

# Improved Algorithms for Dynamic Page Migration<sup>\*</sup>

Marcin Bienkowski<sup>1</sup>, Mirosław Dynia<sup>2</sup>, and Mirosław Korzeniowski<sup>1</sup>

<sup>1</sup> International Graduate School of Dynamic Intelligent Systems,  
University of Paderborn, Germany. {young,rudy}@upb.de

<sup>2</sup> DFG Graduate College “Automatic Configuration in Open Systems”,  
University of Paderborn, Germany. mdynia@upb.de

**Abstract.** The dynamic page migration problem [4] is defined in a distributed network of  $n$  mobile nodes sharing one indivisible memory page of size  $D$ . During runtime, the nodes can both access a unit of data from the page and move with a constant speed, thus changing the costs of communication. The problem is to compute *online* a schedule of page movements to minimize the total communication cost.

In this paper we construct and analyze the first deterministic algorithm for this problem. We prove that it achieves an (up to a constant factor) optimal competitive ratio  $\mathcal{O}(n \cdot \sqrt{D})$ . We show that the randomization of this algorithm improves this ratio to  $\mathcal{O}(\sqrt{D} \cdot \log n)$  (against an oblivious adversary). This substantially improves an  $\mathcal{O}(n \cdot \sqrt{D})$  upper bound from [4]. We also give an almost matching lower bound of  $\Omega(\sqrt{D} \cdot \sqrt{\log n})$  for this problem.

## 1 Introduction

The page migration problem [1, 2, 5, 7, 10] arises in a distributed network of processors which share some global data. Shared variables or memory pages are stored at the local memory of these processors. If a processor wants to access (read or write) a single unit of data from a page, and the page is not stored in its local memory, it has to send a request to the processor holding the page, and appropriate data is sent back. Such transactions incur a cost which is proportional to the distance between these two processors. To avoid the problem of maintaining consistency among multiple copies of the page, the model allows only one copy of the page to be stored in the network. However, to reduce the communication cost, the system can migrate the page between processors. The migration cost is proportional to the cost of sending one unit of data times the size of the memory page. The problem is to decide, online, when and where to

---

<sup>\*</sup> Full version of this paper is available under <http://www.hni.upb.de/publikationen/>. Partially supported by DFG-Sonderforschungsbereich 376 “Massive Parallelität: Algorithmen Entwurfsmethoden Anwendungen” and by the Future and Emerging Technologies programme of the EU under EU Contract 001907 DELIS “Dynamically Evolving, Large Scale Information Systems”.

move the page to minimize the total cost of communication over all sequences of requests. The performance of the online algorithm is measured by competitive analysis [9, 6], i.e. by comparing its total cost to the total cost of the optimal offline algorithm on the same input sequence.

The *dynamic page migration* (DPM) problem introduced by Bienkowski, Korzeniowski and Meyer auf der Heide [4] is an extension to this model where the adversary can change the positions of the nodes during the runtime of the algorithm. This is typical in mobile networks, and also models the dynamics of networks that are not exclusively dedicated to the page migration problem. Obviously, the movement changes the distances and corresponding costs of sending data between the processors. The DPM model imposes a *basic restriction* on the adversary, i.e. the speed of the network changes has to be bounded. In each round a new position of each node has to be chosen within a ball of a constant diameter centered at its previous position. Whereas in the standard models of page migration [5, 10] the cost is equal to the distance between two processors, in DPM the cost of accessing one unit of data is defined as the distance between the requesting node and the node holding the page plus a constant overhead for communication. The reason for this overhead is twofold. First, this overhead represents the fact that there are no zero-cost communications, even if nodes are very close. Second, without the overhead in the definition of cost, the problem becomes infeasible, i.e. no algorithm could achieve a finite competitive ratio.

**The Model.** Following [4], we define the DPM problem as follows. The network is modelled as a set of  $n$  nodes (processors) labelled  $v_0, v_1, \dots, v_{n-1}$  placed in a metric space  $(\mathcal{X}, d)$ . The distances between the nodes are given by the metric  $d$ , but we extend the notion of the distance between two nodes in the following way. If  $v_i$  and  $v_j$  are the same node, which we denote by  $v_i \equiv v_j$ , then we denote the distance between them by  $0_{\mathbb{E}}$ . Note that this is different from the case where they just occupy the same point in the space, in which case we write  $v_i = v_j$  and  $d(v_i, v_j) = 0$ .

We assume discrete time steps  $t = 1, 2, \dots$ . We denote the distance between two nodes  $v_x$  and  $v_y$  in time step  $t$  as  $d_t(v_x, v_y)$ . Since the nodes can move, this distance can change with time. A tuple  $C_t$  describing the positions of all nodes in a time step  $t$  is called a *configuration at time  $t$* .

An input consists of a *configuration sequence*  $(C_t)$  and a *request sequence*  $(\sigma_t)$ , both created by an adversary, where  $\sigma_t$  denotes the node that issues a request at time  $t$ . The basic restriction mentioned above is formalized as follows.

**Definition 1.** A  $\Delta$ -restricted adversary is allowed to choose a new position of each node within a ball of radius  $\Delta$ , centered at the previous position of this node.<sup>3</sup>

For the  $\Delta$ -restricted adversary and any node  $v_x$ , its positions  $x_t$  and  $x_{t+1}$  in two consecutive configurations  $C_t$  and  $C_{t+1}$  cannot be too far apart, i.e.  $d(x_t, x_{t+1}) \leq \Delta$ . Furthermore, by  $\lambda$  we denote a maximum distance allowed between any two nodes at any time step. If we do not impose any such restriction, then  $\lambda = \infty$ .

<sup>3</sup> All adversaries considered in the previous work [4] were 1-restricted.

Any two nodes are able to communicate directly with each other. The cost of sending a unit of data from node  $v_x$  to node  $v_y$  at time step  $t$  is defined by a cost function  $c_t(v_x, v_y)$  as follows. If  $v_x \equiv v_y$  then  $c_t(v_x, v_y) = 0$ . Otherwise  $c_t(v_x, v_y) = d_t(v_x, v_y) + 1$ . We have one shared, indivisible memory page of size  $D$ , initially stored at the node  $v_0$ . The cost of moving the whole page from  $v_x$  to  $v_y$  in time step  $t$  is equal to  $D \cdot c_t(v_x, v_y)$ .

In time step  $t \geq 1$ , first the positions of the nodes are defined by  $C_t$  and then a request is issued at the node  $v_{\sigma_t}$ . For clarity, we abuse the notation and write “node  $\sigma_t$ ” instead of “node  $v_{\sigma_t}$ ”. In this time step the algorithm has its page in some node denoted by  $P_{\text{ALG}}(t)$ . First, it has to pay  $c_t(P_{\text{ALG}}(t), \sigma_t)$  for serving the request. Then it can optionally move the page to a new position  $P'_{\text{ALG}}(t)$  paying the cost  $D \cdot c_t(P_{\text{ALG}}(t), P'_{\text{ALG}}(t))$ . Sometimes, we will abuse the notation by writing that an algorithm is at  $v_i$  or moves to  $v_j$ , meaning that the algorithm’s page is at  $v_i$  or the algorithm moves its page to  $v_j$ .

We consider only online algorithms, i.e. the ones which make decision in step  $t$  solely on the basis of the initial part of the input up to step  $t$ , i.e. on the sequence  $C_1, \sigma_1, C_2, \sigma_2, \dots, C_t, \sigma_t$ . To analyze the performance of online algorithm ALG we use competitive analysis [9, 6]. We say that a deterministic algorithm ALG is  $c$ -competitive if for all input sequences  $(\sigma_t, C_t)$  it holds  $C_{\text{ALG}}((\sigma_t, C_t)) \leq c \cdot C_{\text{OPT}}((\sigma_t, C_t))$ .<sup>4</sup>  $C_{\text{ALG}}((\sigma_t, C_t))$  and  $C_{\text{OPT}}((\sigma_t, C_t))$  are the cost of ALG and the *optimal offline* algorithm, respectively, run on the input sequence  $(\sigma_t, C_t)$ . The factor  $c$  is called the *competitive ratio* of the algorithm. For a randomized algorithm to be  $c$ -competitive, we require that its *expected* cost is not greater than  $c$  times the cost of the optimum. The expected value is taken over all possible random choices of the algorithm. In this paper we consider only oblivious adversaries [3], which have no access to the random bits used by the algorithm.

**Related Work.** For the page migration problem in a static network, Westbrook [10] gave the first randomized  $\mathcal{O}(1)$ -competitive algorithms against oblivious and adaptive adversaries. This result was improved for some network topologies like trees or uniform graphs by Chrobak et al. [7]. The first constant competitive deterministic algorithm was Move-To-Min given by Awerbuch, Bartal and Fiat in [1] and the competitive ratio was subsequently improved by Bartal, Charikar and Indyk [2].

Bienkowski, Korzeniowski and Meyer auf der Heide [4] defined the DPM model. Their randomized algorithm  $\text{ALG}_{\text{DIST}}$  achieved a competitive ratio of  $\mathcal{O}(\min\{n \cdot \sqrt{D}, D, \lambda\})$  against an adaptive-online adversary. This ratio was proven to be up to a constant factor optimal. They also gave two trivial deterministic algorithms achieving competitive ratios of  $\mathcal{O}(D)$  and  $\mathcal{O}(\lambda)$ , respectively. Finally, they proved that the competitive ratio of any randomized algorithm against an oblivious adversary is at least  $\Omega(\min\{\sqrt{D}, \lambda\})$ .

The deterministic and randomized algorithms presented in this paper use the marking technique, which bears a resemblance to the Least Recently Used

---

<sup>4</sup> In literature, this notion is sometimes referred to as *strict competitiveness*.

(LRU) paging algorithm by Sleator and Tarjan [9] and randomized MARK paging algorithm by Fiat et al. [8], respectively.

**Contribution of the Paper.** In this paper we improve and extend the results of [4]. In Sect. 3 we give a first deterministic algorithm, MARK, for the DPM problem. Our algorithm achieves the competitive ratio of  $\mathcal{O}(n \cdot \sqrt{D})$ . It can be combined with two trivial algorithms from [4], yielding an  $\mathcal{O}(\min\{n \cdot \sqrt{D}, D, \lambda\})$ -competitive algorithm. The constants hidden in  $\mathcal{O}$  notation are moderate; in fact they are much better than the constants from the proof of competitiveness of  $\text{ALG}_{\text{DIST}}$  in [4].

In Sect. 4 we randomize MARK and get an algorithm R-MARK which is  $\mathcal{O}(\sqrt{D} \cdot \log n)$ -competitive against an oblivious adversary. Again, it can be combined with  $\mathcal{O}(D)$  and  $\mathcal{O}(\lambda)$ -competitive algorithms, resulting in an  $\mathcal{O}(\min\{\sqrt{D} \cdot \log n, D, \lambda\})$ -competitive algorithm. In Sect. 5 we prove that this ratio is almost optimal by showing that the lower bound on competitiveness of any randomized algorithm against an oblivious adversary is  $\Omega(\min\{\sqrt{D \cdot \log n}, D^{2/3}, \lambda\})$ .

## 2 Preliminaries

In this paper we consider only  $\frac{1}{2}$ -restricted adversaries. This assures that the distance between any two nodes can change only by 1 per time step. The presented proofs can be extended to any constant-restricted adversary with the Reduction Lemma proven in the full version of the paper.

**Lemma 1 (Reduction Lemma).** *Assume that there exists a  $k$ -competitive (possibly randomized) algorithm  $\text{ALG}_A$  against an  $A$ -restricted adversary. Then  $\text{ALG}_A$  is  $k$ -competitive against a  $B$ -restricted adversary for  $B \leq A$ . Additionally, for any  $B \geq A$  there exists a (randomized) algorithm  $\text{ALG}_B$  which is  $\frac{B}{A} \cdot k$ -competitive against a  $B$ -restricted adversary.*

Throughout this paper we will use the following notation.  $\text{OPT}$  denotes the optimal offline algorithm. If  $\text{ALG}$  is any algorithm and  $\mathcal{S}$  any sequence of consecutive time steps, then by  $C_{\text{ALG}}(\mathcal{S})$  we denote the cost of the algorithm  $\text{ALG}$  on  $\mathcal{S}$ . In particular,  $C_{\text{OPT}}(\mathcal{S})$  is the cost of the optimal algorithm on sequence  $\mathcal{S}$ . By  $C_{\text{ALG}}^{\text{req}}(\mathcal{S})$  we denote the cost of serving requests by  $\text{ALG}$ , i.e. not counting the cost of page movements in  $\mathcal{S}$ .

In the proof of competitiveness of any algorithm  $\text{ALG}$  we will follow the following scheme. We take any input sequence  $(C_t, \sigma_t)$ , and we run  $\text{ALG}$  and  $\text{OPT}$  “in parallel” on this sequence. By  $P_{\text{ALG}}(t)$  and  $P_{\text{OPT}}(t)$  we denote the node holding  $\text{ALG}$ ’s and  $\text{OPT}$ ’s pages, respectively, in the  $t$ -th time step. At each step we will be able to compare the cost paid so far, respectively by  $\text{ALG}$  and by  $\text{OPT}$ .

**Jump Sets.** Before we construct and analyze our algorithms for DPM problem, we prove a useful property of the costs incurred during the runtime. We assume that  $\sqrt{D}$  is an integer, and we consider any sequence  $I$  of  $K = \sqrt{D}$  steps, numbered from 1 to  $K$ . We call sequences of such lengths *intervals*. Let  $\sigma_i$  be

the node which issues a request in the  $i$ -th step of  $I$  and  $d_i(\cdot)$ ,  $c_i(\cdot)$  be the distance and cost functions in the  $i$ -th step.

**Definition 2.** A gravity center for  $I$  is a vertex  $v$  which minimizes the sum  $\sum_{i=1}^K c_K(v, \sigma_i)$ . If there is more than one such vertex, then the gravity center is the one labelled with the smallest index.<sup>5</sup>

**Definition 3.** Let  $I$  be any interval (of length  $K$ ) and let  $v_{\min}$  be its gravity center. Then a jump set  $\mathcal{G}(I)$  consists of all the nodes whose  $K$ -th step distance to  $v_{\min}$  is not greater than  $9 \cdot \sqrt{D}$ , i.e.  $\mathcal{G}(I) := \{v \in V : d_K(v, v_{\min}) \leq 9 \cdot \sqrt{D}\}$ .

Intuitively, the gravity center and the jump set try to approximate a “good” position to place a page in the last  $I$  steps. This intuition is formalized in the following lemma, whose proof can be found in the full version of the paper.

**Lemma 2.** Consider any interval  $I$  of  $K$  requests and corresponding jump set  $\mathcal{G}(I)$  with gravity center at  $v_{\min}$ . Consider any algorithm ALG which at the end of the  $K$ -th step is in a node  $P_{\text{ALG}} \notin \mathcal{G}(I)$ . Then  $C_{\text{ALG}}(I) \geq D/4$ .

### 3 A Deterministic Algorithm

In this section we construct and analyze a deterministic algorithm MARK. MARK divides the requests into intervals of length  $K$ . In each interval MARK remains at one node and serves all the requests issued. Then, at the end of the interval, it makes its decision, whether and where to move the page.

Intervals are grouped in epochs; the first epoch starting with the beginning of the input sequence. MARK begins each epoch without any nodes marked. Subsequently, for each node  $v$  we measure the cost incurred by requests issued so far in the current epoch  $\mathcal{E}$ , on the algorithm which remains in  $v$  and never moves. We denote this amount by  $C_v^{\text{req}}(\mathcal{E})$  and when it reaches  $D/4$ , then node  $v$  becomes marked.

MARK waits in one node, denoted  $P_{\text{MARK}}$  for one or more intervals, and moves at the end of an interval in which  $P_{\text{MARK}}$  becomes marked. When MARK wants to move, it computes a jump set  $\mathcal{G}$  on the basis of the last interval and moves to  $v^*$ , an arbitrarily chosen node from the not yet marked nodes of  $\mathcal{G}$ . If there is no unmarked node in  $\mathcal{G}$ , then prior to choosing  $v^*$  the algorithm erases the marks on all the nodes and a new epoch begins.

The pseudocode of MARK is presented in Fig. 1. The algorithm is initialized by setting  $\mathcal{E} = \emptyset$  and *unmarking* all the nodes.

**Theorem 1.** MARK achieves competitive ratio of  $\mathcal{O}(n \cdot \sqrt{D})$  for DPM problem.

*Proof.* To make the proof concise, we introduce a notion of a *phase*. A phase is a sequence of consecutive intervals in which MARK remains at one node. In other words MARK’s page movement divides each epoch into a sequence of phases.

<sup>5</sup> We could apply any tie breaking method here.

```

Serve requests in interval  $I$ 
 $\mathcal{E} := \mathcal{E} \cup I$ 
for each  $v \in V$ 
  if  $C_v^{\text{req}}(\mathcal{E}) \geq D/4$  then mark  $v$ 
if  $P_{\text{MARK}}$  is marked then
   $\mathcal{G} := \mathcal{G}(I)$ 
  if all nodes in  $\mathcal{G}$  are marked then
    unmark all nodes from  $V$ 
     $\mathcal{E} = \emptyset$ 
  Choose any not marked node  $v^*$  from  $\mathcal{G}$ 
  Move the page to  $v^*$ 

```

**Fig. 1.** Algorithm MARK for one interval  $I$

Lemma 2 implies that after any phase  $\mathcal{P} = (I_1, \dots, I_p)$  for all the nodes  $v$  outside  $\mathcal{G}(I_p)$  holds  $C_v^{\text{req}}(\mathcal{P}) \geq C_v^{\text{req}}(I_p) \geq D/4$ . Thus, these nodes are already marked, and, in fact, at the end of  $\mathcal{P}$  the algorithm chooses an arbitrary node  $v^*$  from not yet marked nodes.

As an immediate consequence we get the following two properties. First, each epoch ends exactly when all the nodes are marked. Therefore, although the division into phases depends on the choices of the algorithm, the division into epochs depends only on the input sequence. Second, since each epoch begins with no node marked, and in each phase at least one node becomes marked, each epoch consists of at most  $n$  phases.

For the proof we use amortized analysis and define a potential function  $\Phi_t$  in time step  $t$ .  $\Phi_t = 2 \cdot D \cdot L_t$ , where  $L_t$  is the  $t$ -th step distance between  $P_{\text{MARK}}$  and  $P_{\text{OPT}}$ .

If  $\mathcal{S}$  is some sequence of  $\ell$  steps, numbered from 1 to  $\ell$ , then by  $\Phi_0$  we denote the potential just before this sequence and we define  $\Delta\Phi(\mathcal{S})$  as the difference between the potential after and before phase  $\mathcal{S}$ , i.e.  $\Delta\Phi(\mathcal{S}) := \Phi_\ell - \Phi_0$ . Since in the beginning of the runtime  $\Phi_0 = 0$  and  $\Phi_t$  is non negative in every time step  $t$ , it is sufficient to prove that for any input sequence  $\mathcal{S}$ , it holds  $C_{\text{MARK}}(\mathcal{S}) + \Delta\Phi(\mathcal{S}) \leq \mathcal{O}(n \cdot \sqrt{D}) \cdot C_{\text{OPT}}(\mathcal{S})$ .

First, we prove a bound on the cost of MARK in one phase.

**Lemma 3.** *In each phase  $\mathcal{P}$  of the algorithm MARK, the amortized cost of the algorithm  $C_{\text{MARK}}(\mathcal{P}) + \Delta\Phi(\mathcal{P})$  is not greater than  $\mathcal{O}(\sqrt{D}) \cdot C_{\text{OPT}}(\mathcal{P}) + \mathcal{O}(D \cdot \sqrt{D})$ .*

*Proof.* In this proof we often use the triangle inequality, the inequality  $c_t(x, y) \leq 1 + d_t(x, y)$ , and the fact that the adversary is 1/2-restricted and therefore the distance between two nodes can change by at most  $K$  in one interval.

We assume that  $\mathcal{P}$  consists of  $p$  intervals  $I_1, I_2, \dots, I_p$  and that the algorithm has its page in  $P_{\text{MARK}}$  throughout the whole phase  $\mathcal{P}$ . First, we bound the amortized cost in the intervals  $(I_1, I_2, \dots, I_{p-1})$ . The algorithm does not move within

these intervals, thus the total cost of serving requests is at most  $D/4$ , because otherwise  $P_{\text{MARK}}$  would become marked and the phase would be finished earlier. Therefore,  $C_{\text{MARK}}(I_1, \dots, I_{p-1}) = C_{\text{MARK}}^{\text{req}}(I_1, \dots, I_{p-1}) < D/4$ . For bounding the change of the potential in these intervals we use the following lemma, which is proven in the full version of the paper.

**Lemma 4.** *If the algorithm MARK remains in the same node for a sequence of requests  $\mathcal{S}$  and  $C_{\text{ALG}}^{\text{req}}(\mathcal{S}) \leq D$ , then  $\Delta\Phi(\mathcal{S}) \leq \mathcal{O}(\sqrt{D}) \cdot C_{\text{OPT}}(\mathcal{S}) + \mathcal{O}(D \cdot \sqrt{D})$*

In consequence,  $\Delta\Phi(I_1, \dots, I_{p-1}) \leq \mathcal{O}(\sqrt{D}) \cdot C_{\text{OPT}}(I_1, \dots, I_{p-1}) + \mathcal{O}(D \cdot \sqrt{D})$  and it is sufficient to bound the amortized cost in the interval  $I_p$ .

In the remaining part of the proof of Lemma 3 we use some ideas from the the proof of 7-competitiveness of Move-To-Min algorithm [1]. However, we have to take into account the movement of the nodes and shorter interval lengths.

We number the time steps within  $I_p$  from 1 to  $K$ . Let  $v^*$  be the new node chosen after phase  $\mathcal{P}$ . We have  $c_K(P_{\text{MARK}}, v^*) \leq 1 + d_K(P_{\text{MARK}}, v_{\min}) + d_K(v_{\min}, v^*)$ . Since  $v^* \in \mathcal{G}(I_p)$ ,  $d_K(v_{\min}, v^*) \leq 9\sqrt{D}$ , and therefore  $C_{\text{MARK}}(I_p) = C_{\text{MARK}}^{\text{req}}(I_p) + D \cdot c_K(P_{\text{MARK}}, v^*) \leq C_{\text{MARK}}^{\text{req}}(I_p) + D \cdot d_K(P_{\text{MARK}}, v_{\min}) + \mathcal{O}(D \cdot \sqrt{D})$ .

By  $a_{i-1}$  and  $a_i$  we denote the position of OPT, respectively at the beginning and at the end of the  $i$ -th step. Therefore, the optimal algorithm's cost in interval  $I_p$  is equal to  $C_{\text{OPT}}(I_p) = \sum_{i=1}^K (c_i(a_{i-1}, \sigma_i) + D \cdot c_i(a_{i-1}, a_i))$ . We need two technical lemmas which are proven in the full version of the paper.

**Lemma 5.** *Let  $X_t = \sum_{i=1}^K d_i(a_t, \sigma_i)$  be the cost of serving all the requests from node  $a_t$ . Then for all  $0 \leq t \leq K$  it holds  $X_t \leq C_{\text{OPT}}(I_p) + D$ .*

**Lemma 6.** *Let  $Y_t = D \cdot d_K(a_t, v_{\min})$ . Then for all  $0 \leq t \leq K$  it holds  $Y_t \leq \mathcal{O}(\sqrt{D}) \cdot C_{\text{OPT}}(I_p) + \mathcal{O}(D \cdot \sqrt{D})$ .*

We need to bound the amortized cost of MARK in the interval  $I_p$ ; this cost is equal to  $C_{\text{MARK}}^{\text{req}}(I_p) + D \cdot d_K(P_{\text{MARK}}, v_{\min}) + \mathcal{O}(D \cdot \sqrt{D}) + \Phi_K - \Phi_0$ . The summands can be bounded as described below. The same bounds can be applied to any sequence of at most  $K$  steps.

$$\begin{aligned} C_{\text{MARK}}^{\text{req}}(I_p) &= \sum_{i=1}^K c_i(P_{\text{MARK}}, \sigma_i) \leq \sum_{i=1}^K (1 + d_i(P_{\text{MARK}}, a_0) + d_i(a_0, \sigma_i)) \\ &\leq K + \sum_{i=1}^K (K + d_0(P_{\text{ALG}}, a_0)) + X_0 \\ &\leq \Phi_0/2 + C_{\text{OPT}}(I_p) + \mathcal{O}(D) \end{aligned} \tag{1}$$

$$\begin{aligned} \Phi_K + D \cdot d_K(P_{\text{MARK}}, v_{\min}) &\leq 2 \cdot D \cdot d_K(v^*, a_K) + D \cdot d_K(P_{\text{MARK}}, a_0) + D \cdot d_K(a_0, v_{\min}) \\ &\leq 2 \cdot Y_K + \mathcal{O}(D \cdot \sqrt{D}) + D \cdot (d_0(P_{\text{MARK}}, a_0) + K) + Y_0 \\ &\leq \mathcal{O}(\sqrt{D}) \cdot C_{\text{OPT}}(I_p) + \Phi_0/2 + \mathcal{O}(D \cdot \sqrt{D}) \end{aligned} \tag{2}$$

Thus, the amortized cost in the interval  $I_p$  is bounded by  $\mathcal{O}(\sqrt{D}) \cdot C_{\text{OPT}}(I_p) + \mathcal{O}(D \cdot \sqrt{D})$ , which finishes the proof of Lemma 3.  $\square$

Closely observing the proof of Lemma 3, we can show that the cost of serving requests in a phase which is not yet finished, can be either paid from the potential or is amortized against the cost of the optimal algorithm in this phase.

**Lemma 7.** *Let  $\mathcal{P}'$  be the first  $\ell$  steps of phase  $\mathcal{P}$  of algorithm MARK. Then  $C_{\text{MARK}}(\mathcal{P}') \leq \Phi_B + \mathcal{O}(\sqrt{D}) \cdot C_{\text{OPT}}(\mathcal{P}') + \mathcal{O}(D \cdot \sqrt{D})$ , where  $\Phi_B$  is the potential at the beginning of the phase  $\mathcal{P}$ .*

*Proof.* We divide the phase into  $p$  intervals, i.e.  $\mathcal{P}' = (I_1, I_2, \dots, I_p)$ , where the last interval possibly has less than  $K$  steps. Let  $\Phi_0$  be the potential at the beginning of  $I_p$ . From the proof of Lemma 3 follows that  $C_{\text{MARK}}(I_1, \dots, I_{p-1}) + \Phi_0 \leq \Phi_B + \mathcal{O}(\sqrt{D}) \cdot C_{\text{OPT}}(I_1, \dots, I_{p-1}) + \mathcal{O}(D \cdot \sqrt{D})$ . Therefore, it is sufficient to show that  $C_{\text{MARK}}(I_p) \leq \Phi_0 + C_{\text{OPT}}(I_p) + \mathcal{O}(D \cdot \sqrt{D})$ . This follows from (1) in the proof of Lemma 3.  $\square$

As shown in Lemmas 3 and 7 we would be able to prove good bounds for each phase if we could neglect the additive term of  $\mathcal{O}(D \cdot \sqrt{D})$ . Thus, we have to find some additional lower bounds on the cost of the optimal algorithm. We can do this by using a property of our marking scheme.

**Lemma 8.** *The cost of OPT in any finished  $\mathcal{E}$  is at least  $\Omega(D)$ . Moreover, OPT's cost in the first epoch  $\mathcal{E}_1$  (even if it is unfinished) is either  $C_{\text{MARK}}(\mathcal{E}_1)$  or at least  $\Omega(D)$ .*

*Proof.* First, we consider a finished epoch  $\mathcal{E}$ . If OPT moves its page within  $\mathcal{E}$ , then its cost is at least  $D$ . Otherwise, OPT remains at one node  $P_{\text{OPT}}$ , which becomes marked at some point of the epoch. Therefore, it must have paid a cost of at least  $C_{P_{\text{OPT}}}^{\text{req}} \geq D/4$  in this epoch.

Second, we consider  $\mathcal{E}_1$ . If OPT moves within  $\mathcal{E}_1$ , then it pays at least  $D$ . Otherwise, it remains in  $v_0$  for the whole  $\mathcal{E}_1$  and we have two cases. If  $\mathcal{E}_1$  consists of one unfinished phase only, then  $C_{\text{OPT}}(\mathcal{E}_1) = C_{v_0}^{\text{req}}(\mathcal{E}_1) = C_{\text{MARK}}(\mathcal{E}_1)$ . Otherwise, let  $\mathcal{P}_1$  be the first (finished) phase of  $\mathcal{E}_1$ . Since  $v_0$  becomes marked at the end of  $\mathcal{P}_1$ ,  $C_{\text{OPT}}(\mathcal{E}_1) \geq C_{\text{OPT}}(\mathcal{P}_1) \geq C_{v_0}^{\text{req}} \geq D/4$ .

Therefore, for any sequence  $\mathcal{S}$  consisting of  $k$  epochs, where the last epoch is possibly unfinished, either  $C_{\text{OPT}}(\mathcal{S}) = C_{\text{MARK}}(\mathcal{S})$  or  $C_{\text{OPT}}(\mathcal{S}) \geq \Omega(k \cdot D)$ . In the latter case, we combine the result with Lemmas 3 and 7 obtaining

$$\begin{aligned} C_{\text{MARK}}(\mathcal{S}) &\leq \mathcal{O}(\sqrt{D}) \cdot C_{\text{OPT}}(\mathcal{S}) + k \cdot n \cdot \mathcal{O}(D \cdot \sqrt{D}) \\ &\leq \mathcal{O}(\sqrt{D}) \cdot C_{\text{OPT}}(\mathcal{S}) + \mathcal{O}(n \cdot \sqrt{D}) \cdot C_{\text{OPT}}(\mathcal{S}) , \end{aligned}$$

which finishes the proof of Thm. 1.  $\square$

In [4] two simple deterministic algorithms attaining competitive ratios of  $\mathcal{O}(D)$  and  $\mathcal{O}(\lambda)$ , respectively, were presented. From the analysis of their potential functions follows that it is possible to combine them with MARK in one algorithm which achieves the competitive ratio of  $\mathcal{O}(\min\{n \cdot \sqrt{D}, D, \lambda\})$ . We omit the details here.

This result is up to a constant factor optimal, since the  $\Omega(\min\{n \cdot \sqrt{D}, D, \lambda\})$  lower bound for a randomized algorithm against an adaptive-online adversary presented in [4] applies also to the deterministic case (see [3] for comparison of the power of different adversaries).

## 4 Randomization against an Oblivious Adversary

In this section we show a direct randomization of MARK which yields an algorithm R-MARK. In the formulation of the algorithm, instead of choosing  $v^*$  in deterministic fashion from not yet marked nodes of  $\mathcal{G}$  (see Fig. 1), we choose  $v^*$  uniformly at random from not yet marked nodes of  $\mathcal{G}$ .

**Theorem 2.** *R-MARK is  $\mathcal{O}(\sqrt{D} \cdot \log n)$ -competitive against an oblivious adversary.*

*Proof.* For the analysis we use the same potential function  $\Phi$  as for MARK. Since Lemmas 3 and 7 hold for *any* choice of  $v^*$  from not yet marked nodes of  $\mathcal{G}$ , they hold also in the case when  $v^*$  is chosen randomly. Therefore, for any finished phase  $\mathcal{P}$  it holds  $C_{\text{R-MARK}}(\mathcal{P}) + \Delta\Phi(\mathcal{P}) \leq \mathcal{O}(\sqrt{D}) \cdot C_{\text{OPT}}(\mathcal{P}) + \mathcal{O}(D \cdot \sqrt{D})$ . Similar result holds for an unfinished phase (see Lemma 7). However, for the algorithm R-MARK we can have a bound better than  $n$  (at least on expectation) for the number of phases in an epoch.

**Lemma 9.** *For any epoch  $\mathcal{E}$ , the expected number of phases in this epoch is  $\mathcal{O}(\log n)$ . The expectation is taken over all possible random choices of R-MARK.*

*Proof.* Analogously to the proof of Thm. 1 we can forget about the set  $\mathcal{G}$  and assume that R-MARK chooses  $v^*$  uniformly at random from all not yet marked nodes.

Let  $\{b_i\}_{i=1}^n$  be the nodes in the order they are becoming marked in epoch  $\mathcal{E}$ . Assume that in a phase  $\mathcal{P}$  the algorithm is in a node  $b_k$ . Then at the end of phase  $\mathcal{P}$  it chooses a new node  $v^*$  uniformly at random from  $n - k$  nodes i.e. from the set  $B_k^+ := \{b_i : k + 1 \leq i \leq n\}$ . Actually, it might happen that some of nodes from  $B_k^+$  were also marked at the end of phase  $\mathcal{P}$ , in which case the algorithm has even fewer than  $n - k$  nodes to choose randomly from.

At the beginning of each epoch, except from the first one, the algorithm also chooses a place uniformly at random from the whole set  $V$ . Let  $T_i$  be the expected number of phases, provided that  $i$  nodes are still unmarked. We have a recursive formula

$$T_0 = 0, \quad T_k = 1 + \sum_{i=0}^{k-1} \frac{1}{k} \cdot T_i,$$

where the second equality follows from the linearity of expected value. It can be proven by induction that  $T_i = H_i$ , the  $i$ -th harmonic number. In fact, the expected number of phases can be even smaller if some nodes get marked concurrently and R-MARK has fewer nodes to choose from.  $T_n$  is the bound on the expected number of phases in each epoch except the first one, because the very first node is not chosen randomly. However, this incurs at most one additional phase after which we have our random process. Therefore, the expected number of phases in any epoch is at most  $T_n + 1 = H_n + 1 = \mathcal{O}(\log n)$ .  $\square$

Since Lemma 8 does not depend on the way of choosing  $v^*$ , either, as long it is chosen from the not yet marked nodes, we have that for any sequence  $\mathcal{S}$  consisting of  $k$  epochs we have

$$\begin{aligned} \mathbf{E}[C_{\text{R-MARK}}(\mathcal{S})] &\leq \mathcal{O}(\sqrt{D}) \cdot C_{\text{OPT}}(\mathcal{S}) + k \cdot \mathcal{O}(\log n) \cdot \mathcal{O}(D \cdot \sqrt{D}) \\ &\leq \mathcal{O}(\sqrt{D}) \cdot C_{\text{OPT}}(\mathcal{S}) + \mathcal{O}(\log n \cdot \sqrt{D}) \cdot C_{\text{OPT}}(\mathcal{S}) . \end{aligned}$$

This finishes the proof of Thm. 2.  $\square$

It is also possible to combine R-MARK with  $\mathcal{O}(D)$  and  $\mathcal{O}(\lambda)$ -competitive algorithms obtaining an algorithm which is  $\mathcal{O}(\min\{\sqrt{D} \cdot \log n, D, \lambda\})$ -competitive against an oblivious adversary. We omit the details.

## 5 Lower Bound against an Oblivious Adversary

In this section we show that no online algorithm can achieve a competitive ratio better than  $\Omega(\min\{\sqrt{D} \cdot \log n, D^{2/3}, \lambda\})$ . This improves an  $\Omega(\min\{\sqrt{D}, \lambda\})$  lower bound result from [4]. We present two theorems which, combined, immediately give the result.

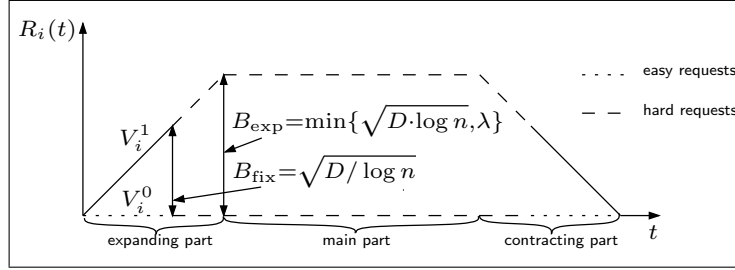
**Theorem 3.** *If  $D \geq \log^3 n$ , then no randomized algorithm can achieve better competitive ratio against an oblivious adversary than  $\Omega(\min\{\sqrt{D} \cdot \log n, \lambda\})$ .*

**Theorem 4.** *If  $D \leq \log^3 n$ , then no randomized algorithm can achieve better competitive ratio against an oblivious adversary than  $\Omega(\min\{D^{2/3}, \lambda\})$ .*

The proofs are almost identical, and therefore we present only the first one. The second one can be found in the full version of the paper.

*Proof (of Thm. 3).* We construct a probability distribution over inputs and prove that no deterministic algorithm, which knows this distribution, can have a competitive ratio better than  $\Omega(\min\{\sqrt{D} \cdot \log n, \lambda\})$ . Then one can apply Yao min-max theorem [11, 6] and Thm. 3 follows immediately.

We assume that  $n$  is a power of 2. If it is not the case, then the adversary can give requests only in the first  $2^{\lfloor \log n \rfloor}$  nodes and put the other nodes exactly in the same point of space  $\mathcal{X}$  as  $v_0$ . Then, for any algorithm ALG, that uses these additional nodes, there exists an algorithm ALG' which uses  $v_0$  instead and has



**Fig. 2.** Lower bound: The  $i$ -th phase of an epoch

cost at most as large as ALG. We can also assume that  $\lambda \geq \sqrt{D}$ . Otherwise we could use the proof from [4] and obtain the lower bound of  $\Omega(\lambda)$ .

We number all nodes from 0 to  $n - 1$ . In the following we identify the nodes with their numbers. For all  $1 \leq i \leq \log n$ , let  $V_i^0$  be the set of all the node numbers whose bit representation has  $i$ -th bit set to 0.  $V_i^1$  is the set of all remaining nodes, i.e. the ones whose bit representation has the  $i$ -th bit set to 1. Let  $B_{\text{main}} = D / \log n$ ,  $B_{\text{exp}} = \min\{\sqrt{D \cdot \log n}, \lambda\}$ , and  $B_{\text{fix}} = \sqrt{D / \log n}$ .

We divide time into epochs, each epoch containing  $\log n$  phases, each of length  $B_{\text{main}} + 2 \cdot B_{\text{exp}}$ . In the  $i$ -th phase we divide the set  $V$  into two sets  $V_i^0$  and  $V_i^1$ . All the nodes from set  $V_i^0$  occupy a certain point in space  $\mathcal{X}$ ; the same holds for nodes from  $V_i^1$ . Therefore, in step  $t$  of phase  $i$ , the distance between the sets  $V_i^0$  and  $V_i^1$  is well defined and we call it  $R_i(t)$ .

First, we describe the movement of nodes in the  $i$ -th phase. The situation is depicted in Fig. 2. Each phase consists of an *expanding part*, which lasts for  $B_{\text{exp}}$  steps, in which the sets  $V_i^0$  and  $V_i^1$  are moved apart, a *main part*, lasting for  $B_{\text{main}}$  steps, and a *contracting part* also of length  $B_{\text{exp}}$ , in which the sets  $V_i^0$  and  $V_i^1$  are brought closer. In the  $j$ -th step of the expanding part  $R_i$  is set to  $j - 1$ . In all the steps of the main part the distance between sets  $V_i^0$  and  $V_i^1$  is exactly  $B_{\text{exp}}$ . Finally, in the  $j$ -th step of the contracting part,  $R_i$  is equal to  $B_{\text{exp}} - j$ .

The first  $B_{\text{fix}}$  requests in the expanding part of a phase and the last  $B_{\text{fix}}$  requests in the contracting part are given always at the node  $v_0$ . We call these requests *easy*. The requests in the remaining steps of a phase are given in one node — with probability  $1/2$  all are given at  $v_0$  (belonging to  $V_i^0$ ), and with probability  $1/2$  all are given at node  $v_{2^i-1}$  (belonging to  $V_i^1$ ). We call these requests *hard*, and we call the set, whose node issued these requests, a *requesting set*. Note that the main part is a (usually proper) subset of a sequence containing all the hard requests.

Consider any randomly generated  $\log n$  phases of epoch  $\mathcal{E}$ . There exists a node  $v^+$  (exactly one) which is in the requesting set for all the phases. As its distance to hard requests in each phase is 0, the cost incurred is at most 1 per request, which sums up to  $\log n \cdot (B_{\text{main}} + 2 \cdot B_{\text{exp}}) = \mathcal{O}(D)$  in total. The optimal algorithm could move to  $v^+$  at the beginning of the epoch, paying  $D$ . Finally,

the cost incurred by easy requests is at most  $2 \cdot \sum_{j=1}^{B_{\text{fix}}} j = \mathcal{O}(D/\log n)$  in each phase which accounts for  $\mathcal{O}(D)$  for all requests in epoch  $\mathcal{E}$ .

On the other hand, for all  $i$ , when hard requests start in the  $i$ -th phase, any deterministic algorithm ALG has its page in the “wrong” set with probability  $1/2$ . At this point  $R_i(t) = B_{\text{fix}}$ . If ALG moves its page to the other set within the hard requests, it pays at least  $D \cdot B_{\text{fix}}$ . Otherwise, it has to pay  $B_{\text{exp}}$  for serving each of  $B_{\text{main}}$  hard requests in the main part of a phase. Therefore, from the linearity of expectation follows that the expected cost of ALG in the whole epoch  $\mathcal{E}$  is

$$\mathbf{E}[C_{\text{ALG}}(\mathcal{E})] = \log n \cdot \frac{\min\{D \cdot B_{\text{fix}}, B_{\text{exp}} \cdot B_{\text{main}}\}}{2} = \Omega(\min\{D\lambda, D\sqrt{D \log n}\}) .$$

Thus, in any epoch the competitive ratio is at least  $\Omega(\min\{\lambda, \sqrt{D \cdot \log n}\})$ . We can generate an arbitrary number of epochs and the bound on ratio still holds. Therefore, the theorem follows.  $\square$

## References

- [1] B. Awerbuch, Y. Bartal, and A. Fiat. Competitive distributed file allocation. In *Proc. of the 25th ACM Symp. on Theory of Computing (STOC)*, pages 164–173, 1993.
- [2] Y. Bartal, M. Charikar, and P. Indyk. On page migration and other relaxed task systems. In *Proc. of the 8th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 43–52, 1997.
- [3] S. Ben-David, A. Borodin, R. M. Karp, G. Tardos, and A. Wigderson. On the power of randomization in online algorithms. In *Proc. of the 22nd ACM Symp. on Theory of Computing (STOC)*, pages 379–386, 1990.
- [4] M. Bienkowski, M. Korzeniowski, and F. Meyer auf der Heide. Fighting against two adversaries: Page migration in dynamic networks. In *Proc. of the 16th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA)*, pages 64–73, 2004.
- [5] 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.
- [6] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
- [7] M. Chrobak, L. L. Larmore, N. Reingold, and J. Westbrook. Page migration algorithms using work functions. In *Proc. of the 4th International Symp. on Algorithms and Computation (ISAAC)*, pages 406–415, 1993.
- [8] A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young. Competitive paging algorithms. *Journal of Algorithms*, 12:685–699, 1991.
- [9] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.
- [10] J. Westbrook. Randomized algorithms for multiprocessor page migration. *DI-MACS Series in Discrete Mathematics and Theoretical Computer Science*, 7:135–150, 1992.
- [11] A. C.-C. Yao. Probabilistic computation: towards a uniform measure of complexity. In *Proc. of the 18th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 222–227, 1977.