SIAM J. COMPUT.
Vol. 29, No. 5, pp. 1683–1702

ON BROADCAST DISK PAGING*

SANJEEV KHANNA† AND VINCENZO LIBERATORE‡

Abstract. *Broadcast disks* are an emerging paradigm for massive data dissemination. In a broadcast disk, data is divided into n equal-sized pages, and pages are broadcast in a round-robin fashion by a server. Broadcast disks are effective because many clients can simultaneously retrieve any transmitted data. Paging is used by the clients to improve performance, much as in virtual memory systems. However, paging on broadcast disks differs from virtual memory paging in at least two fundamental aspects:

- A page fault in the broadcast disk model has a variable cost that depends on the requested page as well as the current state of the broadcast.
- Prefetching is both natural and a provably essential mechanism for achieving significantly better competitive ratios in broadcast disk paging.

In this paper, we design a deterministic algorithm that uses prefetching to achieve an $O(n \log k)$ competitive ratio for the broadcast disk paging problem, where k denotes the size of the client's cache. We also show a matching lower bound of $\Omega(n \log k)$ that applies even when the adversary is not allowed to use prefetching. In contrast, we show that when prefetching is not allowed, no deterministic online algorithm can achieve a competitive ratio better than $\Omega(nk)$. Moreover, we show a lower bound of $\Omega(n \log k)$ on the competitive ratio achievable by any nonprefetching randomized algorithm against an oblivious adversary. These lower bounds are trivially matched from above by known results about deterministic and randomized marking algorithms for paging. An interpretation of our results is that in the broadcast disk paging, prefetching is a perfect substitute for randomization.

Key words. design of algorithms, online algorithms, competitive analysis, paging, distributed systems, client-server architecture, broadcast disks

AMS subject classifications. 68A10, 68Q25, 68H

PII. S0097539798341399

1. Introduction.

In traditional client-server architectures, such as the World Wide Web, data transfers are initiated by clients that request information from servers. Such an architecture is said to be a "pull" system because clients pull data from the servers. An emerging alternative to pull systems is the "push" technology. In a push system, the server repeatedly broadcasts data to clients; thus the server now "pushes" information toward the clients.

Broadcast disks are a widely used type of push technology. In broadcast disks, data are divided into pages of equal sizes, and pages are broadcast in a round-robin fashion. The name "broadcast disk" derives from the broadcast program being a circular repetition of pages.

1.1. A widespread application.

Broadcast disks have been deployed since the early 80's by most national television companies in western Europe. Broadcast disk

*Received by the editors June 27, 1998; accepted for publication (in revised form) July 3, 1999; published electronically March 17, 2000. A preliminary version of this paper appeared in *Proceedings of the 30th Annual ACM Symposium on the Theory of Computing*, 1998, pp. 634–643.
http://www.siam.org/journals/sicomp/29-5/34139.html

†Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104 (sanjeev@cis.upenn.edu, http://www.cis.upenn.edu/~sanjeev). This work was done while at Bell Labs, Murray Hill, NJ 07974.

‡UMIACS, A. V. Williams Building, University of Maryland, College Park, MD 20742 (vliberatore@acm.org, http://www.umiacs.umd.edu/users/liberato). This work was done while at Rutgers University and Bell Labs and was supported by NSF Career Development Award CCR-9501942, NATO grant CRG 960215, NSF/NIH grant BIR 94-12594-03-CONF, and an Alfred P. Sloan Research Fellowship.

technology has since attained nationwide diffusion and reaches most households. It provides a continuous information source and has deeply influenced the lifestyle of the countries where it is operational [10, 17]. Broadcast disks have been used in high throughput multiprocessor database systems over high bandwidth networks [6] and wireless communication [11]. An interested reader is referred to the survey lecture by Franklin and Zdonik [9] that reports results and research directions in the field of broadcast disks. The client storage resources have been recently integrated in the broadcast disk approach [1, 2]. The client decides which pages to keep in its local cache and which pages to evict. If the client finds a requested page in its local cache, then the page request can be satisfied at no cost. However, if the client does not have the requested page, it will have to wait for that page to be broadcast again by the server. The client's objective is to minimize the completion time needed to satisfy a sequence of page requests. The resulting paging system has some affinity to the traditional virtual memory systems [4]. We refer to it as *broadcast disk paging (BDP)*. The objective of this paper is to study paging algorithms for BDP in the framework of competitive analysis (see [5, 12, 18], for instance).

1.2. Theoretical significance. BDP is not reducible to traditional online paging and poses several unexplored theoretical problems. It differs from traditional paging in at least two main ways:

- The cost of faulting on any page changes dynamically with time in the case of BDP.
- Prefetching dramatically improves the competitive ratio of online algorithms for BDP and brings in the same advantage as randomization.

We elaborate these two points next. In traditional paging models, either the cost of a page fault is uniform [4] or depends only on the faulting page [15]. In contrast, the cost of a fault in BDP is the time spent waiting for the requested page to be transmitted. It depends on the time step reached during the transmission schedule and could range from 1 (the desired page is currently being broadcast) to n (the desired page was broadcast just before), where n is the total number of pages in the server broadcast. As a result, the design and analysis of BDP algorithms involve techniques and ideas that are often significantly different from those for traditional paging. Moreover, the final behavior of the BDP problem is very different from virtual memory paging as illustrated by the following elementary example. Suppose that the server broadcasts n pages and that the client has a local cache of size k. We show that, when $n = k + 1$, there is a 1-competitive deterministic BDP online algorithm. Briefly, the 1-competitive algorithm keeps in the cache all pages except the one that is about to be broadcast. Such an algorithm pays a constant cost per fault, and after at most n faults it will have in the cache exactly the same pages as the adversary. Hence, the algorithm cost is only an additive factor away from the optimum. This is to be contrasted with virtual memory paging where the adversary can force a worst-case competitive ratio of k.

BDP also differs from the traditional paging in the role played by *prefetching*. Prefetching is essentially ruled out from traditional paging without loss of generality [15]. In this paper, we will show that in BDP prefetching is critical to dramatically improve the competitive ratio of online algorithms. In fact, we will show that prefetching is an exact substitute for randomization in BDP. The importance of prefetching can be intuitively explained as follows. Prefetching makes it possible for a client to reload a page recently evicted and, in some sense, allows the client to dynamically revise its eviction decisions.

1.3. New results. In the absence of any prefetching, the conventional caching analysis can be extended to show that any marking algorithm is $O(nk)$-competitive. But things are far from clear when either the adversary or the online algorithm is allowed to prefetch arbitrary subsets of pages. Our main result is that there exists a deterministic online algorithm that uses prefetching and is $O(n \log k)$-competitive. Our algorithm uses the history of the request sequence and prefetching to develop a strategy that dynamically rectifies any potential eviction mistake. We also show that no deterministic online algorithm can achieve a competitive ratio better than $\Omega(n \log k)$. Therefore, our algorithm is optimal up to a constant factor. In contrast, we prove that in the absence of prefetching, no deterministic algorithm can achieve a competitive ratio better than $\Omega(nk)$, and that even with the use of randomization, there is a lower bound of $\Omega(n \log k)$ against oblivious adversaries. The corresponding randomized upper bound is $O(n \log k)$.

Finally, we extend all our results to a more general model, called the *delay model*, where at any step, the adversary generates either a page request or a delay request. The delay requests capture the fact that a client may use the pages in its cache for variable amounts of time, before ever requesting an outside page. The difficulty is that the adversary may introduce delays to disrupt the current state of the online algorithm. But we show that all of our results hold unchanged in the general model.

1.4. Paging and scheduling. In traditional paging, the cost of a page fault depends on the cache configuration. In BDP, the cost depends on a dynamic state that is not confined only to the contents of the local cache but reflects the configuration of other areas of the system. Specifically, the BDP cost is affected by the time reached along the broadcast schedule. In fact, BDP is only one of many problems where there is a strong interaction between caching and schedules. For example, paging can be integrated with prefetching strategies and yields a problem where performance depends on cache contents as well as on the current disk state [7]. We suspect that the results in this paper might shed light also on other online problems where caching is interrelated with scheduling problems.

1.5. Organization. In section 2, we formalize our problem, define our notation, and establish some basic properties of BDP that are useful to our study. In section 3, we study a simple special case of BDP that highlights an important difference between BDP and virtual memory paging. We next study deterministic as well as randomized BDP algorithms that do not use prefetching in section 4. In section 5, we establish our main results, namely, an $O(n \log k)$-competitive deterministic algorithm and an $\Omega(n \log k)$ lower bound on the deterministic competitive ratio for BDP with prefetching. In section 6, we examine the general delay model. We conclude the paper in section 7.

2. Preliminaries. In this section, we formally describe various aspects of BDP and establish our notation. In a broadcast disk, a server broadcasts a set $P = \{0, 1, \ldots, n-1\}$ of pages over a network in a round-robin fashion. The pages are received by the clients in the same order as they are transmitted by the server. Each client operates in perfect isolation from other clients and thus the performance of a client is independent of the behavior of other clients. A client can cache at most k pages. We will typically assume that $a \leq k \leq bn$ for some positive integer a and a positive real constant $b < 1$. The assumption reflects the fact that paging is usually done only if there is enough space in the local cache for at least a small number of pages and that the local cache size is typically much smaller than n. Applications

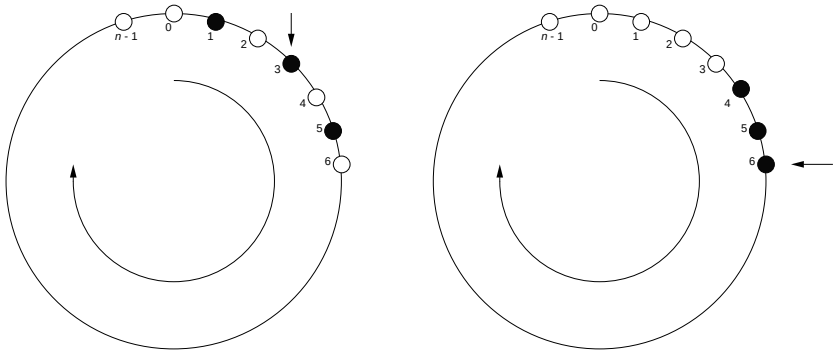

FIG. 2.1. *Examples of client configurations. Black circles represent cached pages and the outer arrow the algorithm position.*

running on the clients generate a sequence σ of page requests. We next define the notion of configuration of a paging algorithm and describe the sequence of actions taken by a client to service a sequence σ of page requests.

2.1. Configuration of an algorithm. The *configuration* of a BDP algorithm G is the contents of its cache along with the step reached along the transmission schedule. Formally, a configuration is a pair $(M, i) \in 2^P \times P$, where

- M is a subset of P of size at most k that is present in G's local cache,
- the index i is the last page that was broadcast and received by the client.

If G's configuration is (M, i), we will say that G is *positioned* over page i or that G's *position* is i. If $M = \{p_1, p_2, \ldots, p_h\}$, then we will denote the memory configuration (M, p_h) as $\{p_1, p_2, \ldots, \triangleright(p_h)\}$.

Example. Figure 2.1(a) represents a broadcast cycle and a configuration (M, i) with $M = \{1, 3, 5\}$ and $i = 3$. The inner arrow represents the order in which pages are broadcast. The outer arrow gives the position of G, that is, the last page that was received by the client. Black circles represent pages in M (contents of G's cache), and white circles represent pages that are not in M.

Let (M, i) be G's configuration. Then, the next page that G will receive is page $i' = (i + 1) \bmod n$. As soon as G has received i', it has the option of loading it into the cache. Thus G's new configuration is of the form (M', i'), where $M' \subseteq M \cup \{i'\}$. Notice that if the cache was full ($|M| = k$) and $i' \in M' - M$, then there must be a page $p \in M - M'$ that is evicted from the cache to make room for i'.

2.2. Page requests. The sequence of page requests generated by a client is denoted by $\sigma = \langle \sigma_1, \sigma_2, \ldots, \sigma_m \rangle \in P^m$. For notational convenience, let $\sigma_0 = k - 1$. The client services a request $\sigma_j \in \sigma$ in an online fashion by repeatedly performing the following sequence of actions: the client receives the next page i from the broadcast and decides whether to cache it or not. This procedure terminates only when the requested page σ_j is in the cache.

Remark. Notice that the client *can* stop the loop iteration when σ_j is in the cache, but it *does not have* to stop the loop the first time σ_j is in the cache. If the client finds it advantageous to keep listening to the broadcast after σ_j is received, it can do so. On the other hand, if σ_j is already in the cache immediately before the jth request is issued, then the client does not have to execute any loop iteration at all. We will prove in Proposition 2.7 that, without loss of generality, clients will stop as soon as σ_j is in the cache.

The cost of a client is the total number of pages it has received from the broadcast, or equivalently, the number of broadcast slots to which the client has listened.

Example. Assume the same set-up as in Figure 2.1(a) and suppose that a request for page 6 is issued. Since page 6 is not in the client's cache, the client listens to the broadcast. Since the client is positioned over page 3, the next page on the broadcast is page 4. The client receives it and decides whether to cache it or not. If it does, it will have to choose a page in $\{1, 3, 5\}$ to evict. In either case, page 6 is not in the cache, so the client keeps listening to the broadcast. Then, page 5 is broadcast. Again, the client can choose whether to cache page 5 or not and continues listening to the broadcast again. Finally, page 6 is broadcast. The client can choose to cache page 6. If it does, it can also choose whether it will keep listening to the broadcast for page 7 or whether it will remain positioned over page 6 and process the next request in σ. Figure 2.1(b) gives an example when pages 4, 5, and 6 have all been cached and the client has stopped immediately after loading page 6. The total cost incurred is three.

Remark. We remark that, according to the previous definition, only the number of received pages contributes to the cost. The definition above entails that when a requested page is in the cache and the client chooses not to move, the cost for that request is zero. In section 6, we will consider a more complex cost model where local computation could cause delays and force the client to skip items in the broadcast. However, for the time being, we will not charge for page requests directly satisfied in the cache.

Paging over a broadcast disk extends the virtual memory, except that now paging is from the network rather than from a physical disk. Clearly, as in virtual memory paging, a page replacement algorithm affects the performance of the system. Virtual memory paging and BDP differ on the following essential point: paging replacement algorithms aim at minimizing the number of page faults (since each fault costs the same), whereas BDP replacement algorithms aim at minimizing the total time spent waiting for pages to arrive from the network.

In what follows, we assume that G is a broadcast disk page replacement algorithm. We will say that G *prefetches* a page p if G loads p even though p is not the currently requested page. In BDP, some pages can be prefetched at no additional cost.

Example. In the example above, page 6 was requested, and page 4 and page 5 were loaded even though they were not requested. Hence page 4 and page 5 were prefetched. Moreover, the client cost is three, independent of whether pages 4 and 5 are prefetched.

We introduce the following notation. The memory configuration reached by G before the jth request will be denoted as $\mathcal{M}(G, j)$ or simply as $\mathcal{M}(G)$ when the time index is clear from the context. Note that the index i in (M, i), and the index j in $\mathcal{M}(G, j) = (M, i)$, denote different quantities. We will assume without loss of generality that the initial configuration of any algorithm G is $\mathcal{M}(G, 1) = \{0, 1, \ldots, \triangleright(k-1)\}$. If $\mathcal{M}(G) = (M, p)$, then we will write $q \in \mathcal{M}(G)$ if and only if $q \in M$. If the jth request $\sigma_j \notin \mathcal{M}(G, j)$, then we will say that G *misses* or *faults* on page σ_j. If $\mathcal{M}(G, j) \neq \mathcal{M}(G, j+1)$, we will say that G *moves* at step j. The algorithm G moves either when it changes cache contents or when it changes position. The cost of G on σ will be denoted as $c(G, \sigma)$ or simply as $c(G)$.

Example. In the example above, G misses and moves from page 3 to page 6. Eventually, G is positioned over page 6.

If $p \in P$ is a page and i a nonnegative integer, then we will write $p + i$ instead of

$(p + i) \bmod n$ when no confusion can arise. If $p, r \in P$ are pages then we will write $p - r$ instead of $(p - r) \bmod n$ when no confusion can arise. Such notation is useful for calculating costs and movements: if G moves from p to q, then its cost is $q - p$, and it is positioned over $q = p + (q - p)$.

2.3. Rotations, transit, and ubiquity. We next define two concepts that we use frequently in our analysis.

DEFINITION 2.1. *The algorithm G executes at least i* rotations *during σ if $c(G, \sigma) \geq i \cdot n$.*

DEFINITION 2.2. *The algorithm G is said to* transit over *page p at step j if it moves from a page $p - h$ to a page $p + l$ ($h > 0, l \geq 0$) at step j.*

Another notion is that of *ubiquitous adversary.* An adversary is said to be ubiquitous if it pays only a unit cost per fault. Since a fault forces any algorithm to receive at least one page, the ubiquitous adversary is at least as strong as an ordinary adversary. We will use the ubiquitous adversary to extend competitiveness results to the delay model. Henceforth, we will assume that the adversary is not ubiquitous unless stated otherwise. A property of the ubiquitous adversary is that it will never prefetch a page [15]. Actually, we could assume without loss of generality that the ubiquitous adversary serves a request sequence in accordance with Belady's algorithm [4], but we will not use this fact.

2.4. Lazy algorithms and hard sequences. We will now define the notions of *lazy algorithms* and *hard sequences* for BDP, compare our definitions with the analogous ones that are given in the context of virtual memory paging, and show that we can assume without loss of generality that algorithms are lazy and sequences hard. First, we define lazy algorithms.

DEFINITION 2.3. *A paging algorithm G is* lazy for virtual memory paging *if G never prefetches a page.*

It can be shown that any algorithm for virtual memory paging can be transformed into a lazy algorithm without any degradation in performance [15]. The definition of lazy algorithms in BDP is as follows.

DEFINITION 2.4. *A paging algorithm G is* lazy for BDP *(or simply* lazy*) if, when G is positioned on page r and faults on page p, its cost is exactly $p - r$.*

In other words, a lazy algorithm stops listening to the broadcast as soon as it receives the faulting page. According to Definition 2.4, a lazy algorithm can prefetch pages as long as those pages are loaded while waiting for the faulting page to be broadcast. Then, a lazy algorithm is not necessarily lazy for virtual memory paging, but a lazy algorithm for virtual memory paging is also lazy for BDP.

DEFINITION 2.5. *A request sequence σ is* hard *for G if G faults on every request in σ.*

The definition above coincides with the one used in virtual memory paging and in BDP. We now claim in the spirit of [15] that it is enough to compare lazy online algorithms to adversaries running lazy algorithms on hard sequences.

LEMMA 2.6. *For any (online) algorithm G, there is an (online) algorithm G' that satisfies $c(G') = c(G)$ and that has the property that, when G' faults on page σ_j, it always loads σ_j the first time G' transits over σ_j.*

Proof. The proof is by induction on the length of σ. Suppose that G satisfies the property before request σ_j, but it violates it on σ_j. The first time G transits over σ_j, it does not load it. Therefore, G will transit over σ_j a second time. Hence, G transits over every page in P after it has transited over σ_j for the first time. The algorithm G' will exactly follow G until page σ_j is received for the first time. Then, G' loads

σ_j and evicts a page p. Afterwards, G' emulates G until either G evicts p, in which case G' evicts σ_j, or G receives p. Notice that G receives p before it receives σ_j for the second time. At that point, G' evicts σ_j and reloads p. Now, the configurations of the two algorithms are identical, and the cost of G' is exactly equal to the cost of G. Finally, we notice that G' does not require any knowledge of the future beyond G's. □

PROPOSITION 2.7. *For any (online) algorithm G, there is an (online) algorithm G' that is lazy and that satisfies $c(G') \leq c(G)$.*

Proof. The proof is by induction on the length of σ. Suppose that G is lazy before request σ_j, but it violates that condition on request σ_j. We can assume without loss of generality that G loads σ_j the first time it transits over it. Afterwards, G will receive an additional set Q of pages. The algorithm G' stops immediately after receiving σ_j. If $j = m$, then G' will not perform any further action and $c(G') < c(G)$. Otherwise, G' services σ_{j+1} by first receiving all pages in Q and then by emulating G on σ_{j+1}. Therefore, $\mathcal{M}(G, j+2) = \mathcal{M}(G', j+2)$ and $c(G') \leq c(G)$. Finally, we notice that G' does not require any knowledge of the future beyond G's. □

PROPOSITION 2.8. *If G is an algorithm that does not move for requests that do not cause a fault, that does not base its eviction decision on nonfaulting requests, and that is c-competitive on all of its hard sequences, then G is c-competitive.*

Proof. Let σ be a sequence of request. Define σ' to be the associated hard sequence, that is, the subsequence of σ consisting of all requests where G faults. Then, $c(G, \sigma) = c(G, \sigma')$. Let H be the optimum algorithm. Then, $c(H, \sigma') \leq c(H, \sigma)$. It follows that, for some constant b,

$$c(G, \sigma) = c(G, \sigma') \leq c \cdot c(H, \sigma') + b \leq c \cdot c(H, \sigma) + b. \qquad \square$$

If G is a lazy algorithm and σ is hard, then G is positioned on σ_{j-1} before the jth request.

3. BDP and virtual memory paging: The case $n = k+1$. In this section, we will analyze the simple case $n = k + 1$ to highlight a fundamental difference between traditional paging and BDP. In traditional paging, the adversary can force a worst-case sequence when $n = k + 1$. In contrast, we show the following result.

PROPOSITION 3.1. *There is a 1-competitive deterministic online algorithm for BDP when $n = k + 1$.*

Proof. Let G be the algorithm that on a faulting request σ_j maintains $\mathcal{M}(G, j+1) = \{\triangleright(\sigma_j)\} \cup \{i \in P : i \neq \sigma_j + 1\}$. In other words, $\sigma_j + 1$ is the only page missing from G's fast memory. When G faults on σ_j, it evicts $\sigma_j + 1$ and loads σ_j. So, G pays a unit cost whenever it faults. Suppose without loss of generality that σ is hard for G, and thus $c(G, \sigma) = m$. In fact, $\sigma_j = k + j$ by induction on j.

Assume without loss of generality that the adversary follows a lazy algorithm H. Suppose that H faults on $\sigma_{j_1}, \sigma_{j_2}, \ldots, \sigma_{j_h}$. The cost of H on σ is then

$$c(H, \sigma) = \sum_{l=1}^{h} (\sigma_{j_l} - \sigma_{j_{l-1}}) = \sum_{l=1}^{h} (k + j_l - k - j_{l-1}) = \sum_{l=1}^{h} (j_l - j_{l-1}) = j_h - 1.$$

Notice that any subsequence of at most n consecutive requests in σ consists of distinct pages. Also, H does not fault on σ_j when $j > j_h$, and so $|\{\sigma_{j_h+1}, \ldots, \sigma_m\}| \leq k-1 < n$, which implies that $\sigma_{j_h+1}, \ldots, \sigma_m$ are all distinct and $m - j_h \leq k - 1$. Therefore, $c(G, \sigma) - c(H, \sigma) = m - (j_h - 1) \leq k$, and the claim is proven. □

The case $n = k + 1$ sets up the general framework for finding lower bounds on the competitive ratio of traditional paging algorithms. Clearly, the general paradigm does not work for BDP. Instead, we will show that all lower bounds can be ultimately traced back to the notion of algorithm in transit, as defined in the previous section.

4. BDP algorithms without prefetching. In this section, we will examine the competitive ratio of online algorithms that do not allow prefetching. In terms of Definition 2.3, such algorithms are lazy for virtual memory paging.

If G is a lazy algorithm for virtual memory paging, it induces an algorithm for BDP that is lazy and whose memory contents after σ_j are exactly the same as G's. For simplicity, we will denote both the virtual memory paging algorithm and its BDP counterpart with the same symbol. A strongly competitive deterministic paging algorithm never incurs more than k times the optimum number of page faults. Since its BDP counterpart is lazy, it follows that a page fault costs at most $n - 1$; the last page that was broadcasted and received must be in the algorithm's cache. Thus, any strongly competitive paging algorithm, e.g., the marking algorithm [13], is trivially $(k(n-1))$-competitive for broadcast disk paging even against a ubiquitous adversary. In fact, we can show a slightly better competitive ratio.

THEOREM 4.1. *Let G be a lazy α-competitive deterministic algorithm for paging. Then, the competitive ratio of G for BDP is $(\alpha - 1)n + 1$.*

Proof. We will assume without loss of generality that the sequence σ is hard for G and that the adversary uses a lazy algorithm H. First, we notice that the cost of H is at least equal to the number of page faults. Indeed, every time H brings a new page into fast memory, H has to transit over that page, paying a unit cost.

Let r_j be the page where the adversary is positioned before the jth request, and define the potential function as $\Phi(j) = r_j - \sigma_{j-1}$. Let $a_j = (\sigma_j - \sigma_{j-1}) + \Phi(j+1) - \Phi(j)$ be the amortized cost of G to serve request σ_j. Notice that if $r_{j+1} = r_j$, then $a_j = (\sigma_j - \sigma_{j-1}) + (r_j - \sigma_j) - (r_j - \sigma_{j-1}) \leq n$. If $\sigma_j \notin \mathcal{M}(H, j)$, then $\Phi(j+1) = 0$ and $a_j = \sigma_j - \sigma_{j-1} + \Phi(j+1) - \Phi(j) = (\sigma_j - r_j) + (r_j - \sigma_{j-1}) - (r_j - \sigma_{j-1})$, which is the real cost of the optimum on that request. Let l be the number of times that the optimum algorithm faults. Then $|\sigma| \leq \alpha l + b$ for some constant b. Moreover, $l \leq c(H, \sigma)$. The resulting cost of G is $c(G, \sigma) = (|\sigma| - l)n + c(H, \sigma) \leq ((\alpha - 1)l + b)n + c(H, \sigma) \leq (n(\alpha - 1) + 1)c(H, \sigma) + bn$, which proves the theorem. ∎

COROLLARY 4.2. *The marking algorithm is $((k-1)n + 1)$-competitive.*

We next establish an essentially matching lower bound.

THEOREM 4.3. *No deterministic online algorithm without prefetching can have a competitive ratio better than $\Omega(nk)$ for $2 \leq k \leq n - 2$ and any fixed $b < 1$, even if the adversary is not allowed to do prefetching.*

Proof. Let G be an online algorithm without prefetching and H be the algorithm used by the adversary. The adversary proceeds in phases. We will maintain that, at the beginning of a phase, $\mathcal{M}(G)$ contains the same pages as $\mathcal{M}(H)$, and that H is positioned on a page q such that $q + 1, q + 2 \notin \mathcal{M}(H)$. Define the set $W = \mathcal{M}(H) \cup \{q + 1, q + 2\}$ and call W the *working set* of the current phase. The gist of the proof is as follows. Notice that $|W| = k + 2$, and so there are always two pages in the working set W that are not in G's fast memory. The adversary will request those pages, and consequently G will fault at every step. Furthermore, we will use the notion of transit to show that G pays $\Omega(n)$ on every two faults. The adversary will generate $\Omega(k)$ requests in the phase and itself serves them at a cost of two. We now detail the argument.

The adversary starts the phase by requesting page $q + 1$ followed by page $q + 2$.

Thus, both G and H are now positioned at $q+2$. The rest of the phase is divided into segments. We will let r be the page where G is positioned at the beginning of a current segment. For example, in the first segment, $r = q+2$. Let α and β be the two pages that are in the working set W but not in G's fast memory at the beginning of a segment. Assume without loss of generality that $\beta - r > \alpha - r$. Then, the adversary requests β followed by α, and the segment is over. The algorithm G faults on the request for β and transits over α while waiting for β to be transmitted; but it does not load α because it is not allowed to prefetch. Therefore, G faults also on the request for α. On the whole, G transits over α at least twice during the segment, and so it executes at least one rotation per segment. We will now describe how many segments are generated. The adversary counts the number of distinct pages that made G fault during all the segments generated so far. The adversary generates segments until G has faulted on a set F of at least $k-2$ distinct pages. Notice that F is contained in the working set. Moreover, during each segment, G faults on at most two new pages, so that $|F| \in \{k-2, k-1\}$. Hence, at least $\lceil (k-2)/2 \rceil$ segments are generated. We will now specify how H serves the requests in the phase. The algorithm H faults on $q+1$ and $q+2$. At this point, its fast memory contains all the pages of $F \cup \{q+2\}$, plus another arbitrary page from W if $|F \cup \{q+2\}| < k$. Notice that H does not prefetch any page and that H does not fault during any segment. If G and H's fast memory configurations still differ after all segments have been generated, the adversary keeps requesting pages in H's memory that are missing from G's memory, until the two memories coincide in their contents. Notice that if the two memories never coincide, then this step will continue indefinitely, and G is not competitive. Henceforth, assume that G's and H's memory contents will eventually be the same. At this point, the phase is over. The cost of G in the phase is at least the cost in the segment, and so hence at least $n(k-2)/2$. The cost of H in the entire phase is 2—due only to the initial faults on $q+1$ and $q+2$.

The adversary issues $(n-k)/2$ request phases paying only a cost of 2 per phase. Then the adversary pays a cost of k and returns to the starting configuration $\{0, 1, \ldots, \triangleright(k-1)\}$. It then keeps requesting pages $0, 1, \ldots, k-1$ until the fast memory configurations of G and H coincide. Therefore, the cost of the adversary on $(n-k)/2$ phases is n, while the cost of G is at least $((n-k)/2)(n(k-2)/2)$, and the result follows. □

The construction above extends to a lower bound for randomized algorithms against an adaptive online adversary.

THEOREM 4.4. *No randomized online algorithm without prefetching can have a competitive ratio better than $\Omega(nk)$ against an adaptive online adversary for $3 \le k \le bn$ and any fixed $b < 1$, even if the adversary is not allowed to do prefetching.*

Proof. The initial setup of the proof is the same as that of Theorem 4.3. The main difference between the proofs of Theorem 4.3 and Theorem 4.4 is that, at the beginning of a phase, the adversary does not know the set F of pages that make G fault. We overcome the difficulty by letting the adversary guess F and showing that the expected number of segments is $\Omega(k)$.

The adversary proceeds in phases. Assume that j is the step of the first phase request, so that $\mathcal{M}(H, j)$ and $\mathcal{M}(G, j)$ denote the configurations of H and G immediately before the current phase begins. We will maintain that $\mathcal{M}(G, j)$ contains the same pages as $\mathcal{M}(H, j)$, and that H is positioned on a page q such that $q+1, q+2 \notin \mathcal{M}(H)$. Define the set $W = \mathcal{M}(H, j) \cup \{q+1, q+2\}$ and call W the working set of the current phase. The adversary starts the phase by requesting $q+1$

followed by $q+2$. Thus, both G and H are now positioned on $q+2$. The algorithm H makes room for $q+1$ and $q+2$ by evicting two random pages r_1 and r_2 from $\mathcal{M}(H,j)$, so that now H's configuration is $\mathcal{M}(H) = (\mathcal{M}(H) - \{r_1, r_2\}) \cup \{q+1, \triangleright(q+2)\}$. Notice that $r_1, r_2 \in \mathcal{M}(H,j)$ implies $\{r_1, r_2\} \cap \{q+1, q+2\} = \emptyset$, a fact that we use later on.

The rest of the phase is divided into segments. A segment is defined exactly as in the proof of Theorem 4.3 and consists of two page requests that force G to execute one rotation. Both segment requests cause a fault for G but no fault for H. Therefore, the adversary generates segments while $|\mathcal{M}(H) - \mathcal{M}(G)| \geq 2$. Notice that the adversary is adaptive, and so it can determine if $|\mathcal{M}(H) - \mathcal{M}(G)| \geq 2$, which pages to request in each segment, and in which order. However, the adversary cannot determine at the beginning of a phase the set F of pages that will make G fault. It remains to estimate the expected number of segments. At all steps, G keeps all the pages in the working set W except two. Let α and β be the two pages that are in the working set but not in G's fast memory immediately after the ith segment has been serviced. Notice that the adversary generates an $(i+1)$st segment only if $|\mathcal{M}(H) - \mathcal{M}(G)| \geq 2$, or, equivalently, only if $\{\alpha, \beta\} \cap \{r_1, r_2\} = \emptyset$. Therefore, G should choose α and β so that $\{\alpha, \beta\} \cap \{r_1, r_2\} \neq \emptyset$. In other words, G should choose the missing pages α and β in order to guess H's missing pages r_1 and r_2. Since $\{r_1, r_2\} \cap \{q+1, q+2\} = \emptyset$, we assume without loss of generality that G always keeps $q+1$ and $q+2$ in fast memory, and so $\{\alpha, \beta\} \cap \{q+1, q+2\} = \emptyset$. We will now estimate the expected number of segments by means of the following simple random experiment. We will have a bin that contains k balls—a white ball for each page in $\mathcal{M}(H,j) - \{r_1, r_2\}$ and a black ball each for r_1 and r_2. In each segment, G extracts two balls α and β from the bin. If both α and β are white, then G gets another round; otherwise, either α or β are black and the experiment stops. The number of segments ℓ is exactly the number of rounds needed to extract a black ball. Clearly, ℓ is minimum if no ball is ever replaced in the bin. Elementary combinatorics now yields $E[\ell] = \Theta(k)$. As an aside, we observe that the case when balls are not replaced in the bin corresponds to the case when G keeps in fast memory a different subset of the working set in different segments.

After the last segment, $|\mathcal{M}(H) - \mathcal{M}(G)| \leq 1$. If $|\mathcal{M}(H) - \mathcal{M}(G)| = 0$, then $\mathcal{M}(G)$ contains the same pages as $\mathcal{M}(H)$, and another phase starts. If $|\mathcal{M}(H) - \mathcal{M}(G)| = 1$, then the adversary keeps requesting pages in $\mathcal{M}(H) - \mathcal{M}(G)$ until $\mathcal{M}(G)$ contains the same pages as $\mathcal{M}(H)$. Again, the adversary is adaptive and knows if $\mathcal{M}(G)$ coincides with $\mathcal{M}(H)$. There are two cases, depending on whether this step ends or not. The expected cost of G conditioned to the event that $\mathcal{M}(G)$ never coincides with $\mathcal{M}(H)$ is infinite, and so G is not competitive. The expected cost of G conditioned to the event that $\mathcal{M}(G)$ eventually coincides with $\mathcal{M}(H)$ is nonnegative. We will assume from now on that indeed G's memory will eventually coincide with H's. After $\mathcal{M}(G)$ and $\mathcal{M}(H)$ contain the same pages, a new phase starts. The expected cost of G in a phase is at least equal to the expected cost in the segments, and so it is at least $nE[\ell] = \Omega(nk)$. Meanwhile, the cost of H in the entire phase is due only to the initial faults on $q+1$ and $q+2$, and so it is 2. The adversary issues $(n-k)/2$ request phases paying only a cost of two per phase. Then the adversary pays a cost of k and returns to the starting configuration $\{0, 1, \ldots, \triangleright(k-1)\}$. It will then keep requesting pages $0, 1, \ldots, k-1$ until the fast memory configurations of G and H coincide. Therefore, the cost of the adversary on $(n-k)/2$ phases is n, while the expected cost of G is $\Omega(nk(n-k))$, and the result follows. □

We next establish an $\Omega(n \log k)$ lower bound on the competitive ratio of randomized online algorithms without prefetching against an oblivious adversary. We start with the following simple proposition.

LEMMA 4.5. *Let G be a deterministic lazy algorithm, $\sigma = (\langle\sigma_1, \sigma_2, \ldots \sigma_l\rangle)^h$ for some $l \leq k$. If $\{\sigma_1, \sigma_2, \ldots, \sigma_l\} - \mathcal{M}(G, lh) \neq \emptyset$, then $c(G, \sigma) \geq h - 1$ for all configurations $\mathcal{M}(G, 1)$.*

Proof. The proof is by induction on c. The claim clearly is true for $c = 1$. Let $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_l\}$ and $\sigma' = \langle\sigma_1, \sigma_2, \ldots, \sigma_l\rangle$. Suppose that $\Sigma - \mathcal{M}(G, l(h)) \neq \emptyset$. Since G is lazy, $\mathcal{M}(G)$ will remain unchanged as soon as $\Sigma \subseteq \mathcal{M}(G)$. Then, $\Sigma - \mathcal{M}(G, l(h-1)) \neq \emptyset$, and by induction hypothesis $c(G, (\sigma')^{h-1}) \geq h-2$. Moreover, G faults at least once during the last repetition of σ', and the lemma follows. □

THEOREM 4.6. *No randomized online algorithm without prefetching can have a competitive ratio better than $\Omega(n \log k)$ for $3 \leq k \leq bn$ and any fixed $b < 1$, even if the adversary is not allowed to do prefetching.*

By the minimax theorem [5], it will be enough to show that there is a probability distribution $\mathcal{P}$ over sequences of page requests such that for any lazy deterministic algorithm G, $E_{\mathcal{P}}\{c(G, \sigma)\} \geq c \cdot c(H, \sigma)$ where $c = \Omega(n \log k)$ and H is an optimal offline strategy.

Proof. The main difference between the proofs of Theorem 4.4 and Theorem 4.6 is that the adversary cannot foresee what pages α and β are in the working set but not in G's fast memory. We sidestep the difficulty by showing that with some probability the adversary is still able to force G to execute one rotation.

The request sequence. The page request sequence σ is of the form $\sigma = (\Gamma_1 \ldots \Gamma_{(n-k)/2}\Gamma_0)^l$, where l is a positive integer, $\Gamma_0 = \langle 0, 1, \ldots, k-1\rangle$, and Γ_i $(1 \leq i \leq (n-k)/2)$ will be defined later. Roughly speaking, the subsequences Γ_i for $i \geq 1$ are the analogues of phases in the previous proofs, and Γ_0 corresponds to the final coordination step that brings about the initial configuration $\{0, 1, \ldots, \triangleright(k-1)\}$. We will now describe the phase Γ_i, a few quantities determined by the Γ_i's, and the value of these quantities at the beginning of Γ_i. A phase Γ_i depends on the set S_{i-1}, which is defined recursively: let $S_0 = \{0, 1, \ldots, k-1\}$, and S_i contains all the pages requested during the phase Γ_i. Throughout we will maintain the following properties:

- $|S_i| = k$ for $1 \leq i \leq (n-k)/2$.
- H holds S_{i-1} in fast memory immediately before the start of phase Γ_i.
- The pages $q_i + 1, q_i + 2 \notin S_{i-1}$, where $q_i = k - 1 + 2i$. In other words, we maintain that $q_i + 1$ and $q_i + 2$ are not in H's fast memory immediately before phase Γ_i starts. For example, k and $k+1$ are not in H's fast memory immediately before Γ_1.

Define the working set in phase Γ_i to be the set $W_i = S_{i-1} \cup \{q_i + 1, q_i + 2\}$. Let $\gamma_1, \gamma_2, \ldots, \gamma_k$ be a random k-permutation of the working set W_i; define $S_i = \{\gamma_1, \gamma_2, \ldots, \gamma_k\}$. The basic plan of the adversary is to request the pages in S_i in the given order $\gamma_1, \gamma_2, \ldots, \gamma_k$. However, the adversary will also request a sequence of other pages between the first request for a page γ_j and the first request for page γ_{j+1}, $1 \leq j \leq (k-1)$, in order to force G's memory configuration. Specifically, $\Gamma_i = \rho_0\langle\gamma_1\rangle\rho_1\langle\gamma_2\rangle\rho_2 \ldots \langle\gamma_k\rangle$, where

- ρ_0 is a repetition for $c+1$ times of S_{i-1}, and
- $\rho_j = (\langle\gamma_1, \gamma_2, \ldots, \gamma_j\rangle)^{c+1}$, where $1 \leq j \leq k$.

We will now turn to estimate the expected cost of G during Γ_i conditioned to the event that either (i) a page in S_{i-1} is not in G's cache immediately before the first request to γ_1, or (ii) one of the $\gamma_1, \ldots, \gamma_{j-1}$ is not in G's cache immediately before the first request to γ_j $(2 \leq j \leq k)$. Suppose that there is a page in S_{i-1} missing from G's cache immediately before the request to γ_1. By the previous lemma, G pays at least $c = \Omega(n \log k)$ on ρ_0. Analogously, suppose that there is one of $\gamma_1, \ldots, \gamma_{j-1}$

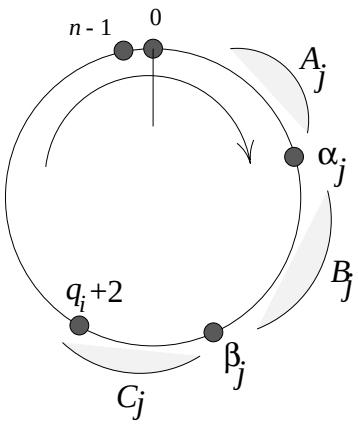

FIG. 4.1. *Lower bound for randomized algorithms without prefetching.*

missing from G's cache immediately before the request to γ_j $(1 \leq j \leq k)$. By the previous lemma, G pays at least $c = \Omega(n \log k)$ on ρ_{j-1}. Therefore, $\Omega(n \log k)$ bounds G's expected cost conditioned to the event that either (i) or (ii) occurred. It remains to show that G's expected cost is $\Omega(n \log k)$ even when neither (i) nor (ii) occurred. We will condition the rest of the analysis to such event, and so we will assume that G has $\gamma_1, \ldots, \gamma_{j-1}$ in fast memory before the request for γ_j where $j = 2, \ldots, k$, and G has all the elements in S_{i-1} before γ_1.

Costs of G and H. The adversary's algorithm H keeps S_i in fast memory throughout Γ_i. Therefore, H pays a cost of at most n on the sequence $\Gamma_1 \ldots \Gamma_{(n-k)/2}\Gamma_0$, exactly as in the previous proofs. We will now show that the expected cost of G on Γ_i $(1 \leq i \leq (n-k)/2)$ is $\Omega(n \log k)$. Let

$$t_j = \begin{cases} 1 & \text{if } G \text{ transits over page 0 while serving } \gamma_j, \\ 0 & \text{otherwise.} \end{cases}$$

The quantity t_j will be referred to as the *transit cost* of G for γ_j. Define $t = \sum_{j=1}^{k} t_j$ to be the transit cost of G in the phase Γ_i. Observe that G executes at least $t-1$ rotations during Γ_i, and so G's cost during Γ_i is at least $n(t-1)$. Hence, it is enough to show that $t = \Omega(\log k)$.

Before estimating t, we will make some definitions and observations. Let Y_{ij} be the set of pages in W_i that have not been requested prior to the first request for γ_j, that is, $Y_{ij} = W_i - \{\gamma_1, \ldots, \gamma_{j-1}\}$, $j = 1, \ldots, k$. Then by assumptions (i) and (ii) above, there are at least two pages in Y_{ij} that are missing from G's fast memory before γ_j is requested. Let $\alpha_j, \beta_j \in Y_{ij}$ be two such pages and assume without loss of generality that $\alpha_j < \beta_j$, as depicted in Figure 4.1. Notice that if G does not fault on γ_j, then G does not move and $\alpha_{j+1} = \alpha_j$ and $\beta_{j+1} = \beta_j$. Let r_j be the position taken by G before the request for γ_j. It can be seen that $r_j \in W_i$ by induction on j. Moreover, G is lazy and so $r_j \neq \alpha_j, \beta_j$. As depicted in Figure 4.1, let

- $A_j = \{p \in W_i : p < \alpha_j\}$,
- $B_j = \{p \in W_i : \alpha_j < p < \beta_j\}$, and
- $C_j = \{p \in W_i : \beta_j < p\}$.

Notice that $r_j \in A_j \cup B_j \cup C_j$.

We now turn to estimate the transit cost t_j. Define the potential of G immediately before γ_j as

$$\Phi(j) = \begin{cases} \frac{1}{2} & \text{if } r_j \in B_j \cup C_j, \\ 0 & \text{otherwise} \end{cases}$$

and the amortized transit cost of G as $t_j + \Phi(j+1) - \Phi(j)$. We will show that the expected amortized transit cost of G during Γ_i is $\Omega(\log k)$. Since the real transit cost of G is equal to the amortized transit cost minus $O(1)$, we obtain that the real transit cost of G during Γ_i is $t = \Omega(\log k)$. Putting together, it will then follow that the actual cost of G during Γ_i is $\Omega(n \log k)$.

If $r_j \in A_j$ and $\beta_j = \gamma_j$, then the potential increases by $1/2$. Moreover, $\beta_j = \gamma_j$ with probability $1/|Y_{ij}| = 1/(k+3-j)$, so that G's expected amortized transit cost when $r_j \in A_j$ is at least $(1/2)/(k+3-j)$. If $r_j \in B_j \cup C_j$ and $\alpha_j = \gamma_j$, then G pays a real transit cost at least equal to 1, and the potential drops by no more than $1/2$. Moreover, $\alpha_j = \gamma_j$ with probability $1/|Y_{ij}| = 1/(k+3-j)$, so that G's expected amortized transit cost when $r_j \in B_j \cup C_j$ is at least $(1/2)/(k+3-j)$. The expected amortized transit cost of G during Γ_i is at least

$$\sum_{j=1}^{k} \left(\frac{1}{2} \frac{1}{k+3-j} \right) = \frac{1}{2} \sum_{j=3}^{k+2} \frac{1}{j} = \Omega(\log k).$$

Consequently, $t = \Omega(\log k)$, and the actual cost of G during Γ_i is $\Omega(n \log k)$.

Finally, we recall that on $\Gamma_1 \Gamma_2 \ldots \Gamma_{(n-k)/2} \Gamma_0$, the cost of H is n, and the cost of G is $\Omega(n^2 \log k)$, which proves the theorem. □

Conversely, the competitive randomized algorithms in [3, 8, 16] immediately imply an $O(n \log k)$-competitive randomized algorithm for BDP even against a ubiquitous adversary.

5. BDP algorithms with prefetching. In this section, we will establish our main theorems. We will first show a lower bound of $\Omega(n \log k)$ on the competitive ratio of any deterministic online algorithm that uses prefetching, even when compared to an adversary that does not do any prefetching. We will then present a deterministic online algorithm that achieves a competitive ratio of $O(n \log k)$ and therefore is optimal up to a constant factor.

5.1. A lower bound. We first prove the lower bound.

THEOREM 5.1. *No deterministic algorithm for BDP can achieve a competitive ratio better than $\Omega(n \log k)$ when $3 \le k \le bn$ for any fixed $b < 1$ even if the adversary is not allowed to do prefetching.*

Proof. The main difference between the present proof and that of Theorem 4.3 is that G might reload α when it faults on β, and so the online algorithm cannot be made to execute one rotation every other request. In fact, we will typically need several requests to have the online algorithm complete a rotation. The basic plan of the adversary is as follows: repeatedly request a page that is not in the online algorithm G's fast memory and is farthest from the page where G is positioned. We will assume without loss of generality that G is lazy. Let H be the algorithm that the adversary uses to satisfy the request sequence.

The request sequence. The adversary proceeds in phases. We will maintain that, at the beginning of a phase, G and H hold the same set of pages W in fast memory and that H is positioned on a page q such that $q+1, q+2 \notin W$. In this proof, the set W will take the function that the working set had in the previous proofs, namely, we will extract requests from W in order to make G fault. The adversary starts the phase by requesting page $q+1$ followed by page $q+2$. We will maintain that there are

always at least two pages in W that G does not have. The adversary will request one of those pages, and if r is the page where G is currently positioned, the adversary will issue a request for the missing page α that maximizes the quantity $\alpha - r$. However, the adversary will also insert other page requests to force G's fast memory configuration. On the whole, the phase has the form $\langle q+1, q+2\rangle \rho_1 \langle \gamma_1 \rangle \rho_2 \langle \gamma_2 \rangle \dots \rho_{k-2} \langle \gamma_{k-2} \rangle$, where

- γ_j is the page in W that is not in G's fast memory, and that is farthest from the current position r of G, $1 \le j \le k-2$,
- ρ_j denotes the sequence of requests $\rho_j = (\langle q+1, q+2, \gamma_1, \dots, \gamma_{j-1}\rangle)^{c+1}$, where $c = \Omega(n \log k)$ and $1 \le j \le k-2$.

Suppose that G and H do not have the same set of pages in fast memory after the request for γ_{k-2}. In that case, the adversary keeps requesting a page that H has, but G does not. Eventually, G and H will have the same set of pages in fast memory, or else G is not competitive, and another phase starts.

Costs of G and H in each phase. Let $R_j = \{q+1, q+2, \gamma_1, \dots, \gamma_{j-1}\}$ denote the set of pages requested in the sequence ρ_j. Notice that R_j is precisely the set of pages requested since the beginning of the phase until the request γ_j, and that $R_j \subset R_{j+1}$. The algorithm H keeps R_{k-1} throughout the phase and pays a cost of two. We can assume the following about the behavior of the algorithm G by Lemma 4.5: (i) G has R_j in its fast memory before the request for γ_j, and (ii) G does not move at all during the ρ_j's, but that it moves only to service the requests for the γ_j's. Thus from here on we can simply assume that a phase merely consists of the sequence of requests $\langle q+1, q+2, \gamma_1, \gamma_2, \dots, \gamma_{k-2}\rangle$. By this assumption, γ_{j+1} is the page in W that is not in G's fast memory and that maximizes $\gamma_{j+1} - \gamma_j$.

We will now show that the cost of G is $\Omega(n \log k)$ on the sequence $\Gamma = \langle q+1, q+2, \gamma_1, \gamma_2, \dots, \gamma_{k-2}\rangle$. To estimate G's cost, we will divide a phase into subphases with the property that G executes one rotation per subphase. Then, it will be enough to count the number of subphases. Specifically, we define subsequences $Z_1, Z_2, \dots, Z_t$ such that

- $Z_1 Z_2 \dots Z_t = \Gamma$, and
- $Z_1 Z_2 \dots Z_i$ is the smallest prefix of Γ on which G completes i rotations ($1 \le i < t$).

Since G executes at least $t-1$ rotations in the phase, it suffices to show that $t = \Omega(\log k)$. The idea is to show that $|Z_i|$ decreases as i increases, and then it will follow that t is large. Let x_i denote the number of pages in W that have not been requested prior to the start of the subsequence Z_i. Then we claim that $|Z_i| \le \lceil (x_i+1)/2 \rceil$. This easily follows from the observations that at each step G has at least 2 pages missing from the set W and we always request the missing page that is farthest. Thus G must complete a rotation after $\lceil (x_i+1)/2 \rceil$ requests.

Observe that $x_1 = k$ and that upon termination, $x_{t+1} = 2$, and so $2 \le x_i \le k$ for all $i = 1, 2, \dots, t+1$. Furthermore, $x_{i+1} = x_i - |Z_i| \ge \lfloor (x_i - 1)/2 \rfloor \ge (x_i/2) - 1$. It is then straightforward to see that $t = \Omega(\log k)$. The cost of G during a phase is thus at least $nt = \Omega(n \log k)$.

Putting it all together. Observe that the adversary can issue $(n-k)/2$ consecutive phases. Then, it will reload $\{0, 1, \dots, \triangleright(k-1)\}$ and another sequence of $(n-k)/2$ phases starts. The total cost of the adversary in such a sequence of phases is n while the total cost of G is $\Omega(n^2 \log k)$, and the result follows. □

5.2. An upper bound: The gray algorithm. We will now design a paging algorithm $\mathcal{G}$, referred to as the *gray algorithm*, that is $O(n \log k)$-competitive. The gray algorithm uses a set of three marks {black, gray, white} and maintains a mark

for each page in P. Notice that $\mathcal{G}$'s marking policy is somewhat different from that of the marking algorithm [13] that uses only two marks and marks only the pages in fast memory. We define b_j to be the number of black pages immediately before the jth request. The gray algorithm $\mathcal{G}$ works as follows:

- Initially, $\mathcal{G}$ marks the pages $\{0, 1, \ldots, k-1\}$ black and all other pages white.
- $\mathcal{G}$ ignores all requests that do not cause a fault. Henceforth, assume that σ is hard for $\mathcal{G}$.
- $\mathcal{G}$ works in phases:
  - A new phase is started when $b_j = k$.
  - At the beginning of a phase, $\mathcal{G}$ marks all gray pages white and all black pages gray.
- When $\mathcal{G}$ faults on σ_j, $\mathcal{G}$ loads σ_j and marks it black.
- Before the jth request, $\mathcal{G}$ keeps in its fast memory
  - the b_j black pages, plus
  - the set of $q_j = k - b_j$ gray pages $\gamma_1, \gamma_2, \ldots, \gamma_{q_j}$ that have the q_j smallest values of $\sigma_{j-1} - \gamma_i$ $(i = 1, 2, \ldots, q_j)$. In other words, the gray pages $\gamma_1, \gamma_2, \ldots, \gamma_{q_j}$ are the q_j gray pages that are most expensive to reload from σ_{j-1}.

We have stated the gray algorithm by assuming that a mark is associated with all n pages. In fact, $\mathcal{G}$ needs only to keep track of the black and gray pages, and a straightforward induction shows that, at any step, there are at most $2k$ gray and black pages.

It is not obvious that the gray algorithm could be implemented to maintain the prescribed set of gray pages without paying a large overhead. The following lemma shows that $\mathcal{G}$ is lazy.

LEMMA 5.2. *The gray algorithm $\mathcal{G}$ is lazy.*

Proof. Let σ be a hard sequence for $\mathcal{G}$. Suppose that $p \in \mathcal{M}(\mathcal{G}, j+1) - \mathcal{M}(\mathcal{G}, j)$ and $p \neq \sigma_{j+1}$. First, we show that p is gray at both step j and $j+1$. Since $p \notin \mathcal{M}(\mathcal{G}, j)$, p is not black at step j, and since $p \neq \sigma_{j+1}$, p is not black at step $j+1$. Since $p \in \mathcal{M}(G, j+1)$, p is not white at step $j+1$, and so it is not white at step j. It follows that p is gray at both steps. If p is in the interval from σ_j to σ_{j+1}, then it can be prefetched at no cost while $\mathcal{G}$ moves from σ_j to σ_{j+1}. We now show that no page that lies in the interval from σ_{j+1} to σ_j is prefetched by $\mathcal{G}$. Suppose, by contradiction, that such a page p is prefetched. First, we observe that σ_{j+1} does not start a new phase. Indeed, if σ_{j+1} started a new phase, then $\mathcal{M}(\mathcal{G}, j)$ is exactly the set of gray pages immediately before step $j+1$. However, p is gray and $p \notin \mathcal{M}(\mathcal{G}, j)$. We conclude that σ_{j+1} did not start a new phase. Therefore, $b_{j+1} = b_j + 1$ and so $q_{j+1} = q_j - 1$. Let ζ_j be the number of gray pages remaining after the jth request and notice that $\zeta_{j+1} \geq \zeta_j - 1$. Since $p \in \mathcal{M}(\mathcal{G}, j+1)$, p has one of the q_{j+1} largest values of $p - \sigma_{j+1}$ among all ζ_{j+1} pages that are gray at step $j+1$. Therefore, there are at least $\zeta_{j+1} - q_{j+1} \geq \zeta_j - q_j$ gray pages between σ_{j+1} and p. Since $p \notin \mathcal{M}(\mathcal{G}, j)$, p does not have one of the q_j largest values of $p - \sigma_j$ among all ζ_j gray pages at step j. Therefore, there are at most $\zeta = \zeta_j - q_j - 1$ gray pages between σ_j and p at step j. Since the number of gray pages never increases during a phase, there are at most ζ gray pages between σ_j and p at step $j+1$. However, $p - \sigma_{j+1} < p - \sigma_j$, which is to say that the interval from σ_{j+1} to p is contained in the interval from σ_j to p. Hence, there are at most $\zeta < \zeta_{j+1} - q_{j+1}$ gray pages between σ_{j+1} and p. Thus, we reach a contradiction and the lemma is proven. □

Remark. The gray algorithm is similar to a marking algorithm once we identify black pages with the marked pages and white pages with the unmarked pages. How-

ever, the gray algorithm differs from the marking algorithm in one important respect: an evicted page might be prefetched and reloaded later on in the phase, without being requested again. By Lemma 5.2, the prefetch operation is executed at no cost. Thus, the gray algorithm adjusts the eviction pattern dynamically according to the requests in a phase. We would like to point out here that an analysis of the standard marking algorithm explicitly uses three marks and marks all n pages [3].

THEOREM 5.3. *The gray algorithm $\mathcal{G}$ is $O(n \log k)$-competitive even against a ubiquitous adversary.*

In the rest of the section, we will prove Theorem 5.3. Let H be the adversary's algorithm and assume without loss of generality that H is lazy. We will assume without loss of generality that the request sequence σ is hard for $\mathcal{G}$. No page is requested more than once in a phase and there are exactly k requests in a phase. Since all black pages are in $\mathcal{G}$'s fast memory, all requests are for gray and white pages. Correspondingly, we will say that a request is gray (white) if the requested page is gray (white) immediately before it is requested. The first request of a phase is white because all gray pages are in $\mathcal{G}$'s fast memory at the beginning of the phase.

The proof is structured as follows.

- We begin by defining the notion of a *segment*; segments allow us to give a lower bound on the cost of H.
- We proceed to examine $\mathcal{G}$'s cost on gray requests in a given segment, and show that it is at most $O(wn \log k)$, where w is the total number of white requests in the segment.
- Finally, we use a potential function argument to show that the amortized cost of $\mathcal{G}$ is no more than $O(n \log k)$ times the cost of H.

5.3. Segments. We now define the notion of segments. Segments start from the second request of a phase and end with the first request of the next phase.

DEFINITION 5.4. *A* segment *is a subsequence of the form*

$$\langle \sigma_{ik+2}, \sigma_{ik+3}, \ldots, \sigma_{(i+1)k+1} \rangle$$

for some $i \geq 0$.

The notion of segment will be central to the rest of the proof because segments allow us to compute algorithm cost. We will estimate the cost of $\mathcal{G}$ and H on each segment and prove that the cost of $\mathcal{G}$ in a segment is no more than $O(n \log k)$ the cost of H in the same segment.

From now on, we will fix our attention on the first segment for simplicity of notation and without loss of generality. The first segment consists of the request sequence $\langle \sigma_2, \sigma_3, \ldots, \sigma_{k+1} \rangle$. Recall that σ_{k+1} is a white request.

5.4. Cost of H. We turn now to examine the cost of H in a segment. Define w^N to be the number of white pages that are requested in the segment and that cause H to fault. Clearly, the cost of H is at least w^N. Suppose that on a white request for a page p, H already has p in its fast memory. Then the ratio of the actual costs of $\mathcal{G}$ and H for the request to p is infinity. This scenario thus requires a careful analysis as we describe below.

DEFINITION 5.5. *A page p is* hidden *at step j if p is white before the jth request and $p \in \mathcal{M}(H, j)$.*

Broadly speaking, our objective is to show that H also implicitly pays a cost on the hidden white pages. We will prove that the cost of H on the segment is at least D, where D is the number of hidden pages at step $k+1$.

LEMMA 5.6. *If a page p is hidden at step $k+1$, then p has not been requested in the segment.*

Proof. If $p \in \{\sigma_2, \ldots, \sigma_k\}$, then p is black after step k and becomes gray before step $k+1$. Therefore, p is not white at step $k+1$ and so p is not hidden at step $k+1$. □

Let p be a hidden page at step $k+1$. Notice that $p \in \mathcal{M}(H, k+1)$, p was not requested in the segment, and H has not prefetched p. We conclude that p has been in $\mathcal{M}(H)$ throughout the segment.

LEMMA 5.7. *The cost of H in a segment is at least D.*

Proof. The proof is reminiscent of those in [12, 13, 18]. Clearly, $\sigma_1 \in \mathcal{M}(H, 2)$. Moreover, the previous lemma implies that all D pages hidden at step $k+1$ are in $\mathcal{M}(H, 2)$. Within a segment, the requests $\sigma_2, \ldots, \sigma_k$ are for pages that are not hidden at step $k+1$ and that are not σ_1. At most $k-D-1$ of those $k-1$ pages are in $\mathcal{M}(H, 2)$, and so the cost of H is at least $k-1-(k-D-1) = D$. □

On the whole, the cost of H in the segment is at least $\max\{w^N, D\} \geq (w^N + D)/2$.

5.5. Potential function. Let us turn now to evaluate the cost of $\mathcal{G}$ in the same segment. Let $\gamma = 3(n-1) + n \ln k$. We will denote by D_j the number of hidden pages at step j, and so $D = D_{k+1}$. Define the potential function at step j to be

$$\Phi(j) = (3(n-1) + n \ln k) D_j = \gamma D_j.$$

Clearly, $\Phi(j) \geq 0$ for all j's. We will analyze the amortized cost of $\mathcal{G}$ by considering the following cases:

- In the first case, $\mathcal{G}$ does not pay any real cost, but the potential might increase as a consequence of an increase of the number D_j of hidden pages.
- In the next two cases, $\mathcal{G}$ indeed pays a real cost.
  - We will examine the cost of $\mathcal{G}$ on gray requests, and finally
  - we will examine the cost of $\mathcal{G}$ on white requests.

5.6. Potential increase. First, we show that the potential increases only at the end of a phase. We will need the following lemma.

LEMMA 5.8. *If a page p is hidden at step j $(2 \leq j \leq k)$, then p was hidden at step* 2.

Proof. It is enough to show that if a page p is hidden at step j $(2 < j \leq k+1)$, then p was hidden at step $j-1$. Suppose to the contrary that p is hidden at step j, but not at step $j-1$. Then, either p is not white at step $j-1$ or $p \notin \mathcal{M}(H, j-1)$. However, $j \leq k$, so that no gray page has turned white. Hence, it must be the case that $p \notin \mathcal{M}(H, j-1)$. Since H does not prefetch, $p = \sigma_{j-1}$, and so p is black at step j, which is a contradiction, and the lemma is proven. □

Lemma 5.8 implies that $D_{j-1} \geq D_j$ for $j = 2, \ldots, k$. It follows that the potential increase increases only at the end of a phase when some gray pages become hidden white pages. The potential increase is γD.

5.7. Gray requests. We now evaluate $\mathcal{G}$'s cost on the gray requests. Define a *gray block* as a maximal sequence of gray requests followed by a white request. The segment can be partitioned into a sequence of gray blocks that alternate with white requests. Notice that there are at most w gray blocks in a phase, where w was defined as the number of white requests in the segment. Actually, the first phase request and the last segment request are both white, so that w is also the number of white requests in the phase. We will argue that the cost incurred by $\mathcal{G}$ during the gray blocks does not exceed $w(2(n-1) + n \ln k)$.

Consider a gray block $B_i = \langle \sigma_j, \sigma_{j+1}, \ldots, \sigma_{j+q} \rangle$. Since σ_{k+1} is a white request, $j + q \leq k$. $\mathcal{G}$'s cost for σ_j is at most $n - 1$. We now estimate $\mathcal{G}$'s cost in the rest of the block. In what follows, we denote

- by α_l, the number of gray pages that are missing from $\mathcal{G}$'s fast memory before the lth request ($j \leq l \leq j + q$),
- by β_l, the number of pages that are gray at the lth request ($j \leq l \leq j + q$),
- by g_l, the number of gray pages requested from the beginning of the phase up to step $l - 1$ inclusive,
- by w_l, the number of white pages requested from the beginning of the phase up to step $l - 1$ inclusive, and
- by $b_l = w_l + g_l$, the number of black pages before the lth request.

The notation β_l is related to that in Lemma 5.2 by $\beta_l = \zeta_{l-1}$. Notice that $\beta_l = k - g_l$ and that $\alpha_l = \beta_l - q_l = k - g_l - (k - b_l) = g_l + w_l - g_l = w_l = w_j$ is a constant throughout the current gray block. Moreover, $\alpha_l \leq w_{k+1}$. Observe that w_{k+1} is the number of white requests in $\langle \sigma_1, \sigma_2, \ldots, \sigma_k \rangle$, w is the number of white requests in $\langle \sigma_2, \ldots, \sigma_k, \sigma_{k+1} \rangle$, and both σ_1 and σ_{k+1} are white requests. Therefore, $w_{k+1} = w$ and $\alpha_l \leq w$.

$\mathcal{M}(\mathcal{G}, l)$ contains all but at most w gray pages, and the missing gray pages are the closest to its current position σ_{l-1}. Thus, once $\mathcal{G}$ is positioned at σ_{l-1}, no more than w gray pages lie in the closed interval between σ_{l-1} and σ_l ($l = j + 1, \ldots, j + q$). Hence, if $\mathcal{G}$ starts from σ_j and moves for more than n units, then there are at most $\beta_j - \left\lceil \frac{\beta_j + 1}{w} \right\rceil \leq \beta_j \left(1 - \frac{1}{w}\right)$ pages that are still gray. Hence, if during the block B_i, $\mathcal{G}$ pays $r_i n + a$, for some $a < n$, then $\beta_{j+q+1} \leq (1 - 1/w)^{r_i} \beta_j$. On the other hand, since $\beta_2 = k$, if $\sum r_i > w \ln k$, no gray pages could be remaining since

$$\left(1 - \frac{1}{w}\right)^{\sum r_i} k < \left(\frac{1}{e}\right)^{\ln k} k = 1.$$

We conclude that $\sum r_i \leq w \ln k$. Notice that the quantity a contributes for at most $n - 1$ per gray block to the cost on gray requests. Recall that σ_j contributed for another $n - 1$ term per gray block. Since there are at most w gray blocks, the cost paid during all gray blocks is at most $2w(n - 1) + n \sum r_i \leq w(2(n - 1) + n \ln k)$.

5.8. White requests. Henceforth, we will assume that $\mathcal{G}$ processes gray requests for free, but each white request is charged $2(n - 1) + n \ln k$. Such a charge is in addition to the cost required to process the white request itself. Notice that on any white request, $\mathcal{G}$ pays a real cost of at most $n - 1$, is charged $2(n - 1) + n \ln k$ for gray requests, and so $\mathcal{G}$'s amortized cost is $3(n - 1) + n \ln k = \gamma$ plus any increase of the potential function. Suppose that σ_j is a white request that causes H to fault. The potential does not increase, and $\mathcal{G}$'s amortized cost is γ. Suppose now that σ_j is a white request that does not cause H to fault. Then, p is hidden, the potential decreases by γ, and $\mathcal{G}$'s amortized cost is naught. Finally, G pays a cost of γD at the end of the phase. On the whole, $\mathcal{G}$'s cost in the segment is $w^N \gamma + D\gamma$, which is no more than $2\gamma = O(n \log k)$ times the actual cost of H in the segment, and the proof is complete.

6. Delay model. We will now describe the delay model for BDP. In the delay model, the adversary has the power of issuing requests of two types: page requests, like in the ordinary BDP, and delays, where one delay request forces any algorithm to listen to one more page. Hence, if the algorithm G is positioned over page i before a

delay request, G will be positioned over $i+1$ after the delay request. The algorithm G is free to decide whether page $i+1$ should be cached or not. Let G be an algorithm for BDP. The algorithm G can be turned into a BDP algorithm G^D in the delay model as follows. On a page request, G^D behaves exactly as the algorithm G would on that request. However, when G^D receives a delay request, it executes at most one full rotation after serving the delay request and returns to the configuration that it had before the delay.

PROPOSITION 6.1. *If G is c-competitive for BDP against a ubiquitous adversary for some $c = \Omega(n)$, then G^D is $O(c)$-competitive in the delay model.*

Proof. Let σ^D be a request sequence consisting of page requests and delays, σ the request sequence where all delays have been removed, and d the number of delay requests in σ^D. Let f be the minimum number of faults on σ and H be the adversary algorithm. Observe that $c(H, \sigma^D) \geq f + d$. Then, $c(G^D, \sigma^D) = c(G, \sigma) + nd \leq cf + nd + b \leq O(c)(f + d) + b \leq O(c)c(H, \sigma^D) + b$ for some constant b. It is then proved that G is $O(c)$-competitive. □

An immediate corollary of the above proposition is as follows.

COROLLARY 6.2. *There is an $O(n \log k)$-competitive randomized algorithm for the delay model with no prefetching and an $O(n \log k)$ deterministic algorithm for the delay model with prefetching.*

Proof. The first result follows from the $O(\log k)$-competitive lazy randomized algorithm for virtual memory paging [3, 8, 16], whereas the second result follows from the fact that the gray algorithm is $O(n \log k)$-competitive against an ubiquitous adversary. □

7. Concluding remarks. We studied deterministic as well as randomized algorithms for broadcast disk paging. An interesting question not resolved by our work is that of the competitive ratio of randomized prefetching algorithms. A lower bound of $\Omega(n)$ is easy to show. It is conceivable that a simultaneous use of randomization and prefetching yields an algorithm that is $o(n \log k)$-competitive.

Recently, the second author performed an empirical evaluation of the gray algorithm on both synthetic and Web traces and found that the gray algorithm always and consistently outperformed the least recently used (LRU) algorithm [14].

Acknowledgments. We thank Martin Farach-Colton for many discussions, Uli Kremer and Stefan Langerman for telling us about the German and Belgian videotext systems, and two anonymous referees for helping us to improve the presentation of this paper.

REFERENCES

[1] S. ACHARYA, R. ALONSO, M. FRANKLIN, AND S. ZDONIK, *Broadcast disks: Data management for asymmetric communication environments*, in Proceedings of the ACM SIGMOD Conference, San Jose, CA, 1995, pp. 199–210.

[2] S. ACHARYA, M. FRANKLIN, AND S. ZDONIK, *Prefetching from a broadcast disk*, in Proceedings of the International Conference on Data Engineering, New Orleans, LA, 1996, pp. 276–285.

[3] D. ACHLIOPTAS, M. CHROBAK, AND J. NOGA, *Competitive analysis of randomized paging algorithms*, in Algorithms—ESA '96, Barcelona, Lecture Notes in Comput. Sci. 1136, Springer-Verlag, Berlin, 1996, pp. 419–430.

[4] L. A. BELADY, *A study of replacement algorithms for a virtual storage computer*, IBM Systems J., 5 (1966), pp. 78–101.

[5] A. BORODIN AND R. EL-YANIV, *Online Computation and Competitive Analysis*, Cambridge University Press, Cambridge, UK, 1998.

[6] T. G. Bowen, G. Gopal, G. Herman, T. Hickey, K. C. Lee, W. H. Mansfield, J. Raitz, and A. Weinrib, *The Datacycle architecture*, Comm. ACM, 35 (1992), pp. 71–81.
[7] P. Cao, E. W. Felten, A. R. Karlin, and K. Li, *A study of integrated prefetching and caching strategies*, in Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS), 1995, pp. 188–196.
[8] A. Fiat, R. Karp, M. Luby, L. A. McGeoch, D. Sleator, and N. Young, *Competitive paging algorithms*, J. Algorithms, 12 (1991), pp. 685–699.
[9] M. Franklin and S. Zdonik, *A framework for scalable dissemination-based systems*, in Proceedings of the International Conference Object Oriented Programming Languages Systems, 1997, pp. 94–105.
[10] D. K. Gifford, *Polychannel systems for mass digital communications*, Comm. ACM, 33 (1990), pp. 141–151.
[11] T. Imielinski and B. Badrinath, *Mobile wireless computing: Challenges in data management*, Comm. ACM, 37 (1994), pp. 18–28.
[12] S. Irani and A. R. Karlin, *Online computation*, in Approximation Algorithms for NP-Hard Problems, D. S. Hochbaum, ed., PWS Publishing, Boston, MA, 1997, pp. 521–564.
[13] A. R. Karlin, M. S. Manasse, L. Rudolph, and D. D. Sleator, *Competitive snoopy caching*, Algorithmica, 3 (1988), pp. 79–119.
[14] V. Liberatore, *Caching and Scheduling for Broadcast Disk Systems*, Technical Report 98-71, UMIACS, University of Maryland, College Park, MD, 1998.
[15] M. S. Manasse, L. A. McGeoch, and D. D. Sleator, *Competitive algorithms for server problems*, J. Algorithms, 11 (1990), pp. 208–230.
[16] L. A. McGeoch and D. D. Sleator, *A strongly competitive randomized paging algorithm*, Algorithmica, 6 (1991), pp. 816–825.
[17] E. Sigel, *Videotext: The Coming Revolution in Home/Office Information Retrieval*, Knowledge Industry Publications, White Plains, NY, 1980.
[18] D. D. Sleator and R. E. Tarjan, *Amortized efficiency of list update and paging rules*, Comm. ACM, 28 (1985), pp. 202–208.