The Distributed k-Server Problem — A Competitive Distributed Translator for k-Server Algorithms *

Yair Bartal Adi Rosén

Department of Computer Science
Tel-Aviv University
Tel-Aviv 69978
Israel

E-mail: yairb@math.tau.ac.il, adiro@math.tau.ac.il

*A preliminary version of this paper has appeared in Proc. of the 33rd Ann. IEEE Symp. on Foundations of Computer Science, pp. 344–353, 1992.

Abstract

We consider the k-server problem [23] in a distributed setting. Given a network of n processors, and k identical mobile servers, requests for service appear at the processors and a server must reach the request-point. Besides modeling problems in computer networks where k identical mobile resources are shared by the processors of the network, this models a realistic situation where the transfer of information is costly and there is no central control that governs the behavior of servers that move around to satisfy requests for service.

We give a general translator to transform any deterministic global-control competitive k-server algorithm into a distributed competitive one. As consequences we get poly(k)-competitive distributed algorithms for the line, trees and the ring.

In contrast to the global-control case where there are k-server algorithms with competitive ratio that depends on k solely [17, 19, 10, 22], we have a lower bound of $\Omega(\max\{k, \frac{1}{D} \cdot \log n / \log\log n\})$ on the competitive ratio of any distributed k-server algorithm, where $\frac{1}{D}$ is the ratio between the cost to transmit a message and the cost to move a server over the same distance.

We also give a distributed version of the Harmonic randomized k-server algorithm [24, 19, 10].

1 Introduction

One of the central on-line problems is the k-server problem, first suggested by Manasse, McGeoch, and Sleator [23]. The problem is that of scheduling the motion of k identical servers that travel among the n points of a metric space as to satisfy requests for service at these points, while minimizing the total distance the servers travel. In the traditional setting communication is free, that is, there is a central algorithm that receives the requests and controls the motion of the servers, incurring no cost for the acquisition of the knowledge about all the requests and for the transmission of motion instructions to the servers.

In this paper we investigate a setting that we call the "costly-message" one, which represents the coordination efforts of dynamically-moving identical servers and of spread-around entities that issue requests for service, so as to satisfy these requests. The transfer of information between the participating parties is costly, and an algorithm for this setting should not only minimize the total distance the servers travel but also the cost it incurs for the transmission of messages. This models situations where the requests for service appear at the locations of service and not at some central control, e.g., when a set of mobile robots (AGV's) carrying tools or spare parts in a factory have to reach different work-cells upon the request of the operators of these work-cells. We assume that the requests are issued at the location of service, and thus there is no central control at which the request is received. Motivated by the fact that the problem is also related to problems of data management [9, 2] (and also mentioned in the context of the mobile user problem [6]), and in order to make our discussion more general, we will discuss the costly-message (distributed) setting in terms of networks and processors. The requests are issued by the processors, and information is passed by messages over network links. A distributed algorithm for this problem should react to a request at a given processor, and send messages and motion instructions that will result in the arrival of a server to the requesting processor.

Since in the distributed setting a k-server algorithm incurs some cost for the transmission of any message, the problem has the additional difficulty that the process of gathering information is costly, thus making it impossible (from a competitive point of view) to inform all processors of all requests. The algorithm has therefore to deal with incomplete information also as to the past, and not only as to the future. Moreover, it has to balance between incurring a cost to gather information, and incurring a cost due to "errors" caused by the lack of information as to the past. In particular, the algorithm has to proceed while not all processors know the location of all the servers.

The cost incurred by a distributed k-server algorithm is composed of the cost due to the travel of the servers, and the cost incurred for the transmission of control messages. Competitive analysis of distributed algorithms was introduced in the context of job scheduling [4] and data management [9, 1, 3]; the cost of a distributed on-line algorithm, including the cost incurred for the transmission of messages, is compared to the cost of an optimal global-control adversary, which does not transmit any control messages. Clearly a result of competitiveness against such adversary will hold against any other distributed algorithm as well.

In some special cases directly deriving distributed k-server algorithms from global-control ones is relatively easy. For the uniform metric space one can transform global-control deterministic algorithms (e.g. [21, 25]) to distributed ones by choosing a leader that will run the global-control algorithm, while ignoring requests on covered points; for the case of $k = n - 1$, one can use the algorithm of Borodin, Linial and Saks for metrical task systems [12] which defines a fixed traversal sequence for the single point which is uncovered at any specific time; only a single message of $O(1)$ bits has to be sent to the new uncovered point. In general, however, the behavior of known k-server algorithms depends on their entire history, making it impossible to directly transform them into distributed ones, as shipping the whole history is too costly.

In this paper we give competitive distributed k-server algorithms for all metric spaces and all k. This is done by building a general *translator* that transforms *any* deterministic competitive *global-control* k-server algorithm for any metric space into a competitive *distributed* one. Thus our result will be applicable to any improved algorithm for the k-server problem to be invented in the future as well. The increase in the competitive ratio is approximately the maximum between a small polynomial in k, the logarithm of the number of memory states of the global-control algorithm, and the competitive ratio of the k-server tracking mechanism (to be discussed later) which is at most $O(\mu \log n)$, where $\mu = \max\{\lceil \log n \rceil, \lceil \log \Delta \rceil\}$ (Δ is the diameter of the network), but is smaller for specific metric spaces. We get poly(k)-competitive distributed k-server algorithms for the line, trees and the ring, by applying our general result to the k-competitive global-control algorithms for the line and trees [13, 14] and the $O(k^3)$-competitive algorithm for the ring [18]. Applying it to the $O(1)$-competitive algorithms for the case of $k = 2$ [15, 20], gives $O(\mu \log n/D)$-competitive distributed algorithms, where D is the ratio between the cost of moving a server and of sending a message across the same distance. Applying our translator to the known deterministic k-server algorithms for general metric spaces [17, 22] is, of course, possible; however, due to the large number of memory states of these algorithms, this yields a distributed algorithm with a competitive ratio which is greater than that of the application of an algorithm for trees to a minimum spanning tree (as previously suggested [14]).

To show the inherent difference between the distributed k-server problem and the "traditional" (global-control) one, we show that there is an infinite number of networks with increasing number of processors n, such that the competitive ratio of any distributed k-server algorithm is $\Omega(\max\{k, \frac{1}{D} \cdot \log n / \log\log n\})$ (using also the lower bound of k [23]), thus depending also on the number of points in the metric space. This is in contrast to the global-control k-server problem where there are algorithms with competitive ratios that depend only on k (and not on n) for any metric space [17, 19, 22].

We also study the problem of devising a distributed version of the Harmonic k-server algorithm [24]. Harmonic is known to be $O(2^k \log k)$-competitive [10], but it is believed that the correct competitive ratio is $O(k^2)$ (see [24, 11]). If the competitive ratio of Harmonic is c_H, then Distributed-Harmonic is $O(c_H \cdot (1 + \frac{1}{D} \cdot \max\{k, \mu\} \cdot (\log \Delta) \log n))$-competitive. The Harmonic algorithm is of practical interest because of its simplicity and the fact that it is memoryless.

The paper is organized as follows: We start with the definition of the model and the problem in Section 2, while Section 3 contains the definitions of the server tracking mechanisms that we use. In Section 4 we give the general competitive distributed translator for k-server algorithms, and in Section 5 discuss its applications. Section 6 gives the distributed version of Harmonic. We conclude in Section 7 with the proof of the lower bound.

2 Model and Statement of Problem

A *network topology* $\mathcal{N}$ is an edge-weighted graph $G = (V, E)$, $|V| = n$. The vertices represent processors, and the weight of each edge represents the length of the link between the two adjacent processors (or equivalently the cost of traversing this link). The weighted graph need not obey the triangle inequality, but a natural metric space can be defined where its points are the vertices and the distance between two points is equal to the length of the shortest path between the vertices in the weighted graph. We denote by $d(a, b)$ the distance between a and b in this metric space. We use the terms network topology, weighted graph, and metric space as called for by the discussion, but they refer to the same underlying interconnection network.

We denote by Δ the diameter of the network topology, assuming the length of the shortest edge is of one unit. The cost of transmitting a $\max\{\lceil \log n \rceil, \lceil \log \Delta \rceil\}$-bit message is $\frac{1}{D}$ times the distance over which the message is sent. Restricting ourselves to the more usual model of $\lceil \log n \rceil$-bit messages will only add a factor of $\frac{\lceil \log \Delta \rceil}{\lceil \log n \rceil}$ to our results. Henceforth let $\delta = \lceil \log \Delta \rceil$, and

$\mu = \max\{\lceil \log n \rceil, \lceil \log \Delta \rceil\}$; μ is sometimes used to indicate the size of a unit-length message. We assume the existence of an underlying routing scheme, that given a message and a list of processors as addressees, routes the message in the most efficient way to reach all the addressees.

The k-server problem [23] is that of scheduling the motion of k identical servers in a metric space as to serve requests for service at the points of the metric space. At each request a server must be sent to the request point before the next request appears, therefore without knowledge of future requests. When a server reaches a request point we say that it *serves* the request. In this setting, that we call *global-control*, the requests are given to some central algorithm. The cost incurred by a global-control k-server algorithm is the total distance its servers travel. An on-line k-server algorithm $\mathcal{A}$ is c_k-competitive if for any initial configuration of the servers and any request sequence σ

$$\mathrm{Cost}_{\mathcal{A}}(\sigma) \leq c_k \cdot \mathrm{Cost}_{\mathrm{Adv}}(\sigma) + a \ ,$$

where $\mathrm{Cost}_{\mathrm{Adv}}$ is the cost of the off-line optimal algorithm and a is an additive term which is not a function of σ.

We will use the following definition to formulate any *global-control* on-line k-server algorithm:

Definition. An on-line k-server algorithm for a specific metric space is a function $f : Q \times M \times R \to Q \times M$, where Q is the set of possible placement configurations of the servers, M is the set of possible memory states of the algorithm, and R is the set of possible requests (the set of points in the metric space).

Note that when the metric space is of n points, then $|Q| \leq n^k$, and that the cardinality of M sometimes depends on the specific metric space that the algorithm is applied to, and on k. We refer to $Q \times M$ as the set of configurations of the algorithm. Any global-control on-line k-server algorithm whose memory size cannot grow unboundly can be formulated in this way.

The *distributed* k-server problem is that of scheduling k identical servers which reside among the processors of a network topology. At times, a request for service appears at a processor and one of the servers must reach that processor. A distributed on-line k-server algorithm is a set of programs, each running in a separate processor. When a request appears in a processor, the request is received by the program running in that processor. This program then sends control messages, possibly causing the transmission of messages by other processors, as to move a server to the request point. The cost of a *distributed* on-line k-server algorithm is the cost incurred for the travel of the servers plus the communication cost incurred for the transmission of any control messages the algorithm sends.

For competitive analysis, on-line distributed algorithms are compared to the global-control optimal adversary. A distributed on-line k-server algorithm $\mathcal{A}$ is c_k-competitive if there exists a value a such that for any request sequence σ, $\mathrm{Cost}_{\mathcal{A}}(\sigma) \leq c_k \cdot \mathrm{Cost}_{\mathrm{Adv}}(\sigma) + a$.

We do not consider here issues of concurrency and assume the requests are serialized.

3 Preliminaries

In our distributed k-server algorithms we will use mechanisms for tracking the locations of the servers. We use two types of mechanisms:

Definition. A c-competitive *k-server tracking* mechanism for tracking the k servers $s_1, s_2, \ldots, s_k$ in a network topology $\mathcal{N}$, that has an *approximation factor* λ, is a distributed algorithm that handles the following types of operations, listed with their respective costs: (1) A move of one of the servers, with an amortized cost of at most $\frac{1}{D} \cdot c$ times the distance moved. (2) The finding, by some processor p, of the location of a particular server, s_i, with an amortized cost of at most $\frac{1}{D} \cdot c$ times the distance from p to s_i. (3) The finding by some processor p, of the location of some *close* server, that is, a server whose distance from p is at most λ times the distance from p to the closest server, with an amortized cost of at most $\frac{1}{D} \cdot c$ times this distance.

The first two operations are the operations of the *mobile user* mechanism [6], and the last one is the *data tracking* [9] operation. Henceforth, let $T_{\mathcal{N},k}$ be the best competitive ratio for a k-server tracking mechanism on network $\mathcal{N}$. If a processor is allowed to have $O(k\delta \log n)$ bits of memory, then it follows from the constructions of [6, 9] that for every $\mathcal{N}$ and k, $T_{\mathcal{N},k} = O(\mu \log n)$ and λ is $O(1)$.

Obviously, for the line and the ring $T_{\mathcal{N},k} = O(1)$, and for trees $T_{\mathcal{N},k} = O(k)$ (even with $O(1)$ memory per processor), since when servers move they can leave track of their direction from each processor, and then can be found by a disjoint path traversal [8].

Definition. A c-competitive *generalized k-server tracking* mechanism is a k-server tracking mechanism that also allows every processor p to find all servers at distance at most d from it, at an amortized cost of at most $\frac{1}{D} \cdot c \cdot d$.

Let $\tilde{T}_{\mathcal{N},k}$ be the best competitive ratio for a generalized k-server tracking algorithm on network $\mathcal{N}$. One can use the constructions of [6, 9] to get this mechanism for every network $\mathcal{N}$ such that $\tilde{T}_{\mathcal{N},k} = O(\max\{k, \mu\} \cdot \log n)$.

4 A Translator to Make k-Server Algorithms Distributed

In this section we give a general translator to transform any global-control deterministic competitive k-server algorithm into a distributed one.

The main idea of the translator is as follows: The set of servers is recursively divided into two disjoint sets; the partition is done at each recursive level according to the minimum spanning tree of the servers, such that the maximum-length edge of this spanning tree separates the two sets. We denote the length of this edge by d and call it the *separation distance* since it follows that the distance between any two servers, each pertaining to a different such set, is at least d. Note that one can communicate between all the servers of the part over a distance of at most the number of servers in the part times d.

Each request is *handled* by a close server —a server whose distance from the request is within a constant factor of the distance to the closest one. This is not necessarily the server that *serves* the request. Each server locally simulates the underlying global-control k-server algorithm, starting from some initial algorithm configuration, and serving those requests that it *handles*. The server counts the cost incurred by the underlying global-control algorithm during this simulated run. Whenever this cost is greater than $\alpha \cdot d$, ($\alpha < 1$ to be determined later), where d is the separation distance of some part of the recursive partition, a *phase* of this part ends. The server that thus ended the phase forces upon all the servers of this part its own simulated algorithm configuration with its associated cost. All these servers then travel to conform to this configuration. It follows that the communication cost (incurred to transmit the configuration and the counted cost) is proportional to the cost incurred for the travel of the servers. Furthermore, the amount of travel each server performs during the phase is bounded by a function of the cost of the simulated run, and at the end of the phase all the servers are at the positions they would occupy had the *competitive* global-control algorithm been run on some mindfully chosen subsequence of the requests. We call this subsequence the *simulation sequence* of the phase. We get, therefore, a competitive distributed algorithm.

In what follows we precisely define the translator.

Given a global-control k-server algorithm $\mathcal{A} : Q \times M \times R \to Q \times M$, we build a distributed k-server algorithm $\mathrm{DS}_{\mathcal{A}}$. $\mathrm{DS}_{\mathcal{A}}$ is defined recursively and proceeds in phases. We define $\mathrm{DS}_{\mathcal{A}}(m, S)$, where S is a set of servers, to be the algorithm that controls the behavior of the servers in the set S. The cardinality of S is m, and is added to the notation for clarity of description. We refer to m as the *cardinality* of the algorithm. When the set of servers is obvious or irrelevant for the discussion we sometimes abuse notation and do not indicate it. Thus, the distributed k-server algorithm, based on the global-control one $\mathcal{A}$, is denoted $\mathrm{DS}_{\mathcal{A}}(k)$.

An algorithm of cardinality m consists of recursively running two separate algorithms of lesser cardinality for two disjoint sets of the m servers until a phase of $\mathrm{DS}_{\mathcal{A}}(m)$ ends. At the end of each phase we associate with the execution of the phase a subsequence of the request sequence handled by the servers of $\mathrm{DS}_{\mathcal{A}}(m)$ during the phase; this subsequence is the *simulation sequence* of the phase. Each server participates simultaneously in a number of algorithms with decreasing cardinality. It maintains a counter for each one of the algorithms it participates in, which we denote $C_{s,j}$, where s is the server number and j is the cardinality of the algorithm the counter is related to. These counters maintain—roughly speaking—the knowledge of the server about the cost of the simulated algorithm $\mathcal{A}$ since the beginning of the current phase, on the simulation sequence. In order to simulate the underlying global-control algorithm $\mathcal{A}$, each server maintains a variable with the simulated configuration of $\mathcal{A}$. It also maintains k variables, each holding an *estimated position* of the other servers, i.e., for each server, it maintains the position of that server when their last lowest-level common phase started. In addition it maintains a tree that describes the recursive partition of the k servers into sets.

For $m > 1$, $\mathrm{DS}_{\mathcal{A}}(m, S)$ is defined recursively as follows: Consider the minimum spanning tree (in the metric space) of the locations of the servers in S at the beginning of a phase, denote by I the maximum-length edge in this tree and denote its length by d. The set S is divided into two disjoint sets A and B ($|A| = m_A$, $|B| = m_B$, $m = m_A + m_B$), such that the edge I separates them in the minimum spanning tree. We call d the *separation distance* of the phase. It follows that for any $a \in A$ and $b \in B$, $d(a, b) \geq d$, and that the weight of the minimum spanning tree is at most $(m-1)d$. All servers of S update the local tree that describes the partition of the servers, and for each other server $x \in S$, update the variable that holds the estimated position of x. They initialize the C variables associated with $\mathrm{DS}_{\mathcal{A}}(t), t \leq m$ to 0, i.e., $C_{x,t} \leftarrow 0$ for every $x \in S$ and $t \leq m$. The simulated configuration held is not re-initialized at the beginning of each phase (when the main algorithm starts, all servers hold the real initial configuration). Two algorithms $\mathrm{DS}_{\mathcal{A}}(m_A, A)$ and $\mathrm{DS}_{\mathcal{A}}(m_B, B)$ are then started; these algorithms run separately until the following occurs:

In some server $s \in S$ $C_{s,m} > \alpha \cdot d$, where α is some constant less than 1 to be determined later.

At this event the current phase of $\mathrm{DS}_{\mathcal{A}}(m, S)$ terminates. We first define the *simulation sequence* of the phase and to do that we "terminate the phase according to s". This is defined recursively as follows: Assume without loss of generality that $s \in A$. First force the termination of the current phase of $\mathrm{DS}_{\mathcal{A}}(m_A, A)$, i.e., recursively "terminate the phase of $\mathrm{DS}_{\mathcal{A}}(m_A, A)$ according to s", and then define the simulation sequence to be the concatenation of the simulation sequences of all the phases of $\mathrm{DS}_{\mathcal{A}}(m_A, A)$ [1].

To end the phase the following *end-of-phase procedure* is performed: Server s sends to all other servers in S messages that include its simulated configuration with its counters. To do that s sends the messages to the positions held in its local variables as the estimated positions of the servers in S. From there, the k-servers tracking mechanism is used to exactly locate the servers and transmit the messages to them. When a server receives this message it overrides its simulated configuration with the simulated configuration of $\mathcal{A}$ as maintained by s, and overrides all its C counters with the values that the corresponding variables in s hold. Thus the counters now reflect the cost of $\mathcal{A}$ on the simulation sequence. It also updates its local variables that hold estimated positions of the servers of S to be according to the current simulated configuration. Finally, the server travels in order to conform to the

[1] Note that this description is meant for the definition of the simulation sequence only; no work is performed.

current simulated configuration.

For $m = 1$ (the base of the recursion), a phase of $\mathrm{DS}_{\mathcal{A}}(m, S)$, $S = s$, starts by initializing the variable $C_{s,1}$ to 0 and consists of a single request handled by its server, the *simulation sequence* of the phase being this single request. At the end of a phase (that is, after handling a single request) the server travels to conform to its location according to its simulated configuration [2].

Upon the appearance of a request, the processor receiving the request uses the k-server tracking mechanism to locate a close server, that is, a server which is at distance within a constant factor of the distance to the closest one. Denote this server by s. Server s *handles* the request (but not necessarily *serves* it): It updates the simulated configuration it holds according to the request (using the underlying algorithm $\mathcal{A}$), and adds the cost thus incurred by $\mathcal{A}$ to $C_{s,l}, l \geq 1$. Then, going from the highest-level algorithm towards the lowest-level one, s checks if the condition that defines the end of a phase has occurred, and ends the phase of the corresponding algorithm.

Note that the updates performed at end-of-phases ensure that at the end of phases all the counters of the servers of S are equal, and all these servers have the same simulated configuration. They also agree, therefore, on the partition of the servers into disjoint sets for the next phase.

4.1 Proof of Correctness and Analysis

In the above description we do not explicitly show which server *serves* each request, and in particular it is not obvious that each request is served when it appears. We will prove shortly, in Lemma 4, that $\mathrm{DS}_{\mathcal{A}}(m, S)$ is a valid m-server algorithm. Before doing that we need some definitions and several technical claims.

We call the server according to which the phase of $\mathrm{DS}_{\mathcal{A}}(m, S)$ terminates, the *terminating server*. At any time, the server $s \in S$, such that $C_{s,m} = \max_{x \in S}\{C_{x,m}\}$, is called the *maximum server* of S. Note that at the time of an end of a phase the terminating server is the maximum server. We also consider an imaginary run of $\mathcal{A}$ defined as follows: the initial server configuration is defined by the servers positions at the beginning of the phase.

[2]Note that the variables $C_{s,1}$ are redundant. We describe the algorithm for $m = 1$ as above to have a description similar to the description of the algorithm for $m > 1$.

The algorithm is initialized with a memory state equal to the memory state that the servers of S hold at the beginning of the phase. The sequence of requests served by $\mathcal{A}$ in this run is the *simulation sequence*. We call this run the *simulation run*. In what follows, a variable with a superscript (e.g., X^t) indicates the value of the variable X at the time t.

Claim 1 *Let $x \in S$ be a server and $\mathrm{DS}_{\mathcal{A}}(m, S)$ an algorithm of cardinality m. Let $l < m$ be such that there is an algorithm $\mathrm{DS}_{\mathcal{A}}(l, S')$ where $S' \subset S$, and $x \in S'$. Consider a phase of $\mathrm{DS}_{\mathcal{A}}(l, S')$, let t_0 be its starting time, and t_f its termination time. Then $C^{t_f}_{x,m} - C^{t_0}_{x,m} = C^{t_f}_{x,l}$.*

Proof. We prove the claim by induction on the cardinality of the algorithm that ended the phase, l.

For $l = 1$, the definition of the algorithm implies the claim.

For $l > 1$ denote by s the terminating server of the phase of $\mathrm{DS}_{\mathcal{A}}(l, S')$. Consider the phases of levels lower than l in which s participates during the phase of $\mathrm{DS}_{\mathcal{A}}(l, S')$. Then we can partition the time of this phase into non-overlapping phases of lower levels. That is, we first pick g times $t_1 < t_2 < \ldots < t_g$, which are the ending times of the phases of the highest-cardinality $k < l$ algorithm in which s participates. Then for the remaining time (which is the time of the last unterminated phase of cardinality k) we take g' times $t_{g+1} < t_{g+2} < \ldots < t_{g+g'}$, which are the termination times of the phases of the highest cardinality $k' < k$ in which s participates. We continue this partition taking phases of lower levels until we reach the time t_f. Since s is the terminating server of the phase under consideration, then the phase of the lowest level (of one server) terminates at the end of the phase of $\mathrm{DS}_{\mathcal{A}}(l, S')$, and we can partition the time frame into non-overlapping ended phases.

We get a sequence of $G + 1$ times $t_0 < t_1 < t_2 < \ldots < t_G = t_f$. For each $i \geq 1$ there is a phase of some algorithm of cardinality $c < l$, s participates in this algorithm, and the phase started at time t_{i-1} and ended at t_i. Now, if we consider such ended phase of cardinality $c < l$, by the induction hypothesis we have $C^{t_i}_{s,c} = C^{t_i}_{s,l} - C^{t_{i-1}}_{s,l}$. At the same time $C^{t_i}_{s,c} = C^{t_i}_{s,m} - C^{t_{i-1}}_{s,m}$, and thus $C^{t_i}_{s,l} - C^{t_{i-1}}_{s,l} = C^{t_i}_{s,m} - C^{t_{i-1}}_{s,m}$. Summing over all t_i we get that $C^{t_f}_{s,m} - C^{t_0}_{s,m} = C^{t_f}_{s,l} - C^{t_0}_{s,l}$. Since $C^{t_0}_{s,l} = 0$, we have $C^{t_f}_{s,m} - C^{t_0}_{s,m} = C^{t_f}_{s,l}$. Since t_0 and t_f are the starting and ending times of a phase of $\mathrm{DS}_{\mathcal{A}}(l, S')$, and $x \in S'$ and $s \in S'$, we have that $C^{t_0}_{s,m} = C^{t_0}_{x,m}$, $C^{t_f}_{s,m} = C^{t_f}_{x,m}$, and $C^{t_f}_{s,l} = C^{t_f}_{x,l}$. The claim follows. □

Claim 2 *For any phase of* $\mathrm{DS}_{\mathcal{A}}(m, S)$ *the following hold:*

1. *At the end of the phase all the servers in* S *hold as simulated configurations the configuration of* $\mathcal{A}$ *at the end of the simulation run.*

2. *Let* s *be the* terminating server. *The cost of* $\mathcal{A}$ *during the* simulation run *is equal to the value* $C^t_{s,m}$ *where* t *is the time at which the phase ends.*

Proof. We prove the claim by induction on m. For $m = 1$ both points are obvious.
For $m > 1$ assume that the claim holds for any $l < m$ and assume without loss of generality that the terminating server $s \in A$.

Consider the sequence of phases of $\mathrm{DS}_{\mathcal{A}}(m_A, A)$ which are executed during the phase of $\mathrm{DS}_{\mathcal{A}}(m, S)$. Denote by p the number of such phases in that sequence (including the forcibly terminated last phase). Let Q^i be the configuration held by the servers of A at the beginning of the i'th phase of $\mathrm{DS}_{\mathcal{A}}(m_A, A)$, and let σ^i be the simulation sequence of this phase. Then, by the induction hypothesis, at the end of the i'th phase of $\mathrm{DS}_{\mathcal{A}}(m_A, A)$, the servers of A hold as simulated configurations the configuration of algorithm $\mathcal{A}$, when run starting with configuration Q^i on the sequence σ^i. Denote this configuration by E^i. Since the servers in A start a phase of $\mathrm{DS}_{\mathcal{A}}(m_A, A)$ with the simulated configuration of the end of the previous phase we have $Q^{i+1} = E^i$. It follows that at the end of the last (p'th) phase of $\mathrm{DS}_{\mathcal{A}}(m_A, A)$ in the sequence, the servers of A hold as simulated configurations the configuration of $\mathcal{A}$, if run starting with configuration Q^1 on the concatenation of the sequences σ^i, $1 \leq i \leq p$. But Q^1 is the simulated configuration at the beginning of the phase of $\mathrm{DS}_{\mathcal{A}}(m, S)$, and the concatenation of the σ^i sequences is the simulation sequence of the phase of $\mathrm{DS}_{\mathcal{A}}(m, S)$. Thus E^p is the simulated configuration of the end of the phase of $\mathrm{DS}_{\mathcal{A}}(m, S)$. Since the terminating server is in A, E^p is forced as a simulated configuration on all servers of S. This ends the proof of point 1.

The cost of $\mathcal{A}$ during the simulation run of the phase of $\mathrm{DS}_{\mathcal{A}}(m, S)$ is equal to the sum of the costs incurred in each of the phases of $\mathrm{DS}_{\mathcal{A}}(m_A, A)$. Let t_i denote the time at which the i'th phase of $\mathrm{DS}_{\mathcal{A}}(m_A, A)$ has ended, and let s_i be the terminating server of this phase. Note that $t = t_p$. Denote by C^i_A the value $C^{t_i}_{s_i, m_A}$. By the induction hypothesis, C^i_A is the cost of $\mathcal{A}$ on the simulation run of this phase. Thus the cost of $\mathcal{A}$ on the simulation run of the phase of $\mathrm{DS}_{\mathcal{A}}(m,S)$ is $\sum_{i=1}^{p} C_A^i$. According to the update procedure performed at the end of the i'th phase of $\mathrm{DS}_{\mathcal{A}}(m_A,A)$, we have that $C_{s,m_A}^{t_i} = C_{s_i,m_A}^{t_i} = C_A^i$. It follows from Claim 1 that $C_{s,m}^{t_i} - C_{s,m}^{t_{i-1}} = C_{s,m_A}^{t_i}$, and thus $\sum_{i=1}^{p} C_A^i = C_{s,m}^t$, proving point 2. □

Claim 3 *Consider a certain time t during a phase of* $\mathrm{DS}_{\mathcal{A}}(m,S)$*, and a server $s \in S$. Let z be the position of s at time t, and p its position at the beginning of the phase. Then, $d(z,p) \le C_{s,m}^t$.*

Proof. The claim holds at the beginning of the phase. Server s performs motions only at end-of-phase procedures, and these are also the only times when $C_{s,m}$ is updated. Thus we will show that the claim holds by inductively considering the events when s participates in an end-of-phase procedure.

Consider a typical end of phase of $\mathrm{DS}_{\mathcal{A}}(l,S')$, where $s \in S'$ and $S' \subset S$, and let x be the terminating server of the phase. We will compare the position of s and the value of $C_{s,m}$ after the end-of-phase procedure to those at the beginning of the phase of $\mathrm{DS}_{\mathcal{A}}(l,S')$.

Let t_0 denote the time at which the phase begins and let t_f be the time at which the phase ends. Let D be the distance between the positions of s according to the simulated configurations as held by x at time t_0 and at time t_f. At the beginning of the phase of $\mathrm{DS}_{\mathcal{A}}(l,S')$ both s and x held the same simulated configuration, and s was located at p according to this simulated configuration. Thus, during the end-of-phase procedure s moves away from p at most distance D. By Claim 2 $C_{x,l}^{t_f}$ holds the cost of $\mathcal{A}$ on the simulation sequence of the phase of $\mathrm{DS}_{\mathcal{A}}(l,S')$. Since this cost is at least D, it follows that $D \le C_{x,l}^{t_f}$.

By Claim 1, $C_{x,m}^{t_f} - C_{x,m}^{t_0} = C_{x,l}^{t_f}$. It follows from the counter updating procedure at the end of a phase that $C_{s,m}^{t_0} = C_{x,m}^{t_0}$ and $C_{s,m}^{t_f} = C_{x,m}^{t_f}$. Therefore $C_{s,m}^{t_f} - C_{s,m}^{t_0} = C_{x,l}^{t_f} \ge D$. □

Next we prove the validity of the algorithm.

Lemma 4 *Each request is served while handling it. (I.e, a server reaches the request point).*

Proof. Assume server s_1 handles a request at point r, and according to the simulation of $\mathcal{A}$ it performs, s_2 is the server that serves it. Let $\mathrm{DS}_{\mathcal{A}}(m,S)$ be the lowest-level algorithm such that $s_1 \in S$ and $s_2 \in S$. Let d be the separation distance of the current phase of this algorithm. We show that the current phase of $\mathrm{DS}_{\mathcal{A}}(m, S)$ terminates while handling the request and thus s_2 will travel to r. If $s_1 = s_2$, then the lowest level algorithm is $\mathrm{DS}_{\mathcal{A}}(1)$, and the claim is obvious. Assume, therefore, that $s_1 \neq s_2$.

For $i = 1, 2$ let z_i denote the position of server s_i when the request appears, and p_i its position at the beginning of the current phase of $\mathrm{DS}_{\mathcal{A}}(m, S)$. Since $d(p_1, p_2) \geq d$, and using the triangle inequality

$$d(p_1, z_1) + d(z_1, r) + d(r, z_2) + d(z_2, p_2) \geq d \ .$$

We have that $d(p_i, z_i) \leq \alpha \cdot d$, otherwise the phase would have terminated earlier. Since s_1 was located by the k-server tracking mechanism, we also have that $d(r, z_1) \leq \lambda d(r, z_2)$. Therefore,

$$d(r, z_2) \geq d - 2\alpha \cdot d - d(z_1, r) \geq d - 2\alpha \cdot d - \lambda d(r, z_2) \ .$$

We get, $d(r, z_2) \geq d \frac{1-2\alpha}{1+\lambda} > 2\alpha \cdot d$ (for $\alpha < \frac{1}{2\lambda+4}$). By the triangle inequality,

$$d(r, p_2) \geq d(r, z_2) - d(p_2, z_2) \ ,$$

and therefore

$$d(r, p_2) > \alpha \cdot d \ ,$$

and when the cost incurred by $\mathcal{A}$ to serve the request at r according to the simulation is added to the counter, the phase terminates. □

We now turn to proving the bound on the competitive ratio of the algorithm. We analyze the cost of the algorithm in several separate parts: its cost for the transmission of messages during end-of-phase procedures, its cost for moving the servers, the cost for the transmission of messages that follow servers, and the cost incurred for the use of the k-server tracking mechanism.

We call the messages sent during the *end-of-phase procedure*, in order to override the simulated algorithm configuration and the counters, *control messages of the end-of-phase procedure.*

Lemma 5 *Consider a phase of* $\mathrm{DS}_{\mathcal{A}}(m, S)$. *Let s be the terminating server of the phase and let t the time at which the phase ends. The control messages of the end-of-phase procedure are sent over a total distance of at most* $2m \frac{1}{\alpha} C^t_{s,m}$.

Proof. When the end-of-phase procedure starts, i.e., *before* the travel of the servers is performed, the terminating server sends the control messages to the estimated positions of the servers of S. These positions must be in the $\alpha \cdot d$-neighborhoods of the positions of the respective servers at the beginning of the phase, otherwise the phase would have terminated before. The real positions of the servers must be in the $\alpha \cdot d$-neighborhoods of the estimated ones, for the same reason. There is a tree, with weight at most $(m-1)d$, that spans the positions of the servers at the beginning of the phase. Therefore, the total distance over which the messages are sent is at most $(m-1)d+2m\alpha d$. Since at the end of the phase $C^t_{s,m} > \alpha \cdot d$, this distance is at most $2m\frac{1}{\alpha}C^t_{s,m}$ (for $\alpha < 1/2$). □

At the end of the phase, the terminating server, s, communicates to all the servers in S the following information: its counters $C_{s,l}, l \geq m$, (there are at most k such counters), the positions of all k servers according to its simulation of $\mathcal{A}$, and the memory state of $\mathcal{A}$ consisting of $\lceil \log |M| \rceil$ bits. In addition a list of addressees is required. In order to know in which processor each server is, we use the k-server tracking mechanism from each server's estimated position. The distance over which this mechanism is used is at most the distance over which the control messages are sent. This cost is accounted for in the final analysis of the competitive ratio.

Lemma 6 *Consider a certain time t during a phase of $\mathrm{DS}_{\mathcal{A}}(m)$. Let s be the maximum server at time t. Then, since the beginning of the phase up to time t, $\mathrm{DS}_{\mathcal{A}}(m)$ has sent control messages of end-of-phase procedures over a total distance of at most $O(m^2 C^t_{s,m})$.*

Proof. We prove by induction on m that $\mathrm{DS}_{\mathcal{A}}(m)$ sends such control messages over a total distance of at most $2\frac{1}{\alpha}m^2 C^t_{s,m}$.
For $m = 1$ no control messages of end-of-phase procedures are sent.
For $m > 1$ assume that the claim holds for any $l < m$. A phase of $\mathrm{DS}_{\mathcal{A}}(m)$, $m > 1$, consists of phases of $\mathrm{DS}_{\mathcal{A}}(m_A, A)$ and $\mathrm{DS}_{\mathcal{A}}(m_B, B)$ run separately. Assume without loss of generality that $s \in A$ and that p phases of $\mathrm{DS}_{\mathcal{A}}(m_A, A)$ have already ended. Note that $C^t_{s,m} = \sum_{i=1}^{p} C^i_A + C^t_{s,m_A}$, where C^i_A is the value of C_{s_i,m_A} at the end of the i'th terminated phase of $\mathrm{DS}_{\mathcal{A}}(m_A, A)$, s_i is the terminating server of this phase. Since at the end of each phase the terminating server is also the maximum server, by the induction hypothesis we know that $\mathrm{DS}_{\mathcal{A}}(m_A, A)$ has sent control messages

during its i'th phase over a total distance of at most $2\frac{1}{\alpha}(m_A)^2C^i_A$. In addition, since $C^t_{s,m} = \max_{x\in A}\{C^t_{x,m}\}$, the same induction hypothesis implies that during its current non-terminated phase $\mathrm{DS}_{\mathcal{A}}(m_A, A)$ has sent control messages over a distance of at most $2\frac{1}{\alpha}(m_A)^2C^t_{s,m_A}$. Thus, $\mathrm{DS}_{\mathcal{A}}(m_A, A)$ has sent control messages during the current phase of $\mathrm{DS}_{\mathcal{A}}(m)$ over a total distance of at most $2\frac{1}{\alpha}(m_A)^2C^t_{s,m}$. Let s' be the the maximum server of B, then by similar arguments applied to s', we have that $\mathrm{DS}_{\mathcal{A}}(m_B, B)$ has sent control messages over a total distance of at most $2\frac{1}{\alpha}(m_B)^2C^t_{s',m}$. But as s is the maximum server of S, $C^t_{s,m} \geq C^t_{s',m}$ and this is at most $2\frac{1}{\alpha}(m_B)^2C^t_{s,m}$. As $m^2 = (m_A + m_B)^2 \geq (m_A)^2 + (m_B)^2$, for any time t before the end of the phase of $\mathrm{DS}_{\mathcal{A}}(m)$, the claim follows from the argument that all communication done during the phase is done by lower level algorithms.

For time t which is the end of the phase, we have that when the end-of-phase procedure is performed, the terminating server is also the maximum server. By Lemma 5 the algorithm sends at this time messages over a distance of at most $2m\frac{1}{\alpha}C^t_{s,m}$. Therefore the total distance over which control messages have been sent since the beginning of the phase is at most

$$2\frac{1}{\alpha}(m_A)^2C^t_{s,m} + 2\frac{1}{\alpha}(m_B)^2C^t_{s,m} + 2m\frac{1}{\alpha}C^t_{s,m} \leq 2\frac{1}{\alpha}m^2C^t_{s,m} \ .$$

$\square$

Lemma 7 *Consider a phase of* $\mathrm{DS}_{\mathcal{A}}(m, S)$. *Let s be the terminating server of the phase, and let t the time at which the phase ends. Then, the total distance traveled by the servers in S during the end-of-phase procedure is at most* $2mC^t_{s,m}$.

Proof. At the end of the phase $\mathrm{DS}_{\mathcal{A}}(m, S)$ instructs the servers in S to travel to their respective positions according to the configuration of $\mathcal{A}$ maintained by s. To do this the servers can travel back to their positions of the beginning of the phase, traveling a distance of at most $\alpha \cdot d$, and then travel to their respective new locations, a total distance of at most $C^t_{s,m}$. Therefore, the total distance traveled is at most $m \cdot \alpha \cdot d + C^t_{s,m}$. Since $C^t_{s,m} > \alpha \cdot d$, this is at most $2mC^t_{s,m}$. $\square$

Lemma 8 *Consider a certain time t during a phase of* $\mathrm{DS}_{\mathcal{A}}(m)$. *Let s be the maximum server at t. Then the servers in S have traveled since the beginning of the phase up to time t a total distance of at most* $O(m^2C^t_{s,m})$.

This lemma follows from arguments similar to those in the proof of Lemma 6, using Lemma 7 to give an upper bound on the travel done during the end-of-phase procedure.

Theorem 9 *Algorithm* $\mathrm{DS}_{\mathcal{A}}(k,S)$ *is* $O(c_k \cdot k^2 \cdot (1+\frac{1}{D}\cdot \max\{k, T_{\mathcal{N},k}, \log|M|/\mu\}))$ *-competitive if* $\mathcal{A}$ *is* c_k*-competitive.*

Proof. To analyze $\mathrm{DS}_{\mathcal{A}}(k)$ we concatenate the simulation sequences of its terminated phases and sum the values of $C_{s_i,k}$ at the end of each phase, s_i being the terminating server of the i'th phase. We then force the termination of the last unfinished phase according to the maximum server at this time. This can be thought of as if $\mathrm{DS}_{\mathcal{A}}(k,S)$ is run as one of the two lower level algorithms of some imaginary algorithm $\mathrm{DS}_{\mathcal{A}}(k+1)$, that never ends a phase. We then terminate the phase according to the maximum server (s) at the time the algorithm halts, and consider the value of $C_{s,k+1}$ together with the simulation sequence of this only phase of $\mathrm{DS}_{\mathcal{A}}(k+1)$.

Summing up the cost of $\mathrm{DS}_{\mathcal{A}}(k)$ we have:

- The cost incurred for the travel of servers is at most $O(k^2 \cdot C_{s,k+1})$ by Lemma 8.
- The cost incurred for the transmission of control messages that follow the travel of servers is, therefore, at most $O(k^2 \cdot C_{s,k+1} \cdot \frac{1}{D} \cdot (k+\log|M|/\mu))$. This is because each server maintains $O(k)$ counters of size δ bits, k simulated positions of servers having $\lceil \log n \rceil$ bits, a memory state of $\mathcal{A}$ of size $\lceil \log |M| \rceil$ bits and a tree describing the partition of the servers and their estimated positions of size $O(k \log n)$ bits.
- By Lemma 6 the communication cost incurred for the transmission of control messages of end-of-phase procedures is at most $O(k^2 \cdot C_{s,k+1} \cdot \frac{1}{D} \cdot (k + \log|M|/\mu))$, since the size of any such message sent by $\mathrm{DS}_{\mathcal{A}}(k)$ is $O(k \cdot \mu + \log|M|)$ bits.
- The cost incurred for the use of the k-server tracking mechanism is at most $O(k^2 \cdot C_{s,k+1} \cdot T_{\mathcal{N},k} \cdot \frac{1}{D})$. This is because we use the k-server tracking mechanism to find a close server over a total distance which is at most the total distance the servers travel, and to find a specific server over a total distance which is at most the total distance over which we send control messages of end-of-phase procedures.

- An additional cost is incurred once for the transmission of the positions of all the servers when the algorithm starts; we denote this cost by a'.

We denote by σ the request sequence served by $\mathrm{DS}_{\mathcal{A}}(k)$ and by σ' the simulation sequence of the phase of $\mathrm{DS}_{\mathcal{A}}(k+1)$. Using Lemma 2 and since $\mathcal{A}$ is c_k-competitive,

$$C_{s,k+1} = \mathrm{Cost}_{\mathcal{A}}(\sigma') \leq c_k \cdot \mathrm{Cost}_{\mathrm{Adv}}(\sigma') + a \ ,$$

where a is an additive term which is not a function of σ'.

The total cost of $\mathrm{DS}_{\mathcal{A}}(k)$ on σ is

$$O(k^2 \cdot C_{s,k+1} \cdot (1 + \frac{1}{D} \cdot \max\{k, T_{\mathcal{N},k}, \log|M|/\mu\})) + a' \ ,$$

therefore,

$$\mathrm{Cost}_{\mathrm{DS}_{\mathcal{A}}(k)}(\sigma) \leq O(k^2 \cdot (1 + \frac{1}{D} \cdot \max\{k, T_{\mathcal{N},k}, \log|M|/\mu\})) \cdot (c_k \cdot \mathrm{Cost}_{\mathrm{Adv}}(\sigma') + a) + a' \ .$$

Clearly $\mathrm{Cost}_{\mathrm{Adv}}(\sigma') \leq \mathrm{Cost}_{\mathrm{Adv}}(\sigma)$, and $\mathrm{DS}_{\mathcal{A}}(k)$ is $O(c_k \cdot k^2 \cdot (1 + \frac{1}{D} \cdot \max\{k, T_{\mathcal{N},k}, \log|M|/\mu\}))$-competitive. □

5 Applications of the Translator

We use discrete versions of the k-competitive Double-Coverage algorithm for the line [13], the k-competitive algorithm for trees [14], and the $O(k^3)$-competitive algorithm for the ring [18]. In these discrete versions a modified metric space is created by adding points at distance of one unit apart along the edges of the metric space, rounding their lengths upwards. Applying the algorithm to this modified metric space increases the competitive ratio by at most a factor of 2, while the servers are always located at one of the points of the modified metric space. The number of memory states of the discrete algorithms is thus at most $(n\Delta)^k$ times the number of memory states of the original algorithm. Together with the best known k-server tracking mechanisms we obtain:

Corollary 10 *There are competitive distributed k-server algorithms with competitive ratios $O(k^3 + k^4/D)$ for the line, $O(k^3 + k^4/D)$ for trees and $O(k^5 + k^6/D)$ for the ring.*

Using the known algorithms for the case of 2 servers [15, 20] we get:

Corollary 11 *There are competitive distributed 2-server algorithms with competitive ratio $O(T_{\mathcal{N},2}/D)$.*

As noted in the introduction, it is possible to apply our translator to known algorithms of general metric spaces [17, 22]. However, due to the large number of memory states used by these algorithms, this yields competitive ratios which are greater than the one obtained by applying the above algorithms for trees, on a minimum spanning tree of the metric space (as previously suggested [14]).

6 Randomized Algorithms

The Harmonic k-server algorithm, proposed by Raghavan and Snir [24], is the algorithm that, given a request at point r, chooses each server to serve the request with probability inversely proportional to its distance to r. Specifically, if the locations of the on-line servers are $s_1, s_2, \ldots, s_k$, and $N = \sum_{i=1}^{k} \frac{1}{d(s_i,r)}$, then server s_i is moved with probability $\frac{1}{d(s_i,r)}/N$. Harmonic is known to be $O(2^k \log k)$-competitive [10]. It has however been conjectured ([24, 11]) that the correct competitive ratio is $O(k^2)$. Let c_H denote the competitive ratio of Harmonic. We present a distributed version of the Harmonic algorithm with a small increase in the competitive ratio.

The Distributed-Harmonic Algorithm

Upon a request initiated by processor r, the algorithm proceeds in phases for each $1 \leq i \leq \delta$, where each phase occurs with some absolute probability $\tilde{p}_{i-1}$ ($\tilde{p}_0 = 1$). The algorithm uses a generalized k-server tracking mechanism to find all the servers at distance at most 2^i from r. If no such server exists then the algorithm proceeds to the next phase. Otherwise, let m_i be the number of servers at r's 2^i-neighborhood, and order the servers, $s_1, s_2, \ldots, s_{m_i}$ by increasing distance from r. Let $N_i = \sum_{j=1}^{m_i} \frac{1}{d(s_j,r)}$, and let $\tilde{M}_i = \frac{k-m_i}{2^i}$. Let $\tilde{p}_i = \tilde{M}_i/(N_i + \tilde{M}_i)$; note that $\tilde{p}_i$ is a function of the locations of the closest m_i servers to r, and therefore can be computed by Distributed-Harmonic in the i'th phase. With absolute probability $1 - \tilde{p}_i$ choose each server s_j, $1 \leq j \leq m_i$, with probability $\frac{1}{d(s_j,r)}/N_i$ to move to the request point, i.e., complete the coin tosses already made in previous phases to achieve this probability. (A technical computation shows that the resulting conditional probability to move s_j, for $j \le m_{i-1}$, is: $\tilde{q}_{i,j} = \frac{1}{d(s_j,r)\tilde{p}_{i-1}}(\frac{1-\tilde{p}_i}{N_i} - \frac{1-\tilde{p}_{i-1}}{N_{i-1}})$.) With absolute probability $\tilde{p}_i$ the algorithm just proceeds with the next phase.

Analysis

The Distributed-Harmonic algorithm is defined so that it moves each server to serve the request with the same probability that the global-control Harmonic algorithm moves it. To do that, it maintains the invariant that after each phase the absolute probability that each server is moved, is a lower bound on the probability that the global-control Harmonic algorithm would have moved it. The probability that the algorithm proceeds to another phase after the i'th phase is $\tilde{p}_i = \tilde{M}_i/(N_i + \tilde{M}_i)$. Assume all k servers are found in the t'th phase. Then $\tilde{p}_t = 0$, and since $N_t = N$, the probability that s_j, $1 \le j \le k$, serves the request by the last phase, reaches that of the global-control Harmonic: $\frac{1}{d(s_j,r)}/N$. It is left to show that for each server s_j, $1 \le j \le k$, the probability that it is moved by the i'th phase, $0 \le i \le t$, is increasing. This probability is:

$$\tilde{z}_{i,j} = (1 - \tilde{p}_i)\frac{1}{d(s_j,r)}/N_i = 1/(N_i + \tilde{M}_i) \cdot \frac{1}{d(s_j,r)} \ .$$

For all $0 \le i \le t-1$, since for all $m_i + 1 \le j \le k$, $d(s_j,r) > 2^i$, we have:

$$\begin{aligned} N_i + \tilde{M}_i &= \sum_{j=1}^{m_i} \frac{1}{d(s_j,r)} + \frac{k-m_i}{2^i} \\ &\ge \sum_{j=1}^{m_{i+1}} \frac{1}{d(s_j,r)} + \frac{k-m_{i+1}}{2^{i+1}} = N_{i+1} + \tilde{M}_{i+1} \ , \end{aligned}$$

and therefore $\tilde{z}_{i,j} \le \tilde{z}_{i+1,j}$.

The expected cost for the (global-control) Harmonic algorithm on the request is:

$$\mathrm{E}[\mathrm{Cost}_{\mathrm{H}}] = \sum_{i=1}^{k} \frac{\frac{1}{d(s_j,r)}}{N} \cdot d(s_j,r) = \frac{k}{N} \ .$$

The expected cost of Distributed-Harmonic for moving the servers is the same. The messages sent by the generalized k-server tracking mechanism when the servers move add an (amortized) overhead of $\tilde{T}_{\mathcal{N},k}$ along the distance traveled by the servers (k/N) giving a total cost of $\mathrm{E}[\mathrm{Cost}_{\mathrm{DH}}{}^1] = \frac{k}{N}(1 + \frac{\tilde{T}_{\mathcal{N},k}}{D})$.

The search messages sent to find the servers in the different phases of the algorithm give an (amortized) overhead of $\tilde{T}_{\mathcal{N},k}$ to the search distance:

$$\begin{aligned}
\mathrm{E}[\mathrm{Cost_{DH}}^2] &= \frac{1}{D}\sum_{i=0}^{\delta-1} \tilde{p}_i \cdot \tilde{T}_{\mathcal{N},k} \cdot 2^{i+1} \\
&= \frac{1}{D}\sum_{i=0}^{\delta-1} \frac{\tilde{M}_i}{N_i + \tilde{M}_i} \cdot \tilde{T}_{\mathcal{N},k} \cdot 2^{i+1} \ .
\end{aligned}$$

Since $\tilde{M}_i = \frac{k-m_i}{2^i}$ and $N_i + \tilde{M}_i \geq N_t + \tilde{M}_t = N$, we obtain:

$$\begin{aligned}
\mathrm{E}[\mathrm{Cost_{DH}}] &= \mathrm{E}[\mathrm{Cost_{DH}}^1] + \mathrm{E}[\mathrm{Cost_{DH}}^2] \\
&\leq \frac{k}{N}(1 + \frac{\tilde{T}_{\mathcal{N},k}}{D}) + \frac{1}{D}\sum_{i=0}^{\delta-1} \frac{k}{N} \cdot \tilde{T}_{\mathcal{N},k} \cdot 2 \\
&\leq (1 + \frac{1}{D} \cdot 3\delta\tilde{T}_{\mathcal{N},k}) \cdot \mathrm{E}[\mathrm{Cost_H}] \ .
\end{aligned}$$

We conclude with the following theorem:

Theorem 12 *The Distributed-Harmonic k-server algorithm is $O(c_H \cdot (1 + \frac{1}{D} \cdot \delta\tilde{T}_{\mathcal{N},k}))$-competitive, against adaptive on-line adversaries.*

Using results on the mobile user and data tracking problems [6, 9] we get:

Corollary 13 *The Distributed-Harmonic algorithm is $O(c_H \cdot (1 + \frac{1}{D} \cdot \max\{k, \mu\}\delta \log n))$-competitive, against adaptive on-line adversaries, for arbitrary network topologies.*

7 Lower Bound

In contrast to the global-control case, where k-server algorithms with competitive ratio that depends solely on k exist for all metric spaces [17, 22], this is not the case in the distributed setting. In the distributed setting at least one message has to be sent over the distance between the requesting processor and the server that serves it (to "find" the server). We thus use a lower bound [5] for the mobile user problem [6], and a reduction, that we present below, from this problem to the distributed k-server problem. This reduction preserves the competitive ratio up to a factor of $O(D)$, and thus gives us a lower bound for our problem. We first define below the Mobile User problem [6].

7.1 The Mobile User Problem

Given a network topology $\mathcal{N}$, and a user U that at each time resides in one of the processors of the network, an algorithm for the problem has to deal with the following requests. A *Move(v)* request, issued at the node, u, in which the user currently is, should result in the User being moved from node u to node v. A *Find*(v) request issued at some node v should result in a message sent to the current location of the user. The cost incurred by the algorithm is the cost incurred for the transmission of messages. The competitive ratio is defined as the ratio between the cost incurred by the algorithm, and the sum of the inherent costs of the requests. For a *move* request the inherent cost is the cost to send a message to the new location of the server from its old location. For a *find* request the inherent cost is the cost to send a message to the location of the user, from the processor that issues the find request.

7.2 The Reduction

Given a network topology $\mathcal{N}$, we create an imaginary network $\mathcal{N}'$ which is the original network plus $k-1$ nodes, all connected to one of the nodes of the original network, denoted by I. This node will simulate the actions of the $k-1$ imaginary nodes. Given a c-competitive distributed k-server algorithm for $\mathcal{N}'$, we build an algorithm for the mobile user problem for $\mathcal{N}$ by generating a sequence of requests for service for each mobile user request, as follows.

Initially $k-1$ servers are placed at the $k-1$ imaginary nodes, and an additional server at the location of the user.

Move(v) A "Move(v)" request to move the user to node v is simulated by a request for service at v plus, if necessary, a sequence of requests at the $k-1$ imaginary nodes and at v until all k servers are placed at these nodes. This configuration must eventually be attained since the k-server algorithm is assumed to be competitive. The sequence of requests can be generated distributively, a single request at a time, in the following way: the termination of the actions due to a previous request can be detected by a technique for termination-detection of centrally initiated distributed computations [16] at a cost proportional to the cost of the original actions. Furthermore, after the destination node has detected the termination of the actions due to its request for service, messages are sent along the paths created by the messages due to this request (augmenting the cost by only a constant factor), to check if these paths reach the node managing the imaginary $k-1$ nodes (I), and if a server has left any of them. In this case control passes to I, which issues requests until $k-1$ servers reach it. Then the same procedure is repeated in order to pass control back to the destination node v if necessary. When all k servers occupy v and the $k-1$ imaginary nodes, v marks itself as the node hosting the user.

Find(v) A "Find(v)" request, that is, a request to access the user, generated at node v, is simulated by a request for service at v. Then a sequence of requests for service is generated at the $k-1$ imaginary points and at v until all servers are at these points, in the same way this is done for a Move request. At this time a subgraph is induced by the edges along which messages have been sent. This subgraph includes v and the node in which the user resides. Node v now sends messages that scan all this subgraph (augmenting the cost by only a constant factor) to find the node marked as the one the user resides in (denoted u). The Find request actions then terminate by an internal "Move(u)" request in order to bring a server back to this node.

Analysis Let σ be a sequence of requests, each request being either Move(v_i) or Find(v_i), for $1 \leq i \leq |\sigma|$. Denote by u_i the location of the user when request i is presented. Denote by T the following sum: $T = \sum_{i=1}^{|\sigma|} d(u_i, v_i)$. Then the cost incurred by an optimal off-line k-server algorithm when presented with the sequence of requests generated by the above described reduction on σ, is $O(T)$. Therefore, the cost incurred by the on-line distributed k-server algorithm is $O(cT)$, where c is its competitive ratio. This is also the cost of the mobile user algorithm generated by the above reduction, on σ. But $\frac{1}{D}T$ is the inherent mobile user cost of σ, and therefore the generated on-line mobile user algorithm is $O(c \cdot D)$ competitive.

¿From a lower bound of $\Omega(\log n / \log\log n)$ for the mobile user problem [5] we get,

Theorem 14 *For all n there is a network topology of n processors, such that the competitive ratio of any distributed k-server algorithm is $\Omega(\max\{k, \frac{1}{D} \cdot (\log n / \log\log n)\})$ against adaptive on-line adversaries.*

8 Conclusions

We have given a general translator that transforms any global-control competitive k-server algorithm into a distributed competitive one with a relatively small increase in the competitive ratio. The increase in the competitive ratio is however a function of the size of the memory needed by the global-control algorithm. As known k-server algorithms for general metric spaces [17, 22] have a large number of memory states, our translator yields distributed algorithms that have (in the case of general networks) a competitive ratio which is linear in the size of the metric space (n), while our lower bound is less than logarithmic in n. It is of interest to close the gap in the dependence of the competitive ratio on n, for distributed k-server algorithms for general networks.

Another interesting question is whether it is possible to devise a similar translator that would be applicable to randomized algorithms as well.

References

[1] B. Awerbuch, Y. Bartal, and A. Fiat. Competitive Distributed File Allocation. In *Proc. of the 25th Ann. ACM Symp. on Theory of Computing*, pages 164–173, May 1993.

[2] B. Awerbuch, Y. Bartal, and A. Fiat. Heat & Dump: Competitive Distributed Paging. In *Proc. of the 34th Ann. IEEE Symp. on Foundations of Computer Science*, pages 22–31, October 1993.

[3] 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.

[4] 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.

[5] N. Alon, G. Kalai, M. Ricklin and L. Stockmeyer. Lower Bounds on the Competitive Ratio for Mobile User Tracking and Distributed Job Scheduling. In *Proc. of the 33rd Ann. IEEE Symp. on Foundations of Computer Science*, pages 334–343, October 1992.

[6] B. Awerbuch and D. Peleg. Concurrent Online Tracking of Mobile Users. In *Proc. SIGCOMM.* Zurich, Sept. 1991.

[7] S. Ben-David, A. Borodin, R.M. Karp, G. Tardos, and A. Wigderson. On the Power of Randomization in Online Algorithms. In *Proc. of the 22nd Ann. ACM Symp. on Theory of Computing*, pages 379–386, May 1990.

[8] R.A. Baeza-Yates, J.C. Culberson, and G.J.E. Rawlins. Searching with Uncertainty. Technical report, University of Waterloo, October 1987.

[9] Y. Bartal, A. Fiat, and Y. Rabani. Competitive Algorithms for Distributed Data Management. In *Proc. of the 24th Ann. ACM Symp. on Theory of Computing*, pages 39–48, May 1992.

[10] Y. Bartal, and E. Grove. The Harmonic k-Server Algorithm is Competitive. *Journal of the ACM.* To appear.

[11] P. Berman, H. Karloff, and G. Tardos. A Competitive 3-Server Algorithm. In *Proc. 1st ACM-SIAM Symp. on Discrete Algorithms*, pages 280–290, January 1990.

[12] 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.

[13] 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.

[14] M. Chrobak and L. Larmore. An Optimal On-Line k-Server Algorithm for Trees. *SIAM Journal of Computing*, 20:144-148, 1991.

[15] M. Chrobak and L. Larmore. On Fast Algorithms for Two Servers. In *Proc. Mathematical Foundations of Computer Science*, Banská Bystrica, 1990. Also, to appear in *Journal of Algorithms.*

[16] W. Dijkstra, C. S. Scholten. Termination Detection for Diffusing Computations. *Information Processing Letters*, 11-1:1–4, 1980.

[17] A. Fiat, Y. Rabani, and Y. Ravid. Competitive k-Server Algorithms. In *Proc. of the 31st Ann. IEEE Symp. on Foundations of Computer Science*, pages 454–463, October 1990. Also, to appear in *JCSS Special FOCS Issue.*

[18] A. Fiat, Y. Rabani, Y. Ravid, and B. Schieber. A Deterministic $O(k^3)$-Competitive k-Server Algorithm for the Circle. Submitted to *Algorithmica.*

[19] E. Grove. The Harmonic k-Server Algorithm is Competitive. In *Proc. of the 23rd Ann. ACM Symp. on Theory of Computing*, pages 260-266, May 1991.

[20] S. Irani and R. Rubinfeld. A Competitive 2-Server Algorithm. *Information Processing Letters*, 39:85–91, 1991.

[21] A.R. Karlin, M.S. Manasse, L. Rudolph, and D.D. Sleator. Competitive Snoopy Caching. *Algorithmica*, 3(1):79-119, 1988.

[22] E. Koutsoupias, and C. Papadimitriou. On the k-Server Conjecture. In *Proc. 26th Annual ACM Symp. on Theory of Computing*, pages 507–511, 1994.

[23] 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.

[24] P. Raghavan and M. Snir. Memory versus Randomization in On-Line Algorithms. In *Proc. 16th ICALP*, July 1989.

[25] D.D. Sleator and R.E. Tarjan. Amortized Efficiency of List Update and Paging Rules. *Communication of the ACM*, 28(2) pages 202–208, 1985.

[26] G. Turpin. Recent Work on the Server Problem. Master's thesis, University of Toronto, 1989.