RANDOMIZED ALGORITHMS FOR MULTIPROCESSOR PAGE MIGRATION

JEFFERY WESTBROOK*

Abstract. The *page migration* problem is to manage a globally addressed shared memory in a multiprocessor system. Each physical page of memory is located at a given processor, and memory references to that page by other processors incur a cost proportional to the network distance. At times the page may migrate between processors at cost proportional to the distance times D, a page size factor. The problem is to schedule movements on-line so that the total cost of memory references is within a constant factor c of the best off-line schedule. An algorithm that does so is called c-competitive. Black and Sleator gave 3-competitive deterministic on-line algorithms for uniform networks (complete graphs with unit edge lengths) and for trees with arbitrary edge lengths. No good deterministic algorithm is known for general networks with arbitrary edge lengths.

We present randomized algorithms for the migration problem that are both simple and better than 3-competitive against an oblivious adversary. We give one algorithm designed for uniform graphs that is approximately 2.28-competitive as D grows large. We also give a second, more powerful algorithm that works on graphs with arbitrary edge distances. This algorithm is approximately 2.62-competitive (or, 1 plus the Golden Ratio) for large D. Both these algorithms use random bits only during an initialization phase, and from then on run deterministically. We also examine the competitiveness of a very simple coin-flipping algorithm.

Key words. on-line algorithms, page migration, competitive analysis, multiprocessors, memory management

AMS subject classifications. 68Q20, 68Q25

1. Introduction. A common design for a shared memory multiprocessor system is a network of processors, each of which has its own local memory [9], [14], [19]. In such a design, a programming abstraction of a single global memory address space is supported by a virtual memory system that distributes one or more copies of each physical page of memory among the processors' local memories. When processor p wishes to read or write to memory address a, located in page b, it first looks to see if page b is contained in its own local memory. If so, then the memory access is done immediately. If not, p determines some processor q that does hold page b, and transmits a memory request to q over the network. The communication cost is dependent on the interconnection distance between p and q. Processor q services the request, and transmits back to p the (updated) value at address a.

Having a given virtual page stored at multiple processors reduces communication overhead during memory reads, but introduces the problem of maintaining consistency among the multiple copies during writes. Most multiprocessors do not provide mechanisms for maintaining consistency [5]. Therefore, various network designers have studied the *page migration* problem [4], [5], [17], which arises when each writeable page is restricted to a single copy. Suppose the single copy of page b is initially located at processor q. If the page is going to be accessed frequently by processor p, then migrating the page from p to q will reduce the communication overhead. On the other hand, moving a full page of memory incurs a large amount of communication overhead, proportional to the size of the page. In addition, moving the page to p may increase the cost of satisfying future memory requests should the pattern

* Department of Computer Science, Yale University, New Haven, CT 06520-8285. Research partially supported by NSF grant CCR-9009753. A preliminary version of this paper appeared in the Proceedings of the DIMACS Workshop on On-line Algorithms, American Mathematical Society, February 1991.

of accesses change. The problem, then, is to design an on-line algorithm to schedule page migrations in response to dynamically changing access patterns.

Formally, an instance of the migration problem consists of an edge-weighted undirected graph G, and a real-valued constant, D, no smaller than 1. The graph G describes the interconnection network; each node contains a processor. We assume the nodes are numbered from 1 to n. One node of G, denoted p, contains the page. Initially the page resides at node 1. The constant D corresponds to the size of the page. A *request at node* r is a reference by processor r to some memory address on the page. A sequence of requests is generated at the nodes of G. Each new request r is satisfied immediately at cost $\delta_{p,r}$, the shortest distance in G between p and r. After satisfying a request, the page can be moved from p to a new node p' at cost $d \cdot \delta_{p,p'}$. By definition, the distance function δ satisfies the triangle inequality: $\delta_{x,y} \leq \delta_{x,z} + \delta_{z,y}$. Since G is undirected, δ is symmetric.

For any sequence of requests, there is an optimum schedule of page movements that minimizes the total cost of requests and migrations. In general, an on-line algorithm cannot compute the optimum schedule, since it lacks knowledge of the future requests. An on-line algorithm is called *c-competitive* if, for any request sequence, its cost is no more than c times the cost of the optimum off-line cost for that sequence, for some constant c. Recently much attention has been given to *competitive analysis* of on-line algorithms [6], [13], [15], [18].

Black and Sleator [5] formalized and studied the page migration problem in the context of competitive analysis. They considered two classes of networks: uniform networks, i.e., complete graphs with each edge having length 1; and trees with arbitrary edge lengths. They developed 3-competitive deterministic algorithms for these two classes for any D. In addition, they showed that for all D, no deterministic algorithm could be better than 3-competitive, even if the graph consists of only a single edge. Little is known about deterministic algorithms for a complete graph with arbitrary edge weights. This problem seems quite hard. The best previous deterministic bound for the general case was either the $2n - 1$ bound for metrical task systems [6][1] or a $2D + 2$ bound given by a simple algorithm that moves the page to the requesting node after each request.

In this paper we use randomization to beat the deterministic lower bound and to give a fast algorithm for the general case. No randomized algorithms were known prior to this work. A randomized on-line algorithm is *c-competitive* (against an oblivious adversary) if, for any request sequence, its expected cost is no more than c times the cost of the optimum off-line cost for that sequence, for some constant c. There are also definitions of randomized competitiveness against stronger adaptive adversaries which are given in Section 2.

For uniform networks, we give a simple algorithm, UNIFORM, that is 2.38-competitive for large values of D. This algorithm is "barely random", in the sense that it only uses a small number of random bits during an initialization phase, and from then on runs deterministically. Such a barely random algorithm has practical value since random bits can be an expensive resource.

For the general problem (which includes trees) we develop a basic algorithm from which variants can be constructed by choosing different probability distributions. We consider two such choices. The first uses random coin flips after each request and gives an algorithm that is $(1 + \phi)$-competitive for large D, where ϕ is the Golden Ratio, approximately 1.62. We then show that a deterministic resetting strategy gives a

[1] Technically, the metrical task bound hold for a slightly different model. See below.

barely random algorithm with a competitive ratio that also tends to $1+\phi$, although it is always slightly larger than the competitive ratio of the first algorithm. Lastly we examine the competitiveness of a very simple coin-flipping algorithm. We show that on any network and for any D it is 3-competitive against an adaptive on-line adversary. The coin flipping algorithm is memoryless, and hence has no storage or network overhead, but needs to generate a random number every request. Applying a result of Ben-David *et al.* [2] to the coin-flipping algorithm and the $(1+\phi)$-competitive algorithm mentioned above, we prove the existence of a $(3+3\phi)$-competitive deterministic algorithm for general networks.

Page migration is one of several problems that arise in managing data in a distributed environment. Black and Sleator [5] have studied the related problem of page replication, in which one may make multiple copies of a read-only page, and Karlin *et al.* [11] studied snoopy caching, which is memory management given a bus-based interconnection network. Other memory management problems have been studied in references [1], [12], [15]. Migration is related to the *1-server with excursions* problem defined by Manasse *et al.* [13]. Migration and 1-server with excursion are also related to the *k-server* problems [3], [7], [13]. Practical issues and applications of page migration are discussed more fully in [4], [5], [17]. Subsequent to the work presented here, Chrobak *et al.* found a $2+1/2D$-competitive algorithm for a single edge and for the tree topology [8], and showed that this bound is tight.

2. Competitive Analysis and Lower Bounds. Let σ be a sequence of requests. The cost of an algorithm A on σ is denoted $\mathrm{A}(\sigma)$. We denote by OPT the off-line algorithm that achieves the optimum cost on σ. Following [6], [13] we say a deterministic algorithm A is *c-competitive* if there is a constant b such that for all request sequences σ,

$$\mathrm{A}(\sigma) \leq c \cdot \mathrm{OPT}(\sigma) + b.$$

For randomized algorithms the competitiveness of an algorithm is defined with respect to an *adversary*. Following [2],[15] we consider three kinds of adversaries.

A randomized on-line algorithm, A, is c-competitive against an *oblivious* or *weak* adversary if there is a constant b such that for all graphs and for all finite request sequences σ,

$$\mathrm{E}\left[\mathrm{A}(\sigma)\right] \leq c \cdot \mathrm{OPT}(\sigma) + b.$$

The expectation is over the random choices made by the on-line algorithm. This is the natural extension of deterministic competitiveness, and models a situation in which the random choices of the algorithm do not affect the choice of requests. We will typically use "c-competitive" as an abbreviation for "c-competitive against an oblivious adversary." When an adaptive adversary is intended we will state so explicitly.

An *adaptive* adversary generates the request sequence on-line, choosing requests based on the actual moves made by the on-line algorithm. That is, an adaptive adversary $\hat{\mathrm{A}}$ is a function that takes as input a sequence of $k-1$ requests and corresponding actions by the on-line algorithm, and outputs the k^{th} request, up to a maximum number of requests, m. (Each adversary has its own value of m.) Since the output of $\hat{\mathrm{A}}$ depends on the random choices of the on-line algorithm, a randomized on-line algorithm and an adaptive adversary together generate a probability distribution over request sequences σ.

There are two ways to charge the adversary for the request sequence. If $\hat{\mathrm{A}}$ is an *adaptive off-line* adversary, then $\hat{\mathrm{A}}(\sigma)$ is simply $\mathrm{OPT}(\sigma)$, the optimum off-line cost

for σ. If $\hat{\text{A}}$ is an *adaptive on-line* adversary, then $\hat{\text{A}}$ is the cost incurred on σ by an auxiliary on-line algorithm $M_{\hat{\text{A}}}$. An on-line algorithm A is c-competitive against an adaptive adversary $\hat{\text{A}}$ if there exists a constant b such that

$$\text{E}\left[\text{A}(\sigma) - c \cdot \hat{\text{A}}(\sigma)\right] \leq b.$$

The adaptive adversaries model a situation in which the random choices of the algorithm may affect the future request sequence. The adaptive on-line adversary is not entirely intuitive, in that the cost measure seems somewhat unnatural, but this adversary is useful in proving theorems about deterministic algorithms.

Black and Sleator showed that in the simple case of two processors linked by a single edge of length 1, no deterministic algorithm can be better than 3-competitive for any D. In addition, Chrobak *et al.* [8] give an example of a 4-processor network on which every deterministic algorithm has a competitive ratio slightly larger than 3 when $D = 1$. By a theorem of [2], these deterministic lower bounds also apply to randomized algorithms facing an adaptive off-line adversary.

The results in this paper demonstrate that against an oblivious adversary, randomized algorithms can beat the deterministic lower bound. By considering the case of a single edge, one can show that for a given page factor D no randomized algorithm can be better than $2 + \frac{1}{2D}$-competitive against an oblivious adversary [8]. We suspect, however, that the lower bound is higher for more complicated graphs.

In Section 5 we show a randomized algorithm that is 3-competitive against adaptive on-line adversaries for all D. One can apply a technique from [10], [16] to show that no algorithm can do better. This lower bound follows from considering the simple case of two processors linked by a single edge of length 1. The adaptive adversary's strategy is to watch the on-line algorithm and always generate requests at the processor that does not contain the on-line algorithm's page. Thus the on-line algorithm pays 1 per request. Prior to generating the first request, the adaptive on-line adversary simulates the on-line algorithm over all possible random choices, using the above strategy to generate a probability distribution over request sequences σ of length m. From the simulation, the adversary computes the expected number of requests at processors p_1 and p_2, and expected total cost of page migrations, B, paid by the on-line algorithm. If $B > m/2$, the adversary initially places its page at the processor most often requested, and never moves it again. The expected cost to the adversary is at most $D + m/2$, while the expected cost to the on-line algorithm is at least $3m/2$. If $B \leq m/2$, the adversary always moves so as to keep its page at the processor that does *not* contain the on-line algorithm's page. In this case, the adversary pays B, while the on-line algorithm pays $B + m \geq 3B$. Thus in either case, the on-line algorithm cannot guarantee to be better than 3-competitive.

The above discussion shows that in the migration problem the oblivious, adaptive on-line, and adaptive off-line adversaries are strictly differentiated from each other in their power. This is the first natural on-line problems for which this is known to be true.

3. Uniform Graphs. In this section we describe and analyze a randomized algorithm, UNIFORM, designed for uniform graphs. The uniform graph is a common and important network topology [5]. In this model, $\delta_{v,v} = 0$ for all processor nodes v, $\delta_{u,v} = 1$ for all pairs $u \neq v$, and the cost of moving the page from node u to node v is $D \geq 1$.

Before describing the algorithm, we remark that when the graph is a single edge between two nodes, this problem is identical to the two-item list update problem. Reingold *et al.* [10], [16] gave a simple randomized algorithm for the list-update problem, which UNIFORM parallels.

Each processor node v of the graph is given an associated counter, C_v, that takes on values between 0 and $k-1$. The optimum value of k depends on D; we will see later how to choose k. Let $\underline{C} = (C_1, C_2, \ldots, C_n)$ be a vector of the counter values at each of the n nodes of the graph. There are k^n such vectors. Let $\underline{C_i}$ be the i^{th} such vector,

Let U^i denote a deterministic algorithm that begins with initial counter vector $\underline{C_i}$ and processes a request at node r as follows:

1. The page request is satisfied at cost at most 1.
2. The counter at r, C_r, is incremented.
3. If C_r has value k, the page is moved to node r, if it is not already there, and C_r is reset to 0.

Prior to processing any requests, the UNIFORM algorithm chooses a deterministic algorithm U^i uniformly at random, and then runs it. The random choice is made by initializing each node counter uniformly at random, using $\Theta(n \log k)$ random bits,

THEOREM 3.1. *UNIFORM is c_k-competitive against an oblivious adversary, where k is the maximum counter value and c_k is the maximum of $1 + \frac{k+1}{2D}$ and $1 + \frac{1}{k}\left(2D + \frac{k+1}{2}\right)$.*

Proof. We show that $\mathrm{E}\left[\mathrm{UNIFORM}(\sigma)\right] \leq c_k \cdot \mathrm{OPT}(\sigma)$ for any finite request sequence σ.

To analyze the expected cost of UNIFORM we imagine that the ensemble of $m = k^n$ different deterministic algorithms U^i is being run simultaneously on the request sequence. The expected cost of UNIFORM is the total cost of the ensemble divided by m. We use the following observation: at any time, each algorithm in the ensemble has a distinct counter vector. This follows by an easy induction on the number of requests. The observation implies that at any time the number of algorithms in the ensemble with the same value of C_v is exactly m/k. Equivalently, at any time during the running of algorithm UNIFORM, $\Pr\left[C_v = j\right] = 1/k$ for $0 \leq j \leq k-1$.

Our proof uses the technique of comparing simultaneous runs of U^i and OPT on the request sequence σ. The actions of U^i and OPT are partitioned into two kinds of events, which together account for all costs to the algorithms. The first kind of event is the servicing of a request by both U^i and OPT. This event may involve U^i moving its page. The second kind of event is OPT moving its page. After the t^{th} event, let p^i denote the node where U^i has the page and *opt* denote the node where OPT has the page. We define a *potential function* Φ:

$$\Phi_t = \begin{cases} 0 & \text{if } p^i = opt \\ D + k - C_{opt} & \text{if } p^i \neq opt \end{cases}$$

Each event has an actual cost to OPT, and an *amortized cost* to U^i, defined to be the actual cost of the event to U^i plus the change in the potential function, $\Delta\Phi = \Phi_t - \Phi_{t-1}$. The potential Φ is always non-negative, and $\Phi_0 = 0$, since OPT and U^i begin with the page at the same node. This implies, by standard amortized analysis arguments, that the total actual cost to U^i is bounded by the sum of the amortized costs for all events. To prove the theorem, we show that for each event the sum of the amortized costs to the algorithms in the ensemble is less than mc_k times the actual cost to OPT.

1. The event is a request. There are two subcases.

(i) The request is at node *opt*. In this case OPT pays 0, so we must show that the amortized cost to each algorithm U^i is also zero. If $p^i = opt$ then the actual and amortized cost to U^i is also 0. Otherwise, U^i pays 1, but since C_{opt} is incremented, the potential decreases by 1. If the counter reaches k, then the page moves to *opt*. This costs D, but decreases the potential by a further D. Thus the amortized cost is always 0.

(ii) The request is at node $r \neq opt$. The cost to OPT is 1. Each algorithm U^i pays at most 1 for the request. For m/k of the algorithms in the ensemble, C_r reaches k and the page is moved to r at cost D. If $p^i \neq opt$ prior to moving, then there is no change in the potential. In the worst case, $p^i = opt$ prior to the move and the potential increases by $D + k - C_{opt}$. No more than m/k^2 algorithms, however, can move from *opt* with $C_{opt} = i$, for $0 \leq i \leq k-1$. Therefore, the total cost of the event to the ensemble is bounded by

$$m + D\frac{m}{k} + \sum_{0 \leq i \leq k-1} \frac{m}{k^2}(D + k - i)$$

Summing and dividing by m gives the expected amortized cost to UNIFORM,

$$1 + \frac{1}{k}\left(2D + \frac{k+1}{2}\right),$$

which is bounded by c_k times the cost to OPT.

2. The event is that OPT moves its page to a new node r. The actual cost to OPT is D, and the actual cost to each algorithm U^i is 0. Now we consider the cost due to changes in potential. If $r = p^i$, the potential decreases by at least D. If $r \neq p^i$ and $opt \neq p^i$, then $\Delta\Phi = C_{opt} - C_r$, which is bounded by $k - C_r$. In the worst case, $r \neq p^i$ and initially $opt = p^i$. Then $\Delta\Phi = D + k - C_r$. Exactly m/k algorithms have $C_r = i$, so the total cost to the ensemble of this event is at most

$$\sum_{0 \leq i \leq k-1} \frac{m}{k}(D + k - i)$$

Summing and dividing by m gives the expected cost to UNIFORM, $D + (k+1)/2$, which is at most c_k times the cost to OPT. □

This completes the proof of Theorem 3.1. Given a value of D, we can choose k to minimize the maximum of $1 + \frac{k+1}{2D}$ and $1 + \frac{1}{k}\left(2D + \frac{k+1}{2}\right)$.

Table 1 shows the best competitive ratio for UNIFORM for values of D up to 10. These values are found by setting

$$1 + \frac{k+1}{2D} = 1 + \frac{1}{k}\left(2D + \frac{k+1}{2}\right)$$

and solving for k in terms of D. Then the best integer approximation to this value is taken. As D tends to infinity, the best competitive ratio decreases and tends to $(5 + \sqrt{17})/4 \approx 2.28$. Note that all these values are better than the deterministic lower bound of 3. It is possible to slightly improve upon the values for small D by using the random reset techniques employed in [16].

The next theorem shows that our analysis is tight.

THEOREM 3.2. *The UNIFORM algorithm is no better than c_k-competitive, where c_k is defined as in Theorem 3.1.*

D	best k	comp. ratio
1	2	2.75
2	5	2.50
3	7	2.43
4	10	2.38
5	12	2.38
6	15	7/3
7	17	2.35
8	20	2.33
9	23	2.33
10	25	2.32

TABLE 1
Best competitive ratios for UNIFORM

Proof. Consider two processors, labelled "1" and "2", connected by a single edge of length 1. Initially the page is located at processor 1. There are two kinds of bad request sequences: $\sigma_a = 2^k$ and $\sigma_b = 21^k$, where 1 and 2 denote page requests at processors 1 and 2 respectively, and i^k denotes k consecutive requests at processor i. Sequence σ_a can be satisfied at cost D, and σ_b can be satisfied at cost 1.

The expected cost to UNIFORM on σ_a is simply the expected time until the page migrates to 2, which is determined by the expected value of the counter at 2, plus the cost of migration. (UNIFORM is guaranteed to move the page eventually, since there are k requests.) Therefore

$$\mathrm{E}\left[\mathrm{UNIFORM}(\sigma_a)\right] = D + \frac{k+1}{2},$$

or $(1 + \frac{k+1}{2D}) \cdot \mathrm{OPT}(\sigma_a)$.

On σ_b, UNIFORM moves its page to 2 after the first request with probability $1/k$, in which case the remainder of the sequence looks identical to σ_a. Hence

$$\mathrm{E}\left[\mathrm{UNIFORM}(\sigma_a)\right] = (1 + \frac{1}{k}(2D + \frac{k+1}{2})) \cdot \mathrm{OPT}(\sigma_b).$$

The adversary can generate arbitrarily long request sequences by repeating σ_a or σ_b, so the UNIFORM algorithm cannot be better than c_k-competitive. □

4. General Graphs. In this section we consider the migration problem on undirected graphs with arbitrary edge distances. Recall that even with arbitrary edge lengths the shortest distance function δ remains symmetric and still satisfies the triangle inequality.

One reason the general case seems harder than the uniform case is that a general graph may contain clusters of processors that are very close to each other but relatively far from other clusters. Such a situation models a collection of local-area networks tied together in a wide-area network. Even though no single processor in a cluster is accessing the page especially often, the cluster as a whole may generate many requests for the page, and it may be advantageous to move the page to some node in the cluster. On the other hand, in the uniform case a cluster of processors that access the page equally often is an easy situation for the on-line algorithm to handle, since both it and OPT must pay about 1 per request.

$$\begin{pmatrix} \alpha_1 & 1 & 0 & 0 & \dots & 0 \\ \alpha_2 & 0 & 1 & 0 & & 0 \\ \alpha_3 & 0 & 0 & \ddots & & 0 \\ \vdots & & & & & \\ \alpha_{k-1} & & & & & 1 \\ \alpha_k & & & & & 0 \end{pmatrix}$$

FIG. 1. *Markov chain transition matrix. Entry* (i, j) *contains the probability of a transition to counter value* i *from counter value* j *following a request,* $1 \leq i, j \leq k$.

The previous best known deterministic competitive ratio for general graphs is $2n - 1$, given by the general metrical task system algorithm [6]. Technically, that result holds for a different model, the "lookahead-1" model, in which the algorithm is allowed to move the page after seeing the next request but before actually servicing it. In our "lookahead-0" model, the request must be serviced immediately it is seen. It is easily checked that an algorithm for the lookahead-1 model can be used to derive an algorithm for the lookahead-0 model that has a competitive ratio no more than 2 times worse than that of the original algorithm. Furthermore, the bounds converge as D grows large. One can also show that the strategy of moving the page to the last requesting node yields a competitive ratio of $2D + 2$. Thus even when the graph is a single triangle, the best known deterministic algorithm for large D is only 5-competitive. Our randomized algorithms for the general case are simple enough to be practical, and beat the deterministic lower bound.

The general algorithm G maintains a *single* counter with value $C \geq 1$. After each request to the page,

1. The counter is decremented by one, regardless of where the request is.
2. If the counter reaches 0, the page is moved to the location of the current request. After the move, the counter is reset according to a *resetting distribution* $\mathcal{D} = (\alpha_1, \alpha_2, \alpha_3, \ldots)$, where α_i is the probability of resetting the counter to value i.

The distribution $\mathcal{D}$ is fixed in advance and is independent of the request sequence. Let $k = \max\{i \mid \alpha_i > 0\}$. We are only interested in distributions for which k exists (*i.e.* is not infinite).

The counter value forms a Markov chain in which state i corresponds to the counter having value i, for $1 \leq i \leq k$. Being in state i means that the page will be moved after i accesses. A state transition occurs every access. The transition matrix for the corresponding Markov chain is shown in Figure 1. Note that the probability of being in a given state depends only on the number of requests that have been processed, not on the nature of the requests.

Let ρ_i denote the steady state probability of being in state i. Given a resetting distribution $\mathcal{D}$, let $\bar{C}_R(\mathcal{D})$ denote the expected value of the counter immediately following a reset and let $\bar{C}_S(\mathcal{D})$ denote the expected value of the counter in steady state. By definition, $\bar{C}_R(\mathcal{D}) = \sum_{i=1}^{k} i\alpha_i$ and $\bar{C}_S(\mathcal{D}) = \sum_{i=1}^{k} i\rho_i$. We will abbreviate the notation to $\bar{C}_R$ and $\bar{C}_S$ when the distribution $\mathcal{D}$ is clear.

The steady state probabilities ρ_i are given by the eigenvector of the transition

matrix corresponding to eigenvalue 1. One may verify that in steady state,

$$\rho_i = \frac{1}{\bar{C}_R} \sum_{j=i}^{k} \alpha_j$$

This implies

$$\begin{aligned} \bar{C}_S &= \frac{1}{\bar{C}_R} \left(\sum_{i=1}^{k} i \sum_{j=1}^{k} \alpha_j \right) \\ &= \frac{1}{\bar{C}_R} \left(\sum_{i=1}^{k} \alpha_i \sum_{j=1}^{i} j \right) \\ &= \frac{1}{\bar{C}_R} \left(\sum_{i=1}^{k} i(i+1)\alpha_i \right). \end{aligned}$$

Prior to starting a run of the algorithm, the counter is initialized according to the steady state distribution, i.e., the counter is initialized to i with probability ρ_i, for $1 \leq i \leq k$.

LEMMA 4.1. *Let* $\mathrm{G}(\mathcal{D})$ *be the general algorithm with resetting distribution* $\mathcal{D}$. $\mathrm{G}(\mathcal{D})$ *is* $c_{\mathcal{D}}$*-competitive, where* $c_{\mathcal{D}}$ *is the maximum of* $2 + \frac{2D}{\bar{C}_R(\mathcal{D})}$ *and* $1 + \frac{\bar{C}_S(\mathcal{D})}{D}$.

Proof. As before, we partition the actions and costs into two events: a request that both $\mathrm{G}(\mathcal{D})$ and OPT satisfy, and a page movement by OPT. A given σ fixes a sequence of events and determines an ensemble of deterministic on-line algorithms that correspond to all possible random choices that can be made throughout the processing of σ. At any time, the fraction of algorithms in this ensemble that have counter value i is given by the steady state probability, ρ_i. The average cost of the algorithms in this ensemble gives the expected cost of one run of G on σ. We will calculate the expected amortized cost per operation of G directly. Let random variable X_t be the amortized cost of the t^{th} event. The total amortized cost is $\sum_t X_t$, and by linearity of expectations the expected total amortized cost is $\sum_t \mathrm{E}[X_t]$. To prove the theorem we bound $\mathrm{E}[X_t]$ in terms of the cost to OPT.

At the time of event t, let *opt* be the node where OPT has its page and g be the node where G has its page. Let C be the value of the counter. Both g and C depend upon previous random choices, but *opt* is fixed by σ.

We use the potential function

$$\Phi_t = (D + C) \cdot \delta_{g,opt}$$

Note that $\Phi = 0$ when OPT and G have their pages at the same node.

Now we analyze the expected amortized cost of each event.

1. The event is a request at node r. Let $\ell_0 = \delta_{r,opt}$, $\ell_1 = \delta_{g,opt}$ and $\ell_2 = \delta_{g,r}$. (See Figure 2.) The cost to OPT is ℓ_0. The actual cost to G is ℓ_2. The counter is decremented, so $\Delta\Phi = -\ell_1$, and hence the amortized cost to G is $\ell_2 - \ell_1$. By the triangle inequality, $\ell_2 \leq \ell_0 + \ell_1$. Therefore, the amortized cost to G to satisfy the request is at most ℓ_0.

With probability ρ_1, the counter value is 1 just prior to the request and becomes 0 after the request is satisfied. Then G moves the page to node r at cost $D\ell_2$ and resets the counter according to $\mathcal{D}$. The potential change is

$$\ell_0(D + C') - D\ell_1$$

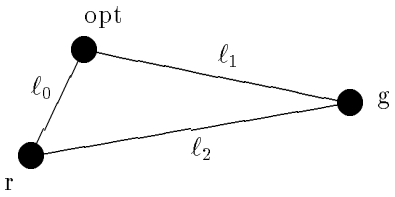

FIG. 2. *Diagram for proof of Lemma 4.1*

where C' is the value to which the counter is reset. The *expected* change in potential is determined by the expected value of the counter after the reset, $\bar{C}_R$. Hence the expected amortized cost of a move is

$$D\ell_2 + \ell_0(D + \bar{C}_R) - D\ell_1 .$$

Again applying the triangle inequality, this is bounded by

$$\ell_0(2D + \bar{C}_R).$$

Combining the cost to satisfy the request with the expected move cost in the case that $C = 1$, the total expected amortized cost to G is bounded by

$$\ell_0 \left(1 + \rho_1(2D + \bar{C}_R)\right) .$$

Since $\rho_1 = \frac{1}{\bar{C}_R}$, this is at most $c_{\mathcal{D}}$ times the cost to OPT.

2. The event is OPT moving its page from node *opt* to a new node *opt'*. Let $\ell_0 = \delta_{opt,opt'}$. The actual cost to OPT is $D\ell_0$ and the actual cost to G is 0. Now we consider the cost due to changes in potential. Let ℓ_1 be $\delta_{g,opt}$ and ℓ_2 be $\delta_{g,opt'}$ The actual potential change is

$$(\ell_2 - \ell_1) \cdot (D + C)$$

where C is the actual counter value. The *expected* potential change is determined by the steady state expected value of the counter, $\bar{C}_S$. Since $\ell_2 \leq \ell_0 + \ell_1$, the expected potential change is bounded by $\ell_0 \cdot (D + \bar{C}_S)$, which is at most $c_{\mathcal{D}}$ times the cost to OPT. □

LEMMA 4.2. *The general algorithm,* $\mathrm{G}(\mathcal{D})$*, is no better than* $c_{\mathcal{D}}$*-competitive, where* $c_{\mathcal{D}}$ *is defined as in Lemma 4.1.*

Proof. As in Theorem 3.2 we consider two processors, connected by an edge of length 1, and two request sequences: $\sigma_a = 2^k$ and $\sigma_b = 21^k$.

The expected cost to $\mathrm{G}(\mathcal{D})$ on σ_a is $\bar{C}_S$, the initial expected time until the page migrates to 2, plus the cost of the migration. Therefore

$$\mathrm{E}\left[\mathrm{G}(\sigma_a)\right] = D + \bar{C}_S ,$$

or $(1 + \frac{\bar{C}_S}{D}) \cdot \mathrm{OPT}(\sigma_a)$.

On σ_b, $\mathrm{G}(\mathcal{D})$ moves its page to 2 after the first request with probability ρ_1. In this case the cost on the remainder of σ_b is determined by the expected time to return the page to 1, given that it has moved initially. This expected time is $\bar{C}_R$. Hence

$$\mathrm{E}\left[\mathrm{G}(\sigma_a)\right] = (1 + \rho_1(2D + \bar{C}_R)) \cdot \mathrm{OPT}(\sigma_b) = (2 + \frac{2D}{\bar{C}_R})\mathrm{OPT}(\sigma_b).$$

The adversary can generate arbitrarily long request sequences by repeating σ_a or σ_b. □

In the next two subsections we analyze two choices for the resetting distribution $\mathcal{D}$. First we give a randomized resetting strategy with a competitive ratio that tends to $1+\phi$, where ϕ is the Golden Ratio, approximately 1.618. Then we discuss the deterministic strategy of resetting the counter to a single value k after a move. We show that the competitive ratio of the deterministic strategy approaches that of the random resetting distribution as D grows large. Deterministic resetting gives a simple, practical, near-optimal barely random algorithm.

4.1. Random Resetting. We restrict our attention to distributions $\mathcal{D}$ that satisfy

$$2 + \frac{2D}{\bar{C}_R} = 1 + \frac{\bar{C}_S}{D}. \tag{1}$$

Then the competitive ratio is given by the choice of resetting probabilities that minimizes $2 + 2D/\bar{C}_R$. This is minimized when $\bar{C}_R$ is maximized. For a fixd value of k, the following linear program computes an optimum resetting distribution subject to constraint 1:

$$\text{maximize } \sum_{i=1}^{k} i\alpha_i$$

subject to the constraints

$$\begin{aligned} \sum_{i=1}^{k} \alpha_i &= 1 \\ \sum_{i=1}^{k} (i^2 - (2D-1)i)\alpha_i &= 4D^2 \\ \alpha_i &\geq 0, \quad 1 \leq i \leq k \end{aligned}$$

Let x be the positive solution to $x^2 - (2D-1)x = 4D^2$. Hence $x = D + \frac{1}{2}((20D^2 - 4D + 1)^{\frac{1}{2}} - 1)$. Then $k_0 = \lceil x \rceil$ is the least integer such that constant on the k_0th term in the second constraint of the linear program is greater than $4D^2$.

LEMMA 4.3. *There is a feasible solution to the linear program with only α_{k_0-1} and α_{k_0} non-zero.*

Proof. Let $\alpha = \alpha_{k_0-1}$. By the first constraint of the linear program, $\alpha = 1 - \alpha_{k_0}$. Setting $\alpha_i = 0$ for all other i, the second constraint becomes

$$0 = k_0^2 - (2D-1)k_0 - 4D^2 + \alpha(2D - 2k_0)$$

Replacing k_0 by $x + \epsilon$, where $0 \leq \epsilon < 1$, and using the fact that $x^2 - (2D-1)x = 4D^2$, this constraint further reduces to

$$0 = (2x - 2D + 1 + \epsilon)\epsilon - (2x - 2D + 2\epsilon)\alpha$$

implying

$$\alpha = \epsilon + \epsilon(1-\epsilon)/(2x - 2D + 2\epsilon)$$

D	best k_0	comp. ratio
1	3	2.800
2	6	2.696
3	9	2.667
4	13	2.655
8	26	2.636
16	52	2.627
32	103	2.622
64	207	2.620
128	414	2.619
256	828	2.619
512	1657	2.618

TABLE 2
Best general competitive ratios, resetting randomly with α_{k_0-1} and α_{k_0}

Observing that $2x - 2D \geq 0$ for all $D \geq 0$, it is easily verified that $0 \leq \alpha \leq 1$. Hence constraint three is satisfied. □

THEOREM 4.4. *The competitive ratio of the algorithm given by choosing k_0 and α as above is $2 + 2D/(k_0 - \alpha)$*

Proof. This theorem follows from straightforward substition into the formula for $\bar{C}_R$. □

Table 2 shows several competitive ratios given by theorem 4.4. As D gets large, the best competitive ratio decreases and tends to $1 + \frac{1+\sqrt{5}}{2}$, which is one plus the Golden Ratio, or approximately 2.62.

4.2. Deterministic Resetting. We can turn the randomized resetting strategy of the previous section into a deterministic resetting strategy by setting either α_{k_0-1} or α_{k_0} to 1, depending on which gives the lowest maximum competitive ratio. Let $s = k_0 - 1$ or k_0 depending on whether we choose $\alpha_{k_0-1} = 1$ or $\alpha_{k_0} = 1$, respectively.

THEOREM 4.5. *The general algorithm with deterministic resetting is c_s-competitive against an oblivious adversary, where c_s is the maximum of $1 + \frac{s+1}{2D}$ and $2 + \frac{2D}{s}$.*

Proof. The counter is initialized by setting it to value i with probability $1/s$ for $1 \leq i \leq s$. At any time thereafter, the counter values are uniformly distributed from 1 to s. Calculating $\bar{C}_R$ and $\bar{C}_S$ from the formulas given above and applying Lemma 4.1 completes the proof. □

Table 3 shows the best competitive ratios achieved by this deterministic resetting strategy for various values of D. As D and hence k_0 grows, the deterministic and random resetting algorithms tend to the same competitiveness. Deterministic resetting has the advantage of greater simplicity, since it requires random bits only during initialization

5. The Coin-Flipping Algorithm. A very simple randomized strategy is to flip a weighted coin at each request, and move to the requested node if the coin comes up heads. That is, at each request, the page is moved to the requesting node with probability $p > 0$. This strategy can be used on a general graph with arbitrary edge weights. It has the advantage of being *memoryless* ; no state information is remembered other than that given by the location of the page. Furthermore, it achieves exactly a competitive ratio of 3 against both adaptive on-line adversaries and oblivious adversaries. On the other hand, it has a poorer competitiveness against

D	best k_0	comp. ratio
1	3	3.000
2	6	2.750
3	9	2.667
4	12	2.667
8	25	2.640
16	51	2.627
32	103	2.625
64	206	2.621
128	413	2.620
256	827	2.619
512	1656	2.618

TABLE 3
Best general competitive ratios, resetting to k_0 with probability 1

an oblivious adversary than the algorithms for the uniform and general problems presented in Sections 3 and 4, respectively, and it requires a random number call every access.

In this section we analyze the competitiveness of the coin-flipping algorithm. First we give a lower bound.

THEOREM 5.1. *The coin-flipping algorithm is no better than 3-competitive against an oblivious adversary.*

Proof. We use our standard scenario of two processors and two request sequences: $\sigma_a = 2^j$ and $\sigma_b = 21^j$. We require that $j > D$. For σ_a the optimum cost is D, and for σ_b the optimum cost is 1.

Let $q = 1 - p$. The expected cost of the coin flipping algorithm on σ_a is $(D + 1/p)(1 - q^j)$. For any $p \leq 1/2D$, this is at least $3(1 - q^j)$ times the optimal cost. The expected cost of the coin flipping algorithm on σ_b is $1 + p(D + (D + 1/p)(1 - q^j)) \geq 2pD + 2(1 - q^j)$. For any $p \geq 1/2D$, this is at least $3 - 2q^j$ times the optimal cost. Hence for any choice of p, there is a sequence and a choice of j such that the competitive ratio of the coin flipping algorithm is arbitrarily close to 3. □

Now we show that the coin-flipping algorithm is 3-competitive against an adaptive on-line adversary.

To perform the analysis, we use the request-answer game framework of [2]. The adversary initially commits to a request sequence length m. The game proceeds as a sequence of m rounds. In each round, the adversary first moves its page, if it so chooses. Then the adversary generates a request, and immediately services it. Then the on-line algorithm services the request and possibly moves its page. At the end of each round, the adversary is told the location of the page of the on-line algorithm.

THEOREM 5.2. *The coin-flipping algorithm with $p = 1/2D$ is 3-competitive against any adaptive on-line adversary $\hat{A}$.*

Proof. We use the following crucial observation. In each round, the probability that the coin-flipping algorithm moves its page is independent both of the request and of the location of the adversary's page when the request is generated. This observation is not true for the algorithms of Sections 3 and 4, which make moves based on counter values that the adaptive adversary knows and can use in generating its requests. This observation is also not true for the coin-flipping algorithm against an adaptive off-line adversary. In the off-line case, the location of the adversary's

page depends deterministically upon future requests, which in turn depend upon the outcomes of future coin-flips by the algorithm.

Using the observation, we show that the expected amortized cost of each event is no more than 3 times the cost to the adversary of that event. Let f denote the processor containing the page of the coin flipping algorithm, a denote the processor containing the page of $\hat{A}$, and ℓ denote the distance from f to a. We use the following potential function:

$$\Phi = 3D\ell$$

There are two events to consider in each round:

1. A request occurs at node r. Let f and a denote the nodes containing the pages of the coin flipping algorithm and the adversary, respectively, just prior to the request. There are three distances of interest: ℓ_0, the distance from f to a; ℓ_1, the distance from a to r; and ℓ_2, the distance from f to r. The cost to $\hat{A}$ is ℓ_1. The expected cost to the coin flipping algorithm is

$$\ell_2 + \frac{1}{2D}(D\ell_2 + 3D\ell_1 - 3D\ell_0)$$

By the triangle inequality, $\ell_2 \leq \ell_1 + \ell_0$. This implies that the expected cost to the coin flipping algorithm is at most $3\ell_1$.

2. $\hat{A}$ moves its page from a to a'. Let ℓ_0 and ℓ_1 denote the distance from f to a and f to a', respectively. Let ℓ_2 be the distance from a to a'. The cost to $\hat{A}$ of the move is $D\ell_2$. The cost to the coin flipping algorithm is $3D\ell_1 - 3D\ell_0$, which is at most $3D\ell_2$, by the triangle inequality. □

COROLLARY 5.3. *The coin-flipping algorithm is 3-competitive against an oblivious adversary.*

Proof. This follows trivially from the fact that an oblivious adversary is simply an adaptive on-line adversary that generates the same requests regardless of the random choices made by the on-line algorithm and services these requests optimally. □

COROLLARY 5.4. *On any graph the competitiveness of the coin-flipping algorithm against an adaptive off-line adversary tends to* $3 + 3\phi \approx 7.86$ *as* d *grows large.*

Proof. Theorem 2.2 of [2] states that if algorithm B is β-competitive against an adaptive on-line algorithm and there exists an algorithm A that is α-competitive against an oblivious adversary, then B is $\alpha\beta$-competitive against an adaptive off-line algorithm. We take A to be the general algorithm with deterministic resetting from Section 4.2 and B to be the coin-flipping algorithm. For various values of D, the competitive factor of the coin-flipping algorithm against an adaptive off-line adversary can be found by multiplying by 3 the appropriate entries in Table 3. □

COROLLARY 5.5. *There exists a deterministic algorithm for the migration problem with competitive ratio that tends to* $3 + 3\phi \approx 7.86$ *as* D *grows large.*

Proof. Theorem 2.1 of [2] states that if there is a randomized algorithm that is α-competitive against any adaptive off-line algorithm, then there is also a α-competitive deterministic algorithm. The theorem gives a method of constructing the deterministic algorithm, but we have not attempted to do so here. □

6. Remarks. This paper presented randomized algorithms for page migration. We have shown

(i) an algorithm for uniform graphs that tends toward being 2.38-competitive as D gets large.

(ii) an algorithm for general graphs that is parameterized by a resetting distribution. Deterministic resetting gives a variant that tends toward being $1+\phi$-competitive as D gets large.

(iii) a very simple coin-flipping algorithm that is 3-competitive against adaptive on-line adversaries.

We have also shown that the analysis is tight for each algorithm and that coin-flipping is optimal against an adaptive on-line adversary.

There are several interesting open problems. One is to improve upon our randomized algorithms, and to improve or generalize the lower bounds. A second is to find a good deterministic algorithm for migration on general graphs. Another open problem is to extend any of these results to the more general problem of 1-server with excursions.

In recent work, Bartal *et al.* [1] studied file allocation in distributed systems. In this situation multiple physical copies of writeable pages or data files are sometimes allowed. In this problem, the competitive ratio rises to $\Theta(\log n)$, where n is the number of processors.

7. Acknowledgements. The author thanks Daniel D. Sleator for suggesting the migration problem and Nick Reingold for helpful conversations.

REFERENCES

[1] Y. Bartal, A. Fiat, and Y. Rabani, *Competitive algorithms for distributed data management*, in Proc. ACM Symp. on Theory of Computing, 1992, pp. 39–50.

[2] S. Ben-David, A. Borodin, R. M. Karp, G. Tárdos, and A. Wigderson, *On the power of randomization in on-line algorithms*, in Proc. 22nd ACM Symp. on Theory of Computing, May 1990, pp. 379–386.

[3] P. Berman, H. J. Karloff, and G. Tardos, *A competitive three-server algorithm*, in Proc. 1st ACM-SIAM Symp. on Discrete Algorithms, 1990, pp. 280–290.

[4] D. Black, A. Gupta, and W. Weber, *Competitive management of distributed shared memory*, in Proceedings, Spring Compcon 1989, IEEE Computer Society, San Francisco, CA., 1989, pp. 184–190.

[5] D. L. Black and D. D. Sleator, *Competitive algorithms for replication and migration problems*, Tech. Report CMU-CS-89-201, Department of Computer Science, Carnegie-Mellon University, 1989.

[6] A. Borodin, N. Linial, and M. Saks, *An optimal online algorithm for metrical task systems*, in Proc. 19th ACM Symp. on Theory of Computing, 1987, pp. 373–382.

[7] M. Chrobak, H. Karloff, T. Payne, and S. Vishwanathan, *New results on server problems*, in Proc. 1st ACM-SIAM Symp. on Discrete Algorithms, 1990, pp. 291–300.

[8] M. Chrobak, L. L. Larmore, N. Reingold, and J. Westbrook, *Page migration algorithms using work functions*, Tech. Report YALEU/DCS/TR-897, Yale University, November 1991.

[9] W. Crowther, J. Goodhue, E. Starr, R. Thomas, W. Milliken, and T. Blackadar, *Performance measurements on a 128-node butterfly parallel processor*, in Proc. International Conf. on Parallel Processing, IEEE Computer Society, 1985, pp. 531–540.

[10] S. Irani, N. Reingold, D. D. Sleator, and J. Westbrook, *Randomized algorithms for the list update problem*, in Proc. 2nd ACM-SIAM Symp. on Discrete Algorithms, 1991, pp. 251–260.

[11] A. Karlin, M. Manasse, L. Rudolph, and D. Sleator, *Competitive snoopy caching*, Algorithmica, 3 (1988), pp. 79–119.

[12] A. R. Karlin, M. S. Manasse, L. A. McGeoch, and S. Owicki, *Competitive randomized algorithms for non-uniform problems*, in Proc. 1st ACM-SIAM Symp. on Discrete Algorithms, 1990, pp. 301–309.

[13] M. Manasse, L. A. McGeoch, and D. Sleator, *Competitive algorithms for on-line problems*, in Proc. 20th ACM Symp. on Theory of Computing, 1988, pp. 322–333.

[14] G. Pfister and et al., *The IBM research parallel processor prototype: Introduction and architecture*, in Proc. International Conf. on Parallel Processing, IEEE Computer Society, 1985, pp. 764–771.

[15] P. Raghavan and M. Snir, *Memory versus randomization in on-line algorithms*, Research Report RC 15622 (No. 69444), IBM T. J. Watson Reseach Center, 1990.

[16] N. Reingold, J. Westbrook, and D. D. Sleator, *Randomized algorithms for the list update problem*, Algorithmica, 11 (1994), pp. 15–32.

[17] C. Scheurich and M. Dubois, *Dynamic page migration in multiprocessors with distributed global memory*, IEEE Transactions on Computers, 38 (1989), pp. 1154–1163.

[18] D. D. Sleator and R. E. Tarjan, *Amortized efficiency of list update and paging rules*, Comm. ACM, 28 (1985), pp. 202–208.

[19] A. Wilson, *Hierarchical cache/bus architecture for shared memory multiprocessors*, in Proc. 14th International Symp. on Computer Architecture, ACM SIGARCH/IEEE Computer Society, 1987, pp. 244–252.