Distributed Paging

Yair Bartal

U.C. Berkeley &
International Computer Science Institute *

Abstract. We survey distributed data management problems including distributed paging, file allocation, and file migration.

1 Introduction

Many modern information services know no national boundaries. The widespread use of the Internet and Internet-related applications such as the World Wide Web is growing fantastically on an annual basis.

This survey deals with *distributed data management* problems. Such problems may arise as a memory management problem for a globally addressed shared memory in a multiprocessor system as well as in a distributed network of processors where data files are kept in different sites and may be accessed for information retrieval by dispersed users and applications. In this context, a file may be a conventional single file, a system database, fragments of a database, or any combination of these.

When a processor wishes to access a file it must send a request to a processor holding the file and the desired information is transmitted back. The communication cost incurred thereby is proportional to the distance between the corresponding processors.

Files can be replicated and allocated so as to reduce communication costs. Such transactions incur however a high communication cost proportional to the file size times the communication distance. In any case, consistency of the data must be preserved. The main issue in distributed data management is the design of a dynamic allocation of file copies in a network in order to achieve low communication costs.

Distributed file assignment problems have been extensively studied in management science, engineering, and computer systems. The 1981 survey paper by Dowdy and Foster [DF81], compares studies on different models, and cites close to a hundred references. The 1990 paper by Gavish and Sheng [GS90], gives a survey of recent research on dynamic file assignment problems, where the locations of file copies may be changed over time. All of these models rely heavily on prior knowledge regarding potential usage patterns of the system databases.

* 1947 Center Street Suite 600, Berkeley, CA 94704-1198. Research supported in part by the Rothschild Postdoctoral fellowship and National Science Foundation operating grants CCR-9304722 and NCR-9416101. E-mail: yairb@icsi.berkeley.edu

This survey describes work concerning with the *competitive analysis* [ST85] of distributed data management problems. This approach makes no prior assumptions on the input sequence, and provides a worst-case guarantee on the performance of the algorithms.

The following are the main problems we survey:

- The *file migration* problem [BS89] allows only one copy of each file to be kept in the network.
- The *file allocation* problem [BFR92] deals with the more general case where files may be replicated and deleted in response to a sequence of read and write requests.
- The *distributed paging* problem [BFR92] further generalizes the model to deal with memory capacity limitations at the network processors.

The *file allocation* problem [BFR92, ABF93a] is the fundamental case where a *single* file resides among the network nodes. Copies of the file may be stored in the local storage of some subset of processors. Copies may be replicated and discarded over time as to optimize communication costs, but multiple copies must be kept consistent and at least one copy must be stored somewhere in the network at all times. An on-line file allocation algorithm must minimize communication costs, over arbitrary sequences of read and write requests issued at different locations over the network.

The file allocation problem can be viewed as the generalization of two other basic problems due to Black and Sleator [BS89]. The *file replication* problem where only read requests are issued, and the *file migration* problem where only one copy of the file may be kept in the network, which admits with write-only file allocation.

The file allocation solutions can be used only if every processor has enough memory to accommodate all files. If this is true, then every file can be handled separately. On the other hand, if this is not true, then the replication of a file into a processor's local memory may require the migration of another file copy onto some other processor.

The *distributed paging* problem is the simultaneous solution of many individual file allocation problems, subject to the constraints of the local memory capacities.

1.1 Network Model

A *network* is modeled by an undirected weighted graph $G = (V, E)$, where processors are represented by vertices, and edge weights the length or cost of a bi-directional link between the two corresponding adjacent processors. Processors communicate only by exchanging messages. The weighted graph need not obey the triangle inequality, but a natural *metric space* can be defined where the points are processors and distance between two points $p, q \in V$ is equal to the length of the shortest path between the processors in the weighted graph, and denoted $d(p, q)$. We use the terms network, weighted graph, and metric space as called for by discussion, but they refer to the same underlying interconnection network.

Let $|V| = n$ be the number of processors in the network. Let Δ denote the diameter of the network, i.e., the maximal distance between any two pair of processors, assuming the minimal distance is 1.

2 File Migration

2.1 Problem Definition

The file migration problem consists of a single copy of a file, F, held at some processor in a network of processors. A sequence of file accesses are issued at different processors over time. Each access to the file costs the distance between the requesting processor and the processor holding the file copy. In addition to serving the access requests, an algorithm is allowed at any time to migrate the file from its current location to another processor at a cost equal to the distance between them times the file size, D.

2.2 Lower Bound

Black and Sleator [BS89] used a result of Karlin et al. [KMRS88] to get a lower bound of 3 for deterministic data migration algorithms.

The following theorem is a modification that gives a lower bound on the competitive ratio of any file migration algorithm in any network topology against adaptive on-line adversaries.

Theorem 1. *Let G be any network over a set of at least two processors. The competitive ratio of any randomized on-line file migration algorithm for G against adaptive on-line adversaries is at least 3.*

Proof. Let two different processors in the network be p and q, and assume the distance between p and q is 1.

Assume the on-line algorithm holds a copy of the file at processor p. We define 3 different adaptive on-line adversaries as follows:

- The *p-adversary*: holds a single copy of the file at p.
- The *q-adversary*: holds a single copy of the file at q.
- The *jumping-adversary*: holds a single copy of the file at a processor not holding a copy of the file by the on-line algorithm (if such exists and otherwise the adversary's configuration remains unchanged) ;I.e., at processor q.

Now a write request is generated at q.

The cost for this request for both the q-adversary and the jumping-adversary is 0. The cost for the request for the p-adversary as well as the on-line algorithm is 1.

Consider next the cost incurred for configuration changes. The p-adversary and the q-adversary never change their configuration. If the on-line algorithm

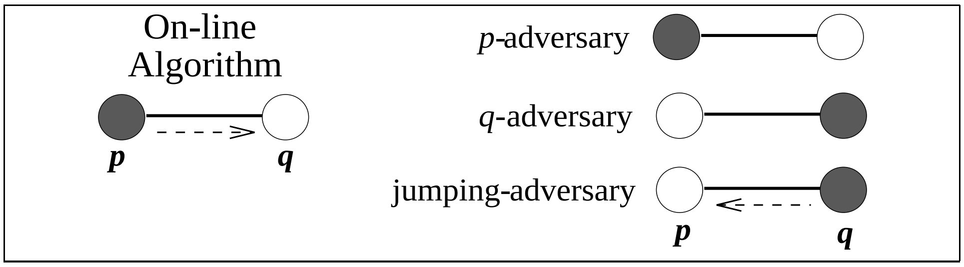

Fig. 1. Lower bound of 3 for file migration on 2 processors.

performs migration the jumping-adversary may change its configuration thereby incurring a cost of D.

We therefore conclude that over the entire sequence of events the online cost is equal to the cost of the 3 adversaries up to an additive term, implying the lower bound. ■

2.3 A Simple Randomized Algorithm

Westbrook [West91] gave a simple randomized algorithm that achieves the optimal competitive ratio of 3 in any network topology against adaptive on-line adversaries.

The Coin-Flipping Algorithm.
Given an access request issued at v move the file to v with probability $1/2D$.

Theorem 2. *The Coin-Flipping algorithm is 3-competitive against an adaptive on-line adversary.*

Proof. Define the potential function $\Phi = 3D \cdot d(a,b)$ where a and b are the adversary's and the on-line's file locations respectively. Clearly, Φ is nonnegative. When the adversary migrates the file the potential function changes by at most 3 times its cost.

Now consider an access request at v. The expected cost incurred by the algorithm is

$$\mathrm{E}(\mathrm{Cost}_{\mathrm{CF}}) = d(b,v) + \frac{1}{2D} \cdot D \cdot d(b,v)$$

The expected change in the potential is

$$\mathrm{E}(\Delta\Phi) = \frac{1}{2D} \cdot 3D \cdot (d(a,v) - d(a,b)).$$

Therefore $\mathrm{E}(\mathrm{Cost}_{\mathrm{CF}}) + \mathrm{E}(\Delta\Phi) = \frac{3}{2}(d(a,v) + d(b,v) - d(a,b)) \leq 3 \cdot d(a,v) = 3 \cdot \mathrm{Cost}_{\mathrm{Adv}}$.
■

2.4 Deterministic Algorithms

Black and Sleator [BS89] give optimal deterministic 3-competitive algorithms for the uniform network and for trees.

In contrast to the competitive ratio of 3 against adaptive on-line adversaries, Chrobak et. al. [CLRW93] show a network with a lower bound greater than 3 for deterministic algorithms, specifically $85/27 \approx 3.148$.

Awerbuch, Bartal and Fiat [ABF93a] gave the first deterministic file migration algorithm called Move-To-Min (MTM) with competitive ratio 7 for all network topologies. The best currently known deterministic file migration algorithm is due to Bartal, Charikar, and Indyk [BCI96], called Move-To-Local-Min (MTLM) which has competitive ratio ~ 4.086.

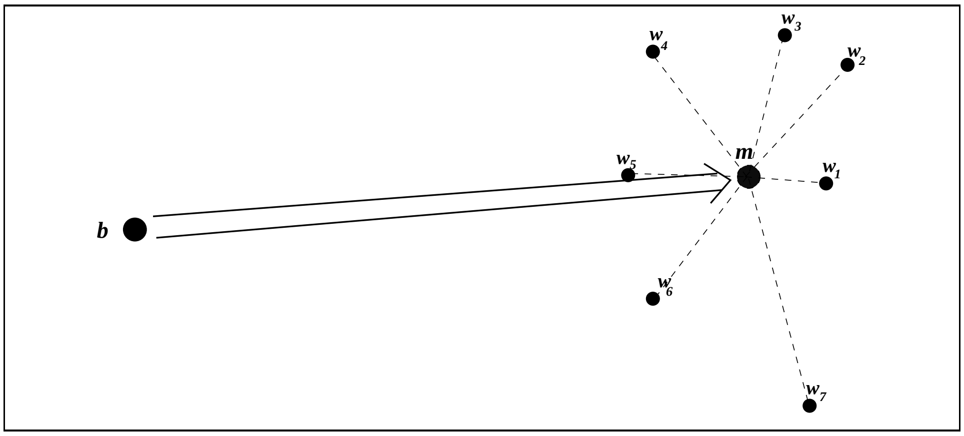

Fig. 2. Algorithm Move-To-Min (MTM). In the example D=7. m minimizes the sum of distances to the D access locations.

We present here the proof of the simpler Move-To-Min algorithm.
Algorithm **Move-To-Min** (MTM).
The algorithm divides the request sequence into phases. Each phase consists of D consecutive accesses to the file at processors $v_1, v_2, \ldots, v_D$. During a phase the algorithm doesn't move the copy of the file. At the end of a phase migrate the copy to a processor m in the network such that the sum of distances from m to the v_i's is minimized.

Theorem 3. *Algorithm Move-To-Min is 7-competitive for file migration on arbitrary network topologies.*

Proof. We analyze the performance of the algorithm in a phase. Let $a = a_0$ denote the position of the adversary copy at the beginning of a phase, and $a_i, 1 \leq i \leq D$ denote its position after the i'th request of the phase. Also let b denote the position of the on-line copy at the beginning of the phase.

During the phase MTM incurs a cost of $d(b, v_i)$ on the i'th request. At the end of the phase its cost for migrating the file is $D \cdot d(b, m)$. Thus its total cost for the phase is

$$\begin{aligned}\mathrm{Cost}_{\mathsf{MTM}} &= \sum_i d(b, v_i) + D \cdot d(b, m) \\ &\le \sum_i d(a, v_i) + D \cdot d(b, a) + D \cdot d(b, m).\end{aligned}$$

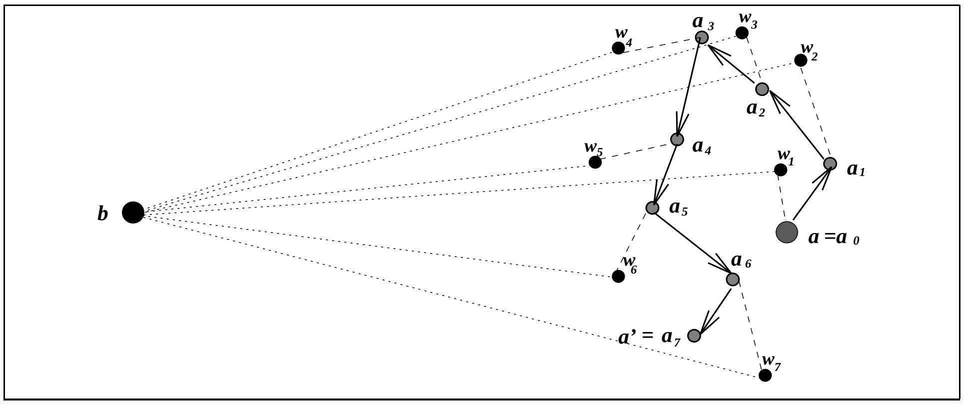

Fig. 3. The costs incurred by the on-line algorithm and the adversary over a phase.

The adversary incurs a cost of $d(a_{i-1}, v_i)$ on the i'th request, plus a cost of $D \cdot d(a_{i-1}, a_i)$ for its migration after the request. Thus the adversary cost during the entire phase is

$$\mathrm{Cost}_{\mathrm{Adv}} = D \cdot \sum_i d(a_{i-1}, a_i) + \sum_i d(a_{i-1}, v_i).$$

Using the triangle inequality we get that for any $0 \le \ell \le D$:

$$\begin{aligned}\mathrm{Cost}_{\mathrm{Adv}} &= D \cdot \sum_i d(a_{i-1}, a_i) + \sum_i d(a_{i-1}, v_i) \\ &\ge \sum_i \{d(a_\ell, a_{i-1}) + d(a_{i-1}, v_i)\} \\ &\ge \sum_i d(a_\ell, v_i).\end{aligned}$$

Thus we obtain

$$D \cdot d(a_\ell, m) \le \sum_i d(a_\ell, v_i) + \sum_i d(m, v_i) \le 2 \sum_i d(a_\ell, v_i) \le 2\mathrm{Cost}_{\mathrm{Adv}}.$$

We use the potential function $\Phi = 2D \cdot d(a, b)$. We show that over a phase it increases by at most 7 times the adversary cost and decreases by at least the on-line cost. Let $a' = a_D$.

$$\begin{aligned}
\Delta\Phi &= 2D \cdot (d(a', m) - d(a, b)) \\
&\leq D(2d(a', m) + d(a, m) - d(b, m) - d(a, b)) \\
&= 2D \cdot d(a', m) + D \cdot d(a, m) \\
&+ \sum_i d(a, v_i) - \sum_i d(a, v_i) - D \cdot d(b, m) - D \cdot d(a, b) \\
&\leq 7\mathrm{Cost}_{\mathrm{Adv}} - \mathrm{Cost}_{\mathsf{MTM}}.
\end{aligned}$$

■

The Move-To-Local-Min (MTLM) algorithm of [BCI96] also operates in phases. The length of a phase is $L = cD$. At the end of the phase, the page is migrated to the node x which minimizes the function $f(x) = \sum_{i=1}^{L} d(x, v_i) + \delta D \cdot d(b, x)$, where $c \approx 1.841$ and $\delta \approx 0.648$.

The first term in the minimizer function $f(x)$ ensures that the page is moved to a node in the network which is close to where the activity is taking place (reflected by the requests in the last phase). The second term reflects the cost of moving the page to the new node, weighted by parameter δ. This ensures that the cost of making this move is not too high. This additional term is similar to, and in fact inspired by the additional term in the minimizer function used by the Work Function Algorithm for the k-server problem and metrical task systems. The competitive ratio of Move-To-Local-Min is approximately 4.086.

2.5 Randomization Against an Oblivious Adversary

Chrobak et al. [CLRW93] studied randomized page migration algorithms against an oblivious adversary. They show that the optimal randomized competitive ratio for a network consisting of two nodes connected by an edge is $2 + \frac{1}{2D}$. They then use the algorithm for an edge to obtain a $(2 + \frac{1}{2D})$-competitive algorithm for trees. Lund et al. [LRWY94] as similar result for uniform networks. All of these algorithms are work-function based, i.e., on the optimal costs of serving the requests and ending at different configurations.

Westbrook [West91] gives randomized algorithms for arbitrary network topologies. The algorithm operates in phases. The length of a phase is drawn out of a fixed probability distribution. The file is not moved during a phase and at the end of a phase it is migrated to the location of the last request. The competitive ratio given by the best choice of a probability distribution approaches $1 + \phi \approx 2.618$ as D goes to infinity.

2.6 k-Page Migration

The *k-page migration* problem is a generalization of the standard file migration problem where there are k copies of a file residing in the network instead of just one. An access request initiated in one of the processors costs the distance to the nearest page copy. Each of the copies may also be migrated at the a cost of D (the size of the page) times the distance traveled. As before the problem is to minimize the total access and migration costs.

The k-page migration problem can be viewed as a special case of a more general framework of "*relaxed task systems*" which are problems that can be formulated as a relaxed version of some other problem where configuration changes are forced. In the case of k-page migration this would be the k-server problem.

Awerbuch, Azar, and Bartal [AAB96] give a randomized transformation from k-server algorithms into k-page migration algorithms with competitive ratio $O(k)$ against an adaptive on-line adversary. Bartal, Charikar and Indyk [BCI96] give a deterministic transformation yielding competitive ratio $O(k^2)$.

[BCI96] also give a $(2k+1)$ lower bound on the competitive ratio for the problem in any network topology. A matching upper bound is given for the uniform network and a $(2k+1)(1+\frac{1}{D})$ competitive ratio for trees.

3 File Allocation

3.1 Problem Definition

The *file allocation* problem [BFR92] assumes that data is organized in indivisible blocks such as files (or pages), consisting of D data units.

Let P denote the set of processors. Initially, a subset $Q \subseteq P$ of processors is each assigned a copy of the file. The algorithm receives a sequence of requests initiated by processors in P. Each request is either a *read* request or a *write* request. A read request at processor r is served by the closest processor p holding a copy of the file. The cost associated with this transmission is the distance between p and r. In response to a *write* request initiated at processor w, the algorithm must transmit an update to all currently held copies of the file – the subset $Q \subseteq P$. It pays a cost equal to the minimum Steiner tree spanning $Q \cup \{w\}$.

In between requests, the algorithm may re-arrange the copies of the database. A processor may delete the copy it is holding, unless it is the last copy in the network, at no cost. The file may also be *replicated* from a processor p, which holds a copy, to a subset $Q' \subset P$. The cost of replicating is equal to D times the minimum Steiner tree spanning $Q' \cup \{p\}$. D represents the ratio between the size of the entire file and the size of the minimal data unit being read or updated. A new current subset Q of processors holding copies of the file is determined as a result of delete and replicate steps. A combination of a replicate step from a processor p to a processor q, followed by a delete at p, is sometimes called a *migration* step. The subset Q is called the configuration of the algorithm.

3.2 Lower Bounds

A lower bound of 3 in any network topology follows from the lower bounds for file migration described in the previous section by restricting the request sequence to write requests.

Bartal, Fiat and Rabani [BFR92] give a lower bound of $\Omega(\log n)$ for a specific family of networks based on a lower bound for the on-line Steiner tree problem [IW91]. This lower bound holds even against an oblivious adversary.

3.3 Specific Network Topologies

[BFR92] give an optimal 3-competitive, deterministic file allocation algorithm for uniform networks. This also gives an alternative algorithm for file migration when restricting to write requests only.

[BFR92] also give a memoryless 3-competitive algorithm for continuous tree. [LRWY94] give a 3-competitive algorithm for trees based on work-functions.

In what follows we describe the optimal algorithm for uniform networks. Let P denote the set of processors in the network.

Algorithm **Count.**

Count is defined for each processor $p \in P$ separately. It maintains a counter c, and performs the following algorithm. We say that Count is *waiting*, if there is a single copy of the file and the processor holding the file is performing step 4 of the algorithm. Initially, set $c := 0$. If p holds a copy of the file, begin at step 4.

1. While $c < D$, if a *read* is initiated by p, or if a *write* is initiated by p, and Count is waiting, increase c by 1.
2. *Replicate* a copy of the file to p.
3. While $c > 0$, if a *write* is initiated by any other processor, decrease c by 1.
4. If p holds the last copy of the file, wait until it is replicated by some other processor.
5. *Delete* the copy held by p.
6. Repeat from step 1.

Theorem 4. *Algorithm* Count *is 3-competitive for uniform networks.*

Proof. Fix a processor p. One iteration of steps 1–6 at p is named a *phase*. Note that if Count is waiting then it is executing step 4 in the single processor holding a copy of the file, and it is executing step 1 in all the other processors. Count's cost is charged on individual processors as follows:

1. A processor initiating a read is charged the cost of the read.
2. If Count is waiting, a processor initiating a write is charged the cost of the write.
3. If Count is not waiting, and a write is initiated, the cost of 1 is charged at each processor holding a copy, except for the initiating processor. Note that the sum of costs charged here is exactly the cost of that write.
4. The cost D of replicating is charged at the processor receiving the copy.

The adversary's cost is charged the same, except that a replication is not charged. Rather, it registers a debit of D at the processor receiving the copy. That debit is paid (and a cost of D is charged) when the copy is deleted. Debits are initially set to 0 for processors *not* holding copies and to D for processors that initially do hold a copy. Note that the charging of the adversary's cost minus the sum of initial debits is a lower bound on its actual cost.

At the beginning Count is waiting after all but one copy are deleted, so that no cost is incurred. Now, during a phase of a processor p, Count's cost charged to p is at most $3D$. Steps 1 and 2 cause a charge of D each. Step 3 causes a charge of D. The total cost of Count is the sum of costs over all phases of all processors. There can be at most n partial phases (which are not over).

The adversary's cost during a full phase (note that the duration of a phase is determined by Count) is at least D. If the adversary ever deletes a copy from the processor during a phase, it is charged D. Otherwise, it either holds a copy at that processor when Count begins step 3 (and therefore not waiting), so it pays D during that step; or, it does not hold a copy at the end of step 1, and since it could not delete during that step, it must have been charged at least D for the requests of step 1. The reason is that during step 1 the processor initiated a total of D requests, counting read requests and write requests initiated while Count was waiting. ■

3.4 Arbitrary Networks

Randomized Algorithm [BFR92] give a randomized algorithm for the file allocation problem on all networks, which is competitive against an adaptive on-line adversary.

This is done by a reduction to the on-line Steiner tree problem. Let G be an arbitrary network. Let A be a strictly (no additive term) c-competitive *Steiner tree* algorithm on G. We use A can to obtain a competitive randomized file allocation algorithm on G. Assume that the initial configuration consists of only one copy of the file.

Algorithm **Steiner Based** (SB).
Algorithm SB simulates a version of the Steiner tree algorithm A starting with p as the initial configuration. At all times, the set of processors in which SB keeps copies of the file is equal to the set of processors covered by A's Steiner tree.

Upon receiving a *read* request initiated at node r, the algorithm serves it, and then with probability $1/D$ feeds A with a new request at vertex r . In response A computes a new Steiner tree T' in place of its previous tree T. SB *replicates* new copies of the file at the processors corresponding to the vertices that A added to its tree.

Upon receiving a *write* request initiated at node w, the algorithm serves it, and with probability $1/\alpha D$ *deletes* all copies of the file, leaving only one copy at the processor closest to w, and then migrates the file to w, initializing a new version of A starting at vertex w as its initial configuration.

SB achieves best performance for $\alpha = \sqrt{3}$.

Theorem 5. *If A is a strictly c-competitive Steiner tree algorithm against adaptive on-line adversaries on a network G, then* SB *is a* $(2+\sqrt{3})c$*-competitive algorithm for the file allocation problem on G against adaptive on-line adversaries.*

In particular by using the greedy Steiner algorithm [IW91] we get an $O(\log n)$ competitive ratio.

Deterministic Algorithm [ABF93a] give a deterministic $O(\log n)$-competitive file allocation algorithm for arbitrary networks.

The algorithm partitions the request sequence into phases. Each phase, except perhaps for the last one, contains exactly D successive *write* requests.

Within each phase, the algorithm deals only with *read* requests.

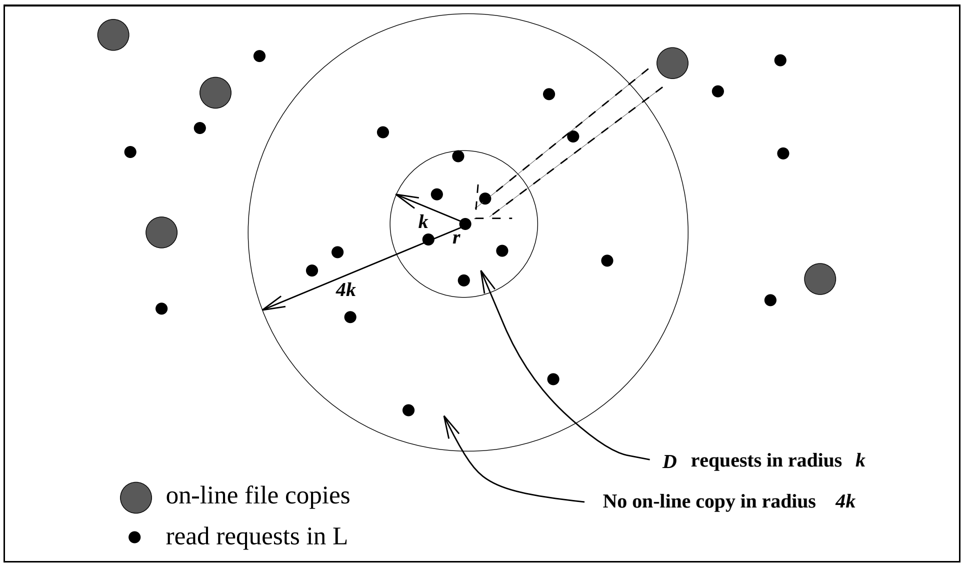

Fig. 4. The file allocation algorithm (FA): dealing with read requests. r is the last read request. ($\lambda = 4$.)

Dealing with read requests. At all times the algorithm maintains a partial list, L, of previous read requests in the current phase.

Upon receiving a new *read* request initiated by processor r, add r to L.

If L consists of more than D *read* requests, then consider the smallest k-neighborhood of r containing D requests.

If the algorithm does not hold a copy of the file at distance less than λk from r, *replicate* a copy of the file to r from the closest processor to r holding a copy, and remove the D requests from L.

Dealing with write requests. When a phase ends, let $w_1, w_2, \ldots w_D$ be the locations of the D *writes* initiated during the phase. Let m be a processor such that the sum of distances from m to the w_i's (i.e., $\sum_i d(m, w_i)$) is minimized. Now, *replicate* a copy of the file to m from the closest processor to it holding a copy, and then *delete* all copies of the file except the one at m. The read requests list, L, is initialized to be empty.

Theorem 6. *Algorithm* FA *is* $O(\min\{\log n, \log(\Delta)\})$*-competitive for file allocation on arbitrary network topologies.*

3.5 Randomization Against an Oblivious Advesary

As mentioned above randomization against an oblivious adversaries cannot help in the general case. It is however interesting to study for specific network topologies.

[LRWY94] give a randomized algorithm for trees with competitive ratio $2 + \frac{1}{D}$. The only lower bound is $2 + \frac{1}{2D}$ which follows from the the lower bound for migration. The problem of obtaining a randomized algorithm for uniform networks with competitive ratio better than 3 is open.

4 Distributed Paging

The *distributed paging* problem [BFR92]. paging is the solution of multiple file allocation problems, constrained by the local memory of the processors. (The problem is also sometimes referred to as *constrained file allocation*).

4.1 Preliminaries

Let $\mathcal{F}$ denote the set of files and $\mathcal{P}$ denote the set of processors . A *file copy* is an ordered pair $< F, p >$ where $F \in \mathcal{F}$ denotes the file and $p \in \mathcal{P}$ denotes the processor.

Reads and writes are issued at some processor for a file. Requests are of the form *read*(p, F) and *write*(p, F), for $F \in \mathcal{F}$ and $p \in \mathcal{P}$.

We assume all files are of the same size. Let $m = \sum_p k_p$, the total number of files that can be stored in the network, and $k = \max_p k_p$, the maximal number of files that can be stored in any one processor.

4.2 Deterministic Algorithms

[BFR92] give a lower bound of $2m - 1$ on the competitive ratio of any deterministic algorithm for the distributed paging problem.

They also describe the Distributed-Flush-When-Full algorithm (DFWF) which is $3m$ for uniform networks.

The algorithm uses the following terminology. We say a processor p is *free* if it holds less than k_p different files. A copy of a file is called *single* if there are no other copies of that file currently in the network.

The algorithm works in phases. Copies of files can be either marked or unmarked. At the beginning of a phase, all counters are zero and all copies are unmarked. Throughout, an unmarked copy is single, a marked copy may be not single.

Algorithm **Distributed-Flush-When-Full** (DFWF).
The algorithm is defined for each processor p separately. Every processor maintains a counter c_F for every file F. Initially, or as a result of a *restart* operation, all counters are set to zero and all markings are erased. Arbitrarily, copies of files are deleted until there is exactly one copy of every file somewhere in the network.

Every processor p follows the following procedure simultaneously for all files F:

1. While $c_F < D$, if a *read*(p, F) request is initiated at p, or if a *write*(p, F) request is initiated at p and F is unmarked, increase c_F by 1, if p does not contain a copy of F.
2. (a) If p is free, *replicate* F to p and mark it. If F was unmarked, *delete* the unmarked copy.
   (b) Otherwise, if all file copies in p are marked then restart.
   (c) Otherwise, choose S to be an arbitrary unmarked copy in p.
      i. If F is unmarked, switch between S and F, and mark F in p.
      ii. Otherwise, if some free processor q is available, *dump* S to q, and *replicate* a copy of F to p, mark this copy.
      iii. Otherwise, restart.
3. While $c_F > 0$, if a *write*(p, F) request is initiated by any other processor, decrease c_F by 1.
4. Restart.

Theorem 7. *Algorithm DFWF is* $3m$*-competitive for distributed paging on uniform networks.*

4.3 Randomization Against an Oblivious Adversary

Awerbuch, Bartal and Fiat [ABF93b] study randomized algorithms against an oblivious adversary for the distributed paging problem.

They prove the following lower bound:

Theorem 8. *Let the* $f > k$ *be the number of different files in the network. Let* $\tilde{S}(r)$ *be the maximal* r*-server lower bound achievable for any metric space over* $r+1$ *points. The competitive ratio of any randomized distributed paging algorithm for* $\mathcal{N}$ *against oblivious adversaries is* $\Omega(\max\{\tilde{S}(\min\{m+1-f, n-1\}), \log k\})$.

In particular it follows from a result of [FKLM+88] that $S(\mathcal{U}, m+1-f) = \Omega(\log(m-f))$. Hence,

Corollary 9. *The competitive ratio of any randomized distributed paging algorithm is* $\Omega(\max\{\log(m-f), \log k\})$ *for uniform networks.*

[ABF93b] also give an $O(\max\{\log(m-f), \log k\})$ competitive algorithm for uniform networks called HEAT & DUMP.

The algorithm is quite complicated and therefore we concentrate on the read-only distributed paging case and assume the file size $D = 1$. Thus the problem considered in the rest of this section is the basic problem where given a request for a file in some processor the file must be replicated into that processor. The algorithm must decide which other file to drop off that processor's local memory and where to migrate ("dump") a copy of that file if the processor holds the last copy of the file.

The HEAT & DUMP **Algorithm** Algorithm HEAT & DUMP is modularly composed of two conflicting strategies.

The first is some *uniprocessor paging strategy* run within each processor's local memory. The paging strategy must deal with the memory constraints of the specific processor and decide which files should be erased to make place for other more popular files.

The paging strategy though may sometimes decide upon erasing the last single copy of some file. We then invoke the *global memory management strategy* which deals with the management of the entire aggregate memory in the network. The global memory management strategy is responsible of making room for keeping a copy of the file that must be removed according to the uniprocessor paging strategy. For that purpose it decides upon erasing a copy of some file, currently having multiple copies.

It follows that the global memory management strategy, may interfere with the local paging algorithms by forcing the deletion of some file required by the paging strategy. Therefore the actual files configuration in every processor somewhat differs the configuration it would have maintained if it was running the uniprocessor paging strategy without interference.

In what follows, we describe the HEAT & DUMP algorithm.

Following the completion of the t'th event in the input sequence, we say that the time is t. At all such times, associated with every processor p is a local paging algorithm $\mathcal{L}_p$.

The *virtual configuration* associated with p is the configuration of the local paging algorithm associated with the processor. The list of files associated with the local processor, had it been a uniprocessor under control of the local paging algorithm, is denoted by $\mathcal{V}(p,t)$. We call $\mathcal{V}(p,t)$ the "virtual configuration" with the understanding that all other configuration data required by the local paging algorithm is also stored.

In reality, the "real" list of files stored at the processor, $\mathcal{R}(p,t)$ contains some other files, not in $\mathcal{V}(p,t)$. These files were "dumped" into the processor p by the global strategy. The *real configuration* may also contain some empty file *slots*, in place of missing "virtual" files.

A *file copy* is an ordered pair $< F, p >$ where $F \in \mathcal{F}$ denotes the file and $p \in \mathcal{P}$ denotes the processor. We associate labels, *hot*, *cold*, and *dumped*, with file copies, a file copy must have exactly one of these labels. The sets of hot,

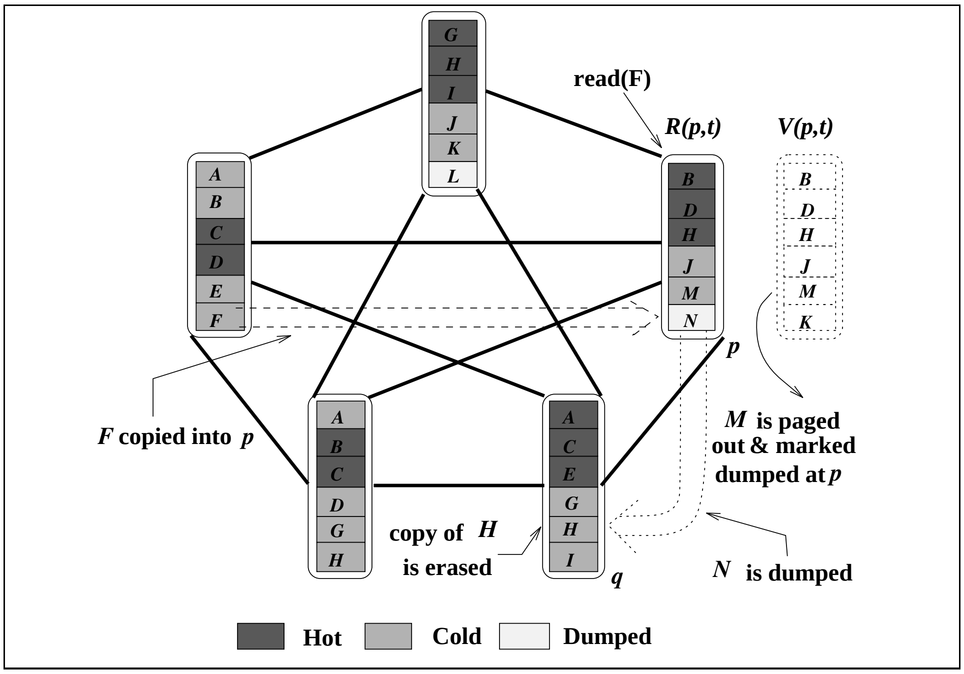

Fig. 5. The HEAT & DUMP algorithm. File copies are marked hot, cold and dumped. F is requested at processor p at time t. M is paged out from the virtual configuration $\mathcal{V}(p,t)$, and is marked dumped at p. p now contains two file copies marked dumped, so an arbitrary dumped file copy (N) is dumped out of p to q overwriting H, a non-single file copy marked cold chosen at random amongst all such candidates in the network.

cold, and dumped files in processor p at time t are denoted $\mathcal{H}(p,t)$, $\mathcal{C}(p,t)$, and $\mathcal{D}(p,t)$ respectively. Note that $\mathcal{R}(p,t) = \mathcal{H}(p,t) \cup \mathcal{C}(p,t) \cup \mathcal{D}(p,t)$.

A file is called *single* if only one copy of the file resides in the network. If more than one copy of the file resides in the network, then the file is called *non-single*.

We may also say that a file is hot, if there exists a hot copy of the file in the network. We say that a file is cold or dumped if all copies of the file are cold or dumped, respectively.

Initially, all files are single and are labeled dumped.

When a request for file F appears at some processor p, the simulated local paging algorithm $\mathcal{L}_p$ is fed with a request for F. This results in erasing some file from the virtual configuration $\mathcal{V}(p,t)$. Files that are ejected from the virtual configuration in a processor are either erased from the processor (if non-single) or become labeled dumped. This assures that p either contains some empty slots or that it contains some files marked dumped.

When a file is ejected from the processor to make room for another file, we say that the file has been *dumped*. The general strategy is to avoid dumps, if possible, and if impossible, to dump out of the processor a dumped file.

The dump destination is chosen to be a processor that has additional storage capacity or a processor that contains a non-single cold file copy, if possible, this choice is made uniformly at random from all non-single cold file copies in the network. If there is no legitimate dump target then a phase is completed.

Just prior to the end of the current phase phase, file copies that have been cold throughout the current phase and that have been used elsewhere during the current phase are labeled dumped. This is to avoid the situation where such a file copy resides in a processor for several phases while labeled cold throughout, unless it is not requested since marked cold.

When a phase ends, and before the next phase actually begins and new activity starts, hot file copies become labeled cold.

After a file slot is evicted to make room for F, F is replicated onto p. If prior to the replication F was labeled dumped then the former dumped copy is erased in the processor from which it was copied.

Let $c(p)$ be the competitive ratio obtained by $\mathcal{L}_p$. Let $c = \max_p c(p)$.

Theorem 10. *Algorithm* Heat & Dump *is* $O(\max\{\log(m - f), c\})$*-competitive against oblivious adversaries, where c is the maximal competitive ratio for any of the local paging algorithms in the network.*

In particular by using local paging algorithms of [MS91, FKLM$^+$88] in all processors we get:

Corollary 11. *Algorithm* Heat & Dump *is* $O(\max\{\log(m-f), \log k\})$*-competitive against oblivious adversaries.*

4.4 Algorithms for Arbitrary Networks

Awerbuch, Bartal and Fiat [ABF96] deal with the distributed paging problem on arbitrary networks. The solution they present holds against some weak adversary as will be explained in the sequel. However they develop a general framework for reducing the problem to the read-only case. This is done by letting the read-only distributed paging algorithm define a set of active and in-active processors for each of the files. Then a modification of the file allocation strategies that can deal with a dynamically changing network are are with these sets of active processors to deal with the write requests.

The goal of giving an algorithm for distributed paging seems to be very difficult. Moreover the best competitive ratio possible for deterministic algorithms is $\Omega(m)$.

This leads [ABF96] to defining a weaker but reasonable adversary, against which they can obtain an algorithm for read-only distributed paging with poly-logarithmic competitive ratio.

The model they use was suggested in [ST85, KMRS88, MMS88, Young91] for uniprocessor paging, where the adversary has a smaller cache size than the on-line algorithm. This can also be viewed as a solution for the problem where the cache size may vary (cf. [Young91]), but the algorithm should pay for enlarging its cache size.

More precisely, we measure the performance of an on-line distributed paging algorithm by two measures: one is the ratio between its cost for serving the request sequence (including it's communication cost in the distributed setting.) and that of the adversary, and the second is the ratio between its cache size and that of the adversary's in every processor.

Definition 12. Given a distributed paging algorithm, DP, and any adversary, Adv, let k_p denote the on-line memory capacity of processor p, and let h_p denote the adversary's memory capacity for processor p.

Let c and s be constants at least 1. If for any adversary, Adv such that for all p, $k_p \geq s \cdot h_p$, the algorithm is c-competitive against Adv, then we say the algorithm is (c, s)-competitive.

In this terminology, [ST85, KMRS88] give deterministic (2,2)-competitive uniprocessor paging algorithms.

Thus, in the uniprocessor case the assumption of some small advantage in memory for the on-line algorithm reduces the competitive ratio dramatically. The situation is similar for the distributed paging problem.

[ABF96] give a read-only distributed paging algorithm called Hierarchical Paging.

The major idea is to use uni-processor paging on "areas of activity". These areas are defined using the sparse partitions of [AP90]. After sufficiently many accesses to a file have been issued by processors within such an "area of activity", a copy of the file is fetched into this area. For every such "area of activity" we simulate a uni-processor paging process that determines what files are to be ejected/erased to make space for new files.

Theorem 13. *The* Hierarchical Paging *algorithm is* (polylog(m,Δ),polylog(m,Δ))-*competitive.*

4.5 Constrained File Migration

It is only natural to consider the migration version of the distributed paging problem where only one copy of each file is allowed to be kept in the network. (This problem is also sometimes referred to as *constrained file migration*.)

This has been first consider by Albers and Koga [AK94] who give $\Theta(k)$ and $\Theta(\log k)$ deterministic and randomized, upper and lower bounds for the problem on uniform networks, respectively.

Bartal [Bart96] gives a deterministic lower bound of $\Omega(m)$, a randomized lower bound of $\Omega(\log m)$ for the problem in some network topology, and an $O(\log m \log^2 n)$ randomized upper bound for arbitrary network topologies.

The upper bound is based on a general method of probabilistically approximating metric spaces by metric spaces from a set of "nice" metric spaces called HST's.

Definition 14. An *r-hierarchically well-separated tree* (r-HST) is defined as a rooted weighted tree with the following properties:

– The edge weight from any node to each of its children is the same.
– The edge weights along any path from the root to a leaf are decreasing by a factor of at least r.

The result of [Bart96] can be summarized as follows:

Theorem 15. *Say there is a c-competitive algorithm on r-HST's. There exists a randomized algorithm with competitive ratio $O(rc \log n \log_r n)$ for arbitrary graphs against an oblivious adversary.*

The result for constrained file migration on arbitrary networks follows from the following theorem

Theorem 16. *Given a network represented by a r-HST, T, for some $r \geq 2$ then there exists a randomized $O(\log m)$ competitive algorithm for the constrained file migration problem on G.*

Proof Sketch. We can concentrate on the case $D = 1$ (as follows from the general reductions in [AAB96, BCI96]). Apply a copy of the marking paging algorithm [FKLM+88] for each subtree as follows. Given a request for a file invoke the algorithm at the topmost level containing the request, where the file is missing, and then repeat until the file arrives at the requesting processor. Now consider a marking algorithm for a particular subtree. Its cache size is at most m. The virtual configuration of the algorithm contains some files that may actually be outside its subtree. Since a file in the virtual configuration has been requested inside the subtree in either the current or the previous marking phases, this may happen only if the file has been later requested outside the subtree. Thus these two requests have an inherent cost proportional to the cost of migrating the file into the subtree. The claim follows since the expected cost on requests for files not in the virtual configuration of the algorithm is bounded by $O(\log m)$ times the optimal cost. ■

Corollary 17. *Given a network G, there exists a randomized $O(\log m \log^2 n)$ competitive algorithm for the constrained file migration problem on G.*

5 Discussion

The fundamental distributed paging problem is still open for arbitrary network topologies. This seems to be an excellent problem for further study.

Obvious generalizations of the problems discussed herein are to deal with network congestion and delay issues rather than simple communication costs. Distributed algorithms for data management problems are discussed in [BFR92, BR92, ABF93a, ABF96, Bart96]. These algorithms make decisions at the network processors based on the local information gathered via communication. However these results assume serialization of the requests, thus issues of concurrency must be addressed. The current file allocation models also assume that all file copies must be updated at every moment. This is a common practical demand. However perhaps there are more efficient solutions if file copies are allowed to be maintained in some clever partially inconsistent manner.

Modern distributed systems allow not only migrating data along the network, but also the migration of processes. The *process allocation* problem can be defined as the dynamic re-allocation of processes as to minimize the sum of communication costs between them (see [FMRW95]).

Perhaps the most promising avenue of research is to attempt to implement some or all of the ideas presented here in real systems. Here, there are additional practical considerations and constraints that have to be addressed. It seems that in practice the issues of distributed control over centralized control are less important and that it is probably our centralized algorithms that are of greater interest. A key practical consideration is how to co-exist with existing networks and protocols.

In many cases, competitive analysis and its variants yield algorithms that seem to be natural for the problem and may prove to have a real advantage in practice over currently used heuristics, and in some cases may even provide the only complete solution for the problem.

For example, the concepts presented herein seem to be highly suitable for implementation in large network caches.

References

[AAB96] B. Awerbuch, Y. Azar, and Y. Bartal. On-line Generalized Steiner Problem. In *Proc. of the 7th Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 68–74, January 1996.

[ABF93a] B. Awerbuch, Y. Bartal, and A. Fiat. Competitive Distributed File Allocation. In *Proc. 25th ACM Symp. on Theory of Computing*, pages 164–173, May 1993.

[ABF93b] B. Awerbuch, Y. Bartal, and A. Fiat. Heat & Dump: Randomized competitive distributed paging. In *Proc. 34rd IEEE Symp. on Foundations of Computer Science*. IEEE, pages 22–31,November 1993.

[ABF96] B. Awerbuch, Y. Bartal, and A. Fiat. Distributed Paging for General Networks. In *Proc. of the 7th Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 574–583, January 1996.

[AK94] S. Albers and H. Koga. New On-line Algorithms for the Page Replication Problem. In *Proc. of the 4th Scandinavian Workshop on Algorithmic Theory*, Aarhus, Denmark, July 1994.

[AK95] S. Albers and H. Koga. Page Migration with Limited Local Memory Capacity. In *Proc. of the 4th Workshop on Algorithms and Data Structures*, , August 1995.

[AKP92] B. Awerbuch, S. Kutten, and D. Peleg. Competitive Distributed Job Scheduling. In *Proc. of the 24th Ann. ACM Symp. on Theory of Computing*, pages 571–580, May 1992.

[AP90] B. Awerbuch and D. Peleg. Sparse Partitions. In *Proc. of the 31st Ann. Symp. on Foundations of Computer science*, pages 503-513, October 1990.

[Bart96] Y. Bartal. Probabilistic Approximation of Metric Spaces and its Algorithmic Applications. To appear in *Proc. of the 37rd Ann. IEEE Symp. on Foundations of Computer Science*, 1996.

[BCI96] Y. Bartal, M. Charikar and P. Indyk. On Page Migration and Other Relaxed Task Systems. To appear in *Proc. of the 8th Ann. ACM-SIAM Symp. on Discrete Algorithms*, January 1997.

[BFR92] Y. Bartal, A. Fiat, and Y. Rabani. Competitive algorithms for distributed data management. In *Proc. 24th ACM Symp. on Theory of Computing*, pages 39–50, 1992. Also accepted for publication in the STOC 92 special issue *Journal of Computer and Systems Sciences*, 1993.

[BLS87] A. Borodin, N. Linial, and M. Saks. An Optimal On-Line Algorithm for Metrical Task Systems. In *Proc. of the 19th Ann. ACM Symp on Theory of Computing*, pages 373–382, May 1987.

[BR92] Y. Bartal and A. Rosén. The Distributed k-Server Problem — A Competitive Distributed Translator for k-Server Algorithms. In *Proc. of the 33rd Ann. IEEE Symp. on Foundations of Computer Science*, pages 344-353, October 1992.

[BS89] 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.

[CLRW93] M. Chrobak, L. Larmore, N. Reingold, and J. Westbrook. Optimal Multiprocessor Migration Algorithms Using Work Functions. In *Proc. of the 4th International Symp. on Algorithms and Computation.* Also *Lecture Notes in Computer Science*, vol. 762, pages 406-415, Hong Kong, 1993, Springer-Verlag.

[DF81] D. Dowdy and D. Foster. Comparative Models of The File Assignment Problem. Computing Surveys, 14(2), June 1982.

[FKLM+88] A. Fiat, R.M. Karp, M. Luby, L.A. McGeoch, D.d. Sleator , and N.E. Young. Competitive Paging Algorithms. Technical Report, Carnegie Mellon University, 1988.

[FMRW95] A. Fiat, Y. Mansour, A. Rosén, and O. Waarts. Competitive Access Time via Dynamic Storage Rearrangement. In *Proc. of the 36th Ann. IEEE Symp. on Foundations of Computer Science*, pages 392–403, October 1995.

[GS90] B. Gavish and O.R.L. Sheng. Dynamic File Migration in Distributed Computer Systems. In *Communications of the ACM*, 33(2):177-189, 1990.

[IW91] M. Imaze and B.M. Waxman. Dynamic Steiner Tree Problem. In *SIAM Journal on Discrete Mathematics*, 4(3):369-384, August 1991.

[KMRS88] A.R. Karlin, M.S. Manasse, L. Rudolph, and D.D. Sleator. Competitive Snoopy Caching. In *Algorithmica*, 3(1):79-119, 1988.

[LRWY94] C. Lund, N. Reingold, J. Westbrook, and D. Yan. On-Line Distributed Data Management. In *Proc. of European Symp. on Algorithms*, 1994.

[MMS88] M.S. Manasse, L.A. McGeoch, and D.D. Sleator. Competitive Algorithms for On-Line Problems. In *Proc. of the 20th Ann. ACM Symp. on Theory of Computing*, pages 322–333, May 1988.

[MS91] L.A. McGeoch and D.D. Sleator. A Strongly Competitive Randomized Paging Algorithm. In *Algorithmica*, 1991.

[ST85] D. Sleator and R.E. Tarjan. Amortized Efficiency of List Update and Paging Rules. *Communications of ACM* , 28(2):202–208, 1985.

[West91] J. Westbrook. Randomized Algorithms for Multiprocess or Page Migration. In *Proc. of DIMACS Workshop on On-Line Algorithms.* American

Mathematical Society, February, 1991.
[Young91] N.E. Young. Competitive Paging as Cache Size Varies. In *Proc. 2nd ACM-SIAM Symp. on Discrete Algorithms*, January 1991.