Competitive analysis of the LRFU paging algorithm *

Edith Cohen

AT&T Labs–Research
180 Park Avenue
Florham Park, NJ 07932 USA
{edith}@research.att.com

Haim Kaplan *Uri Zwick*

School of Computer Science
Tel-Aviv University
Tel Aviv 69978, Israel
{haimk,zwick}@post.tau.ac.il

April 22, 2001

Abstract

We present a competitive analysis of the LRFU paging algorithm, a hybrid of the LRU (Least Recently Used) and LFU (Least Frequently Used) paging algorithms. We show that the competitive ratio of LRFU is $k + \left\lceil \frac{\log(1-\lambda)}{\log \lambda} \right\rceil - 1$, where $\frac{1}{2} \leq \lambda \leq 1$ is the decay parameter used by the LRFU algorithm, and k is the size of the cache. This supplies, in particular, the first natural paging algorithms that are competitive but are not optimally competitive, answering a question of Borodin and El-Yaniv. Although LRFU, as it turns out, is not optimally competitive, it is expected to behave well in practice, as it combines the benefits of both LRU and LFU.

1 Introduction

Paging (cache replacement) algorithms have been extensively studied and deployed. Two important applications are operating system virtual memory paging and caching of Web content. The input to a paging algorithm is a sequence of requests to different *pages* and the size of the *cache*. The algorithm decides which page to evict when a request arrives for a page which is not presently in the cache and the cache is full. Often, the choice of a paging algorithm can have a significant effect on performance.

Two important paging algorithms are *Least Recently Used* (LRU) and *Least Frequently Used* (LFU). LRU is arguably the most commonly deployed in practice. One example application is the popular Squid [7] Web caching software.[1] When LRU has to evict a page from its cache,

* A preliminary version of the paper is to appear in the proceedings of the 7th International Workshop on Algorithms and Data Structures (WADS'01), Providence, Rhode Island, 2001.

[1] Squid uses some variant of LRU where the least recently requested pages are periodically evicted from the cache when it gets close to being full.

it chooses to evict the least recently requested page. LRU exploits temporal locality in request sequences and the *recency* property which states that recently requested objects are more likely to be requested next than objects that have not been requested recently. LFU is another policy that seems to perform well on Web request sequences. LFU evicts the page in the cache that was requested the fewest number of times. LFU comes in two flavors: *in-cache* LFU which counts the number of times a page was requested since it entered the cache, and *perfect* LFU which counts the total number of requests made to the page since the start of the sequence. LFU exploits the *frequency* property of request sequences which states that pages that have been requested more times are more likely to be requested again.

Recent studies (e.g. [2]) comparing cache replacement algorithms concluded that LRU and LFU exhibit comparable performance on Web request sequences. In general, performance depends on the interplay between the recency and frequency properties of the particular sequence. Frequency seems to matter more for larger caches and recency has a more pronounced impact with smaller cache sizes. Similar behavior was observed for page request sequences in operating systems [6]. Motivated by these observations, Lee et al. [6] defined a spectrum of hybrid policies between LRU and LFU. They named these policies LRFU_λ, with the parameter λ varying between $\lambda = 0$ (LRU) and $\lambda = 1$ (LFU). Their experiments based on simulations using page requests generated by different application programs demonstrated that the performance curve as a function of λ is smooth and the dependence on λ is bitonic: the miss-rate first decreases and then increases as λ increases. Thus typically LRFU_λ outperforms at least one of the endpoints and with the best choices of λ, LRFU_λ often outperforms both LRU and LFU. Their results show that LRFU_λ does indeed provide a desirable spectrum in terms of performance.

Competitive analysis is the leading theoretical tool for analyzing the performance of paging algorithms [1]. A paging algorithm A is said to have a *competitive ratio* of at most c if on any request sequence, the number of *misses* of A is at most c times the number of misses of the optimal offline algorithm, plus some constant. A *miss* occurs when a requested page is not in the cache. The competitive ratio of LRU and LFU demonstrates both strengths and shortcomings of the ability of competitive analysis to capture practical behavior. The LRU algorithm is optimally competitive. That is, its competitive ratio of k is the best possible by any deterministic paging algorithm (where k is the maximum number of pages that can fit in the cache). On the other hand, the factor k obtained on worst-case adversarial sequences is very far from the typical ratio on actual sequences. Moreover, LFU, which performs comparably to LRU on Web sequences, has an unbounded competitive ratio (is not competitive). Practical sequences, as opposed to worst-case sequences, typically exhibit the recency and frequency properties exploited by LRU and LFU.

Interestingly, most natural deterministic paging algorithms fall in one of these two categories: either they are optimally-competitive like LRU, or they are noncompetitive like LFU. An open question posed by Borodin and El-Yaniv [1] (open question 3.2 on page 43) is the existence of a natural deterministic paging algorithm with a finite but not optimal competitive ratio. It seems that most conceivable hybrids of a non-competitive algorithm with an optimally-competitive algorithms (such as partitioning the cache between the two policies) are not competitive.

Our main contribution here is a tight competitive analysis of LRFU$_\lambda$. Our analysis reveals that as λ increases we obtain all integral values between the competitive ratio of LRU (k) and that of LFU (∞). This solves the open problem posed by Borodin and El-Yaniv. The full spectrum of values also suggests that LRFU$_\lambda$ is the "right" hybrid of LRU and LFU. In this sense the competitive analysis supports and complements the experimental results.

2 The LRFU$_\lambda$ paging algorithm

The LRFU$_\lambda$ paging policy, for $0 < \lambda \leq 1$, is defined as follows:

1. Each page p currently in the cache has a value $v(p)$ associated with it.
2. Upon a request for a page q, the following operations are performed:
   (a) If a page has to be evicted to make room for q, then the page with the smallest value is evicted. (Ties are broken arbitrarily.) The new page q is temporarily given the value $v(q) \leftarrow 0$.
   (b) The values of the pages in the cache are updated as follows:

$$v(p) \leftarrow \begin{cases} \lambda v(p) & \text{if } p \neq q, \\ 1 + \lambda v(p) & \text{if } p = q. \end{cases}$$

It follows that if a page p is in the cache at time t and if the hits to this page, since it last entered the cache, were at times $t - j_1, t - j_2, \ldots, t - j_n$, then its value at time t is $v(p) = \sum_{i=1}^{n} \lambda^{j_i}$. Note that the value $v(p)$ of a page is always *smaller* than $\sum_{j=0}^{\infty} \lambda^j = \frac{1}{1-\lambda}$.

In particular, if $0 < \lambda \leq \frac{1}{2}$, then LRFU$_\lambda$ behaves just like LRU, since the most recent request time dominates the value of the page (that is, $v(p_1) < v(p_2)$ if and only if the most recent request to p_1 occured before the most recent request to p_2). If $\lambda = 1$, then LRFU$_\lambda$ behaves just like LFU. It is well known that LRU has an optimal competitive ratio of k, while LFU is not competitive. We show that as λ increases from $\frac{1}{2}$ to 1, the competitive ratio of LRFU$_\lambda$ increases from k to ∞ and assumes all possible integral values in this range.

3 Competitiveness of LRFU$_\lambda$

We obtain the following result:

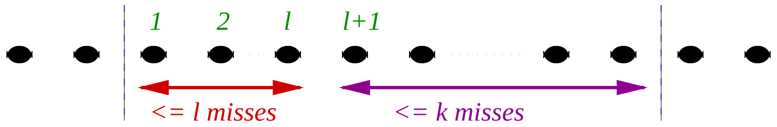

Figure 1: LRFU_λ incurs at most $k+\ell$ misses per k-phase.

Theorem 3.1 *The competitive ratio of* LRFU_λ *for a cache of size* k*, where* $\frac{1}{2} \le \lambda < 1$*, is*

$$k + \left\lceil \frac{\log(1-\lambda)}{\log \lambda} \right\rceil - 1 \, .$$

Another way of stating this result is as follows: the competitive ratio of LRFU_λ is $k+\ell$, where $\ell = \left\lceil \frac{\log(1-\lambda)}{\log \lambda} \right\rceil - 1$. Note that $\frac{\lambda^\ell}{1-\lambda} > 1$ while $\frac{\lambda^{\ell+1}}{1-\lambda} \le 1$. Thus, ℓ has the following property: if p was not referenced in the last $\ell+1$ time units, then $v(p) < 1$. ℓ is the smallest number with this property. Theorem 3.1 follows from Lemmas 3.2 and 3.4 that we obtain next.

3.1 Upper bound on the competitive ratio

Lemma 3.2 *For every* $\frac{1}{2} \le \lambda < 1$, LRFU_λ *is* $(k+\ell)$*-competitive, where* $\ell = \left\lceil \frac{\log(1-\lambda)}{\log \lambda} \right\rceil - 1$.

Proof: As in the classical analysis of LRU (see [1], Chapter 3), we break the sequence of requests into *k-phases*. A k-phase is a maximal contiguous block of requests in which at most k different pages are requested. The first k-phase starts at the beginning of the sequence and ends just before the $(k+1)$-st distinct page is referenced. The second block begins immediately after the first block and so on. It is easy to see that any caching policy must incur, on average, at least one miss for each k-phase. (More precisely, any caching policy must incur at least one miss on each *shifted k-phase*, where a shifted k-phase is a sequence of requests that begins with the second request of an ordinary k-phase and ends with the first request of the next ordinary k-phase.) To show that LRFU_λ is $(k+\ell)$-competitive, we show that LRFU_λ incurs at most $k+\ell$ misses on each k-phase.

We first claim that under the LRFU_λ policy, a page p that enters the cache *after* the first ℓ requests of a k-phase stays in the cache until the end of the phase. This follows from the interpretation of ℓ given after the statement of Theorem 3.1. When p enters the cache, and is assigned the value 1, the value $v(q)$ of a page q that is currently in the cache, but was not referenced yet in the current k-phase, must satisfy $v(q) < 1$. If a miss occurs in the k-phase after p enters the cache, then there is at least one page q in the cache that was not referenced yet in the k-phase. (If all k pages in the cache were referenced in the current phase then a subsequent miss would end the phase.) As $v(q) < v(p)$, p will not be evicted.

It follows, therefore, that at most ℓ misses occur in the first ℓ requests of each k-phase (at most one miss per request), and at most k misses occur in the rest of the k-phase (at most one miss per each page that enters the cache after the ℓ-th request). (See Figure 1.) This completes the proof of the lemma. $\Box$

Note that if $\lambda = \frac{1}{2}$ then $\ell = 0$ (i.e., $v(p) \geq 1$ if and only if p is the last page referenced), and Lemma 3.2 states that the competitive ratio of LRFU$_{1/2}$ =LRU is k. Lemma 3.2 is therefore an extension of the classical competitive analysis of the LRU policy. An easy extension of Lemma 3.2 is the following:

Lemma 3.3 *For every $\frac{1}{2} \leq \lambda < 1$ and $1 \leq h \leq k$,* LRFU$_\lambda$ *with a cache of size k is $(k+\ell)/(k-h+1)$-competitive, where $\ell = \left\lceil \frac{\log(1-\lambda)}{\log \lambda} \right\rceil - 1$, versus an offline adversary that uses a cache of size h.*

Proof: Follows from the fact any algorithm that uses a cache of size h must incur at least $k-h+1$ misses on each shifted k-phase. $\Box$

3.2 Lower bound on the competitive ratio

We now show that the bound given in Lemma 3.2 is tight.

Lemma 3.4 *For every $\frac{1}{2} \leq \lambda < 1$,* LRFU$_\lambda$ *is* at best *$(k+\ell)$-competitive, where $\ell = \left\lceil \frac{\log(1-\lambda)}{\log \lambda} \right\rceil - 1$.*

Proof: For every $k \geq 1$ and $\frac{1}{2} \leq \lambda < 1$, we construct an infinite sequence of requests such that the ratio between the number of misses incurred by LRFU$_\lambda$ and the minimum possible number of misses needed to serve the sequence approaches $k+\ell$. As in the classical case of LRU, only $k+1$ distinct pages are needed for this purpose.

Our sequences are again partitioned into k-phases. The state of the cache at the end of each k-phase is isomorphic to the state of the cache at the beginning of the phase. (This is explained below in more detail.) We show that LRFU$_\lambda$ incurs $k+\ell$ misses in each phase. As there is a total of only $k+1$ pages, the sequence of requests can be served with only one miss per k-phase. (At the first miss of a phase, the optimal algorithm evicts the page that is not going to be referenced in that phase.) The claim of the lemma would thus follow.

Let L (for Large) be a sufficiently large number such that after L consecutive requests to a page p we have $v(p) > \lambda^{-\ell}$ and $v(q) < 1$ for any page q such that $q \neq p$. Such a number exists as by the definition of ℓ we have $\lambda^{-\ell} < \frac{1}{1-\lambda}$.

The first initializing k-phase is composed of the following page requests:

$$p_2, p_3, p_4, \ldots, p_{k+1}^L \ ,$$

where p_i^L stands for L consecutive requests of the page p_i. At the end of this phase, the pages in the cache, in decreasing order of their values, are $p_{k+1}, p_k, \ldots, p_2$, and their values satisfy $v(p_{k+1}) > \lambda^{-\ell}$ and $v(p_i) < 1$, for $2 \leq i \leq k$.

Requested page	Cache content by decreasing value								
	$k+1$	k	$k-1$				4	3	2
1	$k+1$	1	k	$k-1$			5	4	3
2	$k+1$	2	1	k	$k-1$			5	4
					$\cdots$				
					$\cdots$				
ℓ	$k+1$	ℓ	$\ell-1$		k	$k-1$		$\ell+3$	$\ell+2$
$\ell+1$	$\ell+1$	$k+1$	ℓ	$\ell-1$		k	$k-1$		$\ell+3$
					$\cdots$				
					$\cdots$				
k	k	$k-1$		$\ell+1$	$k+1$	ℓ	$\ell-1$		2
1	1	k	$k-1$		$\ell+1$	$k+1$	ℓ		3
2	2	1	k	$k-1$		$\ell+1$	$k+1$		4
					$\cdots$				
					$\cdots$				
ℓ	ℓ	$\ell-1$		2	1	k	$k-1$		$\ell+1$

Figure 2: A typical phase of the worst-case sequence for LRFU_λ

The second phase is now composed of the following page requests:

$$p_{(1)}, p_{(2)}, \ldots, p^L_{(k+\ell)} ,$$

where the parentheses around the indices indicate that they should be interpreted modulo k (i.e., $p_{(k+1)} = p_1$, and more generally $p_{(i)} = p_{1+(i-1) \bmod k}$.

What happens to the cache when this sequence of requests is served? At the beginning of the phase $\lambda^{-\ell} < v(p_{k+1}) < \frac{1}{1-\lambda} \le \lambda^{-(\ell+1)}$ (the last inequality follows from the definition of ℓ). Thus, during the first ℓ requests of the phase, page p_{k+1} still has the largest value among all the pages of the cache, and the page entering the cache becomes the page with the *second* largest value. The page evicted is always the next page requested, and thus each request causes a miss.

When the $(\ell+1)$-st request of the phase is served, the value of $v(p_{k+1})$ drops below 1. The page entering the cache now becomes the page with the largest weight. The rank of p_{k+1} in the cache decreases by 1 at each step. The page evicted is again the next page requested and thus each request still causes a miss. While serving the request for $p_{(k+\ell)}$, page p_{k+1} is finally evicted from the cache.

Overall, there are $k + \ell$ misses in this phase. The content of the cache during such a phase is depicted in Figure 2. (In the figure, it is assumed, for simplicity, that $\ell < k$. The argument given above does not rely on this assumption.)

Now, the state of cache at the end of the second phase is similar to its state at the end of the first phase. One page, namely $p_{(k+\ell)}$, has value larger than $\lambda^{-\ell}$, while all other pages have values that are less than 1. We can therefore use a sequence of requests similar to the one used in the second state, and again create a k-phase in which LRFU_λ incurs $k+\ell$ misses. This process can be repeated indefinitely, resulting in the promised infinite sequence of requests. $\square$

4 Open problems

The plain LRU, LFU, and LRFU_λ algorithms are designed for pages with equal sizes and miss costs. Web objects, however, vary significantly in both size and cost of a miss. This motivated the development of cache replacement algorithms that account for varying page sizes and fetching costs [5, 8, 3, 4]. Experiments showed that the optimally-competitive Landlord algorithm performs well on Web caching sequences [8, 3]. Other experiments, however, show that it can be outperformed by perfect LFU [2]. A natural open problem is thus to extend our results to this more general model. That is, develop natural hybrid policies of Landlord and "weighted" LFU.

References

[1] A. Borodin and R. El Yaniv. *Online computation and competitive analysis.* Cambridge University Press, 1998.

[2] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and zipf-like distributions: Evidence and implications. In *Proceedings of the IEEE INFOCOM'99 Conference*, 1999.

[3] P. Cao and S. Irani. Cost-aware WWW proxy caching algorithms. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, Monterey, California, December 1997. `http://www.usenix.org/events/usits97`.

[4] E. Cohen and H. Kaplan. LP-based analysis of greedy-dual-size. In *Proc. 10th ACM-SIAM Symposium on Discrete Algorithms.* ACM-SIAM, 1999.

[5] S. Irani. Page replacement with multi-size pages and applications to web caching. In *Proc. 29th Annual ACM Symposium on Theory of Computing.* ACM, 1997.

[6] D. Lee, J. Choi, J.-H. Kim, S. H. Noh, S. L. Min, Y. Cho, and C. S. Kim. On the existence of a spectrum of policies that subsumes the Least Recently Used (LRU) and Least Frequently Used (LFU) policies. In *Proceedings of the ACM SIGMETRICS'99 Conference*, 1999.

[7] Squid internet object cache.
`http://squid.nlanr.net/Squid`.

[8] N. Young. On line file caching. In *Proc. 9th ACM-SIAM Symposium on Discrete Algorithms.* ACM-SIAM, 1998.