Heat & Dump: Competitive Distributed Paging

Baruch Awerbuch* Yair Bartal† Amos Fiat†

Abstract

This paper gives a randomized competitive *distributed paging* algorithm called Heat & Dump. The competitive ratio is logarithmic in the total storage capacity of the network, this is optimal to within a constant factor. This is in contrast to the linear optimal deterministic competitive ratio [BFR].

1 Introduction

The basic paradigm: Distributed Virtual Memory. Virtual addressing has the advantage that the physical address is separate from the logical address [TDMF]. Briefly, the name of a memory item is decoupled from its physical location in memory; moreover, physical location may dynamically change in the run-time. With the appearance of the massively parallel machines in the 1980s, it was natural to extend the virtual memory concept from the traditional uniprocessor to distributed shared-memory environment. In other words, the programmer can use the convenient Parallel Random Access Machine (PRAM) abstraction to write the program, which will be then compiled automatically to run on message-passing network-based distributed memory machines.

We consider the problem of *distributed paging* in a distributed-memory multi-processor [LLG+] (or a distributed file server [YR]) configured on a message-passing network. Our goal is to design data management algorithms that exploit the locality of reference of the underlying program by replicating data at the locations where they are frequently requested, thus minimizing the number of network accesses.

More precisely, assume a network of processors, each having limited memory capacity. Each data is stored in at least one, and possibly multiple processors. The total amount of different data items stored by the processors cannot, of course, exceed the total nodal capacity. The algorithm is allowed to dynamically reconfigure contents of the nodal memories by replicating and migrating files between processors, subject to the constraint that each file is stored in at least one memory location.

Uniform cost metric. For the purpose of performance evaluation in this paper, we are only distinguishing between *local* and *remote* (over-the-network) data accesses, in effect assuming "flat" communication cost between any pair of processors, as in LogP model [CKP+].

This assumption indicates that the communication bottleneck is in transmitting and processing the message rather than in delivering it thru the network. In other words, there is plenty of network bandwidth, but not enough processing power; this assumption is considered reasonable in existing multiprocessors [CKP+]. Thus, the goal of the distributed paging algorithm is to minimize number of remote network accesses in the presence of dynamic and unpredictable access pattern.

Performance evaluation: the competitive framework. We deal with the analysis of online algorithms that make decisions based only on the past. An online algorithm is *c-competitive* [ST] if for every request sequence, its incurred cost is bounded by c times the cost expended by an optimal offline adversary plus a constant additive term. For randomized online algorithms, considered in this paper, competitiveness is defined analogously using the expected online cost [FKL+]

*Lab. for Computer Science, MIT. Supported by Air Force Contract TNDGAFOSR-86-0078, ARO contract DAAL03-86-K-0171, NSF contract 9114440-CCR, DARPA contract N00014-J-92-1799, and a special grant from IBM. E-mail: baruch@theory.lcs.mit.edu

†Department of Computer Science, School of Mathematics, Tel-Aviv University, Tel-Aviv 69978, Israel. Supported by a grant from the Israeli Academy of Sciences. E-mail: yairb@math.tau.ac.il, fiat@math.tau.ac.il

Comparison with previously considered settings. In a uni-processor setting, memory management involves moving memory across different levels of storage hierarchy, from bottom (e.g. slow disk) to top (e.g. fast cache); thus we call such data motion *vertical.* In contrast, in a distributed multi-processor with a single level of memory hierarchy, the data is confined to that level, and memory items are moved in the same hierarchy level across different processors; we call such data motion *horizontal.*

Distributed paging setting we consider differs sharply from the one where existence of a file server is assumed, where no space sharing takes place between different processors. In this case the problem is reduced to the problem of achieving efficient uni-processor paging algorithm, for which efficient solutions have been proposed [ST, KMRS, MS, FKL+].

In contrast, we are dealing here with a completely homogeneous environment with *distributed* file service, where we do not distinguish between clients and servers, and the space sharing between clients is allowed. The file server is thus replaced by the "aggregate cache", i.e. the conjunction of the local caches. In the context of parallel multiprocessors, this setting is referred to as *Cache-Only Memory Architecture* (COMA) [HALH], and it has has been employed in a number of recently developed parallel machines such as the Data Diffusion Machine [HALH] and in the new KSR1 machine [Bel], where it is referred to as ALL-CACHE ENGINE. This setting also corresponds to that of a homogeneous distributed file server, comprised of a collection of diskless workstations.

Previous results. In theory community, this problem was first tackled in [BFR] who defined this problem (the *constrained file allocation* problem). [BFR] give online algorithms with competitive factors that grow *linearly* with the memory capacity of the network. While linear competitive factor is proved [BFR] to be optimal for deterministic algorithms considered in[BFR], it is quite unsatisfactory as a performance guarantee in practice. The question of a randomized competitive ratio for this problem was left as an open problem in [BFR].

Main results of this paper. This paper presents a new randomized *distributed paging* algorithm for uniform networks, called Heat & Dump, whose competitive ratio against oblivious adversaries is logarithmic in the total storage capacity of the network. We also provide a matching (up to a constant) lower bound, thus proving that Heat & Dump is optimal to within a constant factor. In fact we give a lower bound for the distributed paging problem in an *arbitrary* metric space that strongly connects between this problem and the *k-server* problem [MMS].

Our approach. The basic idea of the algorithm is to capture locality of the underlying distributed program by migrating and replicating files in a way that brings "most" of the files into the locations where they are "frequently" and "recently" requested. The difficulty here is achieving this goal under the constraints of limited hardware.

Our algorithm will consist of a modular composition of two algorithms:

- *Local memory management strategy:* is invoked each time a file is requested by a processor; it specifies a file to be erased from local memory in order to free space for the currently requested file.
- *Global memory management strategy:* is used to prevent extinction of the last file copies as a result of some other file being over-replicated. This strategy, invoked each time a last copy of the file is going to be erased, determines memory location this "extinct" copy is to be "dumped" into in order to avoid loosing the last copy of the file.

The goal of local strategy is to minimize the number of faults in a uni-processing environment. This is exactly the "paging" problem in the uni-processor environment and we assume existence of an efficient solution for this problem, that will be used modularly as a "black box". The local paging algorithm could be efficient competitive deterministic or randomized paging algorithms ([ST, FKL+]), or other algorithms using some other variant of the competitive ratio (*e.g.*, [BIRS, IKP]). We allow the local strategy to be any strategy.

The focus of our paper is on designing the randomized global memory management strategy.

An interesting feature of our algorithm is the interaction between a "global" memory management strategy whose goal is to prevent last copy from becoming extinct, and the local memory management strategy, whose goal is to "selfishly" exploit locality of reference in the individual processor. In effect, these two strategies "conflict" with other. We show how to resolve these conflicts, and combine two strategies in a "competitive" way.

2 The Model

The basic memory management problem. We assume a network consisting of n processors, where processor p has the capacity to store up to k_p files, and the total memory capacity of the network $m = \sum_p k_p$. The number of different files in the network is no more than m. Let $\mathcal{F}$ denote the set of files in the network and let $f = |\mathcal{F}|$. Let $\mathcal{P}$ be the set of processors and let $n = |\mathcal{P}|$. Each file is stored in at least one, and possibly multiple processors. Input requests are of the form "processor p requires file F". Such request necessitates fetching file F into p, if F is not currently stored at p, and may involve some other configuration changes. The cost of serving such a request is 0 if F is stored in p and 1 if it is stored elsewhere. The cost of a "memory reconfiguration" equals to the total number of memory slots modified, which represents the total number of remote network accesses. The goal is to minimize the total cost of serving an (arbitrary) input sequence, including both the cost of serving the request sequence, and the cost of all memory reconfigurations. The algorithm must operate subject to the constraint that each file is stored in at least one memory location.

Generalizations. The above setting corresponds to a read-only access pattern in a network implementation of a distributed shared memory. If a write is made then the correct response is to treat the write as though it was a read but to discard all other copies of the file in the network. This certainly requires proof but we will not deal with this issue in this paper.

A further generalization is to treat read commands and file replication as different cost operations. The correct treatment here would be to keep a running count of the number of times a file was requested, and to require a file replication when this count reaches the ratio between the two costs, as elaborated in [BFR, ABF].

The analysis of the randomized algorithm herein is sufficiently complicated so as to make these additional features of the model simply confusing. Thus, throughout this paper, we focus on the simplest possible setting (the basic memory management problem above).

3 A Lower Bound

The lower bound is stated in the more general setting of the *distributed paging* on an arbitrary network topology defined in [BFR] (the *constrained file allocation problem*). We show that this problem is strongly related to the *k-server problem* [MMS]. The distributed paging problem in an arbitrary network is he simultaneous solution of *file allocation* problems [BFR] under the constraints of the local memory of the processors. Let $k = \max_p k_p$, the maximal number of files that a single processor can accommodate in its local memory.

Theorem 3.1 *Given a network $\mathcal{N}$ over n processors, let $S(\mathcal{N}, \ell)$ denote the largest randomized lower bound on the competitive ratio for an ℓ-server problem, $\ell < n$, in the metric space defined by $\mathcal{N}$. Let $\mathcal{U}$ denote the uniform metric space of f points. For any $\ell < n$ such that $k+n \leq f+\ell \leq m+1$, the competitive ratio of any randomized distributed paging algorithm for $\mathcal{N}$, against oblivious adversaries, is $\Omega(\max\{S(\mathcal{N}, \ell), S(\mathcal{U}, k)\})$.*

Proof: The $S(\mathcal{U}, k)$ lower bound follows from the lower bound for single processor paging with cache size k and f pages, which is analogous to the k-server problem on an f-points uniform metric space. We now turn the $S(\mathcal{N}, \ell)$ lower bound. We set the file configuration in the network to consist of f different files as follows. Let the $f+\ell-n-1$ different files $F_1, F_2, \ldots, F_{f+\ell-n-1}$ be assigned to processors so that each processor is assigned exactly $k_p - 1$ different files. This can be done since $\max_p (k_p - 1) \leq f + \ell - n - 1 \leq \sum_p (k_p - 1)$. After each request issued as in what follows, the adversary issues requests to each file F_i in its initial location so that any algorithm that does not hold F_i that location will incur a cost of at least the distance to F_i, and thus can be replaced with an algorithm that keeps F_i at its original location with at most a constant factor increase in performance. Henceforth we shall assume the algorithm does not move the files in $\{F_i\}$. The $n - \ell$ different files $D_1, D_2, \ldots, D_{n-\ell}$, are never requested but must reside amongst $n - \ell$ different processors in the network. The additional file Q receives *read* requests. Without loss of generality we may assume that any distributed paging algorithm will keep copies of Q in exactly ℓ different processors. Given a request for a server in the ℓ-servers lower bound at p, we initiate a *read* request for file Q in processor p. Given a distributed paging algorithm $\mathcal{A}$, we define an ℓ-server algorithm $\mathcal{A}'$ that locates servers at the nodes where copies of Q are held by $\mathcal{A}$. It is not difficult to see that this can be done so that the cost $\mathcal{A}$ is at least as large as the cost of $\mathcal{A}'$. Similarly, using the same transformation the adversary can simulate an optimal ℓ-server adversary with at most a constant factor increase in cost. Thus we obtain the lower bound. ∎

Corollary 3.2 *Let the $f > k$ be the number of different files in the network. Let $S(r)$ be the maximal r-server lower bound achievable for any metric space over $r + 1$ points, and assume $S(r) = O(\log r)$. The competitive ratio of any randomized distributed paging algorithm for $\mathcal{N}$ against oblivious adversaries is $\Omega(\max\{S(m + 1 - f), \log k\})$.*

Proof: The corollary follows from theorem 3.1 using $\ell = n-1$ if $f \leq m-n+1$ and $\ell = m+1-f$ otherwise. ∎

From the result of [FKL+] we have:

Corollary 3.3 *The competitive ratio of any randomized distributed paging algorithm is $\Omega(\max\{\log(m - f), \log k\})$ for uniform networks.*

4 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" set 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, 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. $\mathcal{R}(p,t) = \mathcal{H}(p,t) \cup \mathcal{C}(p,t) \cup \mathcal{D}(p,t)$.

A file is called non-single if more than one copy of the file resides in the network. We may also say that a file is hot, if some copy of the file in the network is hot. Similarly, we say that a file is cold or dumped if some copy of the file are cold or dumped, respectively.

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 a processor a dumped file. Files that are ejected from the virtual configuration in a processor are either erased from the processor (if not single) or become labeled dumped. 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 copies in the network. If there is no legitimate dump target then a phase is completed.

Just prior to the end of a phase, file copies that have been cold throughout the previous phase and that have been used elsewhere during the previous phase become labeled dumped. This is to avoid the situation where a file copy resides in a processor for several phases while labeled cold throughout.

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

Pseudo-code for the algorithm is presented in figure 1.

Notations:
  $\mathcal{V}(p,t)$ = current *virtual* configuration
  $\mathcal{R}(p,t)$ = current *real* configuration

Read F in processor p, at time t:
  compute $\mathcal{V}(p,t+1)$
  /* simulate local paging alg. $\mathcal{L}_p$ to */
  /* compute next virtual configuration */
  for all nonsingle file copies in $\mathcal{R}(p,t) - \mathcal{V}(p,t+1)$
    erase these copies from p
  if all processors are filled to capacity
    and there are no non-single *cold* file copies
    then
      mark *dumped* all *cold* file copies that were:
        cold at the end of previous phase
        requested (someplace) during this phase
    { End of Phase }
    set all *hot* file copies in the network to be *cold.*
    { Beginning of Next Phase }
  mark all files in $\mathcal{R}(p,t) - \mathcal{V}(p,t+1)$ *dumped* in p
  if F is in p then mark F *hot* in p and exit.
  if p is now filled to capacity
    then
      select an arbitrary *dumped* file copy B in p
      copy B into buffer, and erase it from p.
  copy F into p
  label F's copy in p *hot*
  if F was labeled *dumped*
    in the processor from which it was copied
    then erase it from that processor.
  if buffer is not empty
    then
      if every processor is filled to capacity
        then select, uniformly at random,
          a non-single cold file copy ($< G,q >$),
          and erase G from q.
        else let q be some processor
          not filled to capacity
      copy B into q and
      mark new copy $< B,q >$ *dumped.*

Figure 1: The Heat & Dump Algorithm (H&D). Initially, all files are single and are labeled *dumped.*

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

The main theorem we prove is the following:

Theorem 4.1 *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 [MS, FKL+] in all processors we get:

Corollary 4.2 *Algorithm* Heat & Dump *is $O(\max\{\log(m-f), \log k\})$-competitive against oblivious adversaries.*

5 Competitive Analysis of — Heat & Dump

This section gives the proof that Heat & Dump is $O(\max\{\log(m-f), c\})$ competitive.

We deal separately with "local issues" that would come up in regular local paging as well and more complicated "global issues". To do this we introduce the notion of *hallucinatory* files.

5.1 The Notion of Hallucination

We assume that all events in the input cycle begin and end at integral times. We also require the notions of "end of the phase" and "beginning of a phase". These are (non integral) points in time highlighted in the algorithm pseudo-code.

Let $\overline{\mathcal{R}}(p,t) = \mathcal{R}(p,t) \setminus \mathcal{D}(p,t)$ be the strongly-real configuration. We define a file to be *hallucinatory* at time t at processor p if it belongs to $\mathcal{V}(p,t)$ but not to $\overline{\mathcal{R}}(p,t)$. We call it *continuously hallucinatory* if it has stayed this way (hallucinatory) since the end of the previous phase. A file is said to be *newly hallucinatory* at time t if it is hallucinatory at time t and is not continuously hallucinatory.

Non-hallucinatory, continuously hallucinatory and newly hallucinatory *requests* in p are requests for non-hallucinatory, continuously hallucinatory and newly hallucinatory files at p, respectively.

Any hallucinatory file must have been in the virtual configuration of the processor since the beginning of the current phase. If it entered the virtual configuration at time $t > t_0$ then it must have also entered the strongly-real configuration (marked hot) and from that point onwards being in the virtual configuration implies that it is also in the strongly-real configuration.

There are two types of newly hallucinatory files, defined according to the type of event that turned them into hallucinatory files:

1. Genuine newly hallucinatory files: The file copy was cold and non-single at some time since the beginning of the phase and then it was erased from p because some other file was dumped into p.
2. Artificial newly hallucinatory files: The file was formerly cold and has been requested during the current phase and was cold at the end of the current phase, and thus was marked dumped at the end of the current phase.

5.2 Proof Overview

Given a sequence of requests $\overline{\sigma}$, let λ denote the sequence of *local coin tosses* used by the local paging algorithms, and let γ denote *global coin tosses* used by H&D for dumping. Let $\mathcal{H\&D}(\overline{\sigma})$ and $\mathcal{ADV}(\overline{\sigma})$ denote the online and adversary costs for serving $\overline{\sigma}$, respectively.

We first deal with the local issues in lemma 5.1. Let $\mathcal{NON}(\overline{\sigma})$ be the cost of non-hallucinatory requests. Fixing the global coin tosses we show this cost is bounded by the costs of the competitive local paging algorithms, obtaining:

$$\mathrm{E}_\lambda(\mathcal{NON}(\overline{\sigma})) \leq c \cdot \mathcal{ADV}(\overline{\sigma}) \tag{1}$$

where $\mathrm{E}_\lambda()$ denotes expectation over the coin tosses λ. From here on, we fix the local coin tosses λ.

We turn to deal with hallucinatory requests. The number of such requests in a phase is bounded by m. Thus by allowing an additive term we may assume that $\overline{\sigma}$ consists only of complete phases.

Lemma 5.2 simplifies dealing separately with each phase by proving that the length of a phase does not depend on the global coin tosses. Thus we can prove our claims by induction on the number of phases in the sequence. We let $\overline{\sigma} = \overline{\sigma}_0\sigma$, where $\overline{\sigma}_0$ is the subsequence of requests prior to the current phase, and σ is the sequence of requests during current phase. Let t_0 denote the time of the end of the previous phase. Let t_e denote the time of the end of the current phase. Also let t_e^- be the time just prior to t_e when requested formerly cold files are labeled dumped.

Now we first need to relate the number of hallucinatory files with the adversary cost. This is done in the following two main lemmas of the proof.

Fix the global coin tosses γ. Let γ_0 and τ be the global coin tosses over $\overline{\sigma}_0$ and over σ, respectively. In section 5.5 we define the amortized adversary cost during the phase at time t, $\mathcal{A}(t)$. Let $\overline{\mathcal{A}}(\overline{\sigma}) = \overline{\mathcal{A}}(\overline{\sigma}_0) + \mathcal{A}(t_e)$.

Lemma 5.3 proves a lower bound on the adversary cost:

$$\mathcal{ADV}(\overline{\sigma}) \geq \Omega(1) \cdot \overline{\mathcal{A}}(\overline{\sigma}) \tag{2}$$

Fix the global coin tosses. Let $nh(\tau, t)$ denote the number of newly hallucinatory files at time t. The amortized adversary cost is shown to be bounded from below by the high water mark of the number of newly hallucinatory files in lemma 5.4:

$$t < t_e^-, \qquad nh(\tau, t) \leq \mathcal{A}(t) \tag{3}$$
$$nh(\tau, t_e) \leq 2 \cdot \mathcal{A}(t_e)$$

The last bound is used in lemma 5.8 to account for the cost of continuously hallucinatory requests. Let $\mathcal{CH}(\overline{\sigma})$ denote this cost.

$$\mathcal{CH}(\overline{\sigma})) \leq 2 \cdot \overline{\mathcal{A}}(\overline{\sigma}) \tag{4}$$

Hence we are left with dealing with requests to newly hallucinatory files. Let $\mathcal{NH}(\overline{\sigma})$ denote the number of such requests. Fix the coin tosses γ_0 over $\overline{\sigma}_0$. Let $\mathcal{NH}(t)$ denote the cost of newly hallucinatory requests during σ up to time t. Analogously define $\mathcal{NON}(t)$. Lemma 5.9 analyzes the expected cost over the global coin tosses during the phase:

$$\begin{aligned} &\mathrm{E}_\tau(\mathcal{NH}(t_e)) \leq O(1) \cdot \mathrm{E}_\tau(\mathcal{NON}(t_e)) + \mathcal{A}(t_e) \\ &\quad + O(\log(m-f)) \cdot \max_\tau \max_{t<t_e^-}\{nh(\tau, t)\} \\ &= O(1) \cdot \mathrm{E}_\tau(\mathcal{NON}(t_e)) + O(\log(m-f)) \cdot \mathcal{A}(t_e) \end{aligned} \tag{5}$$

By induction on the number of phases in the sequence and using (4) we get:

$$\begin{aligned} &\mathrm{E}_\gamma(\mathcal{H}\&\mathcal{D}(\overline{\sigma})) \leq \\ &\quad \mathrm{E}_\gamma(\mathcal{NON}(\overline{\sigma}) + \mathcal{NH}(\overline{\sigma}) + \mathcal{CH}(\overline{\sigma})) \leq \\ &O(1) \cdot \mathrm{E}_\gamma(\mathcal{NON}(\overline{\sigma})) + O(\log(m-f)) \cdot \mathrm{E}_\gamma(\overline{\mathcal{A}}(\overline{\sigma})) \end{aligned} \tag{6}$$

Taking expectation over the local coin tosses λ and using (1) and (2) we conclude the proof:

$$\mathrm{E}_{\gamma,\lambda}(\mathcal{H}\&\mathcal{D}(\overline{\sigma})) \leq \quad O(\max\{\log(m-f), c\}) \cdot \mathcal{ADV}(\overline{\sigma})$$

5.3 Non-Hallucinatory Requests

Lemma 5.1

$$\mathrm{E}_\lambda(\mathcal{NON}(\overline{\sigma})) \leq c \cdot \mathcal{ADV}(\overline{\sigma})$$

Proof: Requests for non-hallucinatory files are part of the requests for files not in the virtual configuration. Thus $\mathcal{NON}(\overline{\sigma}) \leq \sum_p \mathcal{L}_p(\overline{\sigma}(p))$, where $\mathcal{L}_p(\overline{\sigma}(p))$ is the cost for the local paging algorithm $\mathcal{L}_p$ to serve the subsequence $\overline{\sigma}(p)$ of requests in p. From the competitiveness of the local paging algorithms we have: $\sum_p \mathcal{L}_p(\overline{\sigma}(p)) \leq \sum_p \mathcal{ADV}_p(\overline{\sigma}(p))$, where $\mathcal{ADV}_p(\overline{\sigma}(p))$ is the cost of a local paging adversary. The sum of costs of the local paging adversary is at most the cost of the distributed paging adversary since $\mathcal{ADV}_p$ can simulate $\mathcal{ADV}$'s actions in p. ∎

5.4 Fixed Length Phases

Lemma 5.2 *The length of a phase is independent of the global coin tosses of the online algorithm.*

Proof:

A file F is called *perfectly cold* at time t if all copies of F are cold. A file F is called *non-perfectly cold* at time t if there are both cold and hot copies of F.

Let $\mathcal{NS}(t)$ denote the number of nonsingle file copies at time t. Let $ns(t)$ denote the number of nonsingle files at time t. Let $\mathcal{NS}_H(t)$ and $ns_H(t)$ denote the same for hot files. Similarly, let $\mathcal{NS}_{NPC}(t), \mathcal{NS}_{PC}(t)$ and $ns_{PC}(t)$ denote the same for non-perfectly cold (NPC) and perfectly cold (PC) files respectively. Let $Empty(t)$ denote the number of empty slots in the network at time t.

At all times t the following holds:

$$\begin{aligned} m - f &= Empty(t) + \mathcal{NS}(t) - ns(t) \\ &= Empty(t) + \mathcal{NS}_{NPC}(t) + \mathcal{NS}_H(t) - ns_H(t) \\ &+ \mathcal{NS}_{PC}(t) - ns_{PC}(t) \end{aligned}$$

The proof is by induction on the number of phases in the sequence. We assume that the current phase has begun at a fixed time irrespective of the global coin tosses, and we prove it ends at a fixed time as well. At the beginning of the phase there are no hot files. All file copies become hot when requested. A hot copy is erased or labeled dumped iff it is removed from the virtual configuration of the processor. Since local coin tosses are fixed, all file copies are hot without dependency in the global coin tosses.

Therefore the value of the quantity:

$$Empty(t) + \mathcal{NS}_{NPC}(t) + \mathcal{NS}_{PC}(t) - ns_{PC}(t)$$

is independent of the global coin tosses. This quantity $= 0$ at time t iff the there are no empty slots in the network, and no cold nonsingle files in the network at time t; i.e., iff the phase ends when the next request arrives. This proves the lemma. ∎

5.5 Lower Bound on Adversary Cost

At the time of the end of the previous phase t_0, all processors must be filled to capacity and there are no non-single cold file copies in the network.

A file copy that was hot at the end of the previous phase (time t_0) is called *formerly hot*. A file copy that was cold at the end of the previous phase is called *formerly cold*. A file copy that was dumped at the end of the previous phase is called *formerly dumped*.

At the end of a phase, all cold and dumped files are single, thus all copies of the same file have the same label and we may talk of formerly hot, cold, and dumped files, without fear of confusion.

Fix the local and the global coin tosses and consider the current phase.

Lemma 5.3 *Let $a(p)$ be the number of different formerly hot or formerly cold files requested in p such that they were not held in p at the beginning of a phase. Let $b(p)$ be the number of formerly dumped files requested in p. Let $z = \sum_p (a(p) + b(p))$. Let ℓ be the number of formerly dumped files requested anywhere during the phase. We can partition the total adversary cost so as to associate an adversary cost with every phase so that the adversary cost associated with a phase is at least some constant fraction of $z - \ell$.*

Proof: This lemma requires a combinatorial "flow" argument. The underlying idea is that if the adversary cost to deal with these specific requests is low, then the adversary must have already paid for these requests elsewhere. Here, by overcounting the adversary costs by no more than a constant factor we can associate an adversary cost whenever the adversary adds a file to a processor and whenever the adversary deletes a file from a processor.

For any file F counted in $a(p)$ or $b(p)$, if the adversary did not hold F in p at the end of the previous phase, then the adversary pays to deal with (at least) one request for F in p, or for bringing F into p before the request. Let the number of such files be z_1.

Let $\mathcal{A}_R(p)$ be the number of different files F, requested during the current phase in p, such that the adversary did hold in p and the online algorithm did not hold in p at the end of the previous phase. Let $z_2 = \sum_p |\mathcal{A}_R(p)|$. Let $\mathcal{A}_{NR}(p)$ denote the same for the files not requested during the current phase in p.

To make room in p for these files, other files, that the online algorithm holds in p, must be missing from p in the off-line configuration at the end of the previous phase.

Let $\mathcal{HD}_H(p)$, $\mathcal{HD}_C(p)$ and $\mathcal{HD}_D(p)$ denote the number of formerly hot, formerly cold and formerly dumped files, respectively, that were held in p by the on-line algorithm and were not held by the adversary at the end of the previous phase.

It follows that for all p:

$$|\mathcal{HD}_H(p) \cup \mathcal{HD}_C(p) \cup \mathcal{HD}_D(p)| = |\mathcal{A}_R(p) \cup \mathcal{A}_{NR}(p)|$$

Since all cold and dumped files at the end of a phase are single the adversary must have held the files in $\mathcal{HD}_C(p) \cup \mathcal{HD}_D(p)$ in some processor different than p. It follows that:

$$\bigcup_p (\mathcal{HD}_C(p) \cup \mathcal{HD}_D(p)) \subseteq \bigcup_p (\mathcal{A}_R(p) \cup \mathcal{A}_{NR}(p))$$

Therefore we obtain:

$$\begin{gathered}
|\bigcup_p \mathcal{A}_R(p)| + |\bigcup_p \mathcal{A}_{NR}(p)| = \\
|\bigcup_p (\mathcal{HD}_H(p) \cup \mathcal{HD}_C(p) \cup \mathcal{HD}_D(p))| = \\
|\bigcup_p \mathcal{HD}_H(p)| + |\bigcup_p (\mathcal{HD}_C(p) \cup \mathcal{HD}_D(p)) \cap \bigcup_p \mathcal{A}_R(p)| \\
+|\bigcup_p (\mathcal{HD}_C(p) \cup \mathcal{HD}_D(p)) \cap \bigcup_p \mathcal{A}_{NR}(p)| \le \\
|\bigcup_p \mathcal{HD}_H(p)| + |\bigcup_p \mathcal{HD}_C(p) \cap \bigcup_p \mathcal{A}_R(p)| \\
+|\bigcup_p \mathcal{HD}_D(p) \cap \bigcup_p \mathcal{A}_R(p)| + |\bigcup_p \mathcal{A}_{NR}(p)|
\end{gathered}$$

Let $\mathcal{AHD}_D(p)$ be the number of formerly dumped files requested in p during the current phase and held by both online and adversary in p at the end of the previous phase. Let $z_3 = \sum_p |\mathcal{AHD}_D|$.

Note that $z = z_1 + z_2 + z_3$.

Hence,

$$\begin{gathered}
z_2 + z_3 = |\bigcup_p \mathcal{A}_R(p)| + |\bigcup_p \mathcal{AHD}_D(p)| \le \\
|\bigcup_p \mathcal{HD}_H(p)| + |\bigcup_p \mathcal{HD}_C(p) \cap \bigcup_p \mathcal{A}_R(p)| \\
+|(\bigcup_p \mathcal{HD}_D(p) \cap \bigcup_p \mathcal{A}_R(p)) \cup \bigcup_p \mathcal{AHD}_D(p)|
\end{gathered}$$

Let us analyze now the adversary cost we can associate with this phase for each file in the sets above.

For every file $F \in \mathcal{HD}_H(p)$ the adversary must have incurred a cost at the time of the request to F that labeled it *hot* in the previous phase, or for removing it from p before the end of the previous phase. This cost can be associated with the current phase.

The same argument holds for a file $F \in \mathcal{HD}_C(p)$ except that the request to it may have occurred several phases ago. But since we consider only files in $\mathcal{A}_R(q)$ for some $q \neq p$ we have that F was requested at q during this phase and thus either the *cold* copy of F will be deleted by the end of the current phase, or it will be labeled dumped at that time. Thus this is the last consecutive phase in which F will be cold and thus the adversary cost following the past request to F can be associated with this phase.

Finally the number of formerly dumped files that were requested during the current phase, irrespective of where they were held by the adversary is ℓ.

Thus, we can associated with this phase an adversary cost of $z - \ell$.

∎

We define the amortized adversary cost at time t within the phase $\mathcal{A}(t) = z(t) - \ell(t)$, where $z(t)$ and $l(t)$ are the values of z and l as defined above restricted to files requested up to time t. We remark that the amortized adversary cost is a nondecreasing function of the time. Every new request can add one to z or add one to both z and ℓ.

5.6 Credit for Hallucinatory Files

Fix the local and global coin tosses.

Lemma 5.4

$$\textit{At all times } t < t_e^-, \qquad nh(\tau, t) \le \mathcal{A}(t)$$
$$nh(\tau, t_e) \le 2 \cdot \mathcal{A}(t_e)$$

Proof: Here, where everything is deterministic, we use a potential function type argument.

Let t be some time within the current phase. Define the following quantities at time t. Here we mean the value of the function is 1 if the logical expression on the right is true, and otherwise 0.

- $\mathcal{E}_\mathcal{R}(p, F, t) = 1$ iff there exists time s, $t_0 < s < t$, $F \in \overline{\mathcal{R}}(p, s)$ and was nonsingle when a request arrived at processor p, and as a result of the request F was erased from p, and $F \notin \mathcal{H}(p, u)$, for all $s < u \le t$.

- $\mathcal{E}_\mathcal{D}(p, F, t) = 1$ iff there exists time s, $t_0 < s < t$, $F \in \overline{\mathcal{R}}(p, s)$ and F was nonsingle when a file was dumped into processor p, and as a result of the dump F was erased from p, and $F \notin \mathcal{H}(p, u)$, for all $s < u \le t$.

- $\mathcal{D}_\mathcal{R}(p, F, t) = 1$ if there exists time s, $t_0 < s < t$, $F \in \overline{\mathcal{R}}(p, s)$ and F was single when a request arrived at processor p, and as a result of the request F was dumped from p, and $F \notin \mathcal{H}(p, u)$, for all $s < u \le t$, and there exist a time $s < s' < t$ such that F was requested at some other processor at time s'.

- $Empty(p, t) =$ the number of empty slots in p at time t.

Claim 5.5

$$\mathcal{A}(t) \ \ge \ \sum_{p,F} \mathcal{E}_\mathcal{R}(p, F, t) + \sum_{p,F} \mathcal{E}_\mathcal{D}(p, F, t) + \sum_{p,F} \mathcal{D}_\mathcal{R}(p, F, t) - \sum_{p} Empty(p, t)$$

Claim 5.6

$$\sum_{p,F} \mathcal{E}_\mathcal{R}(p, F, t) + \sum_{p,F} \mathcal{D}_\mathcal{R}(p, F, t) \ge \sum_{p} Empty(p, t).$$

Proof of claim 5.5 and claim 5.6:

The proof is by induction on the time. At the beginning of the phase the claim holds, since all quantities above equal to zero. We shall assume the claims are true at time $t-1$ and examine the change in the various quantities defined above at time t. Consider the new state at time t following a request for file F at processor p. Along the rest of the proof we say a quantity increases (resp. decreases) if it increases (resp. decreases) by 1, unless specifically otherwise stated. We

	(a)	(b)
I-1	$\Delta Empty(p) = -1$	$\Delta\mathcal{E}_\mathcal{R}(p, G) = 1$
II-1.1	$\Delta Empty(q) = 1$ $\Delta Empty(p) = -1$ $\Delta\mathcal{D}_\mathcal{R}(u, F) = 1$	$\Delta Empty(q) = 1$ $\Delta\mathcal{E}_\mathcal{R}(p, G) = 1$ $\Delta\mathcal{D}_\mathcal{R}(u, F) = 1$
II-1.2	$\Delta Empty(q) = 1$ $\Delta Empty(p) = -1$	$\Delta Empty(q) = 1$ $\Delta\mathcal{E}_\mathcal{R}(p, G) = 1$
I-2	$\Delta\mathcal{E}_\mathcal{R}(p, F) = -1$ $\Delta Empty(p) = -1$	$\Delta\mathcal{E}_\mathcal{R}(p, F) = -1$ $\Delta\mathcal{E}_\mathcal{R}(p, G) = 1$
I-3	$\Delta\mathcal{E}_\mathcal{D}(p, F) = -1$ $\Delta Empty(p) = -1$	$\Delta\mathcal{E}_\mathcal{D}(p, F) = -1$ $\Delta\mathcal{E}_\mathcal{R}(p, G) = 1$
I-4	$\Delta\mathcal{D}_\mathcal{R}(p, F) = -1$ $\Delta Empty(p) = -1$	$\Delta\mathcal{D}_\mathcal{R}(p, F) = -1$ $\Delta\mathcal{E}_\mathcal{R}(p, G) = 1$
II-2	$\Delta Empty(q) = 1$ $\Delta Empty(p) = -1$ $\Delta\mathcal{D}_\mathcal{R}(u, F) = 1$ $\Delta\mathcal{E}_\mathcal{R}(p, F) = -1$	$\Delta Empty(q) = 1$ $\Delta\mathcal{E}_\mathcal{R}(p, G) = 1$ $\Delta\mathcal{D}_\mathcal{R}(u, F) = 1$ $\Delta\mathcal{E}_\mathcal{R}(p, F) = -1$
II-3	$\Delta Empty(q) = 1$ $\Delta Empty(p) = -1$ $\Delta\mathcal{D}_\mathcal{R}(u, F) = 1$ $\Delta\mathcal{E}_\mathcal{D}(p, F) = -1$	$\Delta Empty(q) = 1$ $\Delta\mathcal{E}_\mathcal{R}(p, G) = 1$ $\Delta\mathcal{D}_\mathcal{R}(u, F) = 1$ $\Delta\mathcal{E}_\mathcal{D}(p, F) = -1$
II-4	$\Delta Empty(q) = 1$ $\Delta Empty(p) = -1$ $\Delta\mathcal{D}_\mathcal{R}(u, F) = 1$ $\Delta\mathcal{D}_\mathcal{R}(p, F) = -1$	$\Delta Empty(q) = 1$ $\Delta\mathcal{E}_\mathcal{R}(p, G) = 1$ $\Delta\mathcal{D}_\mathcal{R}(u, F) = 1$ $\Delta\mathcal{D}_\mathcal{R}(p, F) = -1$

	(c1)	(c2)
I-1	$\Delta Empty(r) = -1$	$\Delta\mathcal{E}_\mathcal{D}(r, H) = 1$ $\sum_p Empty(p, t) = 0$
II-1.1	$\Delta Empty(q) = 1$ $\Delta Empty(r) = -1$ $\Delta\mathcal{D}_\mathcal{R}(u, F) = 1$	——
II-1.2	$\Delta Empty(q) = 1$ $\Delta Empty(r) = -1$	——
I-2	$\Delta\mathcal{E}_\mathcal{R}(p, F) = -1$ $\Delta Empty(r) = -1$	$\Delta\mathcal{E}_\mathcal{R}(p, F) = -1$ $\Delta\mathcal{E}_\mathcal{D}(r, H) = 1$ $\sum_p Empty(p, t) = 0$
I-3	$\Delta\mathcal{E}_\mathcal{D}(p, F) = -1$ $\Delta Empty(r) = -1$	$\Delta\mathcal{E}_\mathcal{D}(p, F) = -1$ $\Delta\mathcal{E}_\mathcal{D}(r, H) = 1$ $\sum_p Empty(p, t) = 0$
I-4	$\Delta\mathcal{D}_\mathcal{R}(p, F) = -1$ $\Delta Empty(r) = -1$	$\Delta\mathcal{D}_\mathcal{R}(p, F) = -1$ $\Delta\mathcal{E}_\mathcal{D}(r, H) = 1$ $\sum_p Empty(p, t) = 0$
II-2	$\Delta Empty(q) = 1$ $\Delta Empty(r) = -1$ $\Delta\mathcal{D}_\mathcal{R}(u, F) = 1$ $\Delta\mathcal{E}_\mathcal{R}(p, F) = -1$	——
II-3	$\Delta Empty(q) = 1$ $\Delta Empty(r) = -1$ $\Delta\mathcal{D}_\mathcal{R}(u, F) = 1$ $\Delta\mathcal{E}_\mathcal{D}(p, F) = -1$	——
II-4	$\Delta Empty(q) = 1$ $\Delta Empty(r) = -1$ $\Delta\mathcal{D}_\mathcal{R}(u, F) = 1$ $\Delta\mathcal{D}_\mathcal{R}(p, F) = -1$	——

Table 1: Case analysis of claims 5.5 and 5.6.
In cases I-1 and II-1.1: $\Delta\mathcal{A} = 1$. Otherwise: $\Delta\mathcal{A} \ge 0$.

assume $F \notin \mathcal{H}(p,t)$ since otherwise non of the quantities above changes. We consider all possible cases of file states and events. We assume that before the request at p, a copy of F was kept in processor q. We distinguish between 4 subcases depending on what happened when F was requested in p:

(a) p was not filled to capacity prior to the request.

(b) Some nonsingle file G was erased from p.

(c1) There was some empty slot in the network prior to this request and some single file G was dumped from p into some other processor r, r was not filled to capacity.

(c2) There were no empty slots in the network prior to this request and some single file G was dumped from p into some other processor r, overwriting some nonsingle cold file H.

We distinguish cases further according to the state of the file F. We have 2 classifications:

I) F is currently hot or cold.

II) F is currently dumped.

1. $F \notin R(p, t_0)$. This case is further classified:
   1.1. F has been requested before in current phase.
   1.2. First request for F in current phase.
2. F was last erased from p to make place for another file requested in p.
3. F was last erased from p to make place for another file dumped into p.
4. F was last dumped from p to make place for another file requested in p.

We first prove claim 5.5.

In cases I-1, and $\mathrm{II}-1.1$ the file requested is either formerly hot, or formerly cold in p, or it is formerly dumped and has been requested before in the phase. Thus $\mathcal{A}(t)$ increases, and we need to prove the right hand of the inequality in claim 5.5 increases by at most 1. For all other cases we show the right hand of the inequality in claim 5.5 does not change.

In table 1 we list by cases all changes in the components of the inequality in claim 5.5.

This follows from the following case analysis:

(a): $\Delta Empty(p) = -1$; (b): $\Delta\mathcal{E}_{\mathcal{R}}(p,G) = 1$; (c1): $\Delta Empty(r) = -1$. (c2): $\Delta\mathcal{E}_{\mathcal{D}}(p,H) = 1$.

II): F is dumped and thus its former copy is deleted after it is replicated in p. Therefore: $\Delta Empty(q) = 1$.

2.: $\Delta\mathcal{E}_{\mathcal{R}}(p,F) = -1$; 3.: $\Delta\mathcal{E}_{\mathcal{D}}(p,F) = -1$; 4.: $\Delta\mathcal{D}_{\mathcal{R}}(p,F) = -1$.

In cases II) together with any of 1.1.,2.,3., or 4.: let u be the last processor for which $F \in \overline{\mathcal{R}}(u,s)$ for some $s < t$. Since F is currently dumped we have that it was dumped from u. Since F is now requested at p, $\Delta\mathcal{D}_{\mathcal{R}}(u,F) = 1$.

There are two objections to the discussion above: We assumed $q \neq r$ and $p \neq u$. If $q = r$ then in case II) together with (c1), we have: $\Delta Empty(q) = 0$. If $p = u$ then in case II) together with 2.,3., or 4., we have: $\Delta\mathcal{D}_{\mathcal{R}}(p,F) = 0$.

We now turn to proving claim 5.6. It follows from what we have proven above that $\sum_{p,F}\mathcal{E}_{\mathcal{R}}(p,F,t) + \sum_{p,F}\mathcal{D}_{\mathcal{R}}(p,F,t) - \sum_p Empty(p,t)$ equals 0 or 1, unless $\mathcal{E}_{\mathcal{D}}(r,H) = 1$. This happens only in case (c2) in which $\sum_p Empty(p,t) = \sum_p Empty(p,t-1) = 0$, and therefore inequality 5.6 holds. ∎

Claim 5.7

$$\sum_{p,F}\mathcal{E}_{\mathcal{D}}(p,F,t) \geq nh(\tau,t).$$

Proof of claim 5.7:
Implied by the definition of genuine newly hallucinatory files. ∎

Therefore the lemma holds up to time t_e^- just prior to the end of the phase.

At time t_e^- more artificial newly hallucinatory files are created. The number of artificial newly hallucinatory files is at most the number of formerly cold files requested during the current phase at some processor not containing them at the beginning of the phase. By lemma 5.3 this is bounded at most $\mathcal{A}(t_e)$. This concludes the proof of the lemma. ∎

5.7 Continuously Hallucinatory Requests

Lemma 5.8

$$\mathcal{CH}(\overline{\sigma}) \leq 2\cdot\overline{\mathcal{A}}(\overline{\sigma})$$

Proof: This follows from the inductive claim that the number of continuously hallucinatory files at the beginning of the phase is bounded above by $2\cdot\overline{\mathcal{A}}(\overline{\sigma}_0)$. The induction step is based on the fact that the continuously hallucinatory files at the beginning of the next step is equal to the number of continuously hallucinatory files that remained hallucinatory throughout the phase, plus the number of newly hallucinatory files at the time of the end of the phase, with the use of lemma 5.4 ∎

5.8 Newly Hallucinatory Requests

Fix the local coin tosses λ and the global coin tosses γ_0 over $\overline{\sigma}_0$.

Lemma 5.9

$$\mathrm{E}_\tau(\mathcal{NH}(t_e)) \leq O(1) \cdot \mathrm{E}_\tau(\mathcal{NON}(t_e)) + \mathcal{A}(t_e)$$
$$+O(\log(m-f)) \cdot \max_\tau \max_{t<t_e^-}\{nh(\tau,t)\}$$

Proof of lemma 5.9:

Consider first cold files that were single at the end of previous phase. This files may become hallucinatory only if they are being requested in current phase at some processor not holding them at time t_0. The number of such files is at most $\mathcal{A}(t_e)$.

Let $nh(t) = \max_\tau\{nh(\tau,t)\}$ and let $nh = \max_{t<t_e^-}\{nh(t)\}$. Given a formerly hot nonsingle file copy F held in p at time t_0, and first requested in p during this phase at time t, we seek an upper bound on the probability that F is newly hallucinatory at p at time t. Denote the i'th request of this type r_i, and denote this probability $\mathrm{P}(r_i)$. Note that there can be at most $m-f$ such requests. The probability $\mathrm{P}(r_i)$ is bounded from above by the maximal number of newly hallucinatory file copies existing concurrently in the network divided by the minimal number of nonsingle cold file copies being candidates for dumps at every time up to time t.

Let $\mathcal{NS}_C(t)$ denote the number of nonsingle cold file copies at time t. We prove by induction on time that:

$$\mathcal{NS}_C(t) \geq (m-f) - 2 \cdot nh(t) - 2 \cdot \mathcal{NON}(t) - i$$

The only cases where $\mathcal{NS}_C(t)$ decreases are:

1. A copy of a nonsingle cold file F is erased from the virtual configuration of some processor, and is therefore erased and may also cause another copy of F to become single. In this case the request is non-hallucinatory.

2. A copy of a nonsingle cold file is erased to make place for a dumped file; i.e, it becomes hallucinatory. Again it may cause another copy of F to become single.

3. A copy of a nonsingle cold file F becomes hot. This decreases the number of nonsingle cold files by exactly 1.

Thus for all $i \leq (m-f) - 2 \cdot nh - 2 \cdot \mathcal{NON}(t_e)$,

$$\mathrm{P}(r_i) \quad \leq \quad \frac{nh}{(m-f) - 2 \cdot nh - 2 \cdot \mathcal{NON}(t_e) - i}$$

Summing over i this is bounded by nh times the (m-f)'th harmonic number. It follows that:

$$\mathrm{E}_\tau(\mathcal{NH}(t_e)) = O(nh \cdot \log(m-f) + \mathcal{NON}(t_e) + \mathcal{A}(t_e))$$

▮

References

[ABF] Baruch Awerbuch, Yair Bartal, and Amos Fiat. Competitive distributed file allocation. In *Proc. 25th ACM Symp. on Theory of Computing*, pages 164–173, May 1993.

[BFR] Yair Bartal, Amos Fiat, and Yuval Rabani. Competitive algorithms for distributed data management. In *Proc. 24th ACM Symp. on Theory of Computing*, pages 39–50, 1992.

[Bel] Gordon Bell. Ultracomputers: A Teraflop before its time. *Comm. of the ACM*, 35(8), 1992.

[BIRS] Alan Borodin, Sandy Irani, Prabhakar Raghavan, and Baruch Schieber. Competitive paging with locality of reference. In *Proc.of the 23rd Ann. ACM Symp. on Theory of Computing*, pages 249–259, may 1991.

[CKP+] David Culler, Richard Karp, David Patterson, Abhijit Sahay, Klaus Erik Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten von Eiken. Logp: Towards a realistic model of parallel computation. In *Proc. of Fourth ACM SIGPLAN Symp. on PPOPP*, pages 1–12, San Diego, 1993.

[FKL+] Amos Fiat, Richard Karp, Mike Luby, McGeoch, Daniel Sleator, and Young. Competitive paging algorithms. *J. of Algorithms*, 12(4):685–699, 1991.

[HALH] E. Hagersten, Par Andersson, Anders Landin, and Seif Haridi. Performance study of the DDM - a cache-only memory architecture. Technical Report R91:17, Swedish Institute of Computer Science, 1991.

[IKP] Sandy Irani, Anna Karlin, and Steven Phillips. Strongly competitive algorithms for paging with locality of reference. In *Proc. 3rd ACM-SIAM Symp. on Discrete Algorithms*, January 1992.

[KMRS] Anna Karlin, Mark Manasse, Larry Rudolpoh, and Daniel Sleator. Competitive snoopy caching. *Algorithmica*, 3(1):79–119, 1988.

[LLG+] L. Lenoski, J. Laundo, K. Gharachorloo, A. Gupta, and J.Hennessy. The directory-based cache coherence protocol for the dash multiprocessor. In *Proc. of 17th Intern. Symp. on Computer Architecture*, pages 148–159, 1990.

[MMS] 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.

[MS] L.A. McGeoch and D.D. Sleator. A strongly competitive randomized paging algorithm. Submitted to *Algorithmica*.

[ST] Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Comm. of the ACM*, 28(2):202–208, 1985.

[TDMF] T.Kilburn, D.Edwards, M.Lanigan, and F.Summer. One-level storage system. *IRE Transactions on Electronic Computers*, 2:223–235, 1962.

[YR] Wang Yongdong and Lawrence A. Rowe. *Cache Consistency and Concurrency Control in a Client/Server DBMS Architecture.* Clifford91, 1991.