

# On Page Migration and Other Relaxed Task Systems

Yair Bartal \*

Moses Charikar †

Piotr Indyk ‡

## Abstract

This paper is concerned with the *page migration* (or file migration) problem [BS89] as part of a large class of on-line problems.

The page migration problem deals with the management of pages residing in a network of processors. In the classical problem there is only one copy of each page which is accessed by different processors over time. The page is allowed to be migrated between processors. However a migration incurs higher communication cost than an access (proportionally to the page size). The problem is that of deciding when and where to migrate the page in order to lower access costs. A more general setting is the *k-page migration* where we wish to maintain  $k$  copies of the page.

The page migration problems are concerned with a dilemma common to many on-line problems: determining when is it beneficial to make configuration changes.

We deal with the *relaxed task systems* model which captures a large class of problems of this type, that can be described as the generalization of some original task system problem [BLS87]. Given a  $c$ -competitive algorithm for a task system we show how to obtain a deterministic  $O(c^2)$  and randomized  $O(c)$  competitive algorithms for the corresponding relaxed task system.

The result implies first deterministic algorithms for *k-page migration* by using  $k$ -server [MMS88] algorithms, and for *network leasing* by using generalized Steiner tree algorithms [AAB96], as well as providing solutions for natural generalizations of other problems (e.g. storage rearrangement [FMRW95]).

We further study some special cases of the  $k$ -page migration problem and get optimal deterministic algorithms. For the classical page migration problem we present a deterministic algorithm that achieves a competitive ratio of  $\sim 4.086$ , improving upon the previously best competitive ratio of 7 [ABF93a]. (The current lower bound on the problem is  $\sim 3.148$  [CLRW93].)

---

\*International Computer Science Institute, Berkeley. Research supported in part by the Rothschild Postdoctoral fellowship and by the National Science Foundation operating grants CCR-9304722 and NCR-9416101. e-mail: yairb@icsi.berkeley.edu

†Department of Computer Science, Stanford University. Research supported in part by Stanford School of Engineering Groszow Fellowship, and by NSF Award CCR-9357849, with matching funds from IBM, Mitsubishi, Schlumberger Foundation, Shell Foundation, and Xerox Corporation. e-mail: moyses@theory.stanford.edu

‡Department of Computer Science, Stanford University. Research supported by NSF Award CCR-9357849, with matching funds from IBM, Mitsubishi, Schlumberger Foundation, Shell Foundation, and Xerox Corporation. e-mail: indyk@theory.stanford.edu

# 1 Introduction

The page migration problem deals with the dynamic reallocation of pages in a network of processors in response to an on-line stream of access requests for the pages.

The page migration problem may arise as a memory management problem for a globally addressed shared memory in a multiprocessor system as well as in a distributed network of processors where files kept in different sites may be accessed by the various processors (thus the problem is sometimes referred to as *file migration*). This situation is very common as a result of the everyday growing use of the Internet and Internet-related applications as the World-Wide-Web. Moreover many of these applications are interactive or real time and therefore efficient access is crucial.

When a processor wishes to access a page it must send a request to a processor holding the page and the desired information is transmitted back. The communication cost incurred thereby is proportional to the distance between the corresponding processors. It is also possible to migrate a page from the local memory of one processor to another. Such transactions incur however a high communication cost proportional to the page size  $D$  times the distance.

In the migration problem it is assumed that only one copy of each page exists in the network. This is usually the case in most distributed systems allowing data reallocation, since such a setting eliminates the problem of maintaining copy consistency. A great deal of research has concentrated on page migration problems including [FH80, PT83, TZ84, Hac86, Sheng86, SD89]. However all existing heuristics heavily rely on the non-realistic prior knowledge of potential usage patterns of the databases (see the survey paper by Gavish and Sheng [GS90]).

Theoretical work on page migration, making no such assumptions, was initiated by Black and Sleator [BS89], comparing the cost of an on-line page migration algorithm to the cost of an optimal algorithm (known as competitive analysis [ST85]). Page migration problems have been further studied in [West91, BFR92, CLRW93, ABF93a, ABF93b, LRWY94, AK95, ABF96, Bart96].

We study this problem and the more general  $k$ -page migration where there may be  $k$  mirror-replicas of the page. The use of mirror copies of a page is very common as a partial solution for reducing communication loads for heavily accessed pages. In the  $k$ -page migration any of the copies of the page may be accessed for obtaining the desired information.

The  $k$ -page migration problem is a special case of the  $k$ -server with *excursions* problem where the excursion cost is proportional to the move cost. This is the first non-trivial case of the problem which is given a solution.

It is therefore a natural question if one can make use of algorithms for the well-studied  $k$ -server problem to produce solutions for the  $k$ -page migration problem.

A similar situation occurs in other settings. The migration problems fall in a large class of on-line problems in which a central dilemma is to decide when is it beneficial to perform an expensive configuration change. Our goal would be to reduce the problem to the simpler case when a configuration change is not expensive. A large class of such problems is that of *relaxed task systems* as defined below.

A major subclass would be that of “rent-or-buy” problems. The most obvious classical example is the *ski-rental* problem (see [Kar92] for survey) where we need to decide whether to rent or buy ski equipment without knowing ahead how many days of skiing we are going to have, while the cost of buying is  $D$  times larger than that of renting. The *file replication* [BS89] and *network leasing* [AAB96] problems have a similar flavor. In the *network leasing* problem for example we need to establish communication paths between pairs of processors. However edge links can be either leased or bought, and we need to decide when edge links should be bought. For such problems our results yield algorithms for the

“rent-or-buy” problem out of algorithms for the corresponding “buy-only” problem.

Other examples for applications are the generalizations of the *storage rearrangement* problem [FMRW95] and *distributed job scheduling* [AKP92] to the case where a configuration change is  $D$  times larger than the distance (note that in both these problems this is a natural parameter).

## 1.1 Relaxed Task Systems

In this section we provide formal definitions of relaxed task systems and description of our results.

The general theorems are formulated in the context of task systems ([BLS87]):

**Definition 1** A task system,  $\mathcal{P}$ , consists of a set of configurations (or states)  $\mathcal{C}$  and a distance function between any two configurations  $C_1, C_2 \in \mathcal{C}$ , denoted  $\text{dist}(C_1, C_2)$ . (this is the move cost between the configurations). The task system consists of a set of requests, called tasks. A task  $r$  is associated with a service cost in each configuration denoted  $\text{task}(C, r)$  (this is the task cost). An algorithm for  $\mathcal{P}$  is associated with a configuration  $C_1$ . Given a request  $r$  the algorithm may serve it by moving to configuration  $C_2$  paying a cost of  $\text{cost}(C_1, C_2, r) = \text{dist}(C_1, C_2) + \text{task}(C_2, r)$ . If the move cost function  $\text{dist}$  forms a metric space over  $\mathcal{C}$ , then the task system is called *metrical*.

Give a specific task system we define a corresponding relaxed problem:

**Definition 2** A  $D$ -relaxed task system,  $D\text{-}\mathcal{P}$ , with respect to a task system  $\mathcal{P}$  and some parameter  $D \geq 1/2$ , is the task system with cost, distance, and task functions denoted  $\text{cost}_D$ ,  $\text{dist}_D$  and  $\text{task}_D$  respectively.  $\text{dist}_D$  and  $\text{task}_D$  are defined as follows: Given  $C_1, C_2 \in \mathcal{C}$ ,  $\text{dist}_D(C_1, C_2) = D \cdot \text{dist}(C_1, C_2)$ . Given  $C \in \mathcal{C}$  and a task  $r$ ,  $\text{task}_D(C, r) = \min_{C'} \text{dist}(C, C') + \text{task}(C', r)$ .

It is also useful to consider the following type of task systems that include “buy-only” type problems:

**Definition 3** A forcing task system [MMS88] task system such that for every request  $r$  and every configuration  $C$   $\text{task}(C, r)$  is either 0 or  $\infty$ . Thus, for every request  $r$  we may associate a set of allowable configurations.

The cost of a task in the relaxed version of a forcing task system is thus just the distance to the closest allowable configuration.

The  $k$ -server, Steiner tree, and generalized Steiner tree problems can all be formulated as forcing task systems. Their corresponding relaxed task systems are the  $k$ -page migration, file replication, and network leasing problems, respectively.

In general however, as well as in the case of the problems of [FMRW95] and [AKP92] the original task system will not be necessarily a forcing task system.

Our definition of relaxed task systems is a generalization of the definition of [AAB96] who define it only in the context of forcing task systems. They also provide randomized algorithms against adaptive on-line adversaries in that case.

We have the following results for the general relaxed task systems model:

- Let  $\mathcal{P}$  be a metrical task system. Given a  $c$ -competitive deterministic algorithm for  $\mathcal{P}$  we construct a  $9c^2$ -competitive algorithm for the  $D$ -relaxed task system  $D\text{-}\mathcal{P}$ .

- Let  $\mathcal{P}$  be a metrical task system. Given a  $c$ -competitive randomized algorithm for  $\mathcal{P}$  against oblivious adversaries we construct a randomized  $3c$ -competitive algorithm for the  $D$ -relaxed task system  $D\text{-}\mathcal{P}$  against oblivious adversaries.

In some special cases our technique yields even better results as for the case of *monotonic* task systems, such as the Steiner tree and generalized Steiner tree problems.

These results are described in Section 2.

## 1.2 Page Migration

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

The classical 1-page migration problem was first proposed by Black and Sleator [BS89] who give a lower bound of 3 in every network and matching upper bounds for uniform and tree metric spaces.

Westbrook [West91] gives a randomized 3-competitive algorithm against adaptive on-line adversaries for any network, and an asymptotically  $(1 + \phi)$ -competitive randomized algorithm against oblivious adversaries, where  $\phi \approx 1.62$  is the golden ratio. Optimal  $2 + 1/(2D)$  randomized file migration algorithms for uniform networks are given in [LRWY94] and for trees in [CLRW93]. Chrobak et. al. [CLRW93] also prove a lower bound greater than 3 in some network topology, specifically  $85/27 \approx 3.148$ . Awerbuch, Bartal and Fiat [ABF93a] give the first deterministic page migration algorithm. This algorithm achieves a competitive ratio of 7 which is the best known prior to this work.

We give a deterministic page migration algorithm achieving a competitive ratio of  $\sim 4.086$ . The algorithm and analysis are described in Section 3.

As mentioned above the general theorem for task systems yields  $O(k^2)$  competitive algorithms for the  $k$ -page migration problem (by using the WFA  $k$ -server algorithm [KP94]).

We give a lower bound of  $2k + 1$  for the  $k$ -page migration problem in any network, and get an optimal algorithm for the uniform network and a nearly-optimal algorithm for trees. These are described in Section 4.

## 2 A Deterministic Algorithm for Relaxed Task Systems

Let  $\text{task}(C, r)$  be the cost of servicing request  $r$  from configuration  $C$  in  $\mathcal{P}$ . Let  $C_{\min}(C, r)$  denote any configuration  $C'$  which minimizes  $\text{dist}(C, C') + \text{task}(C', r)$ . Let  $\text{task}_D(C, r)$  be the cost if servicing request  $r$  from configuration  $C$  in  $D\text{-}\mathcal{P}$ . Then  $\text{task}_D(C, r) = \text{dist}(C, C') + \text{task}(C', r)$ , where  $C' = C_{\min}(C, r)$ .

For any deterministic algorithm  $A$ , request sequence  $\sigma$  and request  $r$ , let  $\text{cost}_A(\sigma, r)$  (or  $\text{cost}_A(r)$  when  $\sigma$  follows from the context) be the cost incurred by  $A$  while servicing  $r$  from the configuration reached by previously servicing  $\sigma$ . Also, let  $\text{cost}_A(\sigma)$  be the total cost of  $A$  on  $\sigma$ . Assuming that  $A$  is  $c$ -competitive for  $\mathcal{P}$ , we define the competitive algorithm  $D\text{-DAI}g$  for  $D\text{-}\mathcal{P}$  as follows.

**Algorithm D-DAI***g*.

Algorithm  $D\text{-DAI}g$  simulates  $2D$  copies  $A_1 \dots A_{2D}$  of  $A$ . The configuration of  $D\text{-DAI}g$  is always the

same as that of  $A_1$ . When given a new request  $r$ , the algorithm gives it one of  $A_i$ s according to the following rule:

- if there exists  $i \geq 2$  such that  $\text{cost}_{A_i}(r) \geq \frac{1}{3c} \text{cost}_{A_1}(r)$ ,  $r$  is given to  $A_i$  (i.e. the simulated configuration of  $A_i$  is updated). Then  $D$ -DAIlg services  $r$  remotely, without changing its configuration.
- otherwise,  $r$  is given to  $A_1$ . Then  $D$ -DAIlg services  $r$  and moves to the new configuration of  $A_1$ .

**Theorem 1** *Let  $\mathcal{P}$  be a metrical task system and let  $A$  be a  $c$ -competitive deterministic algorithm for  $\mathcal{P}$ . Then algorithm  $D$ -DAIlg is  $9c^2$ -competitive for the  $D$ -relaxed task system  $D$ - $\mathcal{P}$ .*

**Proof:** The proof consists of two steps. First, we show that the sum of the costs of algorithms  $A_1 \dots A_{2D}$  is within a factor  $2c$  from the optimal off-line cost of servicing the requests in  $D$ - $\mathcal{P}$ . Then we show that the cost of  $D$ -DAIlg is within a factor  $4.5c$  from the above sum. The result will follow.

**Lemma 1** *Let  $\sigma$  be a request sequence, and let  $\sigma_1 \dots \sigma_{2D}$  be (possibly empty) subsequences of  $\sigma$  such that each request from  $\sigma$  appears in exactly one  $\sigma_i$ . Also, let  $A$  be a  $c$ -competitive algorithm for  $\mathcal{P}$  and let  $\text{cost}_{Adv}(\sigma)$  be the optimal off-line cost of servicing  $\sigma$  in  $D$ - $\mathcal{P}$ . Then*

$$\sum_{i=1}^{2D} \text{cost}_A(\sigma_i) \leq 2c \cdot \text{cost}_{Adv}(\sigma)$$

**Proof:** Let  $\text{cost}_{Adv_i}(\sigma_i)$  be the optimal offline cost of servicing  $\sigma_i$  in  $\mathcal{P}$ . As  $A$  is  $c$ -competitive,  $\text{cost}_A(\sigma_i) \leq c \cdot \text{cost}_{Adv_i}(\sigma_i)$ . Hence it is sufficient to prove that  $\sum_{i=1}^{2D} \text{cost}_{Adv_i}(\sigma_i) \leq 2 \cdot \text{cost}_{Adv}(\sigma)$ . Let  $Adv$  be the optimal offline algorithm for servicing  $\sigma$  in  $D$ - $\mathcal{P}$ . We define algorithms  $Adv'_i, i = 1 \dots 2D$  such that  $Adv'_i$  services  $\sigma_i$  in  $D$ - $\mathcal{P}$ . All the algorithms  $Adv'_i$  always maintain the same configuration as  $Adv$ .

- whenever  $Adv$  changes configuration (say from  $C_1$  to  $C_2$ ), all  $Adv'_i$  change their configuration accordingly. The sum of the costs of  $Adv_i$  is equal to  $2D \cdot \text{dist}(C_1, C_2) = 2 \cdot \text{dist}'(C_1, C_2)$ , which is twice the cost of  $Adv$ ,
- when  $Adv$  services request  $r$  (say from  $C_1$ ), the  $Adv'_i$  for which  $r$  is included in  $\sigma_i$  moves from  $C_1$  to  $C' = C_{\min}(C_1, r)$ , satisfies the request and moves back to  $C_1$ . Again, the cost of  $Adv_i$  is equal to  $2 \cdot \text{dist}(C_1, C') + \text{task}(C', r)$ , which is at most twice  $\text{task}_D(C_1, r)$  i.e the cost of  $Adv$ .

Thus,  $\sum_{i=1}^{2D} \text{cost}_{Adv'_i}(\sigma_i) \leq 2 \cdot \text{cost}_{Adv}(\sigma)$ . Since the optimal offline cost  $\text{cost}_{Adv_i}(\sigma_i) \leq \text{cost}_{Adv'_i}(\sigma_i)$ , the lemma follows.  $\square$

**Lemma 2** *Let  $\sigma_i$  be a sequence of requests given to  $A_i$  while running  $D$ -DAIlg on  $\sigma$ . Then*

$$\text{cost}_{D\text{-DAIlg}}(\sigma) \leq 4.5c \sum_{i=1}^{2D} \text{cost}_{A_i}(\sigma_i)$$

**Proof:** We may split  $\text{cost}_{D\text{-DAIlg}}(\sigma)$  into  $\text{cost}_{D\text{-DAIlg}}^S(\sigma)$  (the cost of servicing requests) and  $\text{cost}_{D\text{-DAIlg}}^M(\sigma)$  (the cost of moving between configurations).

Consider the cost incurred by  $D\text{-DAIlg}$  to service a request  $r$ . If  $r$  is given to  $A_i$ , the cost of servicing  $r$  from the current configuration of  $D\text{-DAIlg}$  is at most  $3c$  times  $\text{cost}_{A_i}(r)$ . Hence, we can bound the total cost of servicing requests by  $3c \sum_{i=1}^{2D} \text{cost}_{A_i}(\sigma_i)$ .

Therefore, it is sufficient to bound  $\text{cost}_{D\text{-DAIlg}}^M(\sigma) = D \cdot \text{cost}_{A_1}(\sigma_1)$  in terms of  $\sum_{i=1}^{2D} \text{cost}_{A_i}(\sigma_i)$ . To this end, consider algorithms  $A'_i$  which simulate  $A_i$  on  $\sigma_i$ , but also service all requests from  $\sigma_1$  in the following way : whenever  $r \in \sigma_1$  appears,  $A'_i$  moves from its current configuration  $C$  to  $C' = C_{\min}(C, r)$ , services  $r$  and moves back to  $C$ , paying  $\text{cost}_{A'_i}(r) := 2 \cdot \text{dist}(C, C') + \text{task}(C', r) \leq 2 \cdot (\text{dist}(C, C') + \text{task}(C', r)) \leq 2 \cdot \text{cost}_{A_i}(r)$ . As  $r$  was given to  $A_1$ , we know that

$$\text{cost}_{A_i}(r) \leq \frac{1}{3c} \text{cost}_{A_1}(r) \quad \Rightarrow \quad \text{cost}_{A'_i}(r) \leq \frac{2}{3c} \text{cost}_{A_1}(r)$$

Hence the total cost of  $A'_i$  (denoted by  $\text{cost}_{A'_i}(\sigma_1)$ ) is bounded by

$$\text{cost}_{A_i}(\sigma_i) + \sum_{r \in \sigma_1} \text{cost}_{A'_i}(r) \leq \text{cost}_{A_i}(\sigma_i) + \frac{2}{3c} \sum_{r \in \sigma_1} \text{cost}_{A_1}(r) = \text{cost}_{A_i}(\sigma_i) + \frac{2}{3c} \text{cost}_{A_1}(\sigma_1)$$

On the other hand, the algorithm  $A_1$  is  $c$ -competitive, so  $\text{cost}_{A_1}(\sigma_1) \leq c \cdot \text{cost}_{A'_1}(\sigma_1)$ . Hence

$$\begin{aligned} \frac{1}{c} \text{cost}_{A_1}(\sigma_1) &\leq \text{cost}_{A'_1}(\sigma_1) \leq \text{cost}_{A_1}(\sigma_1) + \frac{2}{3c} \text{cost}_{A_1}(\sigma_1) \\ \Rightarrow \text{cost}_{A_1}(\sigma_1) &\leq 3c \cdot \text{cost}_{A_1}(\sigma_1) \end{aligned}$$

Now we can bound the moving cost as follows:

$$\begin{aligned} \text{cost}_{D\text{-DAIlg}}^M(\sigma) &= D \cdot \text{cost}_{A_1}(\sigma_1) \leq \frac{1}{2} \sum_{i=1}^{2D} \text{cost}_{A_i}(\sigma_i) \leq \frac{3c}{2} \sum_{i=1}^{2D} \text{cost}_{A_i}(\sigma_i) \\ \text{cost}_{D\text{-DAIlg}}(\sigma) &= \text{cost}_{D\text{-DAIlg}}^S(\sigma) + \text{cost}_{D\text{-DAIlg}}^M(\sigma) \\ &\leq (3 + 1.5)c \sum_{i=1}^{2D} \text{cost}_{A_i}(\sigma_i) = 4.5c \sum_{i=1}^{2D} \text{cost}_{A_i}(\sigma_i) \end{aligned}$$

□

Theorem 1 follows from Lemmas 1 and 2. □

## 2.1 Applications and Improvements

We apply the ideas behind the above construction to obtain algorithms for different problems. In some cases, we can use specific properties of the task system to improve the analysis.

## Monotonic Task Systems

**Definition 4** A Monotonic Task System is a forcing task system with a monotonicity property between configurations as follows. A configuration  $C$  is said to be dominated by  $C'$  if for all tasks for which  $C$  is allowable so is  $C'$ . A forcing task system is monotonic if for every pair of configurations  $C_1, C_2$  there exists a configuration  $C$  dominating both, and for every configuration  $C'_1$  dominated by  $C_1$ ,  $\text{dist}(C_1, C) \leq \text{dist}(C'_1, C_2)$ .

A better ratio of  $4c^2$  may be obtained when the underlying task system  $\mathcal{P}$  is *monotonic*. An example of a monotonic task system is the Steiner tree problem. The corresponding relaxed version is the page replication problem. Another example is the generalized Steiner tree problem; the relaxed version is the network leasing problem.

To get the better bound, we use a modified version of  $D$ -DAIlg, which now simulates  $D$  algorithms  $A_1 \dots A_D$  and gives a requests  $r$  to  $A_i$  for which  $\text{cost}_{A_i}(v) \geq \frac{1}{2c} \text{cost}_{A_i}(r)$  (if such algorithm exists) or to  $A_1$  otherwise. Using monotonicity we improve the bound of Lemma 1 to  $c \cdot \text{cost}_{Adv}(\sigma)$ , and get a better bound of  $2c \cdot \text{cost}_{A_i}(\sigma_i)$  for  $\text{cost}_{A_1}(\sigma_1)$ . The rest of the analysis is identical.

## Randomized Algorithm against Oblivious Adversary

One can define a randomized version of  $D$ -DAIlg, called  $D$ -RAIlg, which is  $3c$ -competitive against an oblivious adversary. For monotonic task systems it is  $2c$ -competitive. The algorithm  $D$ -RAIlg simulates  $2D$  algorithms  $A_1 \dots A_{2D}$  ( $D$  algorithms in the monotonic case). At the beginning it chooses one of them at random (say  $A_i$ ) and then always keeps the same configuration as  $A_i$ . The requests are always given to the algorithm which incurs the *highest* cost. The analysis is similar to the proof of Theorem 1. Observe that the algorithm is *barely random*, i.e. the randomization is only in the first step.

## 0/1 Request Cost

One may obtain an improved ratio of  $2c(c + 1)$  if the cost incurred by any  $A_i$  on any request is either 0 or 1. This can be further improved to  $c(c + 1)$  for monotonic task systems. This gives a 2-competitive algorithm for the network leasing problem on a uniform metric space. For the page replication problem on a uniform metric, we obtain the 2-competitive algorithm of [BS89]. The ratio 2 is optimal for both these problems as there is a matching lower bound even for the 2-point metric space.

## Better Performance for Specific Cases

It is interesting to observe that our method yields known optimal algorithms for some problems. Here, the actual behavior of  $D$ -DAIlg and  $D$ -RAIlg is better than our analysis indicates. For the one page migration problem on a uniform metric space,  $D$ -DAIlg simulates (a slightly modified version of) the Counter algorithm from [BS89], and is hence 3-competitive (the analysis of [BS89] applies almost unchanged).

For one page migration on a 2-point metric space, a version of  $D$ -RAIlg simulating  $2D$  algorithms actually simulates an algorithm EDGE [CLRW93] which is  $2 + \frac{1}{2D}$ -competitive.

## 3 A Better Algorithm for One Page Migration

We present an algorithm, Move-To-Local-Min (MTLM) for one page migration. The algorithm operates in phases of length  $n = cD$ . The page is kept at the same node throughout the phase and migrated to a

new node only at the end of a phase. Let  $v_1, v_2, \dots, v_n$  be the requests in a particular phase. Suppose the page is kept at node  $b$  throughout the phase. At the end of the phase, the page is migrated to the node  $x$  which minimizes the function  $f(x) = \sum_{i=1}^n d(x, v_i) + \delta D \cdot d(b, x)$ . The parameters  $c$  and  $\delta$  will be specified later.

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

We will prove that for a suitable choice of  $c$  and  $\delta$ , MTLM is 4.086 competitive.

### 3.1 Analysis of Move-To-Local-Min

Let MTLM( $c$ ) be the Move-To-Local-Min algorithm, with phase length  $n = cD$  and minimizing function

$$f(x) = \sum_{i=1}^n d(v_i, x) + \delta D \cdot d(x, b)$$

where  $1 \leq c \leq 2, \delta = c/(c+1)$ ,

**Theorem 2** *MTLM( $c$ ) is  $\max(3 + 2/c, 1.5c + \delta/2 + 1)$ -competitive.*

**Proof:** Fix a phase. Let  $v_1, v_2, \dots, v_n$  be the sequence of requests in the phase. Let  $a_i$  be the position of the optimal off-line algorithm OPT before the request  $v_{i+1}$ . ( $a_n$  is the position of OPT at the end of the phase). Let  $b$  be the position of MTLM during the phase and  $x$  be the node to which it moves to at the end of the phase. Let  $C_{MTLM}$  ( $C_{OPT}$  resp.) be the cost of MTLM (OPT resp.) during the current phase. It is easy to see that

$$\begin{aligned} C_{MTLM} &= \sum_{i=1}^n d(b, v_i) + D \cdot d(b, x) \\ &\leq \sum_{i=1}^n (d(b, a_0) + d(a_0, v_i)) + D \cdot d(b, x) \\ &= \sum_{i=1}^n d(a_0, v_i) + cD \cdot d(a_0, b) + D \cdot d(b, x) \\ C_{OPT} &= \alpha + \beta \\ &\text{where } \alpha = \sum_{i=1}^n d(a_{i-1}, v_i) \text{ and } \beta = D \sum_{i=1}^n d(a_{i-1}, a_i). \end{aligned}$$

We use the potential function

$$\Phi = (1+c)D \cdot d(s, t)$$

where  $s$  is the location of OPT and  $t$  is the location of MTLM.

Consider the difference in the potential  $\Delta\Phi$  before and after the phase.

$$\Delta\Phi = (1+c)D[d(a_n, x) - d(a_0, b)]$$

$$\begin{aligned}
&= \frac{1+c}{c} \sum_{i=1}^n d(a_n, x) - (1+c)D \cdot d(a_0, b) \\
&\leq (1+1/c) \sum_{i=1}^n (d(a_n, v_i) + d(v_i, x)) - (1+c)D \cdot d(a_0, b) \\
\Delta\Phi + C_{MTLM} &\leq (1+1/c) \left[ \sum_{i=1}^n d(a_n, v_i) + \sum_{i=1}^n d(x, v_i) \right] - (1+c)D \cdot d(a_0, b) \\
&\quad + \sum_{i=1}^n d(a_0, v_i) + cD \cdot d(a_0, b) + D \cdot d(b, x) \\
&= \left[ \sum_{i=1}^n d(a_0, v_i) + \sum_{i=1}^n d(a_n, v_i) \right] \\
&\quad + \frac{1}{c} \left[ \sum_{i=1}^n d(a_n, v_i) + \left( \sum_{i=0}^n d(x, v_i) + \delta D \cdot d(b, x) \right) - \delta D \cdot d(a_0, b) \right] \\
&\quad + \left[ \left( \sum_{i=1}^n d(x, v_i) + \delta D \cdot d(b, x) \right) - \delta D \cdot d(a_0, b) \right]
\end{aligned}$$

Now, using  $f(x) \leq f(a_0)$  and  $f(x) \leq \frac{1}{2}(f(a_0) + f(a_n))$ , we get:

$$\begin{aligned}
\Delta\Phi + C_{MTLM} &\leq \left[ \sum_{i=1}^n d(a_n, v_i) + \sum_{i=1}^n d(a_0, v_i) \right] + \frac{1}{c} \left[ \sum_{i=1}^n d(a_0, v_i) + \sum_{i=1}^n d(a_n, v_i) \right] \\
&\quad + \frac{1}{2} \left[ \sum_{i=1}^n d(a_n, v_i) + \sum_{i=1}^n d(a_0, v_i) + \delta D(d(a_n, b) - d(a_0, b)) \right] \\
&\leq (3/2 + 1/c) \left[ \sum_{i=1}^n d(a_0, v_i) + \sum_{i=1}^n d(a_n, v_i) \right] + \delta/2 \cdot D \cdot d(a_0, a_n)
\end{aligned}$$

Let  $A = \sum_{i=1}^n [d(a_0, v_i) + d(a_n, v_i)]$ . Then,

$$\begin{aligned}
A &\leq \sum_{i=1}^n d(a_0, a_{i-1}) + \sum_{i=1}^n d(a_n, a_{i-1}) + 2 \sum_{i=1}^n d(a_{i-1}, v_i) \\
&\leq n \sum_{i=1}^n d(a_{i-1}, a_i) + 2 \sum_{i=1}^n d(a_{i-1}, v_i)
\end{aligned}$$

as

$$\begin{aligned}
\sum_{i=1}^n d(a_0, a_{i-1}) + \sum_{i=1}^n d(a_n, a_{i-1}) &= \sum_{i=1}^{n-1} d(a_0, a_i) + \sum_{i=0}^{n-1} d(a_n, a_i) \\
&\leq \sum_{i=1}^{n-1} \sum_{k=1}^i d(a_{k-1}, a_k) + \sum_{i=0}^{n-1} \sum_{k=i}^{n-1} d(a_k, a_{k+1}) \\
&= \sum_{i=0}^{n-1} \sum_{k=1}^n d(a_{k-1}, a_k) = n \sum_{k=1}^n d(a_{k-1}, a_k)
\end{aligned}$$

Thus,  $A \leq 2\alpha + c\beta$ . Clearly  $D \cdot d(a_0, a_n) \leq \beta$ . Therefore,

$$\begin{aligned} \Delta\Phi + C_{MTLM} &\leq (3/2 + 1/c)(2\alpha + c\beta) + \delta/2 \cdot \beta \\ &\leq \max(3 + 2/c, 1.5c + \delta/2 + 1)(\alpha + \beta) \\ \Delta\Phi &\leq \max(3 + 2/c, 1.5c + \delta/2 + 1)C_{OPT} - C_{MTLM} \end{aligned}$$

□

Solving for minimizing the maximum of the two expressions  $3 + 2/c$  and  $1.5c + \delta/2 + 1$  yields  $c \approx 1.841$ ,  $\delta \approx 0.648$  and the competitive ratio is approximately 4.086.

We mention that the above analysis is tight. We can also obtain a lower bound of 3.847 for a general class of *phase-based algorithms*, which operate in phases of the same length fixed in advance, move only at the end of each phase and the decision where to move is a function of the requests which occurred in the last phase and the current position of the page. The details will appear in the full version of this paper.

## 4 More on $k$ -Page Migration

The  $(2k - 1)$ -competitive  $k$ -server algorithm of [KP94] and the results of Section 2 give us a deterministic  $O(k^2)$  competitive algorithm for  $k$ -page migration for general metric spaces. In this section, we describe optimal or near optimal algorithms for  $k$ -page migration on uniform metric spaces and trees. Using techniques very similar to [MMS88, BLS87], we can prove the following lower bound:

**Theorem 3** *The competitive ratio of any algorithm for  $k$ -page migration on a discrete metric space with  $\geq k + 1$  points is at least  $2k + 1$ .*

### 4.1 Uniform Metric

We present a  $2k + 1$  competitive algorithm for  $k$ -page migration on a uniform metric space.

**Relaxed Flush-When-Full (RFFW):**

```

var counter  $c(p)$  (initially 0) for every node  $p$ 
forever
  unmark all nodes
  repeat
    if the request is at an unmarked node  $p$ 
       $c(p) = c(p) + 1$ 
      if  $c(p) = D$ 
        mark  $p$ 
         $c(p) = 0$ 
        migrate the page from any unmarked node to  $p$ 
    until  $k$  nodes marked
  wait until  $D$  requests occur at unmarked nodes
loop

```

**Theorem 4** *Algorithm RFFW is  $2k + 1$  competitive for  $k$ -page migration on a uniform metric space.*

We briefly sketch the proof. Divide the requests into sets; each set corresponds to a particular iteration of the outermost loop. The set consists of all the requests at each of the  $k$  marked nodes (including the  $D$  requests that caused them to be marked) together with the  $D$  requests issued during the waiting step. Consider a particular set of requests. The algorithm pays a cost  $(2k + 1)D$ . The optimal off-line algorithm either pays  $D$  for the requests at one of the  $k$  marked nodes, or pays  $D$  to move to one of the  $k$  nodes, or pays  $D$  for the requests issued in the waiting phase. One can show that costs are not double counted in this process.

## 4.2 Trees

We present an algorithm, Discrete Fractional Converge which has a competitive ratio of  $(2k + 1)(1 + \frac{1}{D})$  for  $k$ -page migration on trees. This algorithm is for the discrete version of the problem where servers can only be placed at discrete points (called nodes) on the tree. It is based on the  $k$  server algorithm of [CL91].

### Discrete Fractional Converge (DFC).

The algorithm maintains  $k$  pairs of servers. Each pair consists of a discrete server and a continuous server. The discrete server is always at a node of the tree, while the continuous server can be on an edge of the tree, between two nodes.

The configuration of the algorithm at any time is given by the positions of the  $k$  discrete servers. Initially, each continuous server is at the same point as the corresponding discrete server. The algorithm always moves one or more continuous servers. In this process, if a continuous server crosses a node, the corresponding discrete server is moved to that node.

**Definition 5** *For a given request  $r$ , a continuous server  $s$  at  $p$  is said to be a neighbor of  $r$  if the unique path from  $p$  to  $r$  does not contain any continuous servers. If the point  $p$  has more than one continuous server, only one of them is said to be a neighbor.*

On receiving a request  $r$ , the algorithm changes its configuration by moving all the neighbors of  $r$  towards  $r$  at the same speed. Observe that the number of moving servers could reduce during the servicing of  $r$ , as a neighbor may cease to be a neighbor.

The movement is done as follows: For a continuous server  $s$  let  $d'(s, r)$  denote the distance of the corresponding discrete server from the request  $r$ . Each continuous server maintains a continuous counter, initially 0. When a continuous server  $s$  is moved a distance  $\epsilon$ , its counter is incremented by  $\epsilon/d'(s, r)$ . Note that since the corresponding discrete server may change position during the move, the change in counter value is not necessarily the same function of distance moved throughout. The movement is stopped when any of the counters reaches the value  $1/2D$ .

**Theorem 5** *Algorithm DFC is  $(2k + 1)(1 + \frac{1}{D})$  competitive for page migration on discrete trees.*

We can prove that this algorithm is  $(2k + 1)$ -competitive in the model where the online algorithm is allowed to move first and then service a request. It follows that in the regular model for page migration where the request must be serviced first, this algorithm is  $(2k + 1)(1 + \frac{1}{D})$ -competitive. The details of the proof will appear in the full version of the paper. We mention that for page migration on continuous trees, we can obtain a  $(2k + 1)$ -competitive algorithm.

## 5 Acknowledgments

The second and third authors would like to thank Chandra Chekuri and Sanjeev Khanna for helpful discussions during the initial part of this work. Thanks also to Rajeev Motwani for his encouragement and for his excellent lectures on online algorithms which initiated this work.

## References

- [AAB96] B. Awerbuch, Y. Azar, and Y. Bartal. On-line Generalized Steiner Problem. In *Proc. of the 7th Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 68–74, January 1996.
- [ABF93a] B. Awerbuch, Y. Bartal, and A. Fiat. Competitive Distributed File Allocation. In *Proc. 25th ACM Symp. on Theory of Computing*, pages 164–173, May 1993.
- [ABF93b] B. Awerbuch, Y. Bartal, and A. Fiat. Heat & Dump: Randomized competitive distributed paging. In *Proc. 34rd IEEE Symp. on Foundations of Computer Science*. IEEE, pages 22–31, November 1993.
- [ABF96] B. Awerbuch, Y. Bartal, and A. Fiat. Distributed Paging for General Networks. In *Proc. of the 7th Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 574–583, January 1996.
- [AK95] S. Albers and H. Koga. Page Migration with Limited Local Memory Capacity. In *Proc. of the 4th Workshop on Algorithms and Data Structures*, , August 1995.
- [AKP92] B. Awerbuch, S. Kutten, and D. Peleg. Competitive Distributed Job Scheduling. In *Proc. of the 24th Ann. ACM Symp. on Theory of Computing*, pages 571–580, May 1992.
- [Bart96] Y. Bartal. Probabilistic Approximation of Metric Spaces and its Algorithmic Applications. To appear in *Proc. of the 37rd Ann. IEEE Symp. on Foundations of Computer Science*, 1996.
- [BFR92] Y. Bartal, A. Fiat, and Y. Rabani. Competitive algorithms for distributed data management. In *Proc. 24th ACM Symp. on Theory of Computing*, pages 39–50, 1992. Also accepted for publication in the STOC 92 special issue *Journal of Computer and Systems Sciences*, 1993.
- [BLS87] A. Borodin, N. Linial, and M. Saks. An Optimal On-Line Algorithm for Metrical Task Systems. In *Proc. of the 19th Ann. ACM Symp on Theory of Computing*, pages 373–382, May 1987.
- [BS89] D.L. Black and D.D. Sleator. Competitive Algorithms for Replication and Migration Problems. Technical Report CMU-CS-89-201, Department of Computer Science, Carnegie-Mellon University, 1989.
- [CKPV90] M. Chrobak, H. Karloff, T. Payne, and S. Vishwanathan. New Results on Server Problems. In *Proc. 1st ACM-SIAM Symp. on Discrete Algorithms*, pages 291–300, January 1990.
- [CL91] M. Chrobak and L. Larmore. An Optimal On-Line  $k$ -Server Algorithm for Trees. *SIAM Journal of Computing*, 20:144-148, 1991.
- [CLRW93] M. Chrobak, L. Larmore, N. Reingold, and J. Westbrook. Optimal Multiprocessor Migration Algorithms Using Work Functions. In *Proc. of the 4th International Symp. on Algorithms and Computation*. Also *Lecture Notes in Computer Science*, vol. 762, pages 406-415, Hong Kong, 1993, Springer-Verlag.
- [FH80] M.L. Fisher and D.S. Hochbaum. Database Location in a Computer Network. In *Journal of the ACM*, 27(4), pages 718-735, 1980.
- [FKLM<sup>+</sup>88] A. Fiat, R.M. Karp, M. Luby, L.A. McGeoch, D.d. Sleator , and N.E. Young. Competitive Paging Algorithms. Technical Report, Carnegie Mellon University, 1988.
- [FMRW95] A. Fiat, Y. Mansour, A. Rosén, and O. Waarts. Competitive Access Time via Dynamic Storage Rearrangement. In *Proc. of the 36th Ann. IEEE Symp. on Foundations of Computer Science*, pages 392–403, October 1995.
- [GS90] B. Gavish and O.R.L. Sheng. Dynamic File Migration in Distributed Computer Systems. In *Communications of the ACM*, 33(2):177-189, 1990.
- [Hac86] File Migration and Process Migration in a Local Area Network. In *Proc. of INFOCOM86*, pages 488-495, 1986.
- [Kar92] R.M. Karp, On-line Algorithms Versus Off-line Algorithms: How Much is it Worth to Know the Future?. In *Proc. World Computer Congress*, 1992.

- [KP94] E. Koutsoupias, and C. Papadimitriou. On the  $k$ -Server Conjecture. In *Proc. 26th Annual ACM Symp. on Theory of Computing*, pages 507–511, 1994.
- [LRWY94] C. Lund, N. Reingold, J. Westbrook, and D. Yan. On-Line Distributed Data Management. In *Proc. of European Symp. on Algorithms*, 1994.
- [MMS88] M.S. Manasse, L.A. McGeoch, and D.D. Sleator. Competitive Algorithms for On-Line Problems. In *Proc. of the 20th Ann. ACM Symp. on Theory of Computing*, pages 322–333, May 1988.
- [PT83] J. Paris and W.F. Tichy. STORK: An Experimental Migrating File System for Computer Networks. In *IEEE Infocom*, 1983.
- [Sheng86] O.L. Sheng. Models for Dynamic File Migration in Distributed Computer Systems. Ph.D. dissertation.
- [SD89] C. Scheurich and M. Dubois. Dynamic Page Migration in Multiprocessors with Distributed Global Memory. In *IEEE Transactions on Computers*, 38(8):1154–1163, 1989.
- [ST85] D. Sleator and R.E. Tarjan. Amortized Efficiency of List Update and Paging Rules. *Communications of ACM* , 28(2):202–208, 1985.
- [TZ84] W. Tichy and R. Zuwang. Towards a Distributed File System. In *Proc. of the 1984 USENIX Summer Conf.*, pages 87-97, 1984.
- [West91] J. Westbrook. Randomized Algorithms for Multiprocess or Page Migration. In *Proc. of DIMACS Workshop on On-Line Algorithms*. American Mathematical Society, February, 1991.