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* problem 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 it is 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 [BLS92]. 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 deterministic algorithms for *k-page migration* by using k-server [MMS90] 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 ~ 4.086, improving upon the previously best competitive ratio of 7 [ABF93a]. (The current lower bound on the problem is ~ 3.148 [CLRW97].)

Keywords: on-line algorithms, page migration, relaxed task systems.

*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 Groswith Fellowship, and by NSF Award CCR-9357849, with matching funds from IBM, Mitsubishi, Schlumberger Foundation, Shell Foundation, and Xerox Corporation. e-mail: `moses@cs.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@cs.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. However, such transactions incur 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 off-line algorithm (known as competitive analysis [ST85]). Page migration problems have been further studied in [West94, BFR92, CLRW97, ABF93a, ABF93b, LRWY94, AK95, ABF98, Bart96].

We study this problem and the more general *k-page migration* problem where there may be k mirror-replicas of the page we call. 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 problem 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 use 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 it is 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 using 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 the cost of 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 a description of our results.

The general theorems are formulated in the context of task systems ([BLS92]):

DEFINITION 1.1. *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* $\mathrm{dist}(C_1, C_2)$. *(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 $task(C,r)$ (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* $\mathrm{cost}(C_1, C_2, r) = \mathrm{dist}(C_1, C_2) + task(C_2, r)$. *If the move cost function* 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 1.2. *A D*-relaxed *task system, D-$\mathcal{P}$, 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* cost_D, dist_D *and $task_D$ respectively.* dist_D *and $task_D$ are defined as follows: Given $C_1, C_2 \in \mathcal{C}$,* $\mathrm{dist}_D(C_1, C_2) = D \cdot \mathrm{dist}(C_1, C_2)$. *Given $C \in \mathcal{C}$ and a task r, $task_D(C, r) = \min_{C'} \mathrm{dist}(C, C') + task(C', r)$.*

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

DEFINITION 1.3. *A* forcing *task system [MMS90] task system such that for every request r and every configuration C, $task(C,r)$ is either 0 or ∞. 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 give a $3c$-competitive randomized algorithm against adaptive on-line adversaries in that case (we note their result can be generalized to our model).

A *non-constructive* proof of [BBK+94] implies that there exists a $9c^2$-competitive deterministic algorithm. They also show how to construct such a deterministic algorithm under certain assumptions. One of their constructions depends on the existence of a certain kind of potential function that is also efficiently computable. The other construction loses a $(1+\epsilon)$ factor in the competitive ratio and requires the task system to satisfy a property called *locality*. Further, the construction requires computing the optimal strategy over a set of request sequences. The size of this set depends on the task system and on ϵ and could be quite large. (See [BBK+94] for details).

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-$\mathcal{P}$. In comparison to [BBK+94], our deterministic algorithm is simple, explicit and efficient.

- 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-$\mathcal{P}$ 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 a cost of D (the size of the page) times the distance traveled. The problem is to minimize the sum of the 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 [West94] gives a randomized algorithm which is 3-competitive 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 [CLRW97]. Chrobak et. al. [CLRW97] 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 ~ 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 Work Function Algorithm for the k-server problem [KP95]).

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-$\mathcal{P}$. 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 σ and request r, let $\text{cost}_A(\sigma, r)$ (or $\text{cost}_A(r)$ when σ follows from the context) be the cost incurred by A while servicing r from the configuration reached by previously servicing σ. Also, let $\text{cost}_A(\sigma)$ be the total cost of A on σ. Assume that A is c-competitive for $\mathcal{P}$, i.e. that for all σ:

$$\text{cost}_A(\sigma) \leq c \cdot \min_A \text{cost}_A(\sigma)$$

This notion is often called *strict competitiveness*, since we do not allow additive constants on the right hand side of the above inequality. We define the competitive algorithm D-DAlg for D-$\mathcal{P}$ as follows.

Algorithm **D-DAlg**.
Algorithm D-DAlg simulates $2D$ copies $A_1 \dots A_{2D}$ of A. The configuration of D-DAlg is always the same as that of A_1. When given a new request r, the algorithm gives it to one of the A_is 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-DAlg services r and retains its current configuration.
- otherwise, r is given to A_1. Then D-DAlg services r and moves to the new configuration of A_1.

THEOREM 2.1. *Let $\mathcal{P}$ be a metrical task system and let A be a c-competitive deterministic algorithm for $\mathcal{P}$. Then algorithm D-*DAlg *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-DAlg is within a factor $4.5c$ from the above sum. The result will follow.

LEMMA 2.1. *Let σ be a request sequence, and let $\sigma_1 \dots \sigma_{2D}$ be (possibly empty) subsequences of σ such that each request from σ appears in exactly one σ_i. Also, let A be a c-competitive algorithm for $\mathcal{P}$ and let $\text{cost}_{D\text{-}Adv}(\sigma)$ be the optimal off-line cost of servicing σ 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 off-line cost of servicing σ_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 $\sum_{i=1}^{2D} \text{cost}_{Adv_i}(\sigma_i) \leq 2 \cdot \text{cost}_{D\text{-}Adv}(\sigma)$. Let D-Adv be the optimal off-line algorithm for servicing σ in D-$\mathcal{P}$. We define algorithms $Adv'_i, i = 1 \dots 2D$ such that Adv'_i services σ_i in $\mathcal{P}$. All the algorithms Adv'_i always maintain the same configuration as D-Adv.

- whenever *D-Adv* changes configuration (say from C to C'), all Adv'_i change their configuration accordingly. The sum of the costs of Adv'_i is equal to $2D \cdot \text{dist}(C, C') = 2 \cdot \text{dist}_D(C, C')$, which is twice the cost of *D-Adv*,

- when *D-Adv* services request r (say from C), the Adv'_i for which r is included in σ_i moves from C to $C' = C_{min}(C, r)$, satisfies the request and moves back to C. Again, the cost of Adv'_i is equal to $2 \cdot \text{dist}(C, C') + \text{task}(C', r)$, which is at most twice $\text{task}_D(C, r)$ i.e. the cost of *D-Adv*.

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

LEMMA 2.2. *Let σ_i be a sequence of requests given to A_i while running D-DAlg on σ. Then*

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

Proof. We may split $\text{cost}_{D\text{-DAlg}}(\sigma)$ into $\text{cost}^S_{D\text{-DAlg}}(\sigma)$ (the cost of servicing requests) and $\text{cost}^M_{D\text{-DAlg}}(\sigma)$ (the cost of moving between configurations).

Consider the cost incurred by D-DAlg to service a request r. If r is given to A_i, the cost of servicing r from the current configuration of D-DAlg 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 $\text{cost}^M_{D\text{-DAlg}}(\sigma) = D \cdot \text{cost}_{A_1}(\sigma_1)$ 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 σ_i, but also service all requests from σ_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

$$\begin{aligned} \text{cost}_{A_i}(\sigma_i) + \sum_{r \in \sigma_1} \text{cost}_{A_i}(r) &\leq \text{cost}_{A_i}(\sigma_i) + \frac{2}{3c} \sum_{r \in \sigma_1} \text{cost}_{A_1}(r) \\ &= \text{cost}_{A_i}(\sigma_i) + \frac{2}{3c}\text{cost}_{A_1}(\sigma_1). \end{aligned}$$

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

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

Now we can bound the moving cost as follows:

$$\text{cost}^M_{D\text{-DAlg}}(\sigma) = D \cdot \text{cost}_{A_1}(\sigma_1) \leq \frac{1}{2} \sum_{i=1}^{2D} \text{cost}_{A_1}(\sigma_1) \leq \frac{3c}{2} \sum_{i=1}^{2D} \text{cost}_{A_i}(\sigma_i).$$

$$\begin{aligned}\mathrm{cost}_{D\text{-DAlg}}(\sigma) &= \mathrm{cost}^{S}_{D\text{-DAlg}}(\sigma) + \mathrm{cost}^{M}_{D\text{-DAlg}}(\sigma)\\ &\le (3+1.5)c\sum_{i=1}^{2D}\mathrm{cost}_{A_i}(\sigma_i)\\ &= 4.5c\sum_{i=1}^{2D}\mathrm{cost}_{A_i}(\sigma_i).\end{aligned}$$

□

Theorem 2.1 follows from Lemmas 2.1 and 2.2. □

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 2.1. *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, $\mathrm{dist}(C_1, C) \le \mathrm{dist}(C_1', C_2)$.*

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-DAlg, which now simulates D algorithms $A_1 \dots A_D$ and gives a request r to A_i for which $\mathrm{cost}_{A_i}(r) \ge \frac{1}{2c}\mathrm{cost}_{A_1}(r)$ (if such an algorithm exists) or to A_1 otherwise. Using monotonicity we improve the bounds of Lemma 2.1 and Lemma 2.2.

LEMMA 2.3. *Let σ be a request sequence, and let $\sigma_1 \dots \sigma_D$ be (possibly empty) subsequences of σ such that each request from σ appears in exactly one σ_i. Also, let A be a c-competitive algorithm for monotonic task system $\mathcal{P}$ and let $\mathrm{cost}_{D\text{-}Adv}(\sigma)$ be the optimal off-line cost of servicing σ in D-$\mathcal{P}$. Then*

$$\sum_{i=1}^{D}\mathrm{cost}_A(\sigma_i) \le c\cdot \mathrm{cost}_{D\text{-}Adv}(\sigma).$$

Proof. Let $\mathrm{cost}_{Adv_i}(\sigma_i)$ be the optimal off-line cost of servicing σ_i in $\mathcal{P}$. As A is c-competitive, $\mathrm{cost}_A(\sigma_i) \le c\cdot\mathrm{cost}_{Adv_i}(\sigma_i)$. Hence it is sufficient to prove that $\sum_{i=1}^{D}\mathrm{cost}_{Adv_i}(\sigma_i) \le \mathrm{cost}_{D\text{-}Adv}(\sigma)$. Let D-Adv be the optimal off-line algorithm for servicing σ in D-$\mathcal{P}$. We define algorithms $Adv_i', i = 1\dots D$ such that Adv_i' services σ_i in $\mathcal{P}$. All the algorithms Adv_i' initially start in the same configuration as D-Adv. As the requests are processed, the following property is always maintained: If D-Adv is in some configuration C, then Adv_i' is in some configuration C_i such that C_i dominates C.

- Suppose D-*Adv* changes configuration (say from C to C'). Adv'_i is in some configuration C_i that dominates C. It now moves to a new configuration C'_i that dominates C_i and C'. By monotonicity $\text{dist}(C_i, C'_i) \leq \text{dist}(C, C')$. The sum of the costs of $Adv'_i, i = 1, \ldots, D$ is at most $D \cdot \text{dist}(C, C') = \text{dist}_D(C, C')$, which is the cost of D-*Adv*,

- when D-*Adv* services request r (say from C), consider the Adv'_i for which r is included in σ_i. It is initially in some configuration C_i that dominates C. It now moves to a new configuration C'_i that dominates C_i and $C' = C_{min}(C, r)$. Recall that $\mathcal{P}$ is a forcing task system, so for a configuration C and request r, $\text{task}(C, r)$ is either 0 or ∞; if it is 0, we say that C is allowable for r. Clearly, C' is allowable for r. Since C'_i dominates C', it follows that C'_i is also allowable for r. Thus Adv'_i can satisfy request r from configuration C'_i. The cost of Adv'_i is equal to $\text{dist}(C_i, C'_i) \leq \text{dist}(C, C') = \text{task}_D(C, r)$ i.e. the cost of D-*Adv*.

Thus, $\sum_{i=1}^{D} \text{cost}_{Adv'_i}(\sigma_i) \leq \text{cost}_{D\text{-}Adv}(\sigma)$. Since the optimal off-line cost $\text{cost}_{Adv_i}(\sigma_i) \leq \text{cost}_{Adv'_i}(\sigma_i)$, the lemma follows. □

LEMMA 2.4. *Let σ_i be the sequence of requests given to A_i while running the modified D-*DAlg *for monotonic task systems on σ. Then*

$$\text{cost}_{D\text{-}DAlg}(\sigma) \leq 4c \sum_{i=1}^{D} \text{cost}_{A_i}(\sigma_i)$$

Proof. We split $\text{cost}_{D\text{-DAlg}}(\sigma)$ into $\text{cost}^S_{D\text{-DAlg}}(\sigma)$ (the cost of servicing requests) and $\text{cost}^M_{D\text{-DAlg}}(\sigma)$ (the cost of moving between configurations).

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

We now bound $\text{cost}^M_{D\text{-DAlg}}(\sigma) = D \cdot \text{cost}_{A_1}(\sigma_1)$ in terms of $\sum_{i=1}^{D} \text{cost}_{A_i}(\sigma_i)$. Consider algorithms A'_i which simulate A_i on σ_i, but also service all requests from σ_1 in the following way. A'_i starts out in the same configuration as A_i. As the requests are serviced, the following property is satisfied: If A_i is in a configuration C, A'_i is in a configuration C_1 such that C_1 dominates C. Thus, if A_i services a request from its configuration C, A'_i can also service the same request from its configuration C_1. When A_i changes configuration from C to C', A'_i moves from configuration C_1 (which dominates C) to configuration C'_1 which dominates both C_1 and C'. By monotonicity, $\text{dist}(C_1, C'_1) \leq \text{dist}(C, C')$. Thus the total cost incurred by A'_i for configuration changes due to changes in the configuration of A_i is at most $\text{cost}_{A_i}(\sigma_i)$. When a request $r \in \sigma_1$ appears, suppose A_i is in configuration C and A'_i is in configuration C_1 that dominates C. A'_i moves from its current configuration C_1 to a new configuration C'_1 which dominates C_1 and $C' = C_{min}(C, r)$. Note that C' is allowable for r and C'_1 dominates C', so C'_1 is also allowable for r. By monotonicity, $\text{dist}(C_1, C'_1) \leq \text{dist}(C, C')$. Hence $\text{cost}_{A'_i}(r) \leq \text{dist}(C, C') = \text{cost}_{A_i}(r)$. As r was given to A_1, we know that

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

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

$$\text{cost}_{A_i}(\sigma_i) + \sum_{r \in \sigma_1} \text{cost}_{A_i}(r) \quad \leq \quad \text{cost}_{A_i}(\sigma_i) + \frac{1}{2c} \sum_{r \in \sigma_1} \text{cost}_{A_1}(r)$$

$$= \mathrm{cost}_{A_i}(\sigma_i) + \frac{1}{2c}\mathrm{cost}_{A_1}(\sigma_1).$$

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

$$\begin{aligned} \frac{1}{c}\mathrm{cost}_{A_1}(\sigma_1) &\le \mathrm{cost}_{A'_i}(\sigma_1) \\ &\le \mathrm{cost}_{A_i}(\sigma_i) + \frac{1}{2c}\mathrm{cost}_{A_1}(\sigma_1) \\ \Rightarrow \mathrm{cost}_{A_1}(\sigma_1) &\le 2c \cdot \mathrm{cost}_{A_i}(\sigma_i). \end{aligned}$$

Now we can bound the moving cost as follows:

$$\mathrm{cost}^M_{D\text{-DAlg}}(\sigma) = D \cdot \mathrm{cost}_{A_1}(\sigma_1) \le 2c \sum_{i=1}^{D} \mathrm{cost}_{A_i}(\sigma_i).$$

$$\begin{aligned} \mathrm{cost}_{D\text{-DAlg}}(\sigma) &= \mathrm{cost}^S_{D\text{-DAlg}}(\sigma) + \mathrm{cost}^M_{D\text{-DAlg}}(\sigma) \\ &\le 4c \sum_{i=1}^{D} \mathrm{cost}_{A_i}(\sigma_i). \end{aligned}$$

□

Lemmas 2.3 and 2.4 imply the following theorem.

THEOREM 2.2. *Let $\mathcal{P}$ be a monotonic task system and let A be a c-competitive deterministic algorithm for $\mathcal{P}$. Then the modified algorithm D-DAlg is $4c^2$-competitive for the D-relaxed task system D-$\mathcal{P}$.*

Randomized Algorithm against Oblivious Adversary

One can define a randomized version of D-DAlg, called D-RAlg, which is $3c$-competitive against an oblivious adversary. For monotonic task systems it is $2c$-competitive. For such systems this improves the $3c$-competitive algorithm from [AAB96]. (though their algorithm works also against adaptive adversaries, which D-RAlg cannot handle). The algorithm D-RAlg simulates $2D$ algorithms $A_1 \ldots A_{2D}$ (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. Observe that the algorithm is *barely random*, *i.e.* the randomization is only in the first step. The following lemma bounds the expected cost of the algorithm D-RAlg.

LEMMA 2.5. *The expected cost of D-RAlg is at most $\frac{3}{2}\sum_{i=1}^{2D} \mathrm{cost}_{A_i}(\sigma_i)$.*

Proof. Suppose the algorithm D-RAlg simulates A_i. We claim that the cost of D-RAlg is at most $D \cdot \mathrm{cost}_{A_i}(\sigma_i) + \sum_{j=1}^{2D} \mathrm{cost}_{A_j}(\sigma_j)$. We will split the total cost of D-RAlg into two parts: the cost of moving between configurations and the cost of servicing requests.

The total movement cost of D-RAlg is at most $D \cdot \mathrm{cost}_{A_i}(\sigma_i)$. Whenever A_i changes configuration from C to C', D-RAlg also changes configuration from C to C'. The cost incurred by A_i is $\mathrm{dist}(C, C')$ and that by D-RAlg is $D \cdot \mathrm{dist}(C, C')$.

Consider now the total request service cost of D-RAlg. Suppose a request r is given to A_j. The cost incurred by D-RAlg to service this request is $\text{cost}_{A_i}(r) \leq \text{cost}_{A_j}(r)$. Hence the total request service cost of D-RAlg is bounded by $\sum_{j=1}^{2D} \text{cost}_{A_j}(\sigma_j)$.

This proves the claim that if D-RAlg chooses A_i, the cost incurred by it is at most $D \cdot \text{cost}_{A_i}(\sigma_i) + \sum_{i=1}^{2D} \text{cost}_{A_i}(\sigma_i)$. Since i is chosen uniformly and at random from the set $1, \ldots, 2D$, the expected cost of D-RAlg is at most $\frac{D}{2D} \sum_{i=1}^{2D} \text{cost}_{A_i}(\sigma_i) + \sum_{i=1}^{2D} \text{cost}_{A_i}(\sigma_i)$ which is $\frac{3}{2} \sum_{i=1}^{2D} \text{cost}_{A_i}(\sigma_i)$. $\square$

Combining Lemma 2.5 with Lemma 2.1, the expected competitive ratio of D-RAlg is bounded by $3c$.

In the case of monotonic task systems, the algorithm D-RAlg simulates D algorithms $A_1, \ldots, A_D$. It is easy to modify the proof of Lemma 2.5 to show that in this case, the expected competitive ratio of D-Ralg is bounded by $\sum_{i=1}^{D} \text{cost}_{A_i}(\sigma_i)$. Combining this with Lemma 2.3, the expected competitive ratio of D-RAlg is bounded by $2c$.

0/1 **Request Cost**

One may obtain an improved ratio of $c(c+2)$ if the underlying task system is such that 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.

The algorithm is the same algorithm D-DAlg. In this case, since each of the simulated algorithms A_i pays either 0 or 1 to service each request, the request distribution rule reduces to the following: Consider a new request r. If there exists an $A_i, i \geq 2$ for which $\text{cost}_{A_i}(r) = 1$, give the request to A_i, else give the request to A_1.

We improve the result of Lemma 2.2 as follows:

LEMMA 2.6. *Let σ_i be a sequence of requests given to A_i while running D-DAlg on σ. If the underlying task system is such that the cost incurred by any A_i on any request is either 0 or 1, then*

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

Proof. We split $\text{cost}_{D\text{-DAlg}}(\sigma)$ into $\text{cost}^S_{D\text{-DAlg}}(\sigma)$ (the cost of servicing requests) and $\text{cost}^M_{D\text{-DAlg}}(\sigma)$ (the cost of moving between configurations).

Consider the cost incurred by D-DAlg to service a request r. If r is given to A_i, the cost of servicing r from the current configuration of D-DAlg is at most $\text{cost}_{A_i}(r)$ (since the cost can only be 0 or 1). Hence, we can bound the total cost of servicing requests by $\sum_{i=1}^{2D} \text{cost}_{A_i}(\sigma_i)$.

We now bound $\text{cost}^M_{D\text{-DAlg}}(\sigma) = D \cdot \text{cost}_{A_1}(\sigma_1)$ in terms of $\sum_{i=1}^{2D} \text{cost}_{A_i}(\sigma_i)$. For this purpose, consider algorithms A'_i which simulate A_i on σ_i and always maintains the same configuration as A_i, but also service all requests from σ_1 as follows. Whenever $r \in \sigma_1$ appears, since the request r was not given to A_i, it must be the case that $\text{cost}_{A_i}(r) = 0$. Thus, $\text{cost}_{A'_i}(r) = 0$, and A'_i simply services r at zero cost. Hence the total cost of A'_i (denoted by $\text{cost}_{A'_i}(\sigma_1)$) is simply $\text{cost}_{A_i}(\sigma_i)$.

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

Now we can bound the moving cost as follows:

$$\text{cost}^M_{D\text{-DAlg}}(\sigma) = D \cdot \text{cost}_{A_1}(\sigma_1) \leq \frac{c}{2} \sum_{i=1}^{2D} \text{cost}_{A_i}(\sigma_i).$$

$$\begin{aligned}\mathrm{cost}_{D\text{-DAlg}}(\sigma_1) &= \mathrm{cost}^S_{D\text{-DAlg}}(\sigma) + \mathrm{cost}^M_{D\text{-DAlg}}(\sigma)\\ &\leq (\frac{c}{2}+1)\sum_{i=1}^{2D}\mathrm{cost}_{A_i}(\sigma_i).\end{aligned}$$

□

The above lemma, combined with Lemma 2.1 implies that for the case of 0/1 costs, D-DAlg has a competitive ratio bounded by $2c(\frac{c}{2}+1) = c(c+2)$.

For monotonic task systems, D-DAlg simulates D algorithms $A_1, \ldots, A_D$. It is easy to modify the proof of the above lemma to show that in this case $\mathrm{cost}_{D\text{-DAlg}}(\sigma) \leq (c+1)\sum_{i=1}^{D}\mathrm{cost}_{A_i}(\sigma_i)$. Combining this with Lemma 2.3 implies that for monotonic task systems with 0/1 costs, D-DAlg has a competitive ratio bounded by $c(c+1)$.

This gives a 2-competitive algorithm for the network leasing problem on a uniform metric space. Note that the corresponding problem for the underlying task system (the generalized Steiner tree on a uniform metric space) can be solved optimally by an algorithm which pays either 0 or 1 on each request. 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.

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 $v_1, v_2, \ldots v_n$ 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 δ 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 δ. This ensures that the cost of making this move is not too high. This additional term is similar to, and in fact inspired by the additional term in the minimizer function used by the Work Function Algorithm for the k-server problem and metrical task systems.

We will prove that for a suitable choice of c and δ, 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 3.1. *MTLM(c) is* $\max(3 + 2/c, 1.5c + \delta/2 + 1)$*-competitive.*

Proof. Fix a phase. Let $v_1, v_2, \ldots v_n$ 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

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

where $\alpha = \sum_{i=1}^{n} d(a_{i-1},v_i)$ and $\beta = D\sum_{i=1}^{n} d(a_{i-1},a_i)$.

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.

$$\begin{aligned}
\Delta\Phi &= (1+c)D[d(a_n,x) - d(a_0,b)] \\
&= \frac{1+c}{c}\sum_{i=1}^{n} d(a_n,x) - (1+c)D \cdot d(a_0,b) \\
&\leq (1+1/c)\sum_{i=1}^{n}(d(a_n,v_i) + d(v_i,x)) - (1+c)D \cdot d(a_0,b).
\end{aligned}$$

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

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

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

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*. Our lower bound holds under the following assumptions:

- the algorithm operates in phases, each phase consisting of a fixed (say N) number of requests
- the page is moved only at the end of each phase
- the decision where to move the page is based solely on the number of requests received at each node during the last phase and the current position of the page

We call the algorithms satisfying the above assumption *phase-based*. It is easy to see that our MTLM algorithm is phase based.

THEOREM 3.2. *For any $\epsilon > 0$ there is D such that any phase-based algorithm for page migration with parameter D is at least $(3.847 - \epsilon)$-competitive.*

Proof. We show the lower bound holds even for the simple metric space with two points only (say p and q). Assume that $d(p, q) = 1$. Let $N = cD$ be the phase length. Then any algorithm A can be represented as two binary vectors M_p and M_q of length $N + 1$ in the following way. Assume that at the beginning of the phase A is at p. Let k be the number of requests placed at q during the phase. Then at the end of the phase the algorithm moves to q iff $M_p[k] = 1$ and stays at p otherwise.

We can assume that neither M_p nor M_q consists entirely of 0's, as the algorithm could be easily fooled otherwise. Similarily we can assume neither of them contains only 1's. Thus we can define:

- a_p^1 - the smallest number a such that $M_p[aN] = 1$
- a_p^0 - the largest number a such that $M_p[aN] = 0$

We define a_q^1 and a_q^0 in the same way. We can assume that $M_p[N] = M_q[N] = 1$. The adversary which we construct will choose one of the following three strategies, depending on the values of c and a's. Each of these strategies is periodic, *i.e.* consists of a fixed sequence of phases repeated arbitrarily large number of times. The starting configurations of these strategies allow the adversary and the algorithm to start from different nodes. This requirement however can be relaxed, as to arrange any configuration the adversary needs to pay only a fixed cost which is vanishingly small compared to its total cost, thus leaving the asymptotic competitive ratio unchanged.

The strategies are as follows.

A. Let $u \in \{p, q\}$ be the point maximizing a_u^0 and let v be the other point. We assume that the algorithm starts from u while the adversary starts from v. During each phase we place $a_u^0 N$ requests at v and $(1 - a_u^0)N$ requests at u. Clearly, the algorithm stays at u forever, paying a cost of $a_u^0 N$ per phase while the adversary pays $(1 - a_u^0)N$ per phase.

B. Let $w \in \{p, q\}$ be the point minimizing a_w^1 and let x be the other point. Both the algorithm and the adversary start at w. The strategy relies on repeating the following request sequence which spans two phases. In the first phase, we place $a_w^1 N$ requests at x and $(1 - a_w^1)N$ at w. After this phase the algorithm moves to x. In the second phase we place all requests at w and thus the algorithm moves back to w.

We observe that the algorithm pays a cost of $a_w^1 N + D + N + D = ((a_w^1 + 1)c + 2)D$, while the adversary pays only $a_w^1 N$.

C. Both the algorithm and the adversary start from p. The strategy relies on repeating the following request sequence which spans four phases. In the first phase the adversary places $(1 - a_p^0)N$ requests at p, moves to q and places $a_p^0 N$ requests at q. Clearly, the algorithm stays at p after the phase is finished. In the phase two all requests are placed at q, so the algorithm moves to q. The phases three and four are symmetric versions of phase one and two.

We observe that the algorithm pays a cost of $a_p^0 N + N + D + a_q^0 N + D + N = (c(a_p^0 + a_q^0 + 2) + 2)D$, while the adversary pays only $2D$.

Therefore, by choosing a proper strategy, the competitive ratio C of the algorithm can be made at least

$$\max\left[\frac{a_u^0}{1 - a_u^0}, \frac{(a_w^1 + 1)c + 2}{c a_w^1}, \frac{c(a_p^0 + a_q^0 + 2) + 2}{2}\right] = \max\left[\frac{a_u^0}{1 - a_u^0}, \frac{c + 2}{c a_w^1} + 1, c\left(\frac{a_p^0 + a_q^0}{2} + 1\right) + 1\right]$$

Define $a^0 = \frac{a_p^0 + a_q^0}{2}$ and $a^1 = \frac{a_p^1 + a_q^1}{2}$. Observe that $a_u^0 \geq a^0$ and $a_w^1 \leq a^1$. Also, one can verify that $a_p^0 \geq a_p^1 - \frac{1}{N}$ and $a_q^0 \geq a_q^1 - \frac{1}{N}$, which implies $a^0 \geq a^1 - \frac{1}{N} \geq a_w^1 - \frac{1}{N}$. Therefore we can write

$$C \geq \max[\frac{a^0}{1-a^0}, \frac{c+2}{c(a^0 + \frac{1}{N})} + 1, c(a^0+1)+1]$$

We can assume that as D tends to infinity, the value of c remains within (say) the interval $[0.1, 10]$, as otherwise either the second or the third ratio exceeds the lower bound to be proved. Thus for D large enough we can approximate C by

$$C \geq \max[\frac{a}{1-a}, \frac{c+2}{ca} + 1, c(a+1)+1] - \epsilon$$

where a^0 has been replaced by a for simplicity. The latter bound can be numerically verified to be greater than 3.847 for all a and c. □

4 More on k-Page Migration

The $(2k-1)$-competitive k-server algorithm of [KP95] 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 [MMS90, BLS92], we can prove the following lower bound:

THEOREM 4.1. *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 (RFWF):

```
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 an unmarked
                    node to p
    until k nodes marked
    wait for D requests at unmarked nodes
loop
```

THEOREM 4.2. *Algorithm* RFWF *is $2k+1$ competitive for k-page migration on a uniform metric space.*

Proof. Assume without loss of generality that the optimal off-line algorithm migrates a page to a node p only if it receives a request at p. We will associate each of the costs incurred by the on-line and off-line algorithms (*i.e.* request service cost and page movement cost) with a request in the request sequence. A slightly different accounting scheme is used for the costs of the on-line and off-line algorithms.

For both the on-line and off-line algorithms, we associate with each request, the cost incurred by the algorithm to service this request; this is either 0 or 1. We call a request *active* if it causes the on-line algorithm to increment a counter value. Suppose the algorithm (on-line or off-line) migrates a page from a node p to node q. It incurs a page movement cost of D. For the on-line algorithm, this cost is associated with the request at node q that caused the on-line algorithm to move the page. For the off-line algorithm, this cost is associated with the last active request at node p before this movement took place; if no such request exists, we ignore this cost in our analysis.

Divide the requests into sets; each set corresponds to a particular iteration of the outermost loop. The set consists of requests at each of the k nodes marked during the iteration: specifically the D active requests that caused them to be marked and all subsequent requests at those nodes till the end of the iteration, together with the D requests issued (at other nodes) during the waiting step. Consider the set of requests corresponding to an iteration. For a page p marked during the iteration, the active requests at p in the set need not have been received during the iteration; however the last active request at p must be received during the iteration. Note that if the set contains an active request at node p received at a particular time t, then it must include all active requests received at node p from time t till the end of the iteration.

Every request belongs to at most one set. If a request is not assigned to any set, it must be an active request. There can be at most D such unassigned requests at every node p. For each such unassigned request, the on-line algorithm pays a service cost of 1 and there is no movement cost associated with it. Thus the total on-line cost associated with unassigned requests is at most nD where n is the number of nodes in the metric space.

Consider the set of requests corresponding to an iteration. The on-line algorithm pays a service cost of 1 for each of the D active requests at each of the k marked nodes and pays a service cost of 0 for subsequent requests at the k marked nodes. Further, it pays a movement cost of D to move pages to each of the k marked nodes. Also, it pays a service cost of 1 for each of the D requests received during the waiting phase. Thus, for the on-line algorithm, the total cost associated with this set of requests is $(2k+1)D$.

We claim that for the optimal off-line algorithm, the total cost associated with the requests in this set is at least D. Consider a marked node p. There must be D active requests at node p in the set. If the algorithm incurred a service cost of 1 for each one of them, the claim is clearly true. If not, the algorithm must have a page at node p for at least one of the D active requests at p. Now if a movement cost of D is associated with one of the active requests at p, then the claim is true. If no such movement cost is associated, then it must be the case that the algorithm has a page at node p for the last active request at node p; further, this page is not moved till the end of the iteration. If the total cost associated with the active requests in the set is less than D, then throughout the waiting phase, the algorithm must have a page at each of the k marked nodes. But then the algorithm incurs a service cost of 1 for each of the requests incurred in the waiting phase, proving the claim.

The analysis shows that the on-line algorithm incurs a cost of at most $(2k+1)C_{OPT} + nD$

where C_{OPT} is the cost of the optimal off-line algorithm. This proves the theorem. □

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 4.1. *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 s' be the location of the corresponding discrete server. Each continuous server maintains a counter, initially 0. The counters are increased as the continuous servers are moved towards r. When all the neighbors are moved a distance ϵ, the counter of continuous server s 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 4.3. *Algorithm* DFC *is $(2k+1)(1+\frac{1}{D})$ competitive for page migration on discrete trees.*

Proof. The regular model for page migration requires that, for every request, the request must be serviced before any server is moved. Consider the modified model where the online algorithm is allowed to move first and then service a request. Suppose that for servicing request r, the algorithm moves its servers a distance d_r and the closest server to r at the end of the move is at a distance s_r from it. Then, when request r was received, there must have been a server at a distance at most $d_r + s_r$ from it. In the modified model, the algorithm pays $C' = D \cdot d_r + s_r$. In the original model, the algorithm pays at most $C = D \cdot d_r + (d_r + s_r) \leq (1 + \frac{1}{D})C'$. Note that the optimal off-line cost for servicing the requests is the same in both the models. We will prove that DFC is $(2k+1)$-competitive in the modified model. It follows that in the regular model for page migration, this algorithm is $(2k + 1)(1 + \frac{1}{D})$-competitive. For the rest of the proof, we restrict our attention to the modified model.

Let OPT denote the optimal off-line algorithm. Let m_t denote the cost of the minimum weight matching between OPT's servers and DFC's continuous servers. Let σ_t be the sum of the pairwise distances between DFC's continuous servers. For a particular continuous server s, let the *separation* of s denote the distance between s and the corresponding discrete server. Let s_t be the sum of the separations of DFC's continuous servers. The potential function Φ is defined as follows:

$$\Phi = D\left[(2k+1)m_t + 2\sigma_t + s_t\right]$$

For the purpose of analysis, we will assume that the following events occur upon each request: request r appears, OPT moves its servers, DFC moves its servers and then both service r.

We break up the movements of DFC's continuous servers into phases. Phases are terminated by *special* events. A special event occurs whenever a continuous server reaches a node in the tree while moving. In this case, the corresponding discrete server is moved to the node and the next phase begins. During a phase, all the neighbors of r are moved towards r at the same speed. A special event could result in a change in the number of neighbors. Let n be the number of phases during the servicing of request r.

We account for the request service costs of DFC and OPT by charging them to the movement of the continuous servers in the manner described below. We explain why this charging scheme gives upper bounds on DFC's cost and lower bounds on OPT's cost.

We charge DFC's request servicing cost to the movement of the continuous servers at a rate of $2D$ per unit movement. We claim that the total cost charged to DFC by this scheme is at least the actual cost paid by DFC to service the request. Let d_i be the distance moved in the ith phase. The total cost charged to DFC by our charging scheme is $2D\sum_{i=1}^{n} d_i$. Consider the continuous server whose counter reaches the value $1/2D$. Let D_i be the distance of the corresponding discrete server from request r during the ith phase. Since the discrete server can only move closer to r, we have $D_1 \geq D_2 \geq \ldots \geq D_n$.

$$\sum_{i=1}^{n}\frac{d_i}{D_i} = \frac{1}{2D}.$$

$$\text{Further,} \quad \sum_{i=1}^{n}\frac{d_i}{D_i} \leq \frac{1}{D_n}\cdot\sum_{i=1}^{n} d_i.$$

$$\text{This gives us} \quad 2D\sum_{i=1}^{n} d_i \geq D_n.$$

Note that at the end of the phase, DFC services the request by paying the distance of its closest discrete server from request r. Since there is a server at a distance of D_n from r, D_n is an upper bound on the request servicing cost paid by DFC. This proves that the total cost charged to DFC by our charging scheme is an upper bound on the actual request servicing cost.

DEFINITION 4.2. *A server s of OPT is said to be* inside *if the path from s to request r does not contain any of DFC's continuous servers. If s is not inside it is said to be* outside.

Note that during the servicing of request r, an inside server of OPT may become outside, but not the other way around. We charge the request service cost for OPT to the movement of the continuous servers as follows: In phases when OPT has at least one inside server, OPT is charged nothing. In phases where all of OPT's servers are outside, OPT is charged at the rate of $2D$ per unit movement of the continuous servers.

We claim that the cost charged to OPT by our charging scheme is a lower bound on the actual cost paid by OPT to service the request. If one of OPT's servers is inside at the end of all the phases, it must have been inside throughout all the phases. OPT is charged nothing and our claim is clearly true. If all of OPT's servers are outside at the end of all the phases, let p be the number of the first phase when all of OPT's servers are outside. The cost charged to OPT by our charging scheme is $2D\sum_{i=p}^{n} d_i$. Let s' be the closest OPT server to r and let $q \leq p$ be the first phase when s' becomes outside.

Let s be the neighbor on the path from s' to r when s' becomes outside for the first time. s remains on the path from s' to r for all the phases from q to n. Let D_i be the distance of the discrete server (corresponding to s) from request r during phase i. Let D_{OPT} be the distance of s' to r. Then OPT pays D_{OPT} to service r. Note that $D_{OPT} \geq D_q \geq \ldots \geq D_n$. The increase in the counter of s in the phases $q, \ldots, n$ is at most $1/2D$.

$$\text{Thus,} \quad \sum_{i=q}^{n} \frac{d_i}{D_i} \leq \frac{1}{2D}.$$

$$\text{Further,} \quad \sum_{i=q}^{n} \frac{d_i}{D_i} \geq \frac{1}{D_{OPT}} \cdot \sum_{i=q}^{n} d_i.$$

$$\text{This gives us} \quad 2D\sum_{i=q}^{n} d_i \leq D_{OPT}.$$

$$\text{Also} \quad 2D\sum_{i=p}^{n} d_i \leq 2D\sum_{i=q}^{n} d_i.$$

Hence the cost charged to OPT by our charging scheme is a lower bound on the actual request service cost for OPT.

We now analyze the change in the potential function for various cases and show that for each,

$$\Delta\Phi \leq (2k+1) \cdot C_{OPT} - C_{DFC}.$$

Case 1: OPT moves a server a distance x.

Consider the change in m_t. At the beginning of the phase, fix the minimum cost matching M between OPT's servers and DFC's continuous servers. We will upper bound the change in the cost of M at the end of the phase. Since M may no longer be the min cost matching at the end of the phase, the actual min cost matching M' could have lower cost than M. However the change in the cost of M gives an upper bound for Δm_t.

Since one of OPT's servers is moved a distance x, the cost of M could increase by at most x. $C_{OPT} = D \cdot x$. $C_{DFC} = 0$. $\Delta m_t \leq x$. $\Delta\sigma_t = 0$. $\Delta s_t = 0$.

$$\Delta\Phi \leq (2k+1) \cdot D \cdot x = (2k+1) \cdot C_{OPT} - C_{DFC}.$$

Case 2: DFC moves a discrete server a distance x.

$C_{OPT} = 0$. $C_{DFC} = D \cdot x$. $\Delta m_t = 0$. $\Delta\sigma_t = 0$. The discrete server which moved decreases its distance from the corresponding continuous server by x. The separation of the other continuous servers remains unchanged. Hence, $\Delta s_t = -x$.

$$\Delta\Phi = -D \cdot x = (2k+1) \cdot C_{OPT} - C_{DFC}.$$

Case 3: DFC moves its continuous servers towards r during a phase.

Suppose there are m neighbors in this phase and all of them are moved towards r by a distance x. For simplicity assume w.l.o.g $x = 1$.

Consider the change in σ_t. The pairwise distances between the non-neighbors do not change. For each of the $k - m$ non-neighbors, one neighbor moves away from it and the other $m - 1$ neighbors move towards it by a unit distance. Further the pairwise distances between the neighbors each decrease by two units.

$$\Delta\sigma_t = -(k - m)(m - 2) - m(m - 1) = -[km + m - 2k]$$

Also, $\Delta s_t \leq m$. We now consider two cases.

Case 3 (a): One of OPT's servers is inside.

To bound the change in m_t, as before we fix a min cost matching M at the beginning of the phase and bound the change in the cost of M. Since m continuous servers move during the phase, only m of the matched pairs in M change their distances. We can assume w.l.o.g that the min cost matching M matches one of OPT's inside servers with a neighbor. The corresponding distance decreases by a unit. The other $m - 1$ continuous servers could at most increase the distance from their matched OPT server by a unit. $\Delta m_t \leq m - 2$. Also, according to our charging scheme, $C_{OPT} = 0$, $C_{DFC} = 2D$.

$$\begin{aligned} \Delta\Phi &= D\left[(2k+1)\Delta m_t + 2\Delta\sigma_t + \Delta s_t\right] \\ &\leq D\left[(2k+1)(m-2) - 2(km + m - 2k) + m\right] \\ &= -2D \leq (2k+1)C_{OPT} - C_{DFC} \end{aligned}$$

Case 3 (b): All of OPT's servers are outside.

Consider the change in m_t. Again, fix a min cost matching at the beginning of the phase. The m distances corresponding to the m neighbors could each increase by at most one unit. $\Delta m_t \leq m$. Also, according to our charging scheme, $C_{OPT} = 2D$, $C_{DFC} = 2D$.

$$\begin{aligned} \Delta\Phi &= D\left[(2k+1)\Delta m_t + 2\Delta\sigma_t + \Delta s_t\right] \\ &\leq D\left[(2k+1)m - 2(km + m - 2k) + m\right] \\ &= 4k \cdot D \leq (2k+1)C_{OPT} - C_{DFC} \end{aligned}$$

□

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. 34th IEEE Symp. on Foundations of Computer Science, IEEE, pages 22–31,November 1993.

[ABF98] B. Awerbuch, Y. Bartal, and A. Fiat, *Distributed Paging for General Networks*, in Journal of Algorithms, July 1998. vol.28, no.1, p. 67-104.

[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*, in Proc. of the 37rd Ann. IEEE Symp. on Foundations of Computer Science, 1996.

[BBK+94] S. Ben-David, A. Borodin, R.M. Karp, G. Tardos, and A. Wigderson. *On the Power of Randomization in Online Algorithms*. In Algorithmica, Jan. 1994. vol.11, no.1, p. 2-14.

[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.

[BLS92] A. Borodin, N. Linial, and M. Saks, *An Optimal On-Line Algorithm for Metrical Task Systems*, in Journal of the ACM, Oct. 1992. vol.39, no.4, p. 745-63.

[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.

[CLRW97] M. Chrobak, L. Larmore, N. Reingold, and J. Westbrook, *Page Migration Algorithms Using Work Functions*, in Journal of Algorithms, July 1997. vol.24, no.1, p. 124-57.

[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+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.

[KP95] E. Koutsoupias, and C. Papadimitriou, *On the k-Server Conjecture*, in Journal of the ACM, Sept. 1995. vol.42, no.5, p. 971-83.

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

[MMS90] M.S. Manasse, L.A. McGeoch, and D.D. Sleator, *Competitive Algorithms for On-Line Problems*, in Journal of Algorithms, June 1990. vol.11, no.2, p. 208-30.

[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.

[West94] J. Westbrook, *Randomized Algorithms for Multiprocessor Page Migration*, in SIAM Journal on Computing, Oct. 1994. vol.23, no.5, p. 951-65.