Page Migration Algorithms Using Work Functions (Extended Abstract)

Marek Chrobak[1]* Lawrence L. Larmore[1]*
Nick Reingold[2]** Jeffery Westbrook[3]***

[1] Department of Computer Science, University of California, Riverside, CA 92521.
[2] AT&T Bell Laboratories, Murray Hill, NJ 07974-0636.
[3] Department of Computer Science, Yale University, New Haven, CT 06520-2158.

Abstract. The *page migration* problem is the management problem for 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 equal to the network distance. At times the page may migrate between processors, at a cost equal to the distance times the page size factor, D. The problem is to schedule movements on-line so as to minimize the total cost of memory references. We consider the problem under the restriction that movement can only occur after a request has been served and before the next request is known. The major results are: we give randomized $(2+\frac{1}{2D})$-competitive on-line algorithms for trees (and products of trees, including the hypercube), and for a uniform space when $D = 1, 2$. We show that these algorithms are optimal. We prove a $\frac{85}{27}$ lower bound on the competitiveness constant of a deterministic algorithm (in arbitrary spaces) with $D = 1$, disproving a conjecture by Black and Sleator. We show a deterministic $(2+\frac{1}{2D})$-competitive algorithm for continuous trees. Our analysis is based on *work functions*, which provide a systematic approach to many competitive analysis problems.

1 Introduction

The *page migration* problem is the management problem for 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 equal to the network distance. At times the page may migrate between processors, at a cost equal to the distance times the page size factor, D. The problem is to schedule movements on-line so as to minimize the total cost of memory references.

The page migration problem has been considered by the architecture community, where it arises as a memory management problem in a globally addressed

* Research partially supported by NSF grant CCR-9112067.
** Research partially supported by NSF grant CCR-8958528.
*** Research partially supported by NSF grant CCR-9009753.

shared memory in a multiprocessor system [3, 4, 16]. A common design for such a system is a network of processors, each of which has its own local memory [8, 14, 18]. 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. If processor p contains a copy of page b, then it can directly read and write memory addresses on b. If p does not have a copy, however, then it must satisfy memory accesses by transmitting a request through the network to some processor q that does have a copy of b and waiting for an answer from q. The communication cost of this request depends upon the network, and may vary depending on the choice of p and q. A very similar situation arises in distributed databases, where multiple copies of a data file may be stored at various local nodes.

Having a given virtual page stored at multiple processors reduces communication overhead during reads, but introduces the problem of maintaining consistency among the multiple copies during writes, something for which most multiprocessors do not provide mechanisms [4]. Therefore, various network designers have proposed restricting each writable page to a single copy [3, 4, 16], and allowing the operating system to move the page between processors in response to changes in access patterns. The *page migration* problem is to decide, on-line, a sequence of page movements that minimizes the cost of communication.

We formalize the page migration problem as follows: An instance of the problem consists of a metric space M, and an integer constant $D \geq 1$ (the page size). Let $s \in M$ denote the current position of the page, which we think of as a "server." A sequence ϱ of requests arrives on line. Each request is a point $r \in M$. Once a request r arrives, it must be immediately served at cost $d(s,r)$, the distance from s to r. This is called the *access cost*. We are then allowed to move the server from s to any other point x at cost $D \cdot d(s,x)$. This is called the *movement cost*. The server is initially placed at point s_0, but it may be moved to an arbitrary point s before the first request, at the usual cost $D \cdot d(s_0, s)$.

Let $\mathcal{A}$ be an on-line algorithm for the page migration problem. We denote the cost of $\mathcal{A}$ on a request sequence ϱ if the initial position is s_0 by $cost_{\mathcal{A}}(s_0, \varrho)$, and we denote the minimum cost of servicing the request sequence ϱ starting from s_0 by $cost_{opt}(s_0, \varrho)$. This minimum cost can be achieved by an off-line algorithm (one which knows the entire sequence ϱ in advance), but, in general, cannot be achieved on-line.

Let $c \geq 1$. Algorithm $\mathcal{A}$ is called *c-competitive* in M if for each $x \in M$ there is a constant a_x

$$cost_{\mathcal{A}}(x, \varrho) \;\leq\; c \cdot cost_{opt}(x, \varrho) + a_x, \tag{1}$$

for all request sequences ϱ. (In our paper all algorithms will have $a_x = 0$.) A randomized algorithm $\mathcal{A}$ is c-competitive if for all sequences ϱ the above inequality holds for the expected cost of $\mathcal{A}$ on ϱ. In this paper we use the notation $cost_{\mathcal{A}}(x, \varrho)$ to denote expected cost of a randomized algorithm $\mathcal{A}$.

In the literature on competitive analysis of randomized algorithms, on-line problems are often viewed as a game played against an adversary that generates requests on-line. An adversary is called *oblivious* if it does not know the location of the page. Our definition of competitiveness corresponds to the oblivious adversary, since such an adversary may as well choose a request sequence in advance and serve it optimally.

Black and Sleator [4] consider two classes of networks: uniform spaces (complete graphs with each edge having length 1), and trees with arbitrary edge lengths. They develop 3-competitive deterministic algorithms for these two cases. In addition, they show that no deterministic algorithm could be better than 3-competitive in any metric space with at least two points. Besides this lower bound, little is known about deterministic algorithms that work for arbitrary metric spaces; the problem seems quite hard. Black and Sleator conjecture that a 3-competitive algorithm exists for all metric spaces. The best currently known upper bound of 7 was given recently by Awerbuch *et al.* [1]. Westbrook [17] gives a randomized algorithm for the uniform network whose competitiveness constant approaches $(5+\sqrt{17})/4 \approx 2.28$ as D approaches infinity, a $(1+\phi)$-competitive randomized algorithm[4] for general networks, and a randomized algorithm that is 3-competitive against an *on-line adaptive adversary*.

Similar memory management problems are examined in [2, 5, 10, 11, 12, 13, 15]. Practical issues and applications of page migration are discussed more fully in [3, 4, 16]. The closely related *file assignment* problem, an important problem in distributed databases, involves the static allocation of multiple copies of data files to the nodes of a distributed system, see [9, 19, 1].

A major difficulty in traditional competitive analysis has been to understand optimal off-line algorithms. In particular, it is almost always impossible to know when, or to where, an optimal algorithm will move its server (the page). The approach in this paper is to use *work functions*, which completely describe the actions of an optimal off-line algorithm, to the extent that an on-line algorithm can know it. Work functions were defined in [7] and are similar to the residues of [13]. Work functions provide a systematic approach to problems of competitive analysis.

Overview of the paper. In Sect. 2 we define work functions and our approach to competitive analysis. In Sect. 3 we consider the simple case where the metric space consists of only two points. We present a $(2+\frac{1}{2D})$-competitive randomized algorithm, EDGE , and prove the matching lower bound. In Sect. 4 we consider uniform spaces of arbitrary size. Quite unexpectedly, this problem appears to be very hard. We give an extension of EDGE to uniform spaces, prove that this extension is optimal for $D = 1, 2$, and conjecture that it is optimal for any $D \geq 3$. In Sect. 5 we present a randomized algorithm for trees called FACTOR. FACTOR can be viewed as a "product" of a number of instances of RAND (one instance for each edge). FACTOR is $(2+\frac{1}{2D})$-competitive, *i.e.*, optimal. FACTOR can also be applied, with the same competitive constant, to metric spaces that are Cartesian products of trees with the $\mathcal{L}^1$ metric, including hypercubes and meshes.

[4] $\phi = 1.618\ldots$, the Golden Ratio.

We then present a number of results about deterministic algorithms. In Sect. 6 we consider continuous trees, and we show how to use continuity to turn a randomized algorithm into a deterministic one with the same constant, $2+\frac{1}{2D}$. In Sect. 7 we disprove the conjecture of Black and Sleator by demonstrating a simple metric space for which there is no algorithm with competitiveness constant better than $\frac{85}{27} \approx 3.148$ for $D = 1$. In Sect. 8, we also give an optimal 3-competitive algorithm for the space consisting of a single triangle and $D = 1$. This algorithm uses a technique called *forgiveness*, that has also been recently applied to define an 11-competitive algorithm for the 3-server problem [6].

2 Work Functions, Algorithms, and Competitiveness

Work Functions. In their original paper on the k-server problem, Manasse *et al.* [13] described a dynamic program for computing the optimum cost in an instance of the k-server algorithm. They discovered that the information carried in the matrix of this dynamic program can be useful both in executing an on-line algorithm and in analyzing its performance.

Work functions (see [7]) extend and generalize this dynamic programming approach. Fix D. We say that a function $\omega : M \to \mathbb{R}^+$ is a *work function* if it satisfies the following two conditions:

(wf1) $\omega(x) - \omega(y) \leq D \cdot d(x, y)$ for all $x, y \in M$.

(wf2) For some point $x \in M$, $\sup_y \{D \cdot d(x, y) - \omega(y)\} < \infty$.

Let W_M be the set of all work functions on M. The *update operator*, $\wedge$, is defined by $(\omega \wedge r)(x) = \inf_z \{\omega(z) + d(z, r) + D \cdot d(z, x)\}$.

It is quite easy to show that $\wedge$ is well-defined, that is, that $\omega \wedge r$ is a work function. In general, the infimum in the definition of $\wedge$ may not be achieved. However, all work functions described in this paper will have a well-defined minimum, and hence we may replace the infimum by a minimum in the above statements. Inductively, we can extend the update operator to sequences of requests: $\omega \wedge \varepsilon = \omega$ and $\omega \wedge \varrho r = (\omega \wedge \varrho) \wedge r$. The *characteristic function* of a point, χ_x, is defined by $\chi_x(y) = D \cdot d(x, y)$.

Lemma 1. *Let ϱ be a request sequence. Then the value $\omega(y) = (\chi_s \wedge \varrho)(y)$ is the minimum cost to service request sequence ϱ starting with the server at point s and ending with the server at point y.*

A work function ω is called *reachable* if $\omega = \chi_x \wedge \varrho$ for some x, ϱ. Note that our definition allows work functions that are not reachable. (This will be useful in our algorithm for the triangle.) When analyzing the behavior of an on-line algorithm after processing request sequence ϱ starting at position s, we will refer to the function $\omega = \chi_s \wedge \varrho$ as the *current* work function. Suppose the next request is r. The new current work is $\omega' \wedge r = \chi_s \wedge \varrho r$. We call the quantity $\inf(\omega') - \inf(\omega)$ the *amortized optimal cost* of request r, denoted $\Delta cost_{opt}$.

An *offset function* is a work function whose infimum value is zero. One special example of a work function which is also an offset function is χ_x. For a work function ω define offset(ω) to be the offset function given by $\omega - \inf(\omega)$.

Our on-line algorithms are based on offset functions. At any time, the algorithm maintains a current offset function that it uses to make decisions about the server movement. The current offset function will be exactly the one defined by $\mathrm{offset}(\chi_s \wedge \varrho)$, where s is the initial position and ϱ is the request sequence up to the current time. Since we will be dealing primarily with offsets, we define another update operator $\triangle$ by $\omega \triangle r = \omega \wedge r - \inf(\omega \wedge r)$. Note that $\omega \triangle r$ is always an offset function. The operator $\triangle$ can be extended to sequences of requests in an analogous way to $\wedge$. We say an offset function is *reachable* if it is of the form $\chi_x \triangle \varrho$ for some $x \in M$ and some sequence of requests $\varrho \in M^*$.

Randomized Algorithms. A probability distribution μ on a metric space M is said to be *finite* if there exists a finite set $X \subseteq M$. such that $\mu(X) = 1$. Let Π_M denote the set of all finite probability distributions on M. Throughout this paper we say "distribution" for short when we mean finite probability distribution. If $\mu \in \Pi_M$ and $\mu(X) = 1$, we will say that X *supports* μ. The smallest set that supports μ is called the *support of* μ. If the support of μ is a singleton, $\{x\}$, by an abuse of notation, we will identify x and μ, *e.g.*, we will write $\mu = x$.

In this paper, a randomized algorithm $\mathcal{A}$ will be viewed as a *deterministic* algorithm whose states, at each step, will be server distributions on M.

We say that a distribution α on $M \times M$ is a *transport* between distributions μ and ν on M, if $\mu(x) = \sum_u \alpha(x, u)$ and $\nu(x) = \sum_u \alpha(u, x)$ for all $x \in M$. A transport describes a way for a randomized algorithm $\mathcal{A}$ to change the distribution of its server from μ to ν, by specifying for each pair of points the probability that the server starts at x and ends at y. If the server position is x, then $\mathcal{A}$ moves its server to position y with probability $\alpha(x, y)/\mu(x)$.

The cost of transport α is defined by $\|\alpha\| = \sum_x \sum_y d(x, y) \cdot \alpha(x, y)$, and the transport distance between μ and ν is $\delta(\mu, \nu) = \min_\alpha \|\alpha\|$, where the minimum is over all transports α between μ and μ. We call an α for which $\|\alpha\| = \delta(\mu, \nu)$ a *minimal transport* from μ to ν.

Note that for $x, y \in M$, $d(x, y)$ (the distance from x to y) and $\delta(x, y)$ (the transport distance between distributions supported by $\{x\}$ and $\{y\}$) yield the same value. The access cost incurred by $\mathcal{A}$ is simply the expected value, given μ, of the distance between the server and r, which is equal to the transport distance $\delta(\mu, r) = \sum_x d(x, r) \cdot \mu(x)$. The movement cost is given by $D \cdot \|\alpha\|$, where α is the transport used by $\mathcal{A}$. We assume that any algorithm uses a minimum transport to move between distributions.

Let $\mathcal{A}(x_0, \varrho)$ be the distribution of the server of $\mathcal{A}$ after processing ϱ. Write $\varrho = r_1 \ldots r_k$, and let $\mu_i = \mathcal{A}(x_0, r_1 \ldots r_{i-1})$ Then

$$cost_{\mathcal{A}}(x_0, \varrho) = \sum_{i=0}^{k-1} \left(\delta(\mu_i, r_{i+1}) + D \cdot \delta(\mu_i, \mu_{i+1})\right). \qquad (2)$$

We call a randomized algorithm *stable* if it has the property that the current distribution depends only on the current offset function. In this case an algorithm can be defined simply as a function $\mathcal{A} : W_M^0 \to \Pi_M$. All algorithms described in this paper are stable.

Competitive Analysis with Offset Functions. Our proofs of competitiveness are based on the following method. Let $\mathcal{A}$ be a stable randomized on-line algorithm. (The method can be easily generalized to other types of on-line algorithms). We say that $\Phi : W_M^0 \to \mathbb{R}^+$ is a *c-potential for* $\mathcal{A}$ (or simply *potential*) if it satisfies the following condition: for any $r \in M$ and any reachable offset $\omega \in W_M^0$

$$\delta(\mu, r) + D \cdot \delta(\mu, \mu') + \Phi(\omega') - \Phi(\omega) \ \leq \ c \cdot \min_z (\omega \wedge r)(z). \tag{3}$$

where $\mu = \mathcal{A}(\omega)$ and $\mu' = \mathcal{A}(\omega')$. The quantity $\inf(\omega \wedge r)$ is called the amortized optimum cost. It is well-known that the existence of a c-potential implies that $\mathcal{A}$ is c-competitive.

We will often write the inequality (3) symbolically as

$$\Delta cost_{\mathcal{A}} + \Delta\Phi \ \leq \ c \cdot \Delta cost_{opt},$$

where $\Delta cost_{\mathcal{A}} = \delta(\mu, r) + D \cdot d(\mu, \mu')$, $\Delta\Phi = \Phi(\omega') - \Phi(\omega)$ and $\Delta cost_{opt} = \inf(\omega \wedge r)$.

3 Migration on Two Points

In this section we consider the page migration problem when the metric space M has only two points, x and y. Without loss of generality we can assume that the distance between these points is 1. In this case, an offset function is a pair of numbers: a value at x and a value at y. We will write $(\omega(x), \omega(y))$ for the offset function ω. The reachable offset functions are those in which one value is 0 and the other is an integer from 0 to D.

Algorithm RAND. Suppose the current offset function is $(0, b)$, where $0 \leq b \leq D$. The algorithm EDGE will use the probability distribution that places mass $p_b = \frac{D+b}{2D}$ on x and $1 - p_b = \frac{D-b}{2D}$ on y. When the offset function is $(b, 0)$, EDGE is defined in the obvious, symmetric manner.

Theorem 2. EDGE *is C_D-competitive for $C_D = 2 + \frac{1}{2D}$. Furthermore, no on-line algorithm for the page migration problem on two points can be less than C_D-competitive.*

4 Uniform Spaces

Let $C_D = 2 + \frac{1}{2D}$. We will define a C_D-competitive randomized algorithm for a uniform space, *i.e.*, a metric space M where $d(x, y) = 1$ for all distinct $x, y \in M$. The algorithm is called UNIFORM. As in the previous section, UNIFORM is stable, and thus it can be described as a function that assigns a probability distribution on M to each offset function. The algorithm is an extension of the edge algorithm.

Let ω be the current offset function. For each $x \in M$ we define its *weight* as $k_\omega(x) = (D - \omega(x))/(D + \omega(x))$. We let UNIFORM have the distribution

$$\mu(x) \;=\; \frac{k_\omega(x)}{\sum_{y \in M} k_\omega(y)}.$$

If M has only two points then UNIFORM is identical to algorithm EDGE .

Theorem 3. *In a uniform space,* UNIFORM *is C_D-competitive for $D = 1, 2$.*

Conjecture: UNIFORM is C_D-competitive for each $D \geq 3$.

Subsequent to this work, Lund and Reingold have constructed a randomized algorithm that is C_D-competitive in any uniform space. Their algorithm is a different generalization of EDGE .

5 Trees

In this section we present a $(2+\frac{1}{2D})$-competitive algorithm, FACTOR, for trees. This algorithm uses a technique called "factoring" that reduces one instance of the page migration problem on a tree to a number of instances of the problem on two-point spaces, one for each edge of the tree.

We define a finite metric space M to be a *tree* if its points are the vertices of an acyclic connected graph whose edges have nonnegative weights, and its distances are the weighted path lengths in that graph. We will denote an edge of M between vertices $x, y \in M$ by $\{x, y\}$, and the set of all edges of M by E. For $x, y \in M$, we denote the set of vertices that are on the unique path in M from x to y by $[x, y]$.

Let M be a tree. We refer to the page migration problem on M as PMP^M. We will show that PMP^M "induces" in a natural way an on-line problem PMP^e on each edge $e = \{x, y\}$, which we view as a space of two points at distance $d(x, y)$. The removal of e disconnects M into two subtrees M_x and M_y, one containing x and the other containing y. For each $z \in M$, we define $z^e = x$ if $z \in M_x$ and $z^e = y$ if $z \in M_y$. This defines a map from request sequences in M to request sequences in each $e = \{x, y\}$. Any algorithm $\mathcal{A}$ on M induces an algorithm $\mathcal{A}^e$ on e as follows: Whenever $\mathcal{A}$ moves its server from u to v, $\mathcal{A}^e$ moves a server from u^e to v^e (u^e may be equal to v^e). Notice that for this move $\mathcal{A}$ pays $D \cdot d(u, v)$ while $\mathcal{A}^e$ pays zero unless e is on the path from u to v, when it pays D times the length of e. Thus for any movement of the server, the cost incurred by $\mathcal{A}$ is the sum of all the movement costs incurred by the $\mathcal{A}^e$. Similary, any access cost incurred by $\mathcal{A}$ is the sum of all the remote access costs incurred by the $\mathcal{A}^e$. Thus for any request sequence, ϱ, $cost_{\mathcal{A}}(x, \varrho) = \sum_e cost_{\mathcal{A}^e}(x^e, \varrho^e)$. Clearly, if each $\mathcal{A}^e$ is C_D-competitive then $\mathcal{A}$ must be, as well.

We will construct an algorithm, FACTOR, such that each FACTOR^e is equivalent to EDGE on e.

For a given work function ω, and an edge $e = \{x, y\}$ we define the *slope of* ω *between* x *and* y as

$$S_\omega(x, y) = \frac{\omega(y) - \omega(x)}{d(x, y)}.$$

Note that $S_\omega(x, y) = -S_\omega(y, x)$. For simplicity, we will write $S(x, y)$ wherever ω is understood.

Algorithm FACTOR. We now define algorithm FACTOR for tree-like networks. Let ω be the current offset function. Then the probability that the server is on $x \in M$ is

$$\mathrm{Pr}_\omega[x] \; = \; 1 - \sum_{y \in N(x)} \frac{D - S_\omega(x, y)}{2D}.$$

Notice that if $\{x, y\}$ were the only edge in M, and $S(x, y) \geq 0$, then the distribution given above is the distribution given in Sect. 3 for an offset function which takes on value 0 on x and $S(x, y) \cdot d(x, y)$ on y. This ensures that FACTOR handles each single edge correctly.

Theorem 4. *If* M *is a tree then* FACTOR *is* C_D*-competitive on* M*.*

FACTOR can be extended to infinite trees, or to dynamic trees, that is trees that can grow branches during our algorithm's computation. It can also be used to design an optimally competitive randomized algorithm for any product of trees, by using one instance of FACTOR in each dimension [4]. The most common examples of such product topologies are hypercubes and meshes.

6 An Optimal Deterministic Algorithm for Continuous Trees

Let T be tree in which each edge e has some length $0 \leq l(e) \leq \infty$. Consider each edge e of T to be an interval of length $l(e)$ and consider the union of these edges, attached according to T. We call this space the *continuous tree based on* T. Any space which is obtained from some (weighted) tree in this fashion is called a continuous tree.

Suppose M is a continuous tree, and μ is any distribution on M. The *second moment* of μ is a function $I^\mu : M \to \mathbb{R}$ defined by

$$I^\mu(z) = \sum_x \frac{1}{2} d(x, z)^2 \cdot \mu(x).$$

We define $\bar{\mu}$, a *barycenter* of μ, to be any point where I^μ is minimized.

Algorithm BARY for continuous trees. If M is a continuous tree, and if ϱ is any sequence of requests, let μ be the distribution of our server on M given by the randomized algorithm FCTOR, after serving the sequence ϱ. After BARY serves ϱ, the server is deterministically at $\bar{\mu}$.

Theorem 5. *BARY is* $(2 + \frac{1}{2D})$*-competitive on any continuous tree.*

7 A Lower Bound

Black and Sleator [4] conjectured the existence of a 3-competitive deterministic page migration algorithm for all metric spaces and for all values of D. In this section we disprove this conjecture by demonstrating that for $D = 1$ no deterministic page migration algorithm for general metric spaces is c-competitive for any $c < \frac{85}{27}$.

Theorem 6. *If* $c < \frac{85}{27}$*, then there is no deterministic* c*-competitive algorithm for the page migration problem.*

8 An Algorithm for Three Points

The results of Black and Sleator [4] imply that there is a 3-competitive algorithm for each D and each metric space with two points. For $D = 1$, we extend it to three points in the theorem below. Our algorithm is based on a new technique called *forgiveness* that has been recently used in an 11-competitive algorithm for 3 servers [6].

Theorem 7. *Let* M *be a triangle, that is a metric space with only three points. Then there is a 3-competitive algorithm in* M *for* $D = 1$.

Four point spaces. One might wonder whether Black and Sleator's conjecture holds for spaces of fixed size. According to a computer program that we have written, there exists a 4-point metric space M such that no on-line algorithm for the page migration problem for $D = 1$ on M is 3-competitive. In fact, it is the same space used for the proof of Theorem 6.

References

1. B. Awerbuch, Y. Bartal and A. Fiat. Competitive distributed file allocation. *Proc. of 25th Symposium on Theory of Computation*, 1993, pages 164–173.
2. P. Berman, H. J. Karloff, and G. Tardos. A competitive three-server algorithm. In *Proc. 1st ACM-SIAM Symp. on Discrete Algorithms*, pages 280–290, 1990.
3. D. Black, A. Gupta, and W. Weber. Competitive management of distributed shared memory. In *Proceedings, Spring Compcon 1989*, pages 184–190. IEEE Computer Society, San Francisco, CA., 1989.

4. D. L. Black and D. D. Sleator. Competitive algorithms for replication and migration problems. Technical Report CMU-CS-89-201, Department of Computer Science, Carnegie-Mellon University, 1989.
5. M. Chrobak, H. Karloff, T. Payne, and S. Vishwanathan. New results on server problems. *SIAM Journal on Discrete Mathematics*, 4: 172–181, 1991. Also in *Proc. 1st ACM-SIAM Symp. on Discrete Algorithms*, pages 291–300, 1990.
6. M. Chrobak and L. L. Larmore, Generosity helps, or an 11-competitive algorithm for three servers. In *Proc. 3rd ACM-SIAM Symp. on Discrete Algorithms*, pages 196–202, 1992.
7. M. Chrobak and L. L. Larmore, The server problem and on-line games, In *Proceedings of the DIMACS Workshop on On-Line Algorithms, American Mathematical Society*, vol. 7, pages 11–64, 1992.
8. 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, pages 531–540, 1985.
9. L. W. Dowdy and D. V. Foster. Comparative models of the file assignment problem. *ACM Computing Surveys*, 14(2):287–313, 1982.
10. Amos Fiat, Richard Karp, Michael Luby, Lyle McGeoch, Daniel D. Sleator, and Neil Young. On competitive algorithms for paging problems. *Journal of Algorithms*, 12(4):685–699, 1991.
11. A. Karlin, M. Manasse, L. Rudolph, and D. Sleator. Competitive snoopy caching. *Algorithmica*, 3(1):79–119, 1988.
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*, pages 301–309, 1990.
13. M. Manasse, L. A. McGeoch, and D. Sleator. Competitive algorithms for on-line problems. *Journal of Algorithms*, 11: 208–230, 1990. Also in *Proc. 20th ACM Symposium on Theory of Computing*, pages 322–333, 1988.
14. G. Pfister et. al. The IBM research parallel processor prototype: Introduction and architecture. In *Proc. International Conf. on Parallel Processing*, IEEE Computer Society, pages 764–771, 1985.
15. P. Raghavan and M. Snir. Memory versus randomization in on-line algorithms. Research Report RC 15622 (No. 69444), IBM T. J. Watson Research Center, 1990. Also in *Proc. 16th International Colloquium on Automata, Languages, and Programming*, Lecture Notes in Computer Science vol. 372, Springer-Verlag, pages 687–703, 1989.
16. C. Scheurich and M. Dubois. Dynamic page migration in multiprocessors with distributed global memory. *IEEE Transactions on Computers*, 38(8):1154–1163, August 1989.
17. J. Westbrook. Randomized algorithms for multiprocessor page migration. In *Proceedings of the DIMACS Workshop on On-Line Algorithms, American Mathematical Society*, vol. 7, pages 135–150, 1992.
18. A. Wilson. Hierarchical cache/bus architecture for shared memory multiprocessors. In *Proc. 14th International Symp. on Computer Architecture*, pages 244–252. ACM SIGARCH/IEEE Computer Society, 1987.
19. O. Wolfson. A distributed algorithm for adaptive replication data. Technical Report CUCS-057-90, Department of Computer Science, Columbia University, 1990.

This article was processed using the LaTeX macro package with LLNCS style