Theoretical Evidence for the Superiority of LRU-2 over LRU for the Paging Problem *

Joan Boyar, Martin R. Ehmsen, and Kim S. Larsen

Department of Mathematics and Computer Science
University of Southern Denmark, Odense
Campusvej 55, DK-5230 Odense M, Denmark
{joan,ehmsen,kslarsen}@imada.sdu.dk
Fax: +45 6550 2325

Abstract. The paging algorithm LRU-2 was proposed for use in database disk buffering and shown experimentally to perform better than LRU [O'Neil, O'Neil, and Weikum, 1993]. We compare LRU-2 and LRU theoretically, using both the standard competitive analysis and the newer relative worst order analysis. The competitive ratio for LRU-2 is shown to be $2k$ for cache size k, which is worse than LRU's competitive ratio of k. However, using relative worst order analysis, we show that LRU-2 and LRU are asymptotically comparable in LRU-2's favor, giving a theoretical justification for the experimental results.

1 Introduction

On many layers in a computer system, one is faced with maintaining a subset of memory units from a relatively slow memory in a significantly smaller fast memory. For ease of terminology, we refer to the fast memory as the *cache* and to the memory units as *pages*. The cache will have size k, meaning that it can hold at most k pages at one time. Pages are requested by the user (possibly indirectly by an operating or a database system) and the requests must be treated one at a time without knowledge of future requests. This makes the problem an *on-line* problem [2]. If a requested page is already in cache, this is referred to as a *hit*. Otherwise, we have a *page fault*. The treatment of a request must entail that the requested page reside in cache. Thus, the only freedom is the choice of a page to evict from cache in order to make room for the requested page in the case of a page fault. An algorithm for this problem is referred to as a *paging algorithm*. Other names for this in the literature are "eviction strategy" or "replacement policy". Various cost models for this problem have been studied. We focus on the classic model of minimizing the number of page faults. The problem is of great importance in database systems where it is often referred to as the *database disk buffering problem*. See [2] for an overview of the paging problem, cost models, and paging algorithms in general.

* This work was supported in part by the Danish Natural Science Research Council (SNF).

Probably the most well-known paging algorithm is LRU (Least-Recently-Used), which on a page fault evicts the least recently used page from cache. The experience from real-life request sequences is that overall LRU performs better than all other paging algorithms which have been proposed up until the introduction of LRU-2 [12]. On a page fault, LRU-2 evicts the page with the least recent second to last request (if there are pages in cache which have been requested only once, the least recently used of these is evicted). Compelling empirical evidence is given in [12] in support of the superiority of LRU-2 over LRU in database systems. We return to this issue below. Since the introduction of LRU-2, there have been other proposals for better paging algorithms; for example [8].

In the on-line community, there are, to our knowledge, no published results on LRU-2. We assume that this is because it has not been possible to explain the experimental results theoretically. In this paper, we provide a theoretical justification of LRU-2's superiority over LRU. More specifically, we show using relative worst order analysis [4] that LRU-2 and LRU are asymptotically comparable in LRU-2's favor. In establishing this result, we prove a general result giving on upper bound on how well any algorithm can perform relative to LRU.

It is well-known that analysis of the paging problem is particularly problematic for the most standard quality measure for on-line algorithms, the competitive ratio [9, 13]. This has lead researchers to investigate alternative methods. See a long list of these in [1]. However, these methods are mostly only applicable to the paging problem.

In contrast, it has been demonstrated that the relative worst order ratio is generally applicable. In most cases, the relative worst order ratio makes the same distinction between algorithms as the competitive ratio does. However, the following is a list of results, where the relative worst order ratio has distinguished algorithms in a situation where the competitive ratio cannot distinguish or even in some cases favors the "wrong" algorithm. This is not an exclusive list; we merely highlight one result from each of these on-line problems:

- For classical bin packing, Worst-Fit is better than Next-Fit [4].
- For dual bin packing, First-Fit is better than Worst-Fit [4].
- For paging, LRU is better than FWF (Flush-When-Full) [5].
- For scheduling, minimizing makespan on two related machines, a post-greedy algorithm is better than scheduling all jobs on the fast machine [7].
- For bin coloring [11], a natural greedy-type algorithm is better than just using one open bin at a time [10].
- For proportional price seat reservation, First-Fit is better than Worst-Fit [6].

We refer the reader to the referenced papers for details and more results. Here, we merely want to point out that the relative worst order ratio is an appropriate tool to apply to on-line problems in general and the paging problem in particular.

2 LRU-2 and Experimental Results

In [12], a new family of paging algorithms, LRU-K, is defined. Here, K is a constant which defines the algorithm. On a page fault, LRU-K evicts the page with the least recent K'th last request (for $K = 1$, LRU-K is LRU). If there are pages in cache which have been requested fewer than K times, then some subsidiary policy must be employed for those pages. In [12], LRU is suggested as a possible subsidiary policy. However, it would also be natural to recursively use LRU-(K − 1). For the case of K = 2, this is the same.

The authors' motivation for considering LRU-2 (or LRU-K in general for various K) is that LRU does not discriminate between pages with very frequent versus very infrequent references. Both types will be held in cache for a long time once they are brought in. This can be at the expense of pages with very frequent references.

The algorithm LFU (Least-Frequently-Used) which evicts the page which is least frequently used is the ultimate algorithm in the direction of focusing on frequency, but it is well-known [13] that this algorithm adjusts way too slowly to changing patterns in the request sequence. The family of algorithms, LRU = LRU-1, LRU-2, LRU-3, ... with recursive subsidiary policies can be viewed as approaching the behavior of LFU.

A conscientious testing in [12] of particularly LRU-2 and LRU-3 up against LRU and LFU lead the authors to conclude that LRU-2 is the algorithm of choice.

The algorithms are tested in a real database system environment using random references from a Zipfian distribution, using real-life data from a CODASYL database system, and finally using data generated to simulate request sequences which would arise from selected applications where LRU-2 is expected to improve performance.

LRU-2 and LRU-3 perform very similarly and in all cases significantly better than the other algorithms. Many test results are reported which can be viewed in different ways. If one should summarize the results in one sentence, we would say that LRU and LFU need 50–100% extra cache space in order to approach the performance of LRU-2.

3 Competitive Ratio Characterizations

Let $\mathbb{A}(I)$ denote the number of page faults $\mathbb{A}$ has on request sequence I. The standard measure for the quality of on-line algorithms is the *competitive ratio*: $\mathrm{CR}(\mathbb{A})$ of $\mathbb{A}$ is $\mathrm{CR}(\mathbb{A}) = \inf\{c \mid \exists b\colon \forall I\colon \mathbb{A}(I) \leq c \cdot \mathrm{OPT}(I) + b\}$, where OPT denotes an optimal off-line algorithm [9, 13].

LRU is known to be both a conservative algorithm [14] and a marking algorithm [3]. Both types of algorithms have competitive ratio k. The following request sequence, $\langle (p_1, p_1), (p_2, p_2), ..., (p_{k+1}, p_{k+1}), (p_1, p_2, p_1, p_2)\rangle$, shows that LRU-2 belongs to neither of these classes, since it faults on all of the last four requests. Thus, it is not obvious that its competitive ratio is k. In fact the lemma below shows that it is larger than k.

Lemma 1. *The competitive ratio of* LRU-2 *is at least* $2k$ *for* k *even.*

Proof. Assume that there are $k+1$ distinct pages, $p_1, p_2, ..., p_{k+1}$, in slow memory, and that k is even.

Let

$$P_1 = \langle (p_2, p_2), (p_3, p_3), \ldots, (p_{k+1}, p_{k+1}) \rangle$$
$$P_2 = \langle (p_2, p_2), (p_3, p_3), \ldots, (p_k, p_k), (p_1, p_1) \rangle$$

and define the request sequence I_l by

$$\langle P_1, (p_1, p_2, p_1, p_2), (p_3, p_4, p_3, p_4), \ldots, (p_{k-1}, p_k, p_{k-1}, p_k),$$
$$P_2, (p_{k+1}, p_2, p_{k+1}, p_2), (p_3, p_4, p_3, p_4), \ldots, (p_{k-1}, p_k, p_{k-1}, p_k) \rangle^l.$$

After LRU-2 processes P_1, the page p_1 will not be in cache. Considering any block $(p_i, p_{i+1}, p_i, p_{i+1})$, for $1 \leq i \leq k+1$ in I_l (where $(k+1)+1$ will be considered 2), it follows inductively that the page p_i is not in LRU-2's cache on the first request in that block. The faults on p_i cause p_{i+1} to be evicted and vice versa until the second fault on p_{i+1}, which causes p_{i+2} (or p_3, if $i = k+1$) to be evicted. Thus, LRU-2 faults k times during the first occurrence of P_1, never on P_1 or P_2 after that, and on all $4kl$ of the remaining requests. OPT, on the other hand, faults k times during the first occurrence of P_1. It also faults on requests to p_1 immediately following P_1 and evicts p_{k+1} each time. Similarly, on the requests to p_{k+1} immediately following P_2, it evicts p_1. Thus it faults $k + 2l$ times in all. Since l can be arbitrarily large, this gives a ratio of $2k$ asymptotically. □

Lemma 2. LRU-2 *is* $2k$*-competitive.*

Proof. First notice that it is enough to prove that in each k-phase (a maximal subsequence of consecutive requests containing exactly k distinct pages) of any sequence I, LRU-2 faults at most two times on each of the k different pages requested in that phase. Suppose, for the sake of contradiction, that LRU-2 faults more than two times on some page in a phase P. Let p be the first page in P with more than two faults. At some point between the second and third faults on p, p must have been evicted by a request to some page q. The page q is one of the k pages in P. Thus, at this point there must be some page r in cache which is not in P. The second to last request to r must be before the start of P and thus before the second to last request to p. Hence, p could not have been evicted at this point. This gives a contradiction, so there are at most $2k$ faults in any k-phase. □

The following theorem follows immediately from the previous two results:

Theorem 1. CR(LRU-2) $= 2k$.

4 Relative Worst Order Characterizations

Now we show the theoretical justification for the empirical result that LRU-2 performs better than LRU. In order to do this, we use a different measure

for the quality of on-line algorithms, the relative worst order ratio [4, 5], which has previously [4, 5, 7, 10, 6] proven capable of differentiating between algorithms in other cases where the competitive ratio failed to give the "correct" result. Instead of comparing on-line algorithms to an optimal off-line algorithm (and then comparing their competitive ratios), two on-line algorithms are compared directly. However, instead of comparing their performance on the exact same sequence, they are compared on their respective worst permutations of the same sequence:

Definition 1. *Let $\sigma(I)$ denote a permutation of the sequence I, let $\mathbb{A}$ and $\mathbb{B}$ be algorithms for the paging problem, and let $\mathbb{A}_W(I) = \max_\sigma\{\mathbb{A}(\sigma(I))\}$. Let S_1 and S_2 be statements about algorithms $\mathbb{A}$ and $\mathbb{B}$ defined in the following way.*

$$S_1(c) \triangleq \exists b\colon \forall I\colon \mathbb{A}_W(I) \leq c\,\mathbb{B}_W(I) + b$$
$$S_2(c) \triangleq \exists b\colon \forall I\colon \mathbb{A}_W(I) \geq c\,\mathbb{B}_W(I) - b$$

The relative worst order ratio $\mathrm{WR}_{\mathbb{A},\mathbb{B}}$ *of algorithm $\mathbb{A}$ to algorithm $\mathbb{B}$ is defined if $S_1(1)$ or $S_2(1)$ holds.*

If $S_1(1)$ holds, then $\mathrm{WR}_{\mathbb{A},\mathbb{B}} = \sup\{r \mid S_2(r)\}$, *and*
if $S_2(1)$ holds, then $\mathrm{WR}_{\mathbb{A},\mathbb{B}} = \inf\{r \mid S_1(r)\}$.

The statements $S_1(1)$ and $S_2(1)$ check that the one algorithm is always at least as good as the other on every sequence (on their respective worst permutations). When one of them holds, the relative worst order ratio is a bound on how much better the one algorithm can be. In some cases, however, the first algorithm can do significantly better better than the second, while the second can sometimes do marginally better than the first. In such cases, we use the following definitions (from [5], but restricted to the paging problem here) and show that the two algorithms are asymptotically comparable in favor of the first algorithm.

Definition 2. *Let $\mathbb{A}$ and $\mathbb{B}$ be algorithms for the paging problem, and let the statement $S_1(c)$ be defined as above. If there exists a positive constant c such that $S_1(c)$ is true, let $c_{\mathbb{A},\mathbb{B}} = \inf\{r \mid S_1(r)\}$. Otherwise, $c_{\mathbb{A},\mathbb{B}}$ is undefined.*

- *If $c_{\mathbb{A},\mathbb{B}}$ and $c_{\mathbb{B},\mathbb{A}}$ are both defined, $\mathbb{A}$ and $\mathbb{B}$ are* $(c_{\mathbb{A},\mathbb{B}}, c_{\mathbb{B},\mathbb{A}})$-related.
- *If $c_{\mathbb{A},\mathbb{B}}$ is defined and $c_{\mathbb{B},\mathbb{A}}$ is undefined, $\mathbb{A}$ and $\mathbb{B}$ are* $(c_{\mathbb{A},\mathbb{B}}, \infty)$-related.
- *If $c_{\mathbb{A},\mathbb{B}}$ is undefined and $c_{\mathbb{B},\mathbb{A}}$ is defined, $\mathbb{A}$ and $\mathbb{B}$ are* $(\infty, c_{\mathbb{B},\mathbb{A}})$-related.

$\mathbb{A}$ and $\mathbb{B}$ are asymptotically comparable*, if*

$$\left(\lim_{k\to\infty}\{c_{\mathbb{A},\mathbb{B}}\} \leq 1 \wedge \lim_{k\to\infty}\{c_{\mathbb{B},\mathbb{A}}\} \geq 1\right) \vee \left(\lim_{k\to\infty}\{c_{\mathbb{A},\mathbb{B}}\} \geq 1 \wedge \lim_{k\to\infty}\{c_{\mathbb{B},\mathbb{A}}\} \leq 1\right)$$

If $\mathbb{A}$ and $\mathbb{B}$ are asymptotically comparable algorithms, then $\mathbb{A}$ and $\mathbb{B}$ are asymptotically comparable in $\mathbb{A}$'s favor *if* $\lim_{k\to\infty}\{c_{\mathbb{B},\mathbb{A}}\} > 1$.

First, we show that LRU-2 can do significantly better than LRU on some sets of input.

Theorem 2. *There exists a family of sequences I_n of page requests and a constant b such that*

$$\mathrm{LRU}_W(I_n) \geq \frac{k+1}{2}\text{LRU-2}_W(I_n) - b,$$

and $\lim_{n\to\infty} \mathrm{LRU}_W(I_n) = \infty$.

Proof. Let I_n consist of n phases, where in each phase, the first $k-1$ requests are to the $k-1$ pages $p_1, p_2, \ldots, p_{k-1}$, always in that order, and the last two requests are to completely new pages. LRU will fault on every page, so it will fault $n(k+1)$ times.

Regardless of the order this sequence is given in, LRU-2 will never evict any page $p' \in \{p_1, p_2, \ldots, p_{k-1}\}$ after the second request to p'. This follows from the fact that there are at most $k-1$ pages in cache with two or more requests at any point in time. Hence, when LRU-2 faults there is at least one page in cache with only one request in its history. By definition, LRU-2 must use its subsidiary policy when there exists pages in cache with less than two requests, and it must choose among these pages. This means that LRU-2 faults at most $2(k-1)+2n$ times.

Asymptotically, the ratio is $\frac{k+1}{2}$. □

The above ratio of $\frac{k+1}{2}$ cannot be improved. In fact no paging algorithm $\mathbb{A}$ can be $(c_{\mathbb{A},\mathrm{LRU}}, c_{\mathrm{LRU},\mathbb{A}})$-related to LRU with $c_{\mathrm{LRU},\mathbb{A}} > \frac{k+1}{2}$.

Theorem 3. *For any paging algorithm $\mathbb{A}$,*

$$c_{\mathrm{LRU},\mathbb{A}} \leq \frac{k+1}{2}.$$

Proof. Suppose there exists a sequence I, where LRU faults s times on its worst permutation, I_{LRU}, $\mathbb{A}$ faults s' times on its worst permutation, $I_{\mathbb{A}}$, and $s > \frac{k+1}{2}s'$. As proven in [5], there exists a worst permutation I_f of I_{LRU} with respect to LRU where all faults appear before all hits. Let I_1 be the prefix of I_f consisting of the s faults. Partition the sequence I_1 into subsequences of length $k+1$ (except possibly the last which may be shorter). We process these subsequences one at time, possibly reordering some of them, so that LRU-2 faults at least twice on all, except possibly the last. (Note that since LRU-2 will fault on the first $k+1$ requests, if $k \geq 3$, this last incomplete subsequence can be ignored. Otherwise, it contributes at most an additive constant of k to the inequality in statement $S_1(\frac{k+1}{2})$.) The first subsequence need not be reordered. Suppose the first i subsequences have been considered and consider the $i+1$st, $I' = \langle r_1, r_2, \ldots, r_{k+1}\rangle$, of consecutive requests in I_1, where $\mathbb{A}$ faults at most once. Since LRU faults on every request, they must be to $k+1$ different pages, $p_1, p_2, ..., p_{k+1}$. Let p be the page requested immediately before I'. Clearly, p must be in $\mathbb{A}$'s cache when it begins to process I' (it is a paging algorithm). If r_{k+1} is not a request to p, then I' contains $k+1$ pages different from p, but at most $k-1$ of them are in $\mathbb{A}$'s cache when it begins to process I' (p is in its cache). Hence, $\mathbb{A}$ must fault

at least twice on the requests in I'. On the other hand, if r_{k+1} is a request to p, there are exactly k requests in I' which are different from p. At least one of them, say p_i, must cause a fault, since at most $k-1$ of them could have been in $\mathbb{A}$'s cache just before it began processing I'. If $\mathbb{A}$ faults on no other page than p_i in I', then all the pages $p, p_1, p_2, \ldots, p_{i-1}, p_{i+1}, \ldots, p_k$ must be in $\mathbb{A}$'s cache just before it starts to process I'. Now, move the request to p_i to the beginning of I' which causes $\mathbb{A}$ to fault and evict one of the pages $p, p_1, p_2, \ldots, p_{i-1}, p_{i+1}, \ldots, p_k$. Hence, it must fault at least one additional time while processing the rest of this reordering of I'. □

Next, we show that LRU can never do significantly better than LRU-2. In order to show this, we need some definitions and lemmas characterizing LRU-2's behavior.

Lemma 3. *For any request sequence I, there exists a worst ordering of I with respect to* LRU-2 *with all faults appearing before all hits.*

Proof. We describe how any permutation I' of I can be transformed, step by step, to a permutation $I_{\text{LRU-2}}$ with all hits appearing at the end of the sequence, without decreasing the number of faults LRU-2 will incur on the sequence. Let I' consist of the requests $r_1, r_2, \ldots, r_n$, in that order.

If all hits in I' appears after all the faults, we are done. Otherwise consider the first hit r_i in I' with respect to LRU-2. We construct a new ordering by moving r_i later in I'.

Let p denote the page requested by r_i. First, we remove r_i from I' and call the resulting sequence I''.

If LRU-2 never evicts p in I'' or evicts p at the same requests in I'' as it does in I', then insert r_i after r_n in I''. This case is trivial since the behavior of LRU-2 on I' and I'' is the same.

Thus, we need only consider the case where p is evicted at some point after r_{i-1} in I'', and is not evicted at the same point in I'. Let r_j, $j > i$, denote the first request causing p to get evicted in I'' but not evicted in I'. Insert r_i just after r_j in I''. The resulting request sequence I'' is shown in Fig. 1 where $r_{p,1}$ and $r_{p,2}$ denote the next two requests to p (if they exist).

$$I' : \langle \ldots, r_{i-1}, r_i, r_{i+1}, \ldots, r_j, \ldots, r_{p,1}, \ldots, r_{p,2}, \ldots \rangle$$
$$I'' : \langle \ldots, r_{i-1}, r_{i+1}, \ldots, r_j, r_i, \ldots, r_{p,1}, \ldots, r_{p,2}, \ldots \rangle$$

Fig. 1. The request sequence I'' after moving r_i.

First note that moving a request to p within the sequence only affects p's position in the queue that LRU-2 evicts from. The relative order of the other pages stays the same. Just before r_{i+1} the content of LRU-2's cache is the same

for both sequences. Therefore, for I'', the behavior of LRU-2 is the same as for I' until p is evicted at r_j. Just after this eviction in I'', p is requested by r_i in I''. Thus, just before r_{j+1}, the cache contents are again the same for both sequences. This means that all pages that are in cache just before r_{j+1}, except p, are evicted no later for I'' than for I'. Hence, no faults are removed on requests to pages different from p, so we only need to count the faults removed on requests to p.

No faults on requests to p are removed on requests after $r_{p,2}$ since after that request the second to last request to p occurs at the same relative position in I' as in I'', so LRU-2 cannot evict it in one and not the other. Hence, the only potential faults that could have been removed are at the two requests $r_{p,1}$ and $r_{p,2}$.

The only case that needs special care is the case where $r_{p,1}$ and $r_{p,2}$ both are faults in I' but both are hits in I''. In all other cases at most one fault is removed which is counterbalanced by the fault created on r_i.

Consider the case where $r_{p,1}$ and $r_{p,2}$ both are faults in I' but neither is in I''. First remove $r_{p,1}$ from I'' and call the resulting sequence I'''. Since $r_{p,2}$ is a fault in I', p must get evicted in the subsequence $\langle r_{p,1}, \ldots, r_{p,2} \rangle$ based on its second to last request in that subsequence, which is the same request for both I' and I'''. Consequently, if p is evicted in that subsequence of I' it must also get evicted in that subsequence of I''' and it follows that $r_{p,2}$ is a fault in both sequences and no faults have been removed.

The situation we are facing with I''' is no different from the situation we faced with I''. We need to insert a (removed) request to p (for I'' it was r_i and for I''' it is $r_{p,1}$) without removing any faults, except that we now have increased the number of faults among the first j requests by at least one, and we have moved the problem of inserting a request to p later in the sequence. We now proceed inductively on I''' in the same manner as we did for I'' until we can insert the request to p without removing any faults or we reach the end of the sequence (in which case we place it there).

Thus, we obtain $I_{\text{LRU-2}}$ in a finite number of steps. □

Thus, when considering a worst case sequence for LRU-2, one can assume that there is a prefix of the sequence containing all of the faults and no hits. In the remaining, we will only be considering such prefixes. We define LRU-2-phases, starting from any request in a request sequence I.

Definition 3. *Let $I = \langle r_1, r_2, ..., r_n \rangle$ be a request sequence for which* LRU-2 *faults on every request. The* LRU-2-phase starting at request r_i *is $P(r_i) = \langle r_i, r_{i+1}, ..., r_j \rangle$, where j is as large as possible under the restriction that no page should be requested three times in $P(r_i)$. A* LRU-2*-phase is* complete *if r_j is not the last request in the I, i.e., r_{j+1} is a page which occurs twice in $P(r_i)$.*

Lemma 4. *Each complete* LRU-2*-phase contains at least $2k + 1$ requests to at least $k + 1$ distinct pages. In addition, it contains two requests to each of at least k different pages.*

Proof. By definition, within a complete LRU-2-phase, P, there is a request to a page p immediately after that phase. This request causes a fault, and p was

requested at least twice within the phase. In order for p to be evicted within the phase P, the second to last request to each of the $k-1$ other pages in cache must have occurred more recently than the first request to p in P. Thus, counting p, at least k distinct pages must have been requested twice in P. In addition, the page causing the second eviction of p within P cannot have been in cache at that point, so P consists of at least $2k+1$ requests. □

Lemma 5. *Let p be the page that starts a complete* LRU-2*-phase containing exactly $2k+1$ requests, then the following phase (if it exists) starts with a request to p.*

Proof. In general after a complete LRU-2-phase, LRU-2 has at least $k-1$ of the pages requested twice in that phase in cache. If the phase ends with the second request to a page, then LRU-2 contains k of the pages requested twice. This fact follows from the observation that by the LRU-2 policy no page with two requests in a phase can get evicted if there is a page in cache with only one request in that phase.

This means that a phase containing only $2k+1$ requests must end with a request to the only page with one request in that phase. Before that request the cache contains k pages which have all been requested twice in the current phase, hence p is the page with the earliest second request. This means p gets evicted on the last request in the phase and hence (by the construction of LRU-2-phases) it must be the page starting the next phase. □

By induction the above shows that if there exist several consecutive phases, each containing $2k+1$ requests, then they must all begin with a request to the same page. This then shows that if $p_1, p_2, \ldots, p_k$ are the k pages requested twice in a phase containing $2k+1$ requests, then after the first request in the following phase (if it exists) all of the pages $p_1, p_2, \ldots, p_k$ are in LRU-2's cache.

Lemma 6. *For any sequence I of page requests,*

$$\text{LRU-2}_W(I) \leq (1 + \frac{1}{2k+2})\text{LRU}_W(I).$$

Proof. Consider any sequence I of requests. By Lemma 3, there exists a worst permutation, $I_{\text{LRU-2}}$, of I such that LRU-2 faults on each request of a prefix I_1 of $I_{\text{LRU-2}}$ and on no requests after I_1. Partition I_1 into LRU-2-phases. We will now inductively transform I_1 into a sequence I_1' such that

$$\text{LRU-2}(I_1) \leq (1 + \frac{1}{2k})\text{LRU}_W(I_1').$$

Start at the beginning of I_1, and consider the LRU-2-phase starting with the first request not already placed in a processed phase. By Lemma 5 each LRU-2-phase contains at least a total of $2k+1$ requests to at least $k+1$ distinct pages. Since each page requested in a LRU-2-phase is at most requested twice, a LRU-2-phase containing at least $2k+2$ requests can be partitioned into two sets, each

containing at least $k+1$ pages, none of which are repeated. Each of these sets of requests can then be ordered so that LRU faults on every request.

Hence, suppose the current LRU-2-phase contains exactly $2k+1$ requests. See Fig. 2 where $|$ marks the beginning of a new phase in I_1 which contains exactly $2k+1$ requests. Let $p_1, p_2, \ldots, p_k$ be the k pages requested twice and q_1 be the page requested once in that phase and let p_1 be the page which begin the following phase. The request r_i to p_1 which starts the following phase must evict q_1 and hence all the pages $p_1, p_2, \ldots, p_k$ are in LRU-2's cache just before the request to $r_{i+1} = q_2$. It follows that $q_2 \notin \{p_1, p_2, \ldots, p_k\}$.

$$\langle \ldots, |p_1, \ldots, q_1, |r_i = p_1, r_{i+1} = q_2, r_{i+2} \ldots \rangle$$

Fig. 2. A LRU-2-phase containing $2k+1$ requests.

By moving r_i to the end of the request sequence it follows from the above that the modified phase in question now contains at least $2(k+1)$ requests (the $2k+1$ requests and the request to q_2) and by the same argument as above it follows that it is possible to make LRU fault on every request. Hence in each such phase LRU faults on (possibly) one request less than LRU-2. The next LRU-2-phase to be processed starts with r_{i+2} or later.

Let l denote the total number of modified phases. For each modified phase i, there are $s_i \geq 2(k+1)$ requests, plus possibly one additional request which LRU-2 faulted on and has been moved to the end. Thus, LRU faults at least $\sum_{i=1}^{l} s_i$ times and LRU-2 faults at most $\sum_{i=1}^{l}(s_i+1)$ times. It follows that

$$\begin{aligned}
\text{LRU-2}_W(I) &\leq \frac{\sum_{i=1}^{l}(s_i+1)}{\sum_{i=1}^{l} s_i}\text{LRU}_W(I) \\
&\leq \frac{l(2k+3)}{l(2k+2)}\text{LRU}_W(I) \\
&= (1+\frac{1}{2k+2})\text{LRU}_W(I)
\end{aligned}$$

□

Combining Theorem 2 and the lemma above gives the following:

Theorem 4. LRU-2 *and* LRU *are* $(1+\frac{1}{2k+2}, \frac{k+1}{2})$*-related, i.e., they are asymptotically comparable in* LRU-2*'s favor.*

5 Concluding Remarks

In contrast to the results using competitive analysis, relative worst order analysis yields a theoretical justification for superiority of LRU-2 over LRU, confirming

previous empirical evidence. It would be interesting to see if these results generalize to LRU-K for $K > 2$. Recently, we have shown that the competitive ratio for LRU-K is kK, and that the separation result showing that LRU-K can be better than LRU holds. The question is: Does the asymptotic comparability still hold.

Although it was shown here that LRU-2 and LRU are asymptotically comparable, it would be interesting to know if the stronger result, that LRU-2 and LRU are comparable using relative worst order analysis, holds. If they are, then the above results show that the relative worst order ratio of LRU to LRU-2 is $\frac{k+1}{2}$.

An algorithm called RLRU was proposed in [5] and shown to be better than LRU using relative worst order analysis. We conjecture that LRU-2 is also asymptotically comparable to RLRU in LRU-2's favor. We have found a family of sequences showing that LRU-2 can be better than RLRU, but would also like to show that the algorithms are asymptotically comparable.

References

1. Susanne Albers. Online algorithms: A survey. In *Proceedings of the 18th International Symposium on Mathematical Programming*, pages 3–26, 2003.
2. Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis.* Cambridge University Press, 1998.
3. Allan Borodin, Sandy Irani, Prabhakar Raghavan, and Baruch Schieber. Competitive Paging with Locality of Reference. *Journal of Computer and System Sciences*, 50(2):244–258, 1995.
4. Joan Boyar and Lene M. Favrholdt. The relative worst order ratio for on-line algorithms. In *Proceedings of the Fifth Italian Conference on Algorithms and Complexity*, volume 2653 of *Lecture Notes in Computer Science*, pages 58–69. Springer-Verlag, 2003. Extended version to appear in *ACM Transactions on Algorithms.*
5. Joan Boyar, Lene M. Favrholdt, and Kim S. Larsen. The Relative Worst Order Ratio Applied to Paging. In *Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 718–727. ACM Press, 2005.
6. Joan Boyar and Paul Medvedev. The Relative Worst Order Ratio Applied to Seat Reservation. In *Proceedings of the Nineth Scandinavian Workshop on Algorithm Theory*, volume 3111 of *Lecture Notes in Computer Science*, pages 90–101. Springer-Verlag, 2004.
7. Leah Epstein, Lene M. Favrholdt, and Jens S. Kohrt. Separating Scheduling Algorithms with the Relative Worst Order Ratio. Tech. report PP-2006-05, Department of Mathematics and Computer Science, University of Southern Denmark, Odense, 2006.
8. Amos Fiat and Ziv Rosen. Experimental Studies of Access Graph Based Heuristics: Beating the LRU Standard? In *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 63–72, 1997.
9. Anna R. Karlin, Mark S. Manasse, Larry Rudolph, and Daniel D. Sleator. Competitive Snoopy Caching. *Algorithmica*, 3:79–119, 1988.
10. Jens Svalgaard Kohrt. *Online Algorithms under New Assumptions.* PhD thesis, Department of Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark, 2004.

11. Sven Oliver Krumke, Willem de Paepe, Jörg Rambau, and Leen Stougie. Online Bin Coloring. In *Proceedings of the Nineth Annual European Symposium on Algorithms*, volume 2161 of *Lecture Notes in Computer Science*, pages 74–85. Springer-Verlag, 2001.
12. Elizabeth J. O'Neil, Patrick E. O'Neil, and Gerhard Weikum. The LRU-K Page Replacement Algorithm for Database Disk Buffering. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 297–306, 1993.
13. Daniel D. Sleator and Robert E. Tarjan. Amortized Efficiency of List Update and Paging Rules. *Communications of the ACM*, 28(2):202–208, 1985.
14. Neal Young. The k-Server Dual and loose Competitiveness for Paging. *Algorithmica*, 11(6):525–541, 1994.