Page Migration with Limited Local Memory Capacity

Susanne Albers* Hisashi Koga**

Abstract. Most previous work on page migration assumes that each processor, in the given distributed environment, has infinite local memory capacity. In this paper we study the migration problem under the realistic assumption that the local memories have limited capacities. We assume that the memories are *direct-mapped*, i.e., the processors use a hash function in order to locate pages in their memory. We show that, for a number of important network topologies, on-line algorithms with a constant competitive ratio can be developed in this model. We also study distributed paging. We examine the *migration version* of this problem in which there exists only one copy of each page. We develop efficient deterministic and randomized on-line algorithms for this problem.

1 Introduction

Many on-line problems of practical significance arise in distributed data management. As a result, there has recently been a lot of research interests in problems such as page migration, page replication and distributed paging, see e.g. [1, 2, 3, 5, 6, 8, 10, 12]. In page migration and replication problems, a set of memory pages must be distributed in a network of processors, each of which has its local memory, so that a sequence of memory accesses can be processed efficiently. Specifically, the goal is to minimize the communication cost. If a processor p wants to read a memory address from a page b that is not in its local memory, then p must send a request to a processor q holding b and the desired information is transmitted from q to p. The communication cost incurred thereby is equal to the distance between q and p. It is also possible to move or copy a page from one local memory to another. However, such a transaction incurs a high communication cost proportional to the page size times the distance between the involved processors.

In the *migration problem* it is assumed that there exists only one copy of each page in the entire distributed system. This model is particularly useful when we deal with writable pages because we do not have to consider the problem of keeping multiple copies of a page consistent. The migration problem is to decide which local memory should contain the single copy of a given page. In the *replication problem*, multiple copies of a page may exist. Hence this model is suitable when we deal with read-only pages. The decision whether a given page should be migrated or replicated from one local memory to another must typically be made *on-line*,

* International Computer Science Institute, Berkeley; and Max-Planck-Institut für Informatik, Saarbrücken, Germany. Supported in part by an Otto Hahn Medal Award of the Max Planck Society and by the ESPRIT Basic Research Actions Program of the EU under contract No. 7141 (ALCOM II). E-mail: albers@icsi.berkeley.edu

** Department of Information Science, The University of Tokyo, Tokyo 113, Japan. Part of this work was done while the author was visiting the Max-Planck-Institut für Informatik. E-mail: nwa@is.s.u-tokyo.ac.jp

i.e., the memory management algorithm does not know which processors will have to access a page in the future. Because of this on-line nature, the performance of migration and replication algorithms is usually evaluated using competitive analysis.

Page migration and replication are extensively studied problems. However, almost all of the research results are developed under the assumption that the capacities of the local memories are infinite: Whenever we want to move or copy a page into the local memory of a processor p, there is room for it; no other page needs to be dropped from p's memory. Assuming infinite local capacity, on-line migration and replication algorithms with a constant competitive ratio can be developed [1, 3, 6, 8, 10, 12]. For example, Black and Sleator [6] presented a deterministic 3-competitive migration algorithm when the network topology is a tree or a complete uniform network. In practice, however, the local capacities are of course not unlimited. Basically the only work that considers local memories with finite capacity is the paper by Bartal *et al.* [5]. They investigate a combination of the migration and replication problem and present an $O(m)$-competitive on-line algorithm for complete uniform networks. Here m is total number of pages that can be stored in the entire network. Unfortunately, this competitive ratio is too high to be meaningful in practice.

In this paper we study the migration problem under the assumption that every local memory has a fixed finite capacity. More precisely, every local memory consists of k block $[1], [2], \ldots, [k]$, each of which can hold one page. We assume that the local memories are *direct-mapped*, i.e., each processor uses a hash function in order to locate pages in its local memory. Specifically, all processors use the same hash function h. This implies that whichever local memory a page b belongs to, it is always stored in block $[h(b) \bmod k + 1]$. Direct-mapped memories constitute an important memory class in practice. From a theoretical point of view they were studied only once before in [9]. We call the migration problem in direct-mapped memories of limited capacity the *direct-mapped constrained migration problem*. We will show that for this problem, we can develop simple on-line algorithms with a constant competitive ratio. Hence this is essentially the first work on page migration that makes realistic assumptions as far as memory is concerned and develops results that are meaningful in practice.

In Section 3 we investigate lower bounds on the competitiveness that can be achieved by deterministic on-line algorithms for the direct-mapped constrained migration problem. We show that, given any network topology, no deterministic on-line algorithm can be better than 3-competitive. We also prove that there are specific network topologies for which no deterministic on-line algorithm can be better than $\Omega(n)$-competitive; n denotes the number of processors in the network. In Section 4 we develop upper bounds. First, we present an optimal 3-competitive deterministic algorithm for networks consisting of two nodes. Next we develop an 8-competitive deterministic algorithm for complete uniform networks. This algorithm achieves a competitiveness of 16 on uniform stars. Finally, we give a 5-competitive randomized and memoryless on-line algorithm for complete uniform networks against adaptive on-line adversaries.

We also study the *distributed paging problem*. In distributed paging, each time

a processor p wants to access a page, this page must be brought into p's local memory, provided the page is not yet present at p. Loosely speaking, the goal is to minimize the number of times at which the requested page is not present in the corresponding local memory. For the *allocation version* of this problem, when multiple copies of a page may exist, Bartal *et al.* [5] presented a deterministic $O(m)$-competitive on-line algorithm; Awerbuch *et al.* [2] developed a randomized $O(\max\{\log(m-l), \log k\})$-competitive algorithm against the oblivious adversary. Again, m is the total number of pages that can be stored in the system, and l is the number of different pages in the system. In this paper we examine the *migration version* of the distributed paging problem; i.e., only one copy of each page may exist. In Section 5 we present an $O(k)$-competitive deterministic and an $O(\log k)$-competitive randomized on-line algorithm (k is the number of pages that each processor can hold). Our randomized algorithm is simpler than that of Awerbuch *et al.* for $l \geq m - k$.

2 Problem definition

We define the direct-mapped constrained migration problem and the distributed paging problem. We also review the notion of competitiveness.

In the *direct-mapped constrained migration problem* we are given an undirected graph $G = (V, E)$. Each node in G corresponds to a processor, and the edges represent the interconnection network. Let $|V| = n$. Associated with each edge is a *length* that is equal to the distance between the connected nodes. Let δ_{uv} denote the length of the shortest path between node u and node v. Each node has its own local memory. Every local memory is divided into k *blocks* $[1], [2], \cdots, [k]$, each of which can hold exactly one page. All nodes use the same hash function $h(b)$ to determine the unique block in which page b will reside. At any time, a node cannot simultaneously hold pages b and c with $h(b) \bmod k = h(c) \bmod k$. On the other hand, there is never a conflict between two pages e and f with $h(e) \bmod k \neq h(f) \bmod k$. Thus we can divide the direct-mapped constrained migration problem into k separate subproblems according to the block number. In the following, we concentrate on one particular block number i ($1 \leq i \leq k$). Let B be the number of pages b such that $h(b) \bmod k + 1 = i$, and let $b_1, b_2, \cdots, b_B$ be the pages whose hash value is equal to i. We always assume $B \leq n$, which is easily realized by a proper choice of h.

We say that *a node v has a page b* if b is contained in block $[i]$ of v's memory. A node v is said to be *empty* if v does not hold a page in block $[i]$. *A request to page b at a node v* occurs if v wants to read or write b. The request can be satisfied at zero cost if v has b. Otherwise the request incurs a cost equal to the distance from v to the node u holding b (i.e. the cost is δ_{uv}). After a request to page b at a node v, b may be migrated into v's local memory. If v is empty, the cost incurred by this migration is $d \cdot \delta_{uv}$. Here d denotes the page size factor. In case v has another page c, we may swap b and c, incurring a cost of $2d \cdot \delta_{uv}$. (Of course, it is also possible to move c to another node, but we will never make use of this possibility.) A direct-mapped constrained migration algorithm is usually presented with an entire sequence of requests that must be served with low total

cost. The algorithm is *on-line* if it serves every request without knowledge of any future requests.

Next we define the *distributed paging problem*. Again, we consider a network consisting of n nodes, each of which can store up to k pages. We only distinguish between local and remote data accesses: A request to page b at node v can be satisfied at zero cost if v has b. Otherwise the request is satisfied by fetching b into v's local memory, which may accompany other page configuration changes. The cost incurred is equal to the number of transferred pages because each page transfer requires exactly one remote access. The goal is to reduce the total number of page transfers. The distributed paging problem is named the *migration version* if the number of copies for any page is restricted to 1.

We analyze the performance of on-line algorithms using *competitive analysis* [11]. That is, the cost incurred by an on-line algorithm is compared to the cost of an *optimal off-line algorithm*. An optimal off-line algorithm knows the entire request sequence in advance and can serve it with minimum cost. Given a request sequence σ, let $C_A(\sigma)$ and $C_{OPT}(\sigma)$ denote the cost of the on-line algorithm A and the optimal off-line algorithm OPT in serving σ. A deterministic on-line algorithm A is c-competitive if there exists a constant a such that for every request sequence $C_A(\sigma) \leq c \cdot C_{OPT}(\sigma) + a$. In case A is a randomized algorithm, the on-line setting is viewed as a request-answer game in which an adversary generates a request sequence σ, see [4]. The expected cost incurred by A is then compared to the cost paid by the adversary. The *oblivious adversary* constructs σ in advance before any actions of A are made; the adversary may serve σ off-line. The *adaptive on-line adversary* constructs σ on-line, knowing the responses of A to previous requests; the adversary also has to serve σ on-line.

3 Lower bounds

Theorem 1 shows that the power of on-line algorithms is limited, no matter how simple the underlying graph structure may be.

Theorem 1. *Let A be a deterministic on-line algorithm for the direct-mapped constrained migration problem. Then A cannot be better than 3-competitive, even on a graph consisting of only two nodes.*

Proof. Consider a 2-node network and let b_1 and b_2 be two pages whose location needs to be managed. Consider request sequences consisting of requests to b_1 only. To process such sequences, A can concentrate on the location of b_1. However, to change the location of b_1, b_2 must also be moved as the result of a swap. Thus, this situation can be regarded as a migration problem with page size factor $2d$. Therefore, the lower bound of 3-competitiveness presented by Black and Sleator [6] for the migration problem also holds for the direct-mapped constrained migration problem. □

Next we prove the existence of specific topologies for which no deterministic on-line algorithm is better than $(n-2)$-competitive. The following star H is an example. Let $v_1, v_2, \cdots, v_n$ be the nodes in H, with v_1 being the center node. The edge lengths are defined as $\delta_{v_1 v_i} = 1$ for $i = 2, \ldots, n-1$ and $\delta_{v_1 v_n} = n-2$.

Theorem 2. *Let A be a deterministic on-line algorithm for the direct-mapped constrained migration problem working on the star H. Then A cannot be better than $(n-2)$-competitive.*

This theorem certifies that there is a difference between the migration problem (when the local memories are infinite) and the direct-mapped constrained migration problem. Recall that, for the migration problem, Black and Sleator [6] developed a deterministic on-line algorithm that is 3-competitive for trees including all stars.

Proof of Theorem 2: We will construct a request sequence σ so that $C_A(\sigma)$ is at least $(n-2)$ times the cost incurred by some off-line algorithm OFF. We assume that initially, both A and OFF have the same page at v_n. The request sequence σ is constructed as follows. An adversary always generates a request at v_1; it asks for the page that A stores in v_n. Therefore, A incurs a cost of $\delta_{v_1 v_n} = n-2$ at each request. We partition σ into phases. The first phase starts with the first request. It ends after $n-1$ distinct pages were requested during the phase and just before the remaining nth page b_r is requested. The second phase begins with the request to b_r and ends in the same way as the end of the first phase. The subsequent phases are determined similarly.

We show that in any phase, the cost incurred by A is at least $(n-2)$ times the cost incurred by OFF. Let σ' be a subsequence of σ that corresponds to a phase, and let l be the length of σ'. A incurs $n-1$ swaps in σ', each of which costs $2(n-2)d$. Thus the total cost for swaps is $2(n-1)(n-2)d$. In addition, A pays a cost of $(n-2)l$ to satisfy the requests. Therefore, $C_A(\sigma') \geq 2(n-1)(n-2)d + (n-2)l$. The following off-line algorithm OFF can serve σ' at a cost of $2(n-1)d + l$. At the beginning of σ', before the first request, OFF swaps the page located at v_n and the page b_r which is requested at the beginning of the next phase. After this swap, OFF does not change the locations of pages throughout the phase. Note that b_r is never requested in σ'. OFF incurs a cost of at most $2(n-1)d$ for the swap, and a cost of at most l to satisfy the requests in σ' because every page requested in σ' is located at one of the nodes $v_1, \ldots, v_{n-1}$. Thus $C_{OFF}(\sigma') \leq 2(n-1)d + l$. By comparing $C_A(\sigma')$ and $C_{OFF}(\sigma')$ we conclude $C_A(\sigma') \geq (n-2)C_{OFF}(\sigma')$. At the beginning of each phase, node v_n has the same page both in A's and OFF's configuration. This implies that we can extend σ arbitrarily by repeating the above construction. □

4 Upper bounds

We develop on-line algorithms for the direct-mapped constrained migration problem. First we present a 3-competitive deterministic algorithm for the case that the network consists of only two nodes. This topology is of course very special, but we have an optimal algorithm for this case. Most of this section deals with important network topologies such as complete uniform graphs and uniform stars. We give $O(1)$-competitive algorithms for these networks.

First consider a 2-node network consisting of nodes u and v. A direct-mapped constrained migration algorithm has to manage the location of two pages b_1 and

b_2. Note that there are only two possible page configurations: u has b_1 and v has b_2; or u has b_2 and v has b_1. Our algorithm TN for 2-node networks is given below. The proof of Theorem 3 is omitted in this extended abstract.

Algorithm TN: The algorithm maintains one global counter that is initialized to 0. Whenever a node requests a page that is not in its local memory, the counter is incremented by 1. When the counter reaches $4d$, the page configuration changes, i.e. the pages are swapped, and the counter is reset to 0.

Theorem 3. *TN is 3-competitive for graphs consisting of two nodes.*

In the remainder of this section we study on-line algorithms for uniform graphs. First we present a deterministic algorithm for complete uniform graphs. We assume w.l.o.g. that all edges in the network have length 1. As the name suggests, our algorithm is thought of as a concurrent version of algorithm M presented by Black and Sleator [6] for the migration problem.

Algorithm Concurrent-M: Each node v has B counters $c_v^{b_i}$ $(1 \leq i \leq B)$. All counters are initialized to 0. Concurrent-M processes a request at node v to page b_i as follows. If v has b_i already, then the request is free and nothing happens. If v does not have b_i, then the algorithm increments $c_v^{b_i}$, and chooses some other non-zero counter among $\{c_w^{b_i} | w \in V\}$, if there is one, and decrements it. When $c_v^{b_i}$ reaches $2d$, one of the following two steps is executed. If v is empty, then b_i is migrated to v and $c_v^{b_i}$ is reset to 0. Otherwise b_i is swapped with the page $b_j (i \neq j)$ that v currently holds, and $c_v^{b_i}$ and $c_u^{b_j}$ are reset to 0. Here u denotes the node that stored b_i before the swap.

In the above swap, we say that b_i is swapped *actively* and that b_j is swapped *passively*.

Theorem 4. *Concurrent-M is 8-competitive for complete uniform graphs.*

The next lemma is crucial for the analysis of Concurrent-M. A similar lemma was shown in [6].

Lemma 5. *For every page b, $\sum_{v \in V} c_v^b \leq 2d$.*

Proof. We prove the lemma by induction. Initially $\sum_{v \in V} c_v^b = 0$. The sum $\sum_{v \in V} c_v^b$ only increases when one counter is incremented and all other counter values are 0. Since the description of the algorithm implies that a counter value cannot exceed $2d$, the sum $\sum_{v \in V} c_v^b$ cannot be larger than $2d$. □

This lemma leads to an important fact: Just before a page b is swapped actively to node v, $c_v^b = 2d$ and all other counters associated with b are 0. After the swap, all counters associated with b are 0.

Proof of Theorem 4: We analyze the algorithm for the case $B = n$. The analysis is easily extended to $B \leq n$. Let $C_{CM}(\sigma)$ be the cost paid by Concurrent-M. We shall show that, for any (on-line and off-line) algorithm A and any request sequence σ, $C_{CM}(\sigma) \leq 8C_A(\sigma)$. Our proof uses the standard technique of comparing simultaneous runs of Concurrent-M and A on σ by merging the actions generated by Concurrent-M and A into a single sequence of events. This sequence contains three types of events: (Type I) Concurrent-M swaps pages, (Type II)

A swaps pages, and (Type III) both A and Concurrent-M satisfy a request. We shall give a non-negative potential function Φ (initially 0) such that the following inequality holds for all kinds of events.

$$\Delta C_{CM} + \Delta\Phi \leq 8\Delta C_A, \tag{1}$$

where Δ indicates the change of the values as the result of the event. If the potential function satisfies the above property for all events, summing up (1) for all events results in $C_{CM}(\sigma) + \Phi_{\text{end}} - \Phi_{\text{start}} \leq 8C_A(\sigma)$, where Φ_{start} denotes the initial value of Φ and Φ_{end} denotes the value of Φ after Concurrent-M and A finish processing σ. Since $\Phi_{\text{start}} = 0$ and $\Phi_{\text{end}} \geq 0$ from the definition of the potential function, we have $C_{CM}(\sigma) \leq 8C_A(\sigma)$, and the proof is complete. It remains to specify the potential function and verify (1) for all events.

The potential function Φ is defined as follows. Let s^b be the node that has page b in Concurrent-M and t^b be the node that has b in A.

$$\Phi = \sum_b \Phi_b, \quad \Phi_b = \begin{cases} 5\sum_{v \in V} c_v^b & \text{if } s^b = t^b. \\ 4d - c_t^b + 3\sum_{\substack{v \in V \\ v \neq t}} c_v^b & \text{if } s^b \neq t^b \end{cases}$$

In the following we prove (1) for all kinds of events. In the subsequent proof we omit the specification of the page in the counter variables when it is obvious.
(Type I): Concurrent-M swaps pages.
Suppose that page b_1 is swapped actively from s to s' and page b_2 is swapped passively from s' to s. As the result of this swap, $c_{s'}^{b_1}$ is reset from $2d$ to 0 and $c_s^{b_2}$ is reset from some non-negative value l to 0. Let t be the location of b_1 and let u be the location of b_2 in A. Then $\Delta C_{CM} = 2d$ and $\Delta C_A = 0$. So we must show that $\Delta\Phi \leq -2d$. Trivially, $\Delta\Phi = \Delta\Phi_{b_1} + \Delta\Phi_{b_2}$. First consider $\Delta\Phi_{b_1}$. There are three cases depending on whether s, s' coincide with t. Lemma 5 and the fact obtained from the lemma make the calculation of $\Delta\Phi_{b_1}$ very simple.

$$s' = t : \Delta\Phi_{b_1} = 5\sum 0 - (4d - 2d + \sum 0) = -2d$$
$$s = t : \Delta\Phi_{b_1} = (4d - 0 - 3\sum 0) - 5 \cdot 2d = -6d$$
$$s, s' \neq t : \Delta\Phi_{b_1} = (4d - 0 - 3\sum 0) - (4d - 0 - 3 \cdot 2d) = -6d$$

Next we calculate $\Delta\Phi_{b_2}$. For clearness, we express the counter value of c_s before the swap simply by $c_s\,(=l)$ and that after the swap by $c'_s\,(=0)$.

$$s = u : \Delta\Phi_{b_2} = 5\sum_{v \in V} c_v - (4d - c_u + 3\sum_{\substack{v \in V \\ v \neq u}} c_v) = 2\sum_{\substack{v \in V \\ v \neq s}} c_v + 5c'_s + c_s - 4d$$
$$= 2\sum_{\substack{v \in V \\ v \neq s}} c_v + c_s - 4d \leq 2\sum_{v \in V} c_v - 4d \leq 0.$$
$$s' = u : \Delta\Phi_{b_2} = (4d - c_u + 3\sum_{\substack{v \in V \\ v \neq u}} c_v) - 5\sum_{v \in V} c_v \leq (4d + 3\sum_{\substack{v \in V \\ v \neq s'}} c_v) - 5\sum_{\substack{v \in V \\ v \neq s'}} c_v$$

$$\leq 4d + 3c'_s - 5c_s - 2 \sum_{\substack{v \in V \\ v \neq s,\, s'}} c_v \leq 4d$$

$$s, s' \neq u : \Delta\Phi_{b_2} = (4d - c_u + 3 \sum_{\substack{v \in V \\ v \neq u}} c_v) - (4d - c_u + 3 \sum_{\substack{v \in V \\ v \neq u}} c_v)$$
$$= 3(c'_s - c_s) = -3l \leq 0.$$

Adding $\Delta\Phi_{b_1}$ and $\Delta\Phi_{b_2}$ we can calculate $\Delta\Phi$. For example, if $s = t$ and $s' = u$, then $\Delta\Phi = \Delta\Phi_{b_1} + \Delta\Phi_{b_2} \leq -6d + 4d = -2d$. The sum $\Delta\Phi_{b_1} + \Delta\Phi_{b_2}$ can only be greater than $-2d$ if $s' = t$ and $s' = u$. However, this case is impossible because a node cannot have both b_1 and b_2 at the same time, and hence t and u cannot be identical. Thus, in all cases $\Delta\Phi \leq -2d$ and (1) holds for (Type I).
(Type II): A swaps pages.
Suppose that page b_1 is swapped from t to t' and that page b_2 is swapped from t' to t. Then $\Delta C_{CM} = 0$ and $\Delta C_A = 2d$. We must show that $\Delta\Phi \leq 16d$. Again we calculate $\Delta\Phi_{b_1}$ and $\Delta\Phi_{b_2}$ separately and then compute $\Delta\Phi$. Let s be the location of b_1 and w be the location of b_2 in Concurrent-M. First consider $\Delta\Phi_{b_1}$.

$$t' = s : \Delta\Phi_{b_1} = 5 \sum_{v \in V} c_v - (4d - c_t + 3 \sum_{\substack{v \in V \\ v \neq t}} c_v) \leq 6 \sum_{v \in V} c_v - 4d \leq 12d - 4d = 8d$$

$$t = s : \Delta\Phi_{b_1} = (4d - c_{t'} + 3 \sum_{\substack{v \in V \\ v \neq t'}} c_v) - 5 \sum_{v \in V} c_v = 4d - 6c_{t'} - 2 \sum_{\substack{v \in V \\ v \neq t'}} c_v \leq 4d$$

$$t, t' \neq s : \Delta\Phi_{b_1} = (4d - c_{t'} + 3 \sum_{\substack{v \in V \\ v \neq t'}} c_v) - (4d - c_t + 3 \sum_{\substack{v \in V \\ v \neq t}} c_v) = 4(c_t - c_{t'}) \leq 8d$$

We conclude $\Delta\Phi_{b_1} \leq 8d$. Next consider $\Delta\Phi_{b_2}$. Since there is no distinction between b_1 and b_2, the same analysis as above gives $\Delta\Phi_{b_2} \leq 8d$. Thus, the total change in potential is $\Delta\Phi = \Delta\Phi_{b_1} + \Delta\Phi_{b_2} \leq 16d$, and (1) holds for (Type II).
(Type III) A request is satisfied by both A and Concurrent-M.
Suppose there is a request at node v to page b. Let s be the node at which Concurrent-M stores b, and let t be the node at which A holds b.
Case 1: $v = s$. $\Delta C_{CM} = 0$. $\Delta C_A \geq 0$. $\Delta\Phi = 0$. Thus (1) is satisfied.
Case 2: $v \neq s$. In this case $\Delta C_{CM} = 1$ because v does not have b in Concurrent-M. The counter c_v^b is incremented by 1. We need to consider three cases.
Case (a): Suppose that $v = t$. $\Delta C_A = 0$. So we have to show that $\Delta\Phi \leq -1$. Note that $s \neq t$. The increment of c_t^b decreases Φ by 1. In case another counter is decremented, then Φ decreases further by 3. Thus $\Delta\Phi \in \{-4, -1\} \leq -1$.
Case (b): Suppose that $v \neq t = s$. $\Delta C_A = 1$. So we must show that $\Delta\Phi \leq 7$. The increment of c_v^b increases Φ by 5. If another counter is decremented, then Φ decreases by 5. Thus $\Delta\Phi \in \{0, 5\} \leq 7$.
Case (c) Suppose that $v \neq t \neq s$. $\Delta C_A = 1$ and we must show that $\Delta\Phi \leq 7$. The increment of c_v^b increases Φ by 3. If no decrement takes place, $\Delta\Phi = 3$. Else if another counter except c_t^b is decremented, Φ decreases by 3 and totally $\Delta\Phi = 0$. If c_t^b is decremented, Φ increases by 1, and in total $\Delta\Phi = 4$. □

We can treat Concurrent-M as an on-line algorithm for uniform stars (stars in which all edges have length 1).

Theorem 6. *Concurrent-M is 16-competitive for uniform stars.*

Proof. Let US be the uniform star consisting of n nodes $v_1, v_2, \cdots, v_n$, with v_1 being the center node. All edges have length 1. Let K_1 and K_2 be two complete uniform graphs consisting of n nodes each; in K_1 all edges have length 1 and in K_2 all edges have length 2. Let $u_1, u_2, \cdots, u_n$ and $w_1, w_2, \cdots, w_n$ be the nodes in K_1 and K_2, respectively. Our analysis maps an arbitrary request sequence σ on US onto two request sequences σ' on K_1 and σ'' on K_2, and then compares simultaneous runs of Concurrent-M on σ, σ' and σ''. Assume that initially, nodes v_i, u_i and w_i have the same page in their memory, for all i $(1 \leq i \leq n)$.

We construct σ' from σ by replacing each request to a page b at node v_i in σ by a request to b at node u_i in σ'. σ'' is derived from σ similarly. If we simultaneously run Concurrent-M on σ, σ' and σ'', the (fixed) counter decrement strategy implies that whenever Concurrent-M moves a page from v_i to v_j in US, the same page is moved from u_i to u_j in K_1 and from w_i to w_j in K_2. Hence, at any time, the page stored at v_i is identical to the page stored at u_i and w_i. Since for any pair of indexes i and j, $\delta_{u_i u_j} \leq \delta_{v_i v_j} \leq \delta_{w_i w_j}$, we have $C_{CM}(\sigma') \leq C_{CM}(\sigma) \leq C_{CM}(\sigma'')$. Similarly, $C_{OPT}(\sigma') \leq C_{OPT}(\sigma) \leq C_{OPT}(\sigma'')$. We have $C_{CM}(\sigma'') \leq 8C_{OPT}(\sigma'')$ because, by Theorem 4, Concurrent-M is 8-competitive for complete uniform graphs. Also, $C_{OPT}(\sigma'') = 2C_{OPT}(\sigma')$ because of the relation between K_1 and K_2. The above formulae give $C_{CM}(\sigma) \leq C_{CM}(\sigma'') \leq 8C_{OPT}(\sigma'') = 16C_{OPT}(\sigma') \leq 16C_{OPT}(\sigma)$ □

Next we present a randomized on-line algorithm for complete uniform graphs. The algorithm is *memoryless*, i.e. it does not need any memory (e.g. for counters) in order to determine when a migration or a swap should take place.

Algorithm COINFLIP: Suppose that there is a request at node v to page b. If v has b, COINFLIP performs no action. If v does not have b, the algorithm serves the request by accessing to the node u that has b. Then with probability $\frac{1}{3d}$, the algorithm migrates b from u to v if v is empty, and moves b from u to v by a swapping operation if v is not empty.

Theorem 7. *COINFLIP is 5-competitive against adaptive on-line adversaries.*

Proof. A detailed proof is omitted; we just give the main idea. Let $\Phi = 5d \cdot |S|$, where S is the set of nodes at which COINFLIP and the adversary A have different pages. Using this potential function we can show $E[C_{CF}(\sigma)] \leq 5C_A(\sigma)$. □

5 On-line algorithms for distributed paging

We present a deterministic on-line algorithm for the migration version of the distributed paging problem. Let B be the number of different pages in the system.

Algorithm DLRU: Each processor v has B counters $c_v[b_i]$ $(1 \leq i \leq B)$. All counters are initialized to 0. The algorithm maintains the invariant that $c_v[b_i] = 0$ if (but not only if) b_i does not belong to v's memory. DLRU serves a request at node v to page b_i as follows. If v has b_i, then the request is free and the algorithm sets $c_v[b_i]$ to k, while all counters whose values were strictly larger than $c_v[b_i]$

before the request are decremented by 1. If v does not have b_i, then b_i is fetched into v from the node u holding b_i, and a number of counters are changed. In node v, $c_v[b_i]$ is set to k and all positive counters are decremented by 1. In node u, $c_u[b_i]$ is reset to 0 and all positive counters whose values were smaller than $c_u[b_i]$ before the request are incremented by 1. In particular, when v is full, a page b_j such that $b_j \in v$ and $c_v[b_j] = 0$ is chosen arbitrarily and is swapped out to u. Such a page b_j can always be found after the counter manipulation.

We mention a simple fact that we will use in the proof of Theorem 8. When a node v has l positive counters, these counters take distinct values in $[k-l+1, k]$.

Theorem 8. *DLRU is $2k$-competitive.*

Proof. We assume $B = kn$. The analysis can be extended to $B < kn$ with only small changes. Let S^v_{opt} be the set of pages stored at v in OPT. We define

$$\Phi = \sum_{v \in V} \sum_{b \in S^v_{opt}} 2(k - c_v[b])$$

as our non-negative potential function. It suffices to prove that, for an arbitrary request sequence σ, $\Delta C_{DL} + \Delta\Phi \leq 2k\Delta C_{OPT}$, for all events contained in the simultaneous run of DLRU and OPT on σ. Here ΔC_{DL} denotes the cost incurred by DLRU during the event. We assume w.l.o.g. that when there is a request, first OPT transfers pages to serve the request and then DLRU starts satisfying it. So when DLRU is serving, the requested page belongs to S^v_{opt}. We have to consider two types of events: (Type I) OPT swaps two pages; (Type II) DLRU satisfies the request. Due to space limitations we prove $\Delta C_{DL} + \Delta\Phi \leq 2k\Delta C_{OPT}$ only for (Type II). Suppose that there is a request to page b_i at node v.

Case 1: DLRU already has b_i at node v.
In this case $\Delta C_{DL} = \Delta C_{OPT} = 0$ and $c_v[b_i]$ is augmented from some non-negative integer $l (\leq k)$ to k. In addition, at most $k - l$ counters in v decrease their values by 1. Since $b_i \in S^v_{opt}$, the change of Φ is smaller than $-2(k-l) + (k-l) \cdot 2 = 0$. Thus we obtain $\Delta C_{DL} + \Delta\Phi \leq 0 + 0 = 0 = 2k\Delta C_{OPT}$.

Case 2: DLRU does not have b_i at node v yet.
Again $\Delta C_{OPT} = 0$. $\Delta C_{DL} = 2$ because DLRU loads b_i into v's local memory, which requires one swap. Let u be the node that stored b_i before the request and let b_j be the page brought from v to u to make room at v for b_i. In v, $c_v[b_i]$ is set from 0 to k, and in the worst case k positive counters are decremented. Since $b_i \in S^v_{opt}$, at least one of the decreased k counters is not in S^v_{opt}, and the change of Φ with respect to v is less than $-2k + (k-1) \cdot 2 = -2$. In node u, $c_u[b_i]$ is reset to 0 and several counters may be incremented. The change of Φ corresponding to u is less than or equal to 0, because the counter increments lower Φ and $b_i \notin S^u_{opt}$. The total change of Φ is the sum of the change at u and v. Hence $\Delta\Phi \leq -2 + 0 = -2$ and $\Delta C_{DL} + \Delta\Phi \leq 2 + (-2) = 0 = 2k\Delta C_{OPT}$. □

Finally, we investigate randomized distributed paging. For uni-processor paging, a well-known randomized on-line algorithm called Marking attains $(2 \log k)$-competitiveness against the oblivious adversary [7]. We can generalize Marking to the migration version of the distributed paging problem.

Algorithm VMARK: The algorithm is defined for each node v separately. Each of the k blocks in node v has a marker bit and a page field associated with it. The marker bit and the page field are called the *attribute* of a block. The page field is used to specify the name of a page; the page stored in a block can be different from that specified in the page field, though. Roughly speaking, a page field memorizes the page which would occupy the corresponding block if there were no requests at any nodes except v. The algorithm works in a series of phases. Like Marking, at the beginning of every phase, all marker bits are reset to 0. As the phase proceeds, the number of marker bits that take the value 1 monotonically increases. After all bits have been marked, the phase is over at the next request to an item not contained in the set of pages written on the k page fields in v. Marker bits and page fields can be modified only if there is a request at v or a page is swapped out to v from other nodes. The details of the algorithm are given in the program style. At a page collision, VMARK moves the evicted page to the block that the incoming page occupied before.

```
type block = record
                mark : 0 or 1
                page : name of the page
             end
```

Procedure Fetchblock /* there is a request at v to b_i */
if b_i belongs to v's local memory **then**
    let BL_i be the block holding b_i.
    if $BL_i.page = b_i$ **then** set $BL_i.mark$ to 1 and exit.
    else choose randomly one block BL_j s.t. $BL_j.mark = 0$.
        copy BL_i's attribute to BL_j's attribute.
        $BL_i.mark \leftarrow 1$. $BL_i.page \leftarrow b_i$.
else /* b_i does not belong to v's local memory */
    if there is a block BL s.t. $BL.page = b_i$ **then**
        swap out a page from BL if BL is not empty. /* page collision */
        fetch b_i to BL. $BL.mark \leftarrow 1$.
    else choose randomly one block BL' s.t. $BL'.mark = 0$.
        swap out a page from BL' if BL' is not empty. /* page collision */
        fetch b_i to BL'. $BL'.mark \leftarrow 1$. $BL'.page \leftarrow b_i$.

Procedure Dropped /* page b_i stored at v is fetched by node u
and b_j is brought into v instead because of a page collision at u */
let BL be the block that b_i occupied before leaving v.
bring b_j to BL.
if there is a block BL' s.t. $BL'.page = b_j$ **then**
    exchange the attributes of BL and BL'.

The program is composed of two procedures, Fetchpage and Dropped. Fetchpage explains the action when there is a request at v. Dropped is called when a page is discarded into v because of a page collision in another node u. Note that if requests are generated at only one node v, the algorithm performs in exactly the same way as Marking. VMARK preserves the following crucial properties. (1) During a phase, exactly k different pages are requested at v. (2) There never exist two blocks BL_1 and BL_2 in a node v so that BL_1 stores a page b and at the same time b is specified in the page field of BL_2. (3) If the page stored in block BL

at v is different from the page b memorized in the page field of BL, then b left v because some other node generated a request to b.

Theorem 9. *VMARK is* $(8 \log k)$*-competitive against the oblivious adversary.*

Proof (Sketch). Let $C_{VM}(\sigma)$ be the cost incurred by VMARK. We analyze the cost by dividing $C_{VM}(\sigma)$ and $C_{OPT}(\sigma)$ among all nodes. Then, for each node v, we compare VMARK's and OPT's cost. The cost $C_{VM}(\sigma)$ is divided as follows. Suppose that there is a request at node v to page b and that VMARK does not have b in v's local memory. Then we charge a cost of 2 to v, even if v is empty and the actual cost would only be 1. This can only overestimate $C_{VM}(\sigma)$. As for OPT, whenever OPT moves a page from u to v, we assign a cost of $\frac{1}{2}$ to both v and u. Using this cost-assignment, when we pay attention to a particular node v, the influence from other nodes (e.g. other nodes generate requests to pages stored in v or drop pages to v as the result of a page collision) does not increase the cost ratio of VMARK to OPT on v. Thus we can assume that requests only occur at node v and can apply the analysis for Marking [7] to v. □

References

1. B. Awerbuch, Y. Bartal and A. Fiat. Competitive distributed file allocation. In *Proc. 25th Annual ACM Symposium on Theory of Computing*, pages 164-173, 1993.
2. B. Awerbuch, Y. Bartal and A. Fiat. Heat & Dump: Competitive Distributed Paging. In *Proc. 34th Annual IEEE Symposium on Foundations of Computer Science*, pages 22-32, 1993.
3. S. Albers and H. Koga. New on-line algorithms for the page replication problem. In *Proc. 4th Scandinavian Workshop on Algorithm Theory*, pages 25-36, 1994.
4. S. Ben-David, A. Borodin, R.M. Karp, G. Tardos and A. Wigderson. On the power of randomization in on-line algorithms. *Algorithmica*, **11**:2-14,1994.
5. Y. Bartal, A. Fiat and Y. Rabani. Competitive algorithms for distributed data management. In *Proc. 24th Annual ACM Symposium on Theory of Computing*, pages 39-50, 1992.
6. D.L. Black and D.D. Sleator. Competitive algorithms for replication and migration problems. Technical Report Carnegie Mellon University, CMU-CS-89-201, 1989.
7. A. Fiat, R.M. Karp, M. Luby, L.A. McGeoch, D.D. Sleator and N.E. Young. Competitive paging algorithm. *Journal of Algorithm*, **12**:685-699, 1991.
8. H. Koga. Randomized on-line algorithms for the page replication problem. In *Proc. 4th International Annual Symposium on Algorithms and Computation*, pages 436-445, 1993.
9. A.R. Karlin, M.S. Manasse, L. Rudolph and D.D. Sleator. Competitive snoopy caching. *Algorithmica*, **3**:79–119, 1988.
10. C. Lund, N. Reingold, J. Westbrook and D. Yan. On-line distributed data management. In *Proc. 2nd Annual European Symposium on Algorithms*, pages 202-214, 1994.
11. D.D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *Communication of the ACM*, **28**:202-208, 1985.
12. J. Westbrook. Randomized Algorithms for the multiprocessor page migration. In *Proc. of the DIMACS Workshop on On-Line Algorithms*, pages 135-149, 1992.