  $\sum_i \pi(i)_{|1} = k \cdot \sum_i \pi(i)_{|2} > k' \cdot \sum_i \pi(i)_{|2}$ . By repeating  $\pi$  sufficiently often  $\sum_i \pi^n(i)_{|1} > k' \cdot \sum_i \pi^n(i)_{|2} + c'$  for any additive constant  $c'$ . *QED*

The proof shows that the additive constant  $c$  can only stem from finite acyclic pre- or suffixes of paths. Once the minimum cycle ratio  $k$  has been determined, finding the appropriate additive (subtractive) constant  $c$  reduces to computing the length of the shortest path through  $\overline{T}_{P,Q}$ , where the edge weights  $w$  are chosen as  $c_{ij} - k\tau_{ij}$ . As  $k$  is the minimum cycle ratio, the graph has no negative cycles. Paths with negative weight correspond to situations where  $P$  can do “worse” than suggested by  $k$  relative to  $Q$  for a limited number of steps. As an example, assume  $P$  to be 2-miss-competitive relative to  $Q$ . Then, there must be paths of length one, such that  $P$  incurs a miss and  $Q$  does not. In Figure 5, we illustrate the two steps of our algorithm for our running example and the case of hit-competitiveness. To improve readability of the example, we omit the node labels. In the course of computing competitiveness values, we generate example access sequences and start states that correspond to the minimum cycle ratio  $k$  and the constant  $c$ . This may help to understand results and to identify patterns, that lead to more general analytical theorems.

## 5. Results

Using our tool, we have obtained a vast amount of competitiveness results for LRU, FIFO, and PLRU at associativities ranging from 2 to 8. In this section, we will present the most interesting results. The full set of results is included at the end of the paper.

We also understand our tool as a means to come up with more general conjectures about relative competitiveness of policies. Competitiveness results for several associativities often reveal a pattern that may then be generalized by the user. In the following description we will point out such cases.

### 5.1 Miss-Competitiveness

Figure 6 depicts relative miss-competitiveness results among FIFO, PLRU, and LRU if compared at the same level of associativity. One observation is that LRU( $k$ ) is  $k$ -miss-competitive relative to FIFO( $k$ ) and vice-versa for all of the investigated associativities. The same holds for FIFO( $k$ ) vs PLRU( $k$ ), but not for PLRU( $k$ ) vs FIFO( $k$ ).

By (Sleator and Tarjan 1985), FIFO( $k$ ) and LRU( $k$ ) are  $k$ -miss-competitive relative to the optimal offline policy MIN (also called OPT). This implies at least  $k$ -miss-competitiveness relative to any online algorithm.

PLRU(2) and LRU(2) are 1-miss-competitive relative to each other, as LRU and PLRU coincide for associativity 2. In contrast, PLRU(4) and PLRU(8) are not  $k$ -miss-competitive relative to LRU(4) and LRU(8), respectively, for any  $k$ . This is particularly interesting as it has been suggested in (Hergenhan and Rosentiel 2000) to model a PLRU( $k$ )-cache by an LRU( $k$ )-cache to simplify WCET prediction, as it “does not add a significant error.”

It is also interesting to compare policies of different associativities. We observe the following for PLRU vs LRU:

$k$	2	4	8	16	32
$l$	2	3	4	5	6
PLRU( $k$ ) vs LRU( $l$ )	1, 0	1, 0	1, 0	1, 0	1, 0

Generalizing this, we conjectured that PLRU( $k$ ) is 1-miss-competitive relative to LRU( $1 + \log_2 k$ ).

**THEOREM 14 (PLRU( $k$ ) vs LRU( $1 + \log_2 k$ )).**  
PLRU( $k$ ) is 1-miss-competitive relative to LRU( $1 + \log_2 k$ ).

**PROOF 6.** The property follows directly from a theorem stated in (Reineke et al. 2007). The theorem claims that a PLRU-cache set of associativity  $k$  always contains the  $1 + \log_2 k$  most-recently-used elements. It therefore subsumes LRU-cache sets of that associativity.

By Theorem 6, PLRU( $k$ ) is also 1-hit-competitive relative to LRU( $1 + \log_2 k$ ). Interestingly, PLRU( $k$ ) is not miss-competitive at all relative to LRU( $l$ ), if  $l > 1 + \log_2 k$ . Due to space limitations we omit the proof.

### 5.2 Hit-Competitiveness

The hit-competitiveness results show less symmetry than the miss-competitiveness results. Figure 7 depicts results relating FIFO, LRU, and PLRU at the same associativities. LRU( $k$ ) is not hit-competitive relative to FIFO( $k$ ) for any  $k$ .

For the case of LRU(3) vs FIFO(3) consider the following example access sequence generated by our tool:

$$[a, b, c]_{\text{LRU}}, [c, b, a]_{\text{FIFO}} \xrightarrow{d} [d, a, b], [d, c, b] \\ \xrightarrow{c} [c, d, a], [d, c, b] \xrightarrow{b} [b, c, d], [d, c, b]$$

The sequence  $d, c, b$  incurs 2 hits in the FIFO-part and no hits in the LRU-part. The final state  $[b, c, d], [d, c, b]$  is equivalent to the first state  $[a, b, c], [c, b, a]$  by the  $\approx$ -relation, i.e. we can go through this sequence arbitrarily often by renaming  $d, c, b$  accordingly.

**THEOREM 15 (LRU( $k$ ) relative to FIFO( $k$ )).**  
LRU( $k$ ) is not hit-competitive relative to FIFO( $k$ ).

**PROOF 7.** For the general case of LRU( $k$ ) vs FIFO( $k$ ) we can generalize the example sequence given above: Start in the state  $[a_1, \dots, a_k]_{\text{LRU}}, [a_k, \dots, a_1]_{\text{FIFO}}$ . Then, incur a miss in both sets, resulting in  $[b, a_1, \dots, a_{k-1}]_{\text{LRU}}, [b, a_k, \dots, a_2]_{\text{FIFO}}$ . Accessing  $\langle a_k, \dots, a_2 \rangle$  will incur  $k - 1$  hits in the LRU-part and no hits in the FIFO-part. Furthermore, the resulting state  $[a_2, \dots, a_k, b]_{\text{LRU}}, [b, a_k, \dots, a_2]_{\text{FIFO}}$  is equivalent to the first state according to the  $\approx$ -relation.  $[a_1, \dots, a_k]_{\text{LRU}}$  and  $[a_k, \dots, a_1]_{\text{FIFO}}$  are compatible, as they are reachable from the initial states  $[\perp, \dots, \perp]_{\text{LRU}}, [\perp, \dots, \perp]_{\text{FIFO}}$  by the sequence  $\langle a_1, \dots, a_k, a_k, \dots, a_1 \rangle$ . *QED*

In contrast, FIFO( $k$ ) is  $\frac{1}{2}$ -hit-competitive relative to LRU( $k$ ) for all of the investigated  $k$ . Consider the following theorem and its proof.

**THEOREM 16 (FIFO( $k$ ) relative to LRU( $k$ )).**  
FIFO( $k$ ) is  $\frac{1}{2}$ -hit-competitive relative to LRU( $k$ ) for all  $k$ .

**PROOF 8.** Consider any access sequence  $s$ . We argue that for every access  $a$  in  $s$  that incurs a hit in LRU but a miss in FIFO, the previous access (if there is one) to  $a$  in  $s$  must have been a hit in FIFO. We prove this by contradiction. Assume the previous access to  $a$  was also a miss in FIFO. Immediately after this access  $a$  is in the “last-in” position. For the next access to  $a$  to be a miss in FIFO, there must be at least  $k$  distinct accesses in between. Otherwise, the next access to  $a$  would have incurred a hit.  $k$  distinct accesses also evict  $a$  from LRU. This is in contradiction to the assumption that the access is a hit in LRU.

There can be at most  $k$  situations where an access to  $a$  incurs a hit in LRU and there is no previous access to  $a$  in the sequence, as the set can hold only  $k$  elements. *QED*



is neither competitive relative to  $FIFO(k)$  nor relative to  $LRU(k)$  it is obviously also not competitive relative to  $MIN(k)$ .

In (Aspnes and Waarts 1996), *relative competitiveness* is used with a different meaning than in our paper. Their definition captures that an algorithm  $A$  is  $k$ -competitive relative to the competitiveness of a subroutine  $B$  that it uses. If  $B$  is in turn implemented by an  $l$ -competitive algorithm, the resulting algorithm is  $k \cdot l$ -competitive overall. This allows to perform modular competitive analyses.

In an effort to overcome weaknesses of traditional competitive analysis, (Koutsoupias and Papadimitriou 1994) propose two refinements of competitive analysis. One of the refinements is coined *comparative analysis*. There, two classes of algorithms  $A$  and  $B$  are compared to determine how powerful  $A$  is relative to  $B$ . Choosing  $A$  and  $B$  to be singletons, *comparative analysis* coincides with our notion of *relative competitive analysis*.

(Smaragdakis et al. 2003) introduce *Early Eviction LRU* (EELRU), an adaptive algorithm based on the LRU stack model. They prove *robustness* of EELRU with respect to LRU. They call a policy  $A$  *robust* with respect to policy  $B$  if  $A$  is  $k$ -miss-competitive relative to  $B$  for some  $k$ . In subsequent work (Smaragdakis 2004) shows how to construct a policy  $AB$  from two given policies  $A, B$ , that is 2-robust, i.e. 2-miss-competitive, with respect to both  $A$  and  $B$ .

There is one major difference between our approach and most of the existing work on competitive analysis that we are aware of: Most work interprets the competitive ratio as an indicator of the quality of a policy. In such work, the definition of competitiveness is relaxed to better distinguish policies that perform very differently in practice, like LRU and FIFO. For this purpose, our work is rather useless. If some policy  $A$  is not competitive relative to some other policy  $B$ , this is not a good indicator of the performance  $A$  and  $B$  are going to show on the average. For our purpose of giving guarantees on the number of hits or misses, the strict worst-case definition is, however, the right choice.

Previous work on cache analysis of set-associative caches mostly considered LRU-replacement (Ferdinand et al. 1997; Ferdinand and Wilhelm 1999; White et al. 1997; Ghosh et al. 1998; Chatterjee et al. 2001). Cache analyses embedded in timing analyses (Ferdinand et al. 1997; Ferdinand and Wilhelm 1999; White et al. 1997) classify individual accesses as hits or misses. Other approaches using Cache Miss Equations (Ghosh et al. 1998) or Presburger formulas (Chatterjee et al. 2001) compute the number of misses of larger parts of a program, like loop nests. The scope of both approaches can be extended by our results. In (Heckmann et al. 2003) a *must*-analysis for an 8-way PLRU is discussed, that maintains the 4 most-recently-used elements. The soundness of this approach can easily be explained by the 1-miss-competitiveness of  $PLRU(8)$  relative to  $LRU(4)$ .

A two-page extended abstract presenting our analysis results appears in SIGMETRICS 2008.

## 7. Summary and Future Work

We have studied the relative competitiveness of three well-known and widely-used families of replacement policies LRU, FIFO, and PLRU. By building a finite quotient structure of the transition system induced by a pair of policies, we were able to compute competitive ratios automatically. The competitiveness analyses revealed interesting previously unknown relations between different policies. The results can be used in two ways: One may bound the number of hits and misses for the execution of a program, or, in the case of 1-competitiveness, one may build sound *may* and *must* analyses. For instance, the relation between  $LRU(2k - 1)$  and  $FIFO(k)$  allows to construct *may* cache analyses for caches with FIFO replacement.

The competitiveness results we are computing hold for arbitrary access sequences. Therefore, they hold for any program. Most pro-

grams do not exhibit the worst-case relative behavior. By restricting the possible access sequence it would be possible to obtain smaller competitive ratios. Such restrictions could be for a particular program or for classes of programs, like structured programs. However, restricting access sequences potentially yields larger quotient transition systems. Another line of future work would be to consider other relations between replacement policies, like the maximal difference of the miss rates.

## References

- Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.
- Hussein Al-Zoubi, Aleksandar Milenkovic, and Milena Milenkovic. Performance evaluation of cache replacement policies for the SPEC CPU2000 benchmark suite. In *ACM-SE 42: Proceedings of the 42nd Annual Southeast Regional Conference*, pages 267–272, New York, NY, USA, 2004.
- James Aspnes and Orli Waarts. Modular competitiveness for distributed algorithms. In *STOC '96: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 237–246, New York, NY, USA, 1996. ACM Press. ISBN 0-89791-785-5.
- L. Belady. A study of replacement algorithms for a virtual storage computer. *IBM Systems Journal*, 5:78–101, 1966.
- Siddhartha Chatterjee, Erin Parker, Philip J. Hanlon, and Alvin R. Lebeck. Exact analysis of the cache behavior of nested loops. In *PLDI '01: Proceedings of the ACM SIGPLAN 2001 conference on Programming language design and implementation*, pages 286–297, New York, NY, USA, 2001. ACM Press.
- Christian Ferdinand and Reinhard Wilhelm. Efficient and precise cache behavior prediction for real-time systems. *Real-Time Systems*, 17(2-3): 131–181, 1999.
- Christian Ferdinand, Florian Martin, and Reinhard Wilhelm. Applying compiler techniques to cache behavior prediction. In *Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers and Tools for Real-Time Systems*, pages 37–46, Las Vegas, Nevada, June 1997.
- Somnath Ghosh, Margaret Martonosi, and Sharad Malik. Precise miss analysis for program transformations with caches of arbitrary associativity. In *ASPLOS-VIII*, pages 228–239, New York, NY, USA, 1998. ACM Press.
- Reinhold Heckmann, Marc Langenbach, Stephan Thesing, and Reinhard Wilhelm. The influence of processor architecture on the design and the results of WCET tools. *Proceedings of the IEEE*, 91(7), 2003.
- A. Hergenhan and W. Rosenstiel. Static timing analysis of embedded software on advanced processor architectures. In *DATE '00*, pages 552–559, New York, NY, USA, 2000. ACM Press.
- Elias Koutsoupias and Christos H. Papadimitriou. Beyond competitive analysis. In *IEEE Symposium on Foundations of Computer Science*, pages 394–400, 1994.
- E.L. Lawler. Optimal cycles in doubly weighted linear graphs. In *Int'l Symp. Theory of Graphs*, pages 209–213, 1966.
- Jan Reineke and Daniel Grund. Relative competitiveness of cache replacement policies [extended abstract]. In *SIGMETRICS*, 2008 (to appear).
- Jan Reineke, Daniel Grund, Christoph Berg, and Reinhard Wilhelm. Timing predictability of cache replacement policies. *Real-Time Systems*, 37(2): 99–122, November 2007.
- Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985.
- Yannis Smaragdakis. General adaptive replacement policies. In *ISMM '04: Proceedings of the 4th international symposium on Memory management*, pages 108–119, New York, NY, USA, 2004. ACM.
- Yannis Smaragdakis, Scott Kaplan, and Paul Wilson. The EELRU adaptive replacement algorithm. *Perform. Eval.*, 53(2):93–123, 2003.
- Randall T. White, Christopher A. Healy, David B. Whalley, Frank Mueller, and Marion G. Harmon. Timing analysis for data caches and set-associative caches. In *RTAS '97*, page 192, Washington, DC, USA, 1997. IEEE Computer Society.

<i>LRU</i> \ <i>FIFO</i>	2	3	4	5	6	7	8
2	2, 1	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
3	1, 0	3, 2	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
4	1, 0	2, 1	4, 3	$\infty$	$\infty$	$\infty$	$\infty$
5	1, 0	1, 0	2, 2	5, 4	$\infty$	$\infty$	$\infty$
6	1, 0	1, 0	2, 1	3, 3	6, 5	$\infty$	$\infty$
7	1, 0	1, 0	1, 0	2, 2	3, 4	7, 6	$\infty$
8	1, 0	1, 0	1, 0	2, 1	2, 3	4, 5	8, 7

(a) Miss-Competitiveness: LRU vs. FIFO

<i>FIFO</i> \ <i>PLRU</i>	2	4	8
2	2, 1	$\infty$	$\infty$
3	1, 1	$\infty$	$\infty$
4	1, 1	4, 4	$\infty$
5	1, 1	5, 4	$\infty$
6	1, 1	2, 4	$\infty$
7	1, 1	7, 4	$\infty$
8	1, 1	8, 4	8, 8

(b) Miss-Competitiveness: FIFO vs. PLRU

<i>PLRU</i> \ <i>FIFO</i>	2	3	4	5	6	7	8
2	2, 1	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
4	1, 0	2, 2	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
8	1, 0	$\frac{4}{3}, 1$	2, 3	$\infty$	$\infty$	$\infty$	$\infty$

(c) Miss-Competitiveness: PLRU vs. FIFO

<i>PLRU</i> \ <i>LRU</i>	2	3	4	5	6	7	8
2	1, 0	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
4	1, 0	1, 0	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
8	1, 0	1, 0	1, 0	$\infty$	$\infty$	$\infty$	$\infty$

(d) Miss-Competitiveness: PLRU vs. LRU

<i>LRU</i> \ <i>PLRU</i>	2	4	8
2	1, 0	$\infty$	$\infty$
3	1, 0	$\infty$	$\infty$
4	1, 0	2, 1	$\infty$
5	1, 0	5, 1	$\infty$
6	1, 0	1, 1	$\infty$
7	1, 0	1, 1	$\infty$
8	1, 0	5, 1	5, 4

(e) Miss-Competitiveness: LRU vs. PLRU

<i>FIFO</i> \ <i>LRU</i>	2	3	4	5	6	7	8
2	2, 1	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
3	1, 1	3, 2	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
4	1, 1	2, 2	4, 3	$\infty$	$\infty$	$\infty$	$\infty$
5	1, 1	1, 2	$\frac{5}{2}, 3$	5, 4	$\infty$	$\infty$	$\infty$
6	1, 1	1, 2	2, 3	3, 4	6, 5	$\infty$	$\infty$
7	1, 1	1, 2	7, 3	$\frac{7}{3}, 4$	7, 5	7, 6	$\infty$
8	1, 1	1, 2	3, 2	2, 4	$\frac{8}{3}, 5$	4, 6	8, 7

(f) Miss-Competitiveness: FIFO vs. LRU

**Figure 8.** Miss-Competitiveness results relating FIFO, PLRU, and LRU. The first component of each cell denotes the competitive ratio of the policy specified in the row relative to the policy specified in the column. The second component shows the additive constant.  $\infty$  indicates that the row-policy is not  $k$ -competitive relative to the column-policy for any  $k$ . As an example, LRU(6) is  $\frac{4}{3}$ -miss-competitive relative to PLRU(4) with additive constant 1.

<i>LRU</i> \ <i>FIFO</i>	2	3	4	5	6	7	8
2	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0
3	1, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0
4	1, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0
5	1, 0	1, 0	0, 0	0, 0	0, 0	0, 0	0, 0
6	1, 0	1, 0	0, 0	0, 0	0, 0	0, 0	0, 0
7	1, 0	1, 0	1, 0	0, 0	0, 0	0, 0	0, 0
8	1, 0	1, 0	1, 0	0, 0	0, 0	0, 0	0, 0

(a) Hit-Competitiveness: LRU vs. FIFO

<i>FIFO</i> \ <i>PLRU</i>	2	4	8
2	1, 1	0, 0	0, 0
3	1, 1	0, 0	0, 0
4	1, 1	$\frac{1}{5}, \frac{4}{3}$	0, 0
5	1, 1	7, 2	0, 0
6	1, 1	13, 2	0, 0
7	1, 1	$\frac{13}{3}, 3$	0, 0
8	1, 1	$\frac{11}{11}, \frac{19}{11}$	0, 0

(b) Hit-Competitiveness: FIFO vs. PLRU

<i>PLRU</i> \ <i>FIFO</i>	2	3	4	5	6	7	8
2	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0
4	1, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0
8	1, 0	$\frac{3}{4}, 1$	$\frac{1}{2}, \frac{3}{2}$	0, 0	0, 0	0, 0	0, 0

(c) Hit-Competitiveness: PLRU vs. FIFO

<i>PLRU</i> \ <i>LRU</i>	2	3	4	5	6	7	8
2	1, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0
4	1, 0	1, 0	$\frac{1}{2}, 1$	0, 0	0, 0	0, 0	0, 0
8	1, 0	1, 0	1, 0	$\frac{2}{3}, \frac{4}{3}$	$\frac{1}{2}, \frac{3}{2}$	$\frac{2}{5}, \frac{8}{5}$	$\frac{1}{4}, \frac{3}{2}$

(d) Hit-Competitiveness: PLRU vs. LRU

<i>LRU</i> \ <i>PLRU</i>	2	4	8
2	1, 0	0, 0	0, 0
3	1, 0	0, 0	0, 0
4	1, 0	1, 1	0, 0
5	1, 0	1, 1	0, 0
6	1, 0	1, 1	0, 0
7	1, 0	1, 1	0, 0
8	1, 0	1, 1	$\frac{1}{8}, \frac{15}{8}$

(e) Hit-Competitiveness: LRU vs. PLRU

<i>FIFO</i> \ <i>LRU</i>	2	3	4	5	6	7	8
2	1, 1	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0
3	1, 1	1, 1	0, 0	0, 0	0, 0	0, 0	0, 0
4	1, 1	1, 1	1, 1	0, 0	0, 0	0, 0	0, 0
5	1, 1	1, 1	1, 1	$\frac{1}{2}, 2$	0, 0	0, 0	0, 0
6	1, 1	1, 1	1, 1	1, 2	$\frac{1}{2}, \frac{5}{2}$	0, 0	0, 0
7	1, 1	1, 1	1, 1	1, 2	1, 3	1, 3	0, 0
8	1, 1	1, 1	1, 1	1, 2	1, 3	1, 3	$\frac{1}{2}, \frac{7}{2}$

(f) Hit-Competitiveness: FIFO vs. LRU

**Figure 9.** Hit-Competitiveness results relating FIFO, PLRU, and LRU.