On-Line File Caching

Neal E. Young*

Abstract

Consider the following file caching problem: in response to a sequence of requests for files, where each file has a specified *size* and *retrieval cost*, maintain a cache of files of total size at most some specified k so as to minimize the total retrieval cost. Specifically, when a requested file is not in the cache, bring it into the cache, pay the retrieval cost, and choose files to remove from the cache so that the total size of files in the cache is at most k. This problem generalizes previous paging and caching problems by allowing objects of arbitrary size *and* cost, both important attributes when caching files for world-wide-web browsers, servers, and proxies.

We give a simple deterministic on-line algorithm that generalizes many well-known paging and weighted-caching strategies, including least-recently-used, first-in-first-out, flush-when-full, and the balance algorithm. On any request sequence, the total cost incurred by the algorithm is at most $k/(k-h+1)$ times the minimum possible using a cache of size $h \leq k$.

For *any* algorithm satisfying the latter bound, we show it is also the case that for *most* choices of k, the retrieval cost is either insignificant or the competitive ratio is *constant*. This helps explain why competitive ratios of many on-line paging algorithms have been typically observed to be constant in practice.

1 Background and Statement of Results

The *file caching* problem is as follows. Given a cache with a specified size k (a positive integer) and a sequence of requests to files, where each file has a specified *size* (a positive integer k) and a specified *retrieval cost* (a non-negative number), maintain files in the cache to satisfy the requests while minimizing the total retrieval cost. Specifically, when a requested file is not in the cache, bring it into the cache, paying the retrieval cost of the file, and choose files to remove from the cache so that the total size of files remaining in the cache is at most k.

Following Sleator and Tarjan [16], we say a file caching algorithm is $c(h,k)$-*competitive* if on any sequence the total retrieval cost incurred by the algorithm using a cache of size k is at most $c(h,k)$ times the minimum possible cost using a cache of size h. An algorithm is *on-line* if its response to a request does not depend on later requests in the sequence.

Uniform sizes, uniform costs. With the restriction that all file sizes and costs are the same, the problem is called *paging*. Paging has been extensively studied. In a seminal paper, Sleator and Tarjan [16] showed that least-recently-used and a number of other deterministic on-line paging strategies were $\frac{k}{k-h+1}$-competitive. Sleator and Tarjan also showed that this performance guarantee is the best possible for any deterministic on-line algorithm.

A simple randomized paging algorithm called the marking algorithm was shown to be $2\ln k$-competitive by Fiat et al. [5]. An optimal $\ln k$-competitive randomized paging algorithm was given by McGeoch and Sleator [15]. In [19], deterministic paging strategies were shown to be *loosely* $O(\ln k)$-competitive. This means roughly that for any sequence, for *most* values of k, the fault rate of the algorithm using a cache of size k is either insignificant or the algorithm is $O(\ln k)$-competitive versus the optimum algorithm using a cache of size k. Similarly, the marking algorithm was shown to be loosely $(2\ln\ln k + O(1))$-competitive.

Uniform sizes, arbitrary costs. The special case of file caching when all file sizes are the same is called *weighted caching*. For weighted caching, Manasse, McGeoch and Sleator [14] showed that an algorithm called the balance algorithm is k-competitive. Subsequently in [19] a generalization of that algorithm called the "greedy-dual" algorithm was shown to be $\frac{k}{k-h+1}$-competitive. The greedy-dual algorithm generalizes many well-known paging and weighted-caching strategies, including least-recently-used, first-in-first-out, flush-when-full, and the balance algorithm.

Arbitrary sizes, cost = 1 or cost = size. Motivated by the importance of file *size* in caching for world-wide-web applications (see comment below), Irani considered two special cases of file caching: when the costs are either all equal (the goal is to minimize the *number* of retrievals), and when each cost equals the file size (the goal is to minimize the total number of *bytes* retrieved). For these two cases, Irani [8] gave $O(\log^2 k)$-competitive randomized on-line algorithms.

Comment: the importance of sizes and costs. File caching is important for world-wide-web applications. For instance, in browsers and proxy servers remote files are cached locally to avoid remote retrieval. In web servers, disk files are cached in fast memory to speed response time. As Irani points out (see [8] and references therein), file *size* is an important consideration; caching policies adapted from memory management ap-

*Dartmouth College, Hanover NH 03755 ney@dartmouth.edu

Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, 82-86 (1998)

plications that don't take size into account do not work well in practice.

Allowing arbitrary *costs* is likely to be important as well. In many cases, the cost (e.g., latency, total transmission time, or network resources used) will neither be uniform across files nor proportional solely to the size. For instance, the cost to retrieve a remote file can depend on the *distance* the file must travel in the network. Even accounting for distance, the cost need not be proportional to the size, e.g., because of economies of scale in routing files through the network. Further, in some applications it makes sense to assign different *kinds* of costs to different kinds of files. For instance, some kinds of documents are displayed by web browsers as they are received, so that the effective delay for the user is determined more by the latency than the total transmission time. Other documents must be fully transmitted before becoming useful. Both kinds of files can be present in a cache. In all these cases, assigning uniform costs or assigning every file's cost to be its size size is not ideal.[1]

This paper: arbitrary sizes, arbitrary costs. This paper presents a simple deterministic on-line algorithm called LANDLORD (shown in Figure 1). LANDLORD handles the problem of file caching with arbitrary costs and integer sizes. The first result is:

THEOREM 1.1. LANDLORD *is* $\frac{k}{k-h+1}$*-competitive for file caching.*

This performance guarantee is the best possible for any deterministic on-line algorithm.[2] File caching is not a special case of the k-server problem, although weighted caching is a special case of both file caching and the k-server problem.

LANDLORD is a generalization of the greedy-dual algorithm [19] for weighted caching, which in turn generalizes least-recently-used and first-in-first-out (paging strategies), as well as the balance algorithm for weighted caching. The analysis uses the potential function $\Phi = (h-1)\sum_{f\in \mathrm{LL}} \mathrm{credit}[f] + k\sum_{f\in \mathrm{OPT}} \mathrm{cost}(f) - \mathrm{credit}[f]$. The analysis is simpler than that of [19] for the special case of weighted caching.

In an independent work [3], Cao and Irani showed that LANDLORD (with step 7 raising credit[g] as much as possible) is k-competitive. They also gave empirical evidence that the algorithm performs well in practice.

This paper: (ϵ, δ)-loosely c-competitiveness. In practice it has been observed that on "typical" request sequences, paging algorithms such as least-recently-used, using a cache of size k, incur a cost within a small constant factor (independent of k) times the minimum possible using a cache of size k [19]. This is in contrast to the theoretically optimal competitive ratio of k. A number of refinements of competitive analysis have been proposed to try to understand the relevant factors. Borodin, Irani, Raghavan, and Schieber [2], in order to model locality of reference, proposed the *access-graph* model which restricts the request sequences to paths in a given graph (related papers include [4, 9, 6]). Karlin, Phillips, and Raghavan [11] proposed a variant in which the graph is a Markov chain (i.e. the edges of the graph are assigned probabilities, and the request sequence corresponds to a random walk) (see also [13]). Koutsoupias and Papadimitriou [12] proposed the *comparative ratio* (for comparing classes of on-line algorithms) and the *diffuse adversary model* (in which the adversary chooses a probability distribution, rather than a sequence, from some restricted class of distributions).

In this paper we introduce a refinement of the aforementioned *loosely competitive* ratio [19] (another previously proposed alternative model). The model is motivated by two observations: In practice, if the retrieval cost is low enough in an *absolute* sense, the competitive ratio is of no concern. For instance, in paging, if the fault rate drops below

$$\frac{\text{time to execute a machine instruction}}{\text{time to retrieve a page from disk}},$$

then the total time to handle page faults ceases to be a bottleneck in the computation. Similar considerations hold for file caching. To formalize this, we introduce a parameter $\epsilon > 0$, and say that "low enough" for a request sequence r means "no more than ϵ times the sum of the retrieval costs" (the sum being taken over all requests). This is tantamount to assuming that handling a file of cost cost(f) requires overhead of ϵ cost(f) whether it is retrieved or not.

The second observation is that in practice, we do not expect the choice of cache size k to be the pessimal one for most of our input sequences. Thus, we are more interested in what happens at a *typical* value of k. To formalize this, we introduce a parameter $\delta > 0$, and

[1] In many applications the actual cost to access a file may vary with time; that issue is not considered here, nor is the issue of cache consistency (i.e., if the remote file changes at the source, how does the local cache get updated? The simplest adaptation of the model here would be to assume that a changed file is treated as a new file; this would require that the local cache strategy learn about the change in some way). Finally, the focus here is on simple *local* caching strategies, rather than distributed strategies in which servers cooperate to cache pages across a network (see e.g. [10]).

[2] Manasse, McGeoch, and Sleator [14] show that no deterministic on-line algorithm for the well-known k-server problem on any metric space of more than k points is better than $\frac{k}{k-h+1}$-competitive. This implies that, at least for any special case when all sizes are 1 (i.e. weighted caching), no deterministic on-line algorithm for file caching is better than $\frac{k}{k-h+1}$-competitive.

Algorithm LANDLORD

Maintain a real value credit[f] with each file f in the cache.
When a file g is requested:
1. **if** g is not in the cache **then**
2. **until** there is room for g in the cache:
3. For each file f in the cache, decrease credit[f] by $\Delta \cdot$ size[f],
4. where $\Delta = \min_{f \in \text{cache}} \text{credit}[f]/\text{size}[f]$.
5. Evict from the cache any file f such that credit[f] $= 0$.
6. Bring g into the cache and set credit[g] $\leftarrow$ cost(g).
7. **else** Reset credit[g] to any value between its current value and cost(g).

Figure 1: The on-line file caching algorithm LANDLORD. Credit is given to each file when it is requested. "Rent" is charged to each file in the cache in proportion to its size. Files are evicted as they run out of credit. Step 7 is not necessary for the worst-case analysis, but it is likely to be important in practice: raising the credit as much as possible in step 7 generalizes the least-recently-used paging strategy; not raising at all generalizes the first-in-first-out paging strategy.

say that a caching strategy is "good enough" if it is good for at least $(1-\delta)n$ of the choices for k in any range $\{1, 2, \ldots, n\}$. These two observations give us the following formalism:

DEFINITION 1.1. *A file caching algorithm A is (ϵ, δ)-*loosely *c-*competitive *if, for any request sequence r and any integer $n > 0$, at least $(1-\delta)n$ of the values $k \in \{1, 2, \ldots, n\}$ satisfy*

$$\text{cost}(A, k, r) \leq \max\left\{c \cdot \text{cost}(\text{OPT}, k, r), \epsilon \cdot \sum_{f \in r} \text{cost}(f)\right\}. \tag{1.1}$$

Here cost(A, k, r) denotes the cost incurred by algorithm A using a cache of size k on sequence r. OPT denotes the optimal algorithm, so that cost(OPT, k, r) is the minimum possible cost to handle the sequence r using a cache of size k. The sum on the right ranges over all requests in r, so that if a file is requested more than once, its cost is counted for each request.

Since the standard competitive ratio grows with k, it is not clear apriori that any on-line algorithm could be (ϵ, δ)-loosely c-competitive for any c that depends only on ϵ and δ. Our second result is the following.

THEOREM 1.2. *Every $\frac{k}{k-h+1}$-competitive algorithm is (ϵ, δ)-loosely c-competitive for any $0 < \epsilon, \delta < 1$ and*

$$c \doteq e\frac{1}{\delta}\left\lceil \ln \frac{1}{\epsilon} \right\rceil.$$

The interpretation is that for *most* choices of k, the retrieval cost is either insignificant or the competitive ratio is constant.

This result supports the intuition that it is meaningful to compare an algorithm against a "handicapped" optimal algorithm (most competitive analyses consider the case $h = k$). A strong performance guarantee, even against a handicapped optimal algorithm, may be as (or more) meaningful than a weak performance guarantee against a non-handicapped adversary.

Our proof is similar in spirit to the proof in [19] for the special case of paging. (The proof here is simpler, more general, and gives a stronger result.)

Of course the following corollary is immediate:

COROLLARY 1.1. LANDLORD *is (ϵ, δ)-loosely c-competitive for $c \doteq e\frac{1}{\delta}\lceil \ln \frac{1}{\epsilon} \rceil$.*

This helps explain why the competitive ratios of the many on-line algorithms that LANDLORD generalizes are typically observed to be constant.

Theorem 1.2 and Corollary 1.1 are tight in the following sense:

CLAIM 1.1. *For any ϵ and δ with $0 < \epsilon, \delta < 1$, there is a constant $c = \Omega(\frac{1}{\delta} \ln \frac{1}{\epsilon})$ such that* LANDLORD *is not (ϵ, δ)-loosely c-competitive.*

The proof will be given in the full paper. For completeness, we also consider randomized algorithms:

CLAIM 1.2. *Every $O\left(\ln \frac{k}{k-h}\right)$-competitive algorithm is (ϵ, δ)-loosely c-competitive for any $0 < \epsilon, \delta < 1$ and*

$$c \doteq O\left(1 + \ln \frac{1}{\delta} + \ln \ln \frac{1}{\epsilon}\right).$$

We also leave the proof of this result to the main paper. (This proof is similar to the proof of Theorem 1.2.)

Since it is shown in [17, 18] that the marking algorithm (a randomized on-line algorithm) is $(1 + 2\ln\frac{k}{k-h})$-competitive for paging, it follows that

CLAIM 1.3. *The marking algorithm is (ϵ,δ)-loosely c-competitive for paging for*

$$c \doteq O\Big(1 + \ln\frac{1}{\delta} + \ln\ln\frac{1}{\epsilon}\Big).$$

We estimate the constant in the big-O to be about $2e$.

2 Proofs of Theorems 1.1 and 1.2.

THEOREM 1.1. LANDLORD *is $\frac{k}{k-h+1}$-competitive for file caching.*

Proof. Define potential function

$$\Phi \doteq (h-1)\cdot\sum_{f\in \text{LL}} \text{credit}[f] + k\cdot\sum_{f\in\text{OPT}} \text{cost}(f) - \text{credit}[f].$$

Here LL denotes the cache of LANDLORD; OPT denotes the cache of OPT. For $f \notin$ LL, by convention credit$[f] \doteq 0$. Before the first request of a sequence, when both caches are empty, Φ is zero. After all requests have been processed (and in fact at all times), $\Phi \geq 0$. Below we show that at each request:

- when OPT retrieves a file of cost c, Φ increases by at most kc;
- when LANDLORD retrieves a file of cost c, Φ decreases by at least $(k-h+1)c$;
- at all other times Φ does not increase.

These facts imply that the cost incurred by LANDLORD is bounded by $k/(k-h+1)$ times the cost incurred by OPT.

The actions affecting Φ following each request can be broken down into a sequence of steps, with each step being one of the following. We analyze the effect of each step on Φ.

- **OPT evicts a file f.**

  Since credit$[f] \leq$ cost(f), Φ cannot increase.

- **OPT retrieves a file g.**

  In this step OPT pays the retrieval cost cost(g).

  Since credit$[g] \geq 0$, Φ can increase by at most $k\cdot$cost(g).

- **LANDLORD decreases credit$[f]$ for all $f \in$ LL.**

  Since the decrease of a given credit$[f]$ is Δ size(f), the net decrease in Φ is Δ times

  $$(h-1)\,\text{size}(\text{LL}) - k\,\text{size}(\text{OPT}\cap\text{LL}),$$

  where size(X) denotes $\sum_{f\in X}$ size(f).

  When this step occurs, we can assume that the requested file g has already been retrieved by OPT but is not in LL. Thus, size(OPT$\cap$LL) $\leq h -$ size(g).

  Further, there is not room for g in LL, so that size(LL) $\geq k -$ size$(g) + 1$ (recall that sizes are assumed to be integers). Thus the decrease in the potential function is at least Δ times

  $$(h-1)(k - \text{size}(g) + 1) - k(h - \text{size}(g)).$$

  Since size$(g) \geq 1$ and $k \geq h$, the expression above is at least $(h-1)(k-1+1) - k(h-1) = 0$.

- **LANDLORD evicts a file f.**

  LANDLORD only evicts f when credit$[f] = 0$. Thus, Φ is unchanged.

- **LANDLORD retrieves the requested file g and sets credit$[g]$ to cost(g).**

  In this step LANDLORD pays the retrieval cost cost(g).

  Since g was not previously in the cache (and credit$[g]$ was zero), and because we can assume that $g \in$ OPT, Φ decreases by $-(h-1)\text{cost}(g) + k\,\text{cost}(g) = (k-h+1)\text{cost}(g)$.

- **LANDLORD resets credit$[g]$ between its current value and cost(g).**

  Again, we can assume $g \in$ OPT. If credit$[g]$ changes, it can only increase. In this case, since $(h-1) < k$, Φ decreases. ◇

THEOREM 1.2. *Every $\frac{k}{k-h+1}$-competitive algorithm is (ϵ,δ)-loosely c-competitive for any $0 < \epsilon,\delta < 1$ and $c \doteq e\frac{1}{\delta}\lceil\ln\frac{1}{\epsilon}\rceil$.*

Proof. Let A be any $\frac{k}{k-h+1}$-competitive algorithm. Let r be any request sequence and $n > 0$ any integer. Fix any $\epsilon,\delta > 0$. Let $c = e\frac{1}{\delta}\lceil\ln\frac{1}{\epsilon}\rceil$.

Our goal is to show that at most δn of the values $k \in \{1,2,\ldots,n\}$ satisfy

$$\text{cost}(A,k,r) > \max\Big\{c(k)\cdot\text{cost}(\text{OPT},k,r), \epsilon\cdot\sum_{f\in r}\text{cost}(f)\Big\}. \tag{2.2}$$

Call the values of k satisfying condition (2.2) "bad" values, and suppose for contradiction that there are more than δn of them. Then there are at least

$$B \doteq 1 + \Big\lfloor\frac{\delta n}{cn/c}\Big\rfloor = 1 + \Big\lfloor\frac{\delta c}{e}\Big\rfloor \geq 1 + \ln\frac{1}{\epsilon} \tag{2.3}$$

bad values $1 \le k_1 < k_2 < \cdots < k_B \le n$ such that for each $i = 2, 3, \ldots, B$,

$$k_i - k_{i-1} + 1 \ge en/c \ge ek_i/c. \tag{2.4}$$

Since A is $\frac{k}{k-h+1}$-competitive, choosing $k = k_i$ and $h = k_{i-1}$ shows that

$$\begin{aligned} \text{cost}(A, k_i, r) &\le \frac{k_i}{k_i - k_{i-1} + 1}\text{cost}(\text{OPT}, k_{i-1}, r) \\ &\le \frac{c}{e}\text{cost}(\text{OPT}, k_{i-1}, r). \end{aligned} \tag{2.5}$$

Since $\text{cost}(A, k_{i-1}, r) \ge c\,\text{cost}(\text{OPT}, k_{i-1}, r)$ (by condition (2.2)), bound (2.5) implies

$$\text{cost}(A, k_i, r) \le \frac{1}{e}\text{cost}(A, k_{i-1}, r). \tag{2.6}$$

From condition (2.2) and induction on bound (2.6) it follows that

$$\begin{aligned} \epsilon \sum_{f \in r} \text{cost}(f) &< \text{cost}(A, k_B, r) \\ &\le \frac{1}{e^{B-1}}\text{cost}(A, k_1, r) \\ &\le \frac{1}{e^{B-1}} \sum_{f \in r} \text{cost}(f). \end{aligned}$$

Thus, $B - 1 < \ln 1/\epsilon$. But this contradicts the choice of B in definition (2.3). ◇

Acknowledgements

Thanks to Dan Gessel for useful discussions and to Pei Cao for pointing out to the author the importance of file size in web caching.

References

[1] *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, El Paso, Texas, 4–6 May 1997.

[2] 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, April 1995.

[3] Pei Cao and Sandy Irani. Cost-aware WWW proxy caching algorithms. In *USENIX Symposium on Internet Technologies and Systems*, December 1997.

[4] Amos Fiat and Anna R. Karlin. Randomized and multipointer paging with locality of reference. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing*, pages 626–634, Las Vegas, Nevada, 29 May–1 June 1995.

[5] Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel D. Sleator, and Neal E. Young. Competitive paging algorithms. *Journal of Algorithms*, 12(4):685–699, December 1991.

[6] Amos Fiat and Ziv Rosen. Experimental studies of access graph based heuristics: Beating the LRU standard? In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 63–72, New Orleans, Louisiana, 5–7 January 1997.

[7] IEEE. *35th Annual Symposium on Foundations of Computer Science*, Santa Fe, New Mexico, 20–22 November 1994.

[8] Sandy Irani. Page replacement with multi-size pages and applications to Web caching. In ACM [1], pages 701–710.

[9] Sandy Irani, Anna R. Karlin, and Steven Phillips. Strongly competitive algorithms for paging with locality of reference. *SIAM Journal on Computing*, 25(3):477–497, June 1996.

[10] David Karger, Eric Lehman, Tom Leighton, Matthew Levine, Daniel Lewin, and Rina Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In ACM [1], pages 654–663.

[11] Anna R. Karlin, Steven J. Phillips, and Prabhakar Raghavan. Markov paging (extended abstract). In *33rd Annual Symposium on Foundations of Computer Science*, pages 208–217, Pittsburgh, Pennsylvania, 24–27 October 1992. IEEE.

[12] Elias Koutsoupias and Christos H. Papadimitriou. Beyond competitive analysis. In *35th Annual Symposium on Foundations of Computer Science* [7], pages 394–400.

[13] Carsten Lund, Steven Phillips, and Nick Reingold. IP over connection-oriented networks and distributional paging. In *35th Annual Symposium on Foundations of Computer Science* [7], pages 424–434.

[14] M. S. Manasse, L. A. McGeoch, and D. D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11:208–230, 1990.

[15] Lyle A. McGeoch and Daniel D. Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6:816–825, 1991.

[16] Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Comm. ACM*, 28(2):202–208, February 1985.

[17] Neal E. Young. Competitive paging and dual-guided algorithms for weighted caching and matching. (Thesis) Tech. Rep. CS-TR-348-91, Computer Science Department, Princeton University, October 1991.

[18] Neal E. Young. On-line caching as cache size varies. In *Proc. of the Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 241–250, 1991.

[19] Neal E. Young. The k-server dual and loose competitiveness for paging. *Algorithmica*, 11(6):525–541, June 1994.