Yale University
Department of Computer Science

Randomized Algorithms for Multiprocessor Page Migration

Jeffery Westbrook[1]

YALEU/DCS/TR-844
January 1991

[1]Department of Computer Science, Yale University, New Haven, CT 06520-2158. Research partially supported by NSF grant CCR-9009753

1 Introduction

Recently much attention has been given to *competitive analysis* of on-line algorithms [5, 11, 13, 15]. Roughly speaking, an on-line algorithm is c-competitive if, for any request sequence, its cost is no more than c times the cost of the optimum off-line algorithm for that sequence.

Black and Sleator [4] and Black, Gupta, and Weber [3] studied several on-line problems that arise in a shared memory multiprocessor system. A common design for such a system is a network of processors, each of which has its own local memory [7, 12, 16]. In such a design, a programming abstraction of a single global memory address space is supported by a virtual memory system that distributes the physical pages of memory among the processor local memories.

In the page migration problem, the assumption is made that there is only one copy of each page of memory. This avoids the problem of maintaining consistency between multiple copies of a given page. When processor p wishes to read or write to memory address a, located in page k, it first looks to see if page k is contained in its own local memory. If so, then the memory access is done immediately. If not, p determines the processor q that does hold page k, 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. 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 memory requests from other processors.

For any sequence of page access requests, there is an optimum schedule of page migrations that minimizes the total communication overhead. If the entire sequence is known, this schedule can be computed by an off-line algorithm. The page migration problem is to design an on-line algorithm that, on any sequence of requests, is c-competitive with the optimum schedule. The on-line algorithm must make its decisions without knowledge of the future requests.

More precisely, an instance of the migration problem consists of an edge-weighted graph G, corresponding to the interconnection network, and a sequence of requests σ. One node of G, denoted P, is distinguished; this is the node that contains the page. A *request at node* r corresponds to a memory reference to the page. The cost of a request is the (shortest) distance in the graph between r and P. After satisfying a request, the algorithm may move the page from P to a new node P'. The cost of the move is d times the distance between P and P', where d is a constant no smaller than 1. (Constant d corresponds to the size of the page).

Black and Sleator considered two kinds of graphs: complete graphs with each edge having length 1, and trees with arbitrary edge lengths. They developed deterministic algorithms for these two cases that are 3-competitive. In addition, they showed that

no deterministic algorithm could be better than 3-competitive, even if the graph consists of only a single edge. In this paper we develop randomized algorithms that beat this bound against oblivious adversaries. We give a simple algorithm, UNIFORM, for the complete graph with unit edge lengths. This algorithm is 2.38-competitive for large values of d. Then we develop an randomized algorithm for any graph whose distances satisfy the triangle inequality. This algorithm is $(1+\phi)$-competitive for large d, where ϕ is the Golden Ratio, approximately 1.62. We also give a third algorithm that is better for small values of d. Note that the best known deterministic algorithm for a general graph is $2d+2$-competitive. The randomized algorithms are "barely random", in the sense that they only use a small number of random bits during an initialization phase, and from then on run deterministically. Such barely random algorithms have practical value since random bits are often an expensive resource. Our randomized algorithms are also simpler than the deterministic ones of Black and Sleator.

Similar memory management problems have been studied by [8, 10, 13]. Migration is a special case of the *1-server with excursions* problem defined by Manasse et. al. [11]. Migration and 1-server with excursion are also related to the *k-server* problems that have recently received much attention [2, 6, 11]. Practical issues and applications of page migration are discussed more fully in Black and Sleator [4] and Black, Gupta, and Weber [3].

2 Competitive Analysis

An *on-line* list update algorithm must service each request without any knowledge of future requests. An *off-line* algorithm is shown the entire sequence in advance; the optimum cost can always be achieved by an off-line algorithm. Following [5, 11] we say a deterministic list update algorithm, A, is *c-competitive* if there is a constant k such that for all size lists, all off-line algorithms, $\hat{A}$, and all request sequences σ, $A(\sigma) \leq c \cdot \hat{A}(\sigma) + k$. For randomized list update algorithms the competitiveness of an algorithm is defined with respect to an *adversary*. Following [1, 13] we consider three kinds of adversaries. The *oblivious* or *weak* adversary generates a complete request sequence before the on-line algorithm begins to process it, and is charged the cost of the optimum off-line algorithm for that sequence. The *adaptive on-line* or *medium* adversary is allowed to watch the on-line algorithm in action, and generate the next request based on all previous moves made by the on-line algorithm. The adversary must also service the requests on-line, however. The *adaptive off-line* or *strong* adversary also generates the requests adaptively, but is charged the optimum cost for the sequence.

A randomized on-line algorithm, A, is *c-competitive against weak (respectively medium, strong) adversaries* if there is a constant k such that for all size lists, all weak (respectively medium, strong) adversaries, $\hat{A}$, and all request sequences σ,

$\mathrm{E}\left[A(\sigma) - c \cdot \hat{A}(\sigma)\right] \leq k$. The expectation is over the random choices made by the on-line algorithm. If a randomized algorithm is c-competitive against oblivious adversaries then for any sequence σ, the expected cost of processing σ using algorithm A is no more than c times the cost of the optimum off-line algorithm plus some constant k.

The results in this paper all apply to oblivious adversaries, which seem the most natural to consider. One can apply results from [9, 14] to show that even if the migration graph consists of only a single edge, no randomized algorithm can be better than 3-competitive against either an adaptive on-line or an adaptive off-line adversary. One can also show that for $d = 1$, no randomized algorithm can be better than 1.5-competitive.

3 Uniform Graphs

In this section we describe and analyze a family of randomized algorithms UNIFORM(d) for use on complete graphs with unit distance between any pair of nodes. The cost of moving the page across an edge is a fixed multiple d of the cost of accessing the page across an edge.

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. [9, 14] gave a simple randomized algorithm for the list-update problem, which our UNIFORM(d) family closely parallels.

Each node of the graph has an associated counter that counts from 0 to k. The optimum value of k depends on d; we will see later how to choose k. Prior to processing any requests, the counter at each node is randomly initialized to a value between 0 and $k-1$. After this initialization, which uses $\Theta(n \log k)$ random bits, UNIFORM(d) services the request sequence deterministically as follows. Consider an request at node x. The request is satisfied (at cost 1), and then the counter at node x is incremented. If the counter at x has value k, the page is moved to node x and the counter is reset to 0.

Note that the initial value of each counter is uniformly distributed between 0 and $k-1$, and hence after j accesses to item x the value of its counter is still takes on any value between 0 and $k-1$ with equal probability. We now analyze the competitiveness of UNIFORM(d) against an oblivious adversary.

Theorem 3.1 *Let σ be any sequence of m requests, and let OPT be any deterministic off-line algorithm that services σ. Assuming that UNFORM(d) and OPT begin with the page at the same node, UNIFORM(d) is c_k-competitive against OPT, where c_k is the maximum of $1 + \frac{k+1}{2d}$ and $1 + \left(\frac{1}{k}\right)\left(2d + \frac{k+1}{2}\right)$.*

This theorem immediately implies that UNIFORM is c_k-competitive against an oblivious adversary, since such an adversary chooses a request sequence ahead of time,

with no knowledge of the random choices UNIFORM(d) makes, and is charged the optimum cost among off-line algorithms for that sequence. (Several algorithms may achieve the optimum cost.) For the remainder of this proof we assume that d is given and refer to the appropriate UNIFORM(d) algorithm simply as UNIFORM.

The proof uses the following observation. Since OPT is deterministic, OPT's page is always located at node x_j at the time of the j^{th} request. Meanwhile, the values of the UNIFORM's counters are uniformly distributed between 0 and $k-1$. Thus the location of the OPT's page conveys no information about the value of UNIFORM's counters. (This argument can be made more rigorous by examining the sample space associated with the events "counter at x has value j and OPT's page is at node y", for all nodes x and y and values j.) This observation is not true if the adversary is adaptive, since the adversary may adjust the request sequence, and hence the position of its page, depending on the random choices made by the on-line algorithm.

Let u be the node containing the on-line algorithm's page and *opt* be the node containing OPT's page. Let C_x denote the value of the counter at any node x. To prove the theorem we use the following potential function Φ that maps a two-tuple (x, y) to the integers, where x and y are nodes.

$$\Phi = \begin{cases} 0 & \text{if } u = opt \\ (d + k - C_{opt}) & \text{if } u \neq opt \end{cases}$$

The amortized cost of an event performed by either OPT or UNIFORM is a random variable defined as the actual cost of the operation to UNIFORM plus the change in the potential function, $\Delta\Phi$. Let e be an event and let

$$X = (\text{the cost of } e \text{ to UNIFORM}) + \Delta\Phi - c_k \cdot (\text{the cost of } e \text{ to OPT}).$$

To prove the theorem, we show that for each operation $\mathrm{E}[X] \leq 0$.
First, suppose the event is a request.

1. The request is from node u. In this case UNIFORM pays 0 and OPT pays at least zero, so $\mathrm{E}[X] \leq 0$.

2. The request is from node $r \neq u, opt$. The cost to OPT is 1. UNIFORM pays 1 for the request. With probability $1/k$ the incremented counter at node r takes on value k, in which case UNIFORM moves the page to r at cost d. In the worst case, $u = opt$ prior to this move, and the potential increases by $d + k - C_{opt}$. We have

$$\mathrm{E}[C_{opt}] = \sum_{0 \leq i \leq k-1} \frac{i}{k} = \frac{k-1}{2}$$

Thus the expected amortized cost to UNIFORM of this request is at most

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

which is c_k times the cost to OPT.

3. There is an access to node $opt \neq u$. In this case OPT pays zero, so we must show that the amortized cost to UNIFORM is also zero. The actual cost of the request to UNIFORM is 1. Since the counter is always incremented, the potential decreases by at least 1. If the counter reaches k, then UNIFORM's page moves to opt. This costs d, but decreases the potential by a further d. Thus the amortized cost is always 0.

The other possible event is that OPT moves its page to a new node r. The actual cost to OPT is d, and the actual cost to UNIFORM is 0. Now we must consider the cost to UNIFORM due to changes in potential. There are three cases:

1. $r = u$. In this case, the potential decreases by at least d, so $X \leq 0$.

2. $r \neq u$ and initially $opt = u$. In this case, the potential increases by $d + k - C_R$. The expected value of C_R is $\frac{k-1}{2}$, as above. Thus the expected cost to UNIFORM is $d + (k+1)/2$, which is at most c_k times the cost to OPT.

3. $r \neq u$ and initially $opt \neq u$. In this case the potential changes by $C_{opt} - C_R$. In the worst case C_{opt} is k, while the expected value of C_R is $\frac{k-1}{2}$. Thus the expected cost to UNIFORM is less than 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 $1 + \left(\frac{1}{k}\right)(2d + \frac{k+1}{2})$.

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

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

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 [14].

4 General Graphs

In this section we consider the migration problem on graphs with arbitrary distances between nodes. The only assumption is that the distances satisfy the triangle inequality. This is clearly reasonable; an algorithm can ignore edges that do not satisfy the triangle inequality by choosing the shortest path between the endpoints. This observation also allows us to handle non-complete graphs such as trees; the algorithm can behave as though the graph were complete by adding pseudo-edges, taking the

	UNIFORM	
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(d) and RANDOM RESET

shortest path between two nodes in the original graph whenever moving across a psuedo-edge.

The best known competitive ratio for general graphs is the minimum of $2d + 2$ and $2n - 1$; the first bound comes from a simple algorithm that moves its page to the requesting node at each request, and the second from applying the general metrical taski system algorithm [5]. Thus even when the graph is a single triangle, the best known deterministic algorithm is only 5-competitive. Furthermore, Black and Sleator's algorithm for trees is somewhat complicated. Our randomized algorithms for this problem are simple enough to be practical, and beat the deterministic lower bound.

The basic approach is to maintain for the page a single counter that runs between 0 and k. After each request, the counter is incremented by one. When the counter reaches k, the page is moved to the location of the current request. After the move, the counter is reset according to a given resetting distribution. We consider two possible resetting distributions. The first resets the counter to 0 with probability 1. We begin the algorithm by randomly initializing the counter value. This resetting choice produces a barely random algorithm that gives best results when d is large.

The second resetting choice resets the page counter to each of the possible values between 0 and $k - 2$ with equal probability, $\frac{1}{k-1}$. We begin the algorithm by setting the counters to the steady state distribution of the Markov chain that describes the algorithm. This distribution gives better results for smaller values of d, but uses a number of random bits proportional to the length of the request sequence.

4.1 A General Algorithm with Unique Resetting

Theorem 4.1 *Let σ be any sequence of m requests, and let OPT be any deterministic off-line algorithm that services σ. The general algorithm G that resets to 0 with*

probability 1 is c_k-competitive, where c_k is the maximum of $1+\frac{k+1}{2d}$ and $1+\left(\frac{1}{k}\right)(2d+k)$) (assuming that the on-line algorithm and OPT begin with the page at the same node).

Note that the initial value of the counter is uniformly distributed between 0 and $k-1$, and hence after j requests the value of the counter still takes on any value between 0 and $k-1$ with equal probability.

Again we use the observation that since OPT is deterministic, OPT's page is always located at node x_j at the time of the j^{th} request. Similarly, OPT always makes the same move at time j, irrespective of either the location of the on-line algorithm's page or the values of the on-line algorithm's counter. In particular, the value of the counter at time j is given by it's steady state distribution.

At any time, let g be the node containing the on-line algorithm's page and let *opt* be the node containing OPT's page. Let C denote the value of the counter. Let $\ell_{x,y}$ be the length of the edge connecting nodes x and y. For convenience, we abbreviate $\ell_{g,opt}$ by ℓ_G.

To prove the theorem we use the potential function

$$\Phi = \begin{cases} 0 & \text{if } g = opt \\ (d+k-C)\ell_G & \text{if } g \neq opt \end{cases}$$

First we examine the cost of satisfying a request when $g = opt$. If the request occurs at g then both G and OPT pay zero. Suppose the request occurs at node r. Both OPT and G pay $\ell_{g,r}$ to satisfy the request. With probability $1/k$, G's counter reaches k, and G moves the page to node r at cost $d\ell_{g,r}$. After the move the counter is reset to 0, and $\ell_G = \ell_{g,r}$, so the increase in potential is $\ell_{g,r}(d+k)$. Therefore the expected cost to G of the access is

$$\ell_{g,r}\left(1+\frac{1}{k}(2d+k)\right).$$

This is at most c_k times the cost to OPT.

Now we consider the cost of a request when $g \neq opt$. When the request is to g, G clearly pays less than OPT. Suppose the request is to *opt*. OPT pays zero, while G pays ℓ_G. Since the counter increases, Φ changes by $-\ell_G$. If the counter reaches k, then the page is moved to *opt*. This costs $d\ell_G$, but decreases Φ by the same amount. Therefore the decrease in the potential always pays for the cost of G's actions, and the amortized cost to G is zero.

Lastly we consider the case that $g \neq opt$ and the request is to a third node r. In this case, G pays $\ell_{g,r}$ while OPT pays $\ell_{opt,r}$. The counter increases by one, so the potential decreases by ℓ_G. With probability $\frac{1}{k}$, G moves its page to node r. This has actual cost $d\ell_{g,r}$, and changes the potential by $(d+k)\ell_{opt,r} - d\ell_G$.

Therefore, the expected amortized cost of this request is

$$\ell_{g,r} - \ell_G + \frac{1}{k}\left(d\ell_{g,r} + \ell_{opt,r}(d+k) - d\ell_G\right).$$

Using the triangle inequality, we substitute $\ell_{g,r} \leq \ell_G + \ell_{opt,r}$ to find that the expected amortized cost is at most

$$\ell_{opt,r} + \frac{1}{k}\left(d\ell_{opt,r} + (d+k)\ell_{opt,r}\right).$$

Therefore the expected amortized cost is at most $1 + \frac{1}{k}(2d+k)$ times the cost to OPT.

This handles the cost of satisfying a request. The other possible event is that OPT moves its page from node *opt* to a new node *opt'*. The actual cost to OPT is $d\ell_{opt,opt'}$, and the actual cost to the on-line algorithm is 0. Now we consider the cost due to changes in potential. There are three cases:

1. $opt' = g$. In this case, the potential decreases by at least $d\ell_{opt,g}$, so the expected cost is at most 0.

2. $opt = g$ and $opt' \neq g$. In this case, the potential increases by $d + k - C_{opt'}$. Since each counter value is equally likely, the expected value of $C_{opt'}$ is $\frac{k-1}{2}$. Thus the expected cost to the on-line algorithm is is $\ell_{g,opt'}(d + \frac{k+1}{2})$, which is at most c_k times the cost to OPT.

3. $opt \neq g$ and $opt' \neq g$. The cost to OPT is $d\ell_{opt,opt'}$. The change in potential is $(d+k-C)(\ell_{g,opt'} - \ell_{g,opt})$. Again, the expected value of C is $\frac{k-1}{2}$, and we use the triangle inequality to conclude that $\ell_{g,opt'} \leq \ell_{g,opt} + \ell_{opt,opt'}$. Thus the expected amortized cost to the on-line algorithm is at most $\ell_{opt,opt'}(d + \frac{k+1}{2})$, which is at most c_k times the cost to OPT.

This completes the proof of Theorem 4.1. Given a value of d, we can choose k to minimize the maximum of $1 + \frac{k+1}{2d}$ and $1 + \left(\frac{1}{k}\right)(2d+k)$.

Table 2 shows the best competitive ratio for this general algorithm values of d up to 10. These values are found by setting

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

and solving for k in terms of d. Then the best integer approximation to this value is taken. The optimum value of k is given by

$$d + \frac{1}{2}(-1 + \sqrt{20d^2 - 4d + 1})$$

As d gets large, the best competitive ratio decreases and tends to $1 + \frac{1+\sqrt{5}}{2}$. This is one plus the Golden Ratio, or approximately 2.62.

d	best k	comp. ratio
1	3	3.00
2	6	2.75
3	9	2.67
4	12	2.67
5	15	2.67
6	18	2.67
7	22	2.64
8	25	2.64
9	28	2.64
10	31	2.65

Table 2: Best competitive ratios for general algorithm, resetting to zero with probability 1

4.2 A General Algorithm with Almost Uniform Resetting

The second resetting distribution we consider resets the counter each of the possible values between 0 and $k-2$ with equal probability, $\frac{1}{k-1}$. The counter value forms a Markov chain in which state i corresponds to the counter having value $k-i$, for $1 \leq i \leq k$. The transition matrix for this Markov chain is as follows. (Entry (i,j) contains the probability of transiting to counter value $k-i$ from counter value $k-j$).

$$\begin{pmatrix} 0 & 1 & 0 & 0 & \dots & 0 \\ \frac{1}{k-1} & 0 & 1 & 0 & & 0 \\ \frac{1}{k-1} & 0 & 0 & \ddots & & 0 \\ \vdots & & & & & \\ \frac{1}{k-1} & & & & & 1 \\ \frac{1}{k-1} & & & & & 0 \end{pmatrix}$$

The steady state distribution of this Markov chain is given by the eigenvector corresponding to eigenvalue 1. In steady state, $p_1 = \frac{2}{k+2}$ and $p_i = p_1 \frac{k+1-i}{k-1}$ for $2 \leq i \leq k$. Here p_i is the probability of being in state i (the counter having value $k-i$).

The algorithm begins by initializing the counter according to the steady state distribution. From then on, it increments the counter each request, until the counter reaches value k, at which time it resets according the reset distribution.

Theorem 4.2 *Let σ be any sequence of m requests, and let OPT be any deterministic off-line algorithm that services σ. The general algorithm that resets uniformly to all values between 0 and $k-1$ is c_k-competitive, where c_k is the maximum of $2+\frac{4d}{k+2}$ and $1+\frac{k^2+4k+6}{3d(k+2)}$. (assuming that the on-line algorithm and OPT begin with the page at the same node).*