The (h, k)-Server Problem on Bounded Depth Trees[*]

Nikhil Bansal[†] Marek Eliáš[†] Łukasz Jeż[‡] Grigorios Koumoutsos[†]

Abstract

We study the k-server problem in the resource augmentation setting i.e., when the performance of the online algorithm with k servers is compared to the offline optimal solution with $h \leq k$ servers. The problem is very poorly understood beyond uniform metrics. For this special case, the classic k-server algorithms are roughly $(1 + 1/\epsilon)$-competitive when $k = (1 + \epsilon)h$, for any $\epsilon > 0$. Surprisingly however, no $o(h)$-competitive algorithm is known even for HSTs of depth 2 and even when k/h is arbitrarily large.

We obtain several new results for the problem. First we show that the known k-server algorithms do not work even on very simple metrics. In particular, the Double Coverage algorithm has competitive ratio $\Omega(h)$ irrespective of the value of k, even for depth-2 HSTs. Similarly the Work Function Algorithm, that is believed to be optimal for all metric spaces when $k = h$, has competitive ratio $\Omega(h)$ on depth-3 HSTs even if $k = 2h$. Our main result is a new algorithm that is $O(1)$-competitive for constant depth trees, whenever $k = (1 + \epsilon)h$ for any $\epsilon > 0$. Finally, we give a general lower bound that any deterministic online algorithm has competitive ratio at least 2.4 even for depth-2 HSTs and when k/h is arbitrarily large. This gives a surprising qualitative separation between uniform metrics and depth-2 HSTs for the (h, k)-server problem, and gives the strongest known lower bound for the problem on general metrics.

1 Introduction

The classic k-server problem, introduced by Manasse et al. [10], is a broad generalization of various online problems and is defined as follows. There are k servers that reside on some points of a given metric space. At each step, a request arrives at some point of the metric space and must be served by moving some server to that point. The goal is to minimize the total distance traveled by the servers.

In this paper, we study the resource augmentation setting of the problem, also known as the "weak adversary" model [8], where the online algorithm has k servers, but its performance is compared to a "weak" offline optimum with $h \leq k$ servers. We will refer to this as the (h, k)-server problem. Our motivation is twofold. Typically, the resource augmentation setting gives a much more refined view of the problem and allows one to bypass overly pessimistic worst case bounds, see e.g. [7]. Second, as we discuss below, the (h, k)-server problem is much less understood than the k-server problem and seems much more intriguing.

1.1 Previous work

The k-server problem has been extensively studied; here we will focus only on deterministic algorithms. It is well-known that no algorithm can be better than k-competitive for any metric space on more than k points [10]. In their breakthrough result, Koutsoupias and Papadimitriou [9] showed that the Work Function Algorithm (WFA) is $(2k-1)$-competitive in any metric space. For special metrics such as the uniform metrics[1], the line, and trees, tight k-competitive algorithms are known (cf. [4]). It is widely believed that a k-competitive algorithm exists for every metric space (the celebrated k-server conjecture), and it is also plausible that the WFA achieves this guarantee. Qualitatively, this means that general metrics are believed to be no harder than the simplest possible case of uniform metrics.

The (h, k)-server problem. Much less is known for the (h, k)-server problem. In their seminal paper [11], Sleator and Tarjan gave several $(k/(k-h+1))$-competitive algorithms for the uniform metrics and also showed that this is the best possible ratio. This bound was later extended to the weighted star metric (weighted paging) [12]. Note that this guarantee equals k for $k = h$ (the usual k-server setting), and tends to 1 as k/h approaches infinity. In particular, for $k = 2h$, this is smaller than 2.

It might seem natural to conjecture that, analogously to the k-server case, general metrics are no harder than the uniform metrics, and hence that $k/(k - h + 1)$ is the right bound for the (h, k)-server problem in all metrics. However, surprisingly, Bar-Noy and Schieber (cf. [4, p. 175]) showed this to be false: In the line metric, for $h = 2$, no deterministic algorithm can be better than 2-competitive, regardless of the value of k. This is the best known lower bound for the general (h, k)-server problem.

On the other hand, the best known upper bound is

[*]This work was supported by NWO grant 639.022.211, ERC consolidator grant 617951, and NCN grant DEC-2013/09/B/ST6/01538. It was carried out while Ł. Jeż was a postdoc at TU/e.

[†]TU Eindhoven, Netherlands. {n.bansal,m.elias,g.koumoutsos}@tue.nl

[‡]University of Wrocław, Poland. lje@cs.uni.wroc.pl

[1]The k-server problem in uniform metrics is equivalent to the paging problem.

$2h$, even when $k/h \to \infty$. In particular, Koutsoupias [8] showed that the WFA with k-servers is (about) $2h$-competitive against an offline optimum with h servers. Note that one way to achieve a guarantee of $2h-1$ is simply to disable the $k-h$ extra online servers and use WFA with h servers only. The interesting thing about the result of [8] is that the online algorithm does not know h and is $2h$-competitive simultaneously for every $h \leq k$. But, even if we ignore this issue of whether the online algorithm knows h or not, no guarantee better than h is known, even for very special metrics such as depth-2 HSTs or the line, and even when $k/h \to \infty$.

1.2 Our Results Motivated by the huge gap between the known lower and upper bounds even for very simple metrics, we consider bounded-depth HSTs (defined formally in Section 1.3).

We first show very strong lower bounds on all the previously known algorithms (beyond uniform metrics), specifically the Double Coverage (DC) algorithm of Chrobak et al. [5, 6] and the WFA. This is perhaps surprising because, for the k-server problem, DC is optimal in trees and WFA is believed to be optimal in all metrics.

THEOREM 1.1. *The competitive ratio of DC in depth-2 HSTs is $\Omega(h)$, even when $k/h \to \infty$.*

In particular, DC is unable to use the extra servers in a useful way. A similar lower bound was recently shown for the line by a superset of the authors [1].

For the WFA, we present the following lower bound.

THEOREM 1.2. *The competitive ratio of the WFA is at least $h + 1/3$ in a depth-3 HST for $k = 2h$.*

Surprisingly, the lower bound here is strictly larger than h! Recall that the WFA is believed to be h-competitive for $k = h$. Although the lower bound instance is quite simple, the analysis is rather involved as we need to consider how the various work function values evolve over time. The lower bound in Theorem 1.2 can also be extended to the line metric. Interestingly, it exactly matches the upper bound $(h+1)\mathrm{OPT}_h - \mathrm{OPT}_k$ implied by results of [8, 3] for the WFA in the line. We describe the details in Appendix B.

Our main result is the first $o(h)$-competitive algorithm for depth-d trees with the following guarantee.

THEOREM 1.3. *There is an algorithm that is $O_d(1)$-competitive on any depth-d tree, whenever $k = \delta h$ for $\delta > 1$. More precisely, its competitive ratio for is $O(d \cdot 2^{d+1})$ for $\delta \in [2^d, +\infty)$, and $O(d \cdot (\frac{\delta^{1/d}}{\delta^{1/d}-1})^{d+1})$ for $\delta \in (1, 2^d)$. If δ is very small, i.e. $\delta = 1 + \epsilon$ for $0 < \epsilon \leq 1$, the later bound equals to $O(d \cdot (2d/\epsilon)^{d+1})$.*

The algorithm is designed to overcome the drawbacks of DC and WFA, and can be viewed as a more aggressive and cost-sensitive version of DC. It moves the servers more aggressively at non-uniform speeds towards the region of the current request, giving a higher speed to a server located in a region containing many servers. It does not require the knowledge of h, and is simultaneously competitive against all h strictly smaller than k.

Finally, we give an improved general lower bound. Bar-Noy and Schieber (cf. [4, p. 175]) showed that there is no better than 2-competitive algorithm for the (h,k)-server problem in general metrics, by constructing their lower bound in the line metric. Our next result shows that even a 2-competitive algorithm is not possible. In particular, we present a construction in a depth-2 HST showing that no 2.4-competitive algorithm is possible.

THEOREM 1.4. *There is no 2.4-competitive deterministic algorithm for general metrics, even when $k/h \to \infty$, provided that h is larger than some constant independent of k.*

This shows that depth-2 HSTs are qualitatively quite different from depth-1 HSTs (same as uniform metrics) which allow a ratio $k/(k-h+1)$. We have not tried to optimize the constant 2.4 above, but computer experiments suggest that the bound can be improved to about 2.88.

1.3 Notation and Preliminaries In the (h,k)-setting, we define the competitive ratio as follows. An online algorithm ALG is R-competitive in metric M, if $ALG_k(I) \leq R \cdot OPT_h(I) + \alpha$ holds for any finite request sequence I of points in M. Here, $ALG_k(I)$ denotes the cost of ALG serving I with k servers, $OPT_h(I)$ denotes the optimal cost to serve I with h servers, and α is a constant independent of I. An excellent reference on competitive analysis is [4].

A *depth-d tree* is an edge-weighted rooted tree with each leaf at depth exactly d. In the (h,k)-server problem in a depth-d tree, the requests arrive only at leaves, and the distance between two leaves is defined as the distance in the underlying tree. A depth-d HST is a depth-d tree with the additional property that the distances decrease geometrically away from the root (see e.g. Bartal [2]). We will first present our algorithm for general depth-d trees (without the HST requirement), and later show how to (easily) extend it to arbitrary trees with bounded diameter, where requests are also allowed in the internal nodes.

DC Algorithm for Trees. Our algorithm can be viewed as a non-uniform version of the DC algorithm for trees [5, 6], which works as follows. When a request arrives at a vertex v of the tree, all the servers adjacent to v move towards it along the edges at the same speed until one eventually arrives at v. Here, a server s is *adjacent* to a point x if there is no other server on the (unique) path from s to x. If multiple servers are located at a single point, only one of them is chosen arbitrarily.

Work Function Algorithm. Consider a request sequence $\sigma = r_1, r_2, \dots, r_m$. For each $i = 1, \dots, m$, let $w_i(A)$ denote the optimal cost to serve requests $r_1, r_2, \dots, r_i$ and end up in the configuration A, which is specified by the set of locations of the servers. The function w_i is called *work function*. The Work Function Algorithm (WFA) decides its moves depending on the values of the work function. Specifically, if the algorithm is in a configuration A and a request $r_i \notin A$ arrives, it moves to the configuration X such that $r_i \in X$ and $w_i(X) + d(A, X)$ is minimized. For more background on the WFA, see [4].

1.4 Organization In Section 2, we describe the lower bound for the DC in depth-2 HSTs. The shortcomings of the DC might help the reader to understand the motivation behind the design of our algorithm for depth-d trees, which we describe in Section 3. Its extension to the bounded-diameter trees can be found in Appendix A.2. In Section 4 we describe the general lower bound (Theorem 1.4) and the lower bound for the WFA (Theorem 1.2) for depth-3 HSTs. The extension of Theorem 1.2 to the line is discussed in Appendix B.

2 Lower Bound for the DC Algorithm on depth-2 HSTs

We now show a lower bound of $\Omega(h)$ on the competitive ratio of the DC algorithm.

Let T be a depth-2 HST with $k + 1$ subtrees and edge lengths chosen as follows. Edges from the root r to its children have length $1 - \epsilon$, and edges from the leaves to their parents length ϵ for some $\epsilon \ll 1$. Let T_u be a subtree rooted at an internal node $u \neq r$. A *branch* B_u is defined as T_u together with the edge e connecting T_u to the root. We call B_u *empty*, if there is no online server in T_u nor in the interior of e. Since T contains $k + 1$ branches, at least one of them is always empty.

The idea behind the lower bound is quite simple. The adversary moves all its h servers to the leaves of an empty branch B_u, and keeps requesting those leaves until DC brings h servers to T_u. Then, another branch has to become empty, and the adversary moves all its servers there, starting a new *phase*. The adversary can execute an arbitrary number of such phases.

The key observation is that DC is "too slow" when bringing new servers to T_u, and incurs a cost of order $\Omega(h^2)$ during each phase, while the adversary only pays $O(h)$.

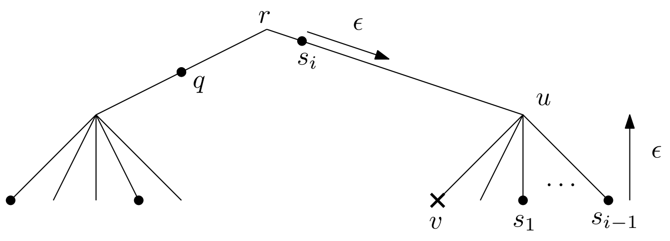

Figure 1: Move of DC during Step i. Servers $s_1, \dots, s_{i-1}$ are moving towards u by distance ϵ and s_i is moving down the edge (r, u) by the same distance. While s_i is in the interior of (r, u), no server q from some other branch is adjacent to v because the unique path between v and q passes through s_i.

Theorem 1.1 (restated). *The competitive ratio of DC in depth-2 HSTs is $\Omega(h)$, even when $k/h \to \infty$.*

Proof. We describe a phase, which can be repeated arbitrarily many times. The adversary places all its h servers at different leaves of an empty branch B_u and does not move until the end of the phase. At each time during the phase, a request arrives at such a leaf, which is occupied by some offline server, but contains no online servers. The phase ends at the moment when the hth server of DC arrives to T_u.

Let ALG denote the cost of the DC algorithm and ADV the cost of the adversary during the phase. Clearly, $ADV = 2h$ in each phase: The adversary moves its h servers to T_u and does not incur any additional cost until the end of the phase. However, we claim that $ALG = \Omega(h^2)$, no matter where exactly the DC servers are located when the phase starts. To see that, let us call Step i the part of the phase when DC has exactly $i - 1$ servers in T_u. Clearly, Step 1 consists of only a single request, which causes DC to bring one server to the requested leaf. So the cost of DC for Step 1 is at least 1. To bound the cost in the subsequent steps, we make the following observation.

Observation 2.1. *At the moment when a new server s enters the subtree T_u, no other DC servers are located along the edge $e = (r, u)$.*

This follows from the construction of DC, which moves only servers adjacent to the request. At the moment when s enters the edge e, no other server above s can be inside e; see Figure 1.

We now focus on Step i for $2 \leq i \leq h$. There are already $i-1$ servers in T_u, and let s_i be the next one which is to arrive to T_u.

Crucially, s_i moves if and only if all the servers of DC in the subtree T_u move from the leaves towards u, like in Figure 1: When the request arrives at v, s_i moves by ϵ and the servers inside T_u pay together $(i-1)\epsilon$. However, such a moment does not occur, until all servers $s_1, \dots, s_{i-1}$ are again at leaves, i.e. they incur an additional cost $(i-1)\epsilon$. To sum up, while s_i moves by ϵ, the servers inside T_u incur cost $2(i-1)\epsilon$.

When Step i starts, the distance of s_i from u is at least $1-\epsilon$, and therefore s_i moves by distance ϵ at least $\lfloor \frac{1-\epsilon}{\epsilon} \rfloor$ times, before it enters T_u. So, during Step i, DC pays at least

$$\left\lfloor \frac{1-\epsilon}{\epsilon} \right\rfloor (2(i-1)\epsilon + \epsilon) \geq \frac{1-2\epsilon}{\epsilon} \cdot \epsilon\big(2(i-1)+1\big) =$$

$$= (1-2\epsilon)(2i-1)$$

By summing over all steps $i = 1, \dots, h$ and choosing $\epsilon \leq 1/4$, we get

$$ALG \geq 1 + \sum_{i=2}^{h} (1-2\epsilon)(2i-1) \geq$$

$$\sum_{i=1}^{h} (1-2\epsilon)(2i-1) = (1-2\epsilon)h^2 \geq \frac{h^2}{2}$$

To conclude the proof, we note that $ALG/ADV \geq (h^2/2)/(2h) = h/4 = \Omega(h)$ for all phases. $\square$

3 Algorithm for depth-d trees

In this section we prove Theorem 1.3.

Recall that a depth-d tree is a rooted-tree and we allow the requests to appear only at the leaves. However, to simplify the algorithm description, we will allow the online servers to reside at any node or at any location on an edge (similar to that in DC). To serve a request at a leaf v, the algorithm moves all the adjacent servers slowly towards v, where the speed of each server coming from a different branch of the tree depends on the number of the online servers "below" it.

To describe the algorithm formally, we state the following definitions. For a point $x \in T$ (either a node or a location on some edge), we define T_x as the subtree consisting of all points below x including x itself, and we denote k_x the number of the online servers inside T_x. If s is a server located at a point x, we denote $T_s = T_x$ and $k_s = k_x$. We also denote $T_x^- = T_x \setminus \{x\}$, and k_x^- the corresponding number of the algorithm's servers in T_x^-. If u is a level-1 node, we call T_u an *elementary* subtree, see Figure 2.

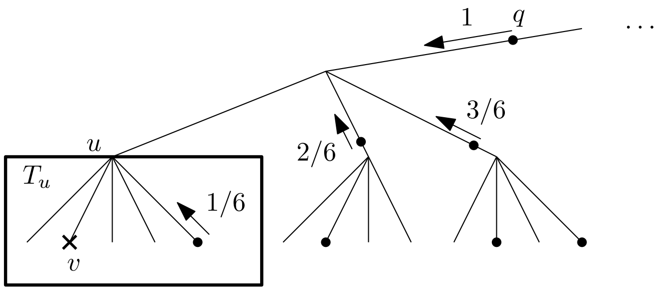

Figure 2: A request at v inside of the elementary tree T_u, and Phase 2 of Algorithm 1. Note that k_q^- equals 6 in the visualised case. Speed is noted next to each server moving.

3.1 Algorithm Description

Suppose a request arrives at a leaf v that lies in the elementary subtree T_u. The algorithm proceeds in two phases, depending on whether there is a server along the path from v to the root r or not. We set speeds as described in Algorithm 1 below and move the servers towards v either until the phase ends or the set A of servers adjacent to v changes. This defines the elementary moves (where A stays unchanged). Note that if there are some servers in the path between v and the root r, only the lowest of them belongs to A. We denote this server q during an elementary move. Figure 2 shows an elementary subtree T_u and the progress of Phase 2.

Algorithm 1: Serving request at leaf v in elementary subtree T_u.

```
**Phase 1:** While there is no server along the path $r - v$
    For each $s \in T_u$: move $s$ at speed $1/k_u$
    For each $s \in A \setminus T_u$: move $s$ at speed $k_s/(k - k_u)$
**Phase 2:** While no server reached $v$; Server $q \in A$ moves down along the path $r - v$
    For server $q$: move it at speed 1
    For each $s \in A \setminus \{q\}$: move it at speed $k_s/k_q^-$
```

We note two properties, the first of which follows directly from the design of Algorithm 1.

OBSERVATION 3.1. *No edge $e \in T$ contains more than one server of Algorithm 1 in its interior.*

Note that during both the phases, the various servers move at non-uniform speeds depending on their respective k_s. The following observations about these speeds will be useful.

Observation 3.2. *During Phase 1, the total speed of servers inside T_u is 1, unless there are no servers inside T_u. This follows as there are k_u servers moving at speed $1/k_u$. Similarly, the total speed of servers outside T_u (if there are any) is also 1. This follows as $\sum_{s\in A\setminus T_u} k_s = k - k_u$.*

Analogously, for Phase 2: the total speed of servers inside T_q^- is 1, if there are any. This follows as $\sum_{s\in A\setminus\{q\}} k_s = k_q^-$.

The intuition behind the algorithm is the following. Recall that the problem with DC is that it brings the outside servers too slowly into T_u when requests start arriving there. Therefore, our algorithm changes the speeds of the servers adjacent to v to make the algorithm more aggressive. The servers outside T_u in Phase 1 and the server q in Phase 2 are *helpers* coming to aid the servers inside T_u and T_q^-. From each region, these helpers move at the speed proportional to the number of the online servers in that region, which helps in removing them quickly from regions with excess servers. Of course, the online algorithm does not know which regions have excess servers and which ones do not (as it does not know the offline state), but the number of servers in a region serves as a good measure of this. The second main idea is to keep the total speed of the helpers proportional to the total speed of the servers inside T_u and T_q^-. This prevents the algorithm from becoming overly aggressive and keeps the cost of the moving helpers comparable to the cost incurred within T_u and T_q^{-1}. As for DC, the analysis of our algorithm is based on a carefully designed potential function.

3.2 Analysis We will analyze the algorithm based on a suitable potential function $\Phi(t)$. Let ALG(t) and OPT(t) denote the cost of the algorithm and of the adversary respectively, for serving the request at time t. Let $\Delta_t\Phi = \Phi(t) - \Phi(t-1)$ denote the change of the potential at time t. We will ensure that Φ is non-negative and bounded from above by a function of h, k, d, and length of the longest edge in T. Therefore, to show R competitiveness, it suffices to show that the following holds at each time t: ALG$(t) + \Delta_t\Phi \leq R\cdot$ OPT(t).

To show this, we split the analysis into two parts: First, we let the adversary move its servers to serve the request. Then we consider the move of the algorithm. Let $\Delta_t^{OPT}\Phi$ and $\Delta_t^{ALG}\Phi$ denote the changes in Φ due to the move of the adversary and the algorithm respectively. Clearly, $\Delta_t\Phi = \Delta_t^{OPT}\Phi + \Delta_t^{ALG}\Phi$, and thus it suffices to show the following two inequalities:

$$\Delta_t^{OPT}\Phi \leq R\cdot OPT(t) \tag{3.1}$$

$$ALG(t) + \Delta_t^{ALG}\Phi \leq 0 \tag{3.2}$$

3.2.1 Potential function Before we define the potential, we need to formalize the notion of excess and deficiency in the subtrees of T. Let $d(a,b)$ denote the distance of points a and b. For $e = (u,v) \in T$, where v is the node closer to the root, we define $k_e := k_u + \frac{1}{d(u,v)}\sum_{s\in e} d(s,v)$. Note that this is the number of online servers in T_u, plus the possible single server in e counted fractionally, proportionally to its position along e. Let h_u denote the number of offline servers inside T_u. For an edge e, let $\ell(e)$ denote its level with the convention that the edges from leaf to their parents have level 1, and the edges from root to its children have level d. For $\ell = 1,\ldots,d$, let β_ℓ be some geometrically increasing constants that will be defined later. For and edge e we define the *excess E_e of e* and the *deficiency D_e of e* as follows

$$E_e = \max\{k_e - \lfloor\beta_{\ell(e)}\cdot h_u\rfloor, 0\}\cdot d(u,v)$$
$$D_e = \max\{\lfloor\beta_{\ell(e)}\cdot h_u\rfloor - k_e, 0\}\cdot d(u,v).$$

Note that these compare k_e to h_u with respect to the *excess threshold $\beta_{\ell(e)}$*. We call an edge *excessive*, if $E_e > 0$, otherwise we call it *deficient*. Let us state a few basic properties of these two terms.

Observation 3.3. *Let e be an edge containing an algorithm's server s in its interior. If e is excessive, it cannot become deficient unless s moves upwards completely outside of the interior of e. Similarly, if e is deficient, it cannot become excessive unless s leaves interior of e completely.*

Note that no other server can pass through e while s still resides there and the contribution of s to k_e is a nonzero value strictly smaller than 1, while $\lfloor\beta_\ell h_u\rfloor$ is an integer.

Observation 3.4. *Let e be an edge and s be an algorithm's server in its interior moving by a distance x. Then either D_e or E_e changes exactly by x.*

This is because k_e changes by $x/d(u,v)$, and therefore the change of D_e (resp. E_e) is x.

Observation 3.5. *If an adversary server passes through the edge e, change in D_e (resp. E_e) will be at most $\lceil\beta_\ell\rceil\cdot d(u,v)$.*

To see this, note that $\lfloor \beta_{\ell(e)} h_u \rfloor \leq \lfloor \beta_{\ell(e)}(h_u - 1) \rfloor + \lceil \beta_{\ell(e)} \rceil$.

We now fix the excess thresholds. We first define β depending on $\delta = k/h$ as,

$$\beta = 2 \text{ if } \delta \geq 2^d, \text{ and } \beta = \delta^{1/d} \text{ for } \delta \leq 2^d.$$

For each $\ell = 1, \dots, d$, we define the excess threshold for all edges in the level ℓ as $\beta_\ell := \beta^{\ell-1}$. We also denote $\gamma := \frac{\beta}{\beta-1}$. Note that, for all possible $\delta > 1$, our choices satisfy $1 < \beta \leq 2$ and $\gamma \geq 2$. Now, we can define the potential. Let

$$\Phi := \sum_{e \in T} \left(\alpha^D_{\ell(e)} D_e + \alpha^E_{\ell(e)} E_e \right),$$

where the coefficients α^D_ℓ and α^E_ℓ are as follows:

$$\begin{aligned} \alpha^D_\ell &:= 2\ell - 1 && \text{for } 1 \leq \ell \leq d \\ \alpha^E_d &:= \frac{\delta}{\delta - \beta_d} \left(3 + \frac{\beta_d}{\delta} \alpha^D_d \right) \\ \alpha^E_\ell &:= \sum_{i=\ell}^{d-1} \gamma^{i-\ell+1} \left(2 + \frac{1}{\beta} \alpha^D_i \right) + \gamma^{d-\ell} \alpha^E_d && \text{for } 1 \leq \ell < d. \end{aligned}$$

Note that $\alpha^D_\ell > \alpha^D_{\ell-1}$ and $\alpha^E_\ell < \alpha^E_{\ell-1}$ for all $1 < \ell \leq d$. The latter follows as the multipliers $\gamma^{i-\ell+1}$ and $\gamma^{d-\ell}$ decrease with increasing ℓ and moreover the summation in the first term of α^E_ℓ has fewer terms as ℓ increases.

To prove the desired competitive ratio for Algorithm 1, the idea will be to show that the *good* moves (when a server enters a region with deficiency, or leaves a region with excess) contribute more than the *bad* moves (when a server enters a region with excess, or leaves a region that is already deficient).

As the dynamics of the servers can be decomposed into elementary moves, it suffices to only analyze these. We will also assume that no servers of A are located at a node. This is without loss of generality, as only the moving servers can cause a change in the potential, and each server appears at a node just for an infinitesimal moment during its motion.

The following two lemmas give some properties of the deficient and excessive subtrees, which will be used later in the proof of Theorem 1.3. The proofs of both these Lemmas are located in Appendix A.1.

Lemma 3.1. (Excess in Phase 1) *Let us assume that no server of $A' = A \setminus T_u$ resides at a node. For the set $E = \{s \in A' \mid s \in e, E_e > 0\}$ of servers which are located in the excessive edges, and for $D = A' \setminus E$, the following holds. If $k_u \leq \lfloor \beta_2 h_u \rfloor$, then we have*

$$\sum_{s \in D} k_s \leq \beta_d (h - h_u) \qquad \text{and} \qquad \sum_{s \in E} k_s \geq k - \beta_d h.$$

Lemma 3.2. (Excess in Phase 2) *Let us assume that no server of $A' = A \setminus \{q\}$ resides at a node. For the set $E = \{s \in A' \mid s \in e, E_e > 0\}$ of servers which are located in the excessive edges, and for $D = A' \setminus E$, the following holds. If $k^-_q \geq \lfloor \beta_\ell h^-_q \rfloor$, then we have*

$$\sum_{s \in D} k_s \leq \frac{1}{\beta} k^-_q \qquad \text{and} \qquad \sum_{s \in E} k_s \geq \frac{1}{\gamma} k^-_q.$$

3.2.2 Proof of Theorem 1.3

We now show the main technical result, which directly implies Theorem 1.3.

Theorem 3.1. *The competitive ratio of Algorithm 1 in depth-d trees is $O(d \cdot \gamma^{d+1})$.*

This implies Theorem 1.3 as follows. If $\delta \geq 2^d$, we have $\beta = 2$ and $\gamma = 2$, and we get the competitive ratio $O(d \cdot 2^{d+1})$. For $1 < \delta < 2^d$, we have $\beta = \delta^{1/d}$ and therefore the competitive ratio is $O(d \cdot (\frac{\delta^{1/d}}{\delta^{1/d}-1})^{d+1})$. Moreover if $\delta = (1+\epsilon)$ for some $0 < \epsilon \leq 1$, we have $\beta = (1+\epsilon)^{1/d} \geq 1 + \frac{1}{2d}$. In this case, we get the ratio $O(d \cdot (\frac{2d}{\epsilon})^{d+1})$, as $\gamma \leq (1+\epsilon)^{1/d} \cdot \frac{2d}{\epsilon}$.

We now prove Theorem 3.1.

Proof. [Proof of Theorem 3.1] As Φ is non-negative and bounded from above by a function of h, k, d, and the length of the longest edge in T, it suffices to show the inequalities (3.1) and (3.2).

We start with (3.1) which is straightforward. By Observation 3.5, the move of a single adversary's server through an edge e of length x_e changes D_e or E_e in the potential by at most $\lceil \beta_{\ell(e)} \rceil x_e$. As the adversary incurs cost x_e during this move, we need to show the following inequalities:

$$\begin{aligned} \lceil \beta_\ell \rceil x_e \cdot \alpha^D_\ell &\leq R \cdot x_e && \text{for all } 1 \leq \ell \leq d \\ \lceil \beta_\ell \rceil x_e \cdot \alpha^E_\ell &\leq R \cdot x_e && \text{for all } 1 \leq \ell \leq d. \end{aligned}$$

As we show in Lemma 3.5 below, $\lceil \beta_\ell \rceil \alpha^D_\ell$ and $\lceil \beta_\ell \rceil \alpha^E_\ell$ are of order $O(d \cdot \gamma^{d+1})$. Therefore, there is $R = \Theta(d \cdot \gamma^{d+1})$, which satisfies (3.1).

We now consider (3.2) which is much more challenging to show. Let us denote A_E the set of edges containing some server from A in their interior. We call an *elementary step* a part of the motion of the algorithm during which A and A_E remain unchanged, and all the servers of A are located in the interior of the edges of T. Lemmas 3.3 and 3.4 below show that (3.2) holds during an elementary step, and the theorem would follow by summing (3.2) over all the elementary steps. □

Lemma 3.3. *During an elementary step in Phase 1 of Algorithm 1, the inequality (3.2) holds.*

Proof. Without loss of generality, let us assume that the elementary step lasted exactly 1 unit of time. This makes the distance traveled by each server equals to its speed, and makes calculations cleaner.

Let us first note that the cost ALG incurred by the algorithm during this step is at most 2. Indeed, by Observation 3.2, the total speed of the servers in T_u as well as in $A \setminus T_u$ is 1 (or 0 if there are none), and thus the total speed of all servers is at most 2.

To estimate the change of the potential $\Delta\Phi$, we decompose A into four sets: $D, \bar{D}$ are the servers outside (resp. inside) T_u residing in the deficient edges, and $E, \bar{E}$ are the servers outside (resp. inside) T_u residing in the excessive edges. Next, we evaluate $\Delta\Phi$ due to the movement of the servers from each class separately. We distinguish two cases:

1. *When* $k_u \geq \lfloor \beta_2 h_u \rfloor = \lfloor \beta h_u \rfloor$. The servers from $\bar{D}$ are exactly those which have an offline server located below them and their rise contributes to the increase of deficiency in their edges. As one server of the adversary is already located at the requested point where no online server resides, we have $|\bar{D}| \leq h_u - 1$. Movement of the servers from $\bar{E}$ causes a decrease of the excess in their edges and we have $|\bar{E}| \geq k_u - |\bar{D}| = k_u - h_u + 1$. Since all the servers in T_u have speed $1/k_u$, the change of the potential due to their movement is at most $\frac{h_u-1}{k_u}\alpha_1^D - (1 - \frac{h_u-1}{k_u})\alpha_1^E$. So, by our assumption, $k_u \geq \lfloor \beta h_u \rfloor \geq \beta h_u - 1 \geq \beta(h_u - 1)$, and thus $(h_u - 1)/k_u \leq 1/\beta$.

   As for the servers outside T_u, all of them might be contained in D in the worst case, but they can cause an increase of Φ by at most α_d^D. Summing this up and using $\frac{1}{\gamma} = 1 - \frac{1}{\beta}$, we have

$$\mathrm{ALG} + \Delta\Phi \leq$$
$$2 + \left(\frac{h_u - 1}{k_u}\right)\alpha_1^D - \left(1 - \frac{h_u - 1}{k_u}\right)\alpha_1^E + \alpha_d^D$$
$$\leq 2 + \frac{1}{\beta}\alpha_1^D - \frac{1}{\gamma}\alpha_1^E + \alpha_d^D \leq 0,$$

   where the last inequality holds for the following reason: We have $\alpha_1^E/\gamma \geq 2 + \frac{1}{\beta}\alpha_1^D + \gamma^{d-2}\alpha_d^E$ (for $d \geq 2$), which cancels $2 + \frac{1}{\beta}\alpha_1^D$. Also $\gamma^{d-2}\alpha_d^E$ cancels α_d^D, since $\gamma \geq 2$, and thereby $\gamma^{d-2}\alpha_d^E \geq 2^{d-2} \cdot 3 \geq 2d - 1 = \alpha_d^D$.

2. *When* $k_u < \lfloor \beta_2 h_u \rfloor$. Here we rely on the decrease of the excess outside T_u, as all the movement inside T_u might contribute to the deficiency increase, causing Φ to increase by α_1^D. As for the servers in D, their contribution to the increase of Φ satisfies $\sum_{s \in D} k_s/(k - k_u)\alpha_{\ell(s)}^D \leq (\beta_d/\delta)\alpha_d^D$, as

$$\frac{\sum_{s\in D} k_s}{k - k_u} \leq \frac{\beta_d(h - h_u)}{k - k_u} \leq \frac{\beta_d(h - h_u)}{\delta(h - k_u/\delta)},$$

$$\text{and } \frac{h - h_u}{h - k_u/\delta} < 1.$$

   The first inequality follows from Lemma 3.1. Fortunately, the movement of the servers in E assures the desired decrease of the potential as, by Lemma 3.1, we have $\sum_{s\in E} k_s \geq k - \beta_d h$. Thus, the movement of servers from E causes the decrease of Φ by at least $\sum_{s\in E} k_s/(k - k_u)\alpha_{\ell(s)}^E \geq ((\delta - \beta_d)/\delta)\alpha_d^E$, as

$$\frac{\sum_{s\in E} k_s}{k - k_u} \geq \frac{k - \beta_d h}{k} \geq \frac{h(\delta - \beta_d)}{h\delta}.$$

   Summing everything up, we obtain

$$ALG + \Delta\Phi \leq 2 + \alpha_1^D + \frac{\beta_d}{\delta}\alpha_d^D - \frac{\delta - \beta_d}{\delta}\alpha_d^E \leq 0,$$

   where the second inequality holds by our choice of $\alpha_1^D = 1$ and $\alpha_d^E = \frac{\delta}{\delta - \beta_d}\left(3 + \frac{\beta_d}{\delta}\alpha_d^D\right)$.

□

LEMMA 3.4. *During an elementary step in Phase 2 of Algorithm 1, the inequality* (3.2) *holds.*

Proof. Similarly to the proof of the preceding lemma, we assume (without loss of generality) that the duration of the elementary step is exactly 1 time unit, so that the speed of each server equals the distance it travels. As the speed of the server q is 1, it also moves by distance 1. The servers in T_q^- (strictly below q) move in total by $\sum_{s\in A\setminus\{q\}} \frac{k_s}{k_q^-} = 1$, and therefore $ALG \leq 2$. We denote ℓ the level of the edge containing q. To estimate the change of the potential, we again consider two cases.

1. *When* $k_q^- \geq \beta_\ell h_q^-$. Here the movement of q increases the excess in the edge containing q. Let us denote E (resp. D) the servers of $A \setminus \{q\}$ residing in the excessive resp. deficient edges. By taking the largest possible α^D and the smallest possible α^E coefficient in the estimation, we can bound the change of the potential due to the move of the servers in T_q^- as,

$$\alpha_{\ell-1}^D \sum_{s\in D} \frac{k_s}{k_q^-} - \alpha_{\ell-1}^E \sum_{s\in E} \frac{k_s}{k_q^-} \leq \frac{1}{\beta}\alpha_{\ell-1}^D - \frac{1}{\gamma}\alpha_{\ell-1}^E,$$

   where the above inequality holds due to the Lemma 3.2.

As the movement of q itself causes an increase of Φ by α^E_ℓ, we have

$$ALG + \Delta\Phi \leq 2 + \alpha^D_{\ell-1}/\beta - \frac{1}{\gamma}\alpha^E_{\ell-1} + \alpha^E_\ell.$$

To see that this is non-positive, recall that $\alpha^E_\ell = \sum_{i=\ell}^{d-1} \gamma^{i-\ell+1}\left(2 + \frac{1}{\beta}\alpha^D_i\right) + \gamma^{d-\ell}\alpha^E_d$. Therefore

$$\frac{1}{\gamma}\alpha^E_{\ell-1} = \frac{1}{\gamma}\sum_{i=\ell-1}^{d-1} \gamma^{i-\ell+2}\big(2 + \frac{1}{\beta}\alpha^D_i\big) + \frac{1}{\gamma}\gamma^{d-\ell+1}\alpha^E_d$$

$$= \big(2 + \frac{1}{\beta}\alpha^D_{\ell-1}\big) + \sum_{i=\ell}^{d-1} \gamma^{i-\ell+1}\big(2 + \frac{1}{\beta}\alpha^D_i\big) + \gamma^{d-\ell}\alpha^E_d$$

$$= 2 + \frac{1}{\beta}\alpha^D_{\ell-1} + \alpha^E_\ell.$$

2. *When* $k^-_q < \lfloor \beta_\ell h^-_q \rfloor$. This case is much simpler. All the movement inside of T^-_q might contribute to the increase of deficiency at level at most $\ell - 1$. On the other hand, q then causes a decrease of deficiency at level ℓ and we have $ALG + \Delta\Phi \leq 2 + \alpha^D_{\ell-1} - \alpha^D_\ell$. This is less or equal to 0, as $\alpha^D_\ell \geq \alpha^D_{\ell-1} + 2$.

□

LEMMA 3.5. *For each* $1 \leq \ell \leq d$, *both* $\lceil \beta_\ell \rceil \alpha^D_\ell$ *and* $\lceil \beta_\ell \rceil \alpha^E_\ell$ *are of order* $O(d \cdot \gamma^{d+1})$.

Proof. [Proof of Lemma 3.5] We have defined $\alpha^D_\ell = 2\ell - 1$, and therefore

$$\lceil \beta_\ell \rceil \alpha^D_\ell \leq 2 \cdot \beta^\ell (2\ell - 1) \leq \beta^d (2d - 1) = O(d \cdot \gamma^{d+1}),$$

since we have chosen $1 < \beta \leq 2$ and $\gamma \geq 2$ for any possible δ.

Now we estimate the value of α^E_d. Since $\delta \geq \beta \cdot \beta_d$ and $\frac{\delta}{\delta - \beta_d} \leq \frac{1}{1 - \beta_d/\delta} \leq \frac{1}{1 - 1/\beta} = \gamma$, we have

$$\alpha^E_d = \frac{\delta}{\delta - \beta_d}\left(3 + \frac{\beta_d}{\delta}\alpha^D_d\right) \leq \gamma(3 + \frac{1}{\beta}\alpha^D_d),$$

which is $O(d \cdot \gamma)$, as $\beta > 1$ and $\alpha^D_d = O(d)$. For α^E_ℓ, we have

$$\alpha^E_\ell \leq (2 + \frac{1}{\beta}\alpha^D_d)\sum_{i=\ell}^{d-1}\gamma^{i-\ell+1} + \gamma^{d-\ell}\alpha^E_d$$

$$\leq \gamma^{d-\ell+1}\big(2 + \frac{2d-3}{\beta} + \alpha^E_d\big) = O(\gamma^{d-\ell+2} \cdot d),$$

because $1 < \beta \leq 2$ and $\alpha^E_d = O(d)$. The second inequality follows by summing-up the geometric series and using the fact that $\gamma \geq 2$. Therefore, as $\beta_\ell \leq \gamma^{\ell-1}$, we have $\lceil \beta_\ell \rceil \alpha^E_\ell = O(\gamma^{d+1} d)$, and this concludes the proof. □

4 Lower Bounds

In this section we prove Theorems 1.4 and 1.2. We first show a general lower bound on the competitive ratio of any algorithm for depth-2 HSTs. Then we give lower bound on the competitive ratio of WFA.

4.1 General lower bound for depth-2 HSTs We now give a lower bound on the competitive ratio of any deterministic online algorithm on depth-2 HSTs. In particular, we show that for sufficiently large h, any deterministic online algorithm has competitive ratio at least 2.4.

The metric space is a depth-2 HST T with the following properties: T contains at least $k+1$ elementary subtrees and each one of them has at least h leaves. To ease our calculations, we assume that edges of the lower level have length $\epsilon \ll 1$ and edges of the upper level have length $1 - \epsilon$. So the distance between leaves of different elementary subtrees is 2.

THEOREM 1.4 (RESTATED). *For sufficiently large* h, *even when* $k/h \to \infty$, *there is no 2.4-competitive deterministic algorithm for general metrics, and in particular even for depth-*2 *HSTs.*

Proof. For a level 1 node u, let T_u denote the elementary subtree rooted at u. We assume without loss of generality that all the offline and online servers are always located at the leaves. We say that a server is inside subtree T_u, if it is located at some leaf of T_u. If there are no online servers at the leaves of T_u, we say that T_u is *empty*. Observe that at any given time there exists at least one empty elementary subtree.

Let $\mathcal{A}$ be an online algorithm. The adversarial strategy consists of arbitrarily many iterations of a phase. During a phase, some offline servers are moved to an empty elementary subtree T_u and requests are made there until the cost incurred by $\mathcal{A}$ is sufficiently large. At this point the phase ends and a new phase may start in another empty subtree. Let ALG and ADV denote the cost of $\mathcal{A}$ and adversary respectively during a phase. We will ensure that for all phases ALG $\geq 2.4 \cdot$ ADV. This implies the lower bound on the competitive ratio of $\mathcal{A}$.

We describe a phase of the adversarial strategy. The adversary moves some $\ell \leq h$ servers to the empty elementary subtree T_u and makes requests at leaves of T_u until $\mathcal{A}$ brings m servers there. In particular, each request appears at a leaf of T_u that is not occupied by a server of $\mathcal{A}$. We denote by $s(i)$ the cost that $\mathcal{A}$ has to incur for serving requests inside T_u until it moves its ith server there (this does not include the cost of moving the server from outside T_u). Clearly, $s(1) = 0$ and $s(i) \leq s(i+1)$ for all $i > 0$. We restrict our attention to $i \leq h$. The choice of ℓ and m depends on the values $s(i)$ for $2 \leq i \leq h$. We will now show that for any values of $s(i)$'s, the adversary can choose ℓ and m such that $\text{ALG} \geq 2.4 \cdot \text{ADV}$.

First, if there exists an i such that $s(i) \geq 3i$, we set $\ell = m = i$. Intuitively, the algorithm is too slow in bringing his servers in this case. Both $\mathcal{A}$ and the adversary incur a cost of $2i$ to move i servers to T_u. However, $\mathcal{A}$ pays a cost of $s(i)$ for serving requests inside T_u, while the adversary can serve all those requests at zero cost (all requests can be located at leaves occupied by offline servers). Overall, the cost of $\mathcal{A}$ is $2i+s(i) \geq 5i$, while the offline cost is $2i$. Thus we get that $\text{ALG} \geq 2.5 \cdot \text{ADV}$.

Similarly, if $s(i) \leq (10i-24)/7$ for some i, we choose $\ell = 1$ and $m = i$. Roughly speaking, in that case the algorithm is too "aggressive" in bringing its first i servers, thus incurring a large movement cost. Here, the adversary only moves one server to T_u. Each request is issued at an empty leaf of T_u. Therefore $\mathcal{A}$ pays for each request in T_u and the same holds for the single server of the adversary.

So, $\text{ALG} = 2i + s(i)$ and $\text{ADV} = 2 + s(i)$. By our assumption on $s(i)$, this gives

$$\text{ALG} - 2.4 \cdot \text{ADV} = 2i + s(i) - 4.8 - 2.4 \cdot s(i)$$

$$= 2i - 4.8 - 1.4 \cdot s(i) \geq 0$$

We can thus restrict our attention to the case that $s(i) \in (\frac{10i-24}{7}, 3i)$, for all $2 \leq i \leq h$. Now, for $1 < \ell < h$ and $m = h$, we can upper bound the offline cost as follows:

$$\text{ADV} \ \leq 2\ell + (s(h) - s(\ell)) \cdot \frac{h-\ell+1}{h}$$

The first term is the cost of moving ℓ servers to T_u. The second term upper bounds the cost for serving requests in T_u for the following reason: the adversary has to pay only during the part when $\mathcal{A}$ has at least ℓ servers at T_u. The cost of the adversary during this time equals the cost of $\mathcal{A}$ (which is $s(h) - s(\ell)$) divided by its competitive ratio on a uniform metric (which is clearly larger than $\frac{h}{h-\ell+1}$). Let us denote $c = s(h)/h$. Note that assuming h is large enough, $c \in (1,3)$. We get that

$$\frac{\text{ALG}}{\text{ADV}} \geq \frac{2h + s(h)}{2\ell + (s(h) - s(\ell)) \cdot \frac{h-\ell+1}{h}} \geq \frac{2h + ch}{2\ell + (ch - \frac{10\ell-24}{7}) \cdot \frac{h-\ell+1}{h}} \tag{4.3}$$

We now show that for every value of $c \in (1,4)$, there is an $\ell = \beta h$, where β is a constant depending on c, such that the right hand side of (4.3) is at least 2.4. First, as h is large enough, the right hand side is arbitrarily close to

$$\frac{2h + ch}{2\ell + (ch - 10\ell/7) \cdot (1 - \ell/h)} =$$

$$\frac{2 + c}{2\beta + (c - 10\beta/7)(1 - \beta)} =$$

$$\frac{2 + c}{10/7\beta^2 + (4/7 - c)\beta + c}$$

We choose $\beta := (c - 4/7)/(20/7)$. Note that, as $c \in (1,3)$, we have that $\beta < 1$ and hence $\ell < h$. The expression above then evaluates to $(2+c)/(c - (c - \frac{4}{7})^2/\frac{40}{7})$. By standard calculus, this expression is minimized at $c = (2\sqrt{221} - 14)/7$ where it attains a value higher than 2.419.

□

4.2 Lower bound for WFA on depth-3 HSTs

We give an $\Omega(h)$ lower bound on the competitive ratio of the Work Function Algorithm (WFA) in the (h,k)-setting. More precisely, we show a lower bound of $h + 1/3$ for $k = 2h$. We first present the lower bound for a depth-3 HST. In Appendix B we show how the construction can be adapted to work for the line. We also show that this exactly matches the upper bound of $(h+1) \cdot \text{OPT}_h - \text{OPT}_k$ implied by results of [8, 3] for the line.

The basic idea behind the lower bound is to trick the WFA to use only h servers for servicing requests in an "active region" for a long time before it moves its extra available online servers. Moreover, we make sure that during the time the WFA uses h servers in that region, it incurs an $\Omega(h)$ times higher cost than the adversary. Finally, when WFA realizes that it should bring its extra servers, the adversary moves all its servers to some different region and starts making requests there. So eventually, WFA is unable to use its additional servers in a useful way to improve its performance.

Theorem 1.2 (restated). *The competitive ratio of WFA in a depth-3 HST for $k = 2h$ is at least $h + 1/3$.*

Proof. Let T be a depth-3 HST. We assume that the lengths of the edges in a root-to-leaf path are $\frac{1-\epsilon}{2}, \frac{\epsilon-\epsilon'}{2}, \frac{\epsilon'}{2}$, for $\epsilon' \ll \epsilon \ll 1$. So the diameter of T is 1 and the diameter of depth-2 subtrees is ϵ. Without loss of generality we will also assume that $\frac{2}{\epsilon}$ is integer. Let L and R be two subtrees of depth 2. Inside each one of them, we focus on 2 elementary subtrees L_1, L_2 and R_1, R_2 respectively (see figure 3). All requests appear at leaves of those four elementary subtrees. Note that both

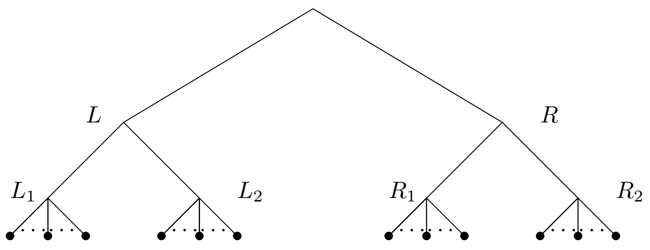

Figure 3: The tree where the lower bound for WFA is applied: All requests arrive at leaves of L_1, L_2, R_1 and R_2.

the WFA and the adversary always have their servers at leaves. This way, we say that some servers of the WFA (or the adversary) are inside a subtree, meaning that they are located at some leaves of that subtree.

The adversarial strategy consists of arbitrary many iterations of a phase. At the beginning of the phase, the WFA and the adversary have all their servers in the same subtree, either L or R. At the end of the phase, all servers are moved to the other subtree (R or L resp.), so a new phase may start. Before describing the phase, we give the basic strategy, which is repeated many times during a phase. For any elementary subtree T, we define strategy $S(T)$.

Strategy $S(T)$:

1. Adversary moves all its h servers to leaves $t_1, t_2, \dots, t_h$ of T.
2. While the WFA has $i < h$ servers in T: Request points $t_1, t_2, \dots, t_{i+1}$ in an adversarial way (i.e each time request a point where the WFA does not have a server) until $(i+1)$th server arrives at T.

We now describe a left-to-right phase, i.e., a phase which starts with all servers at L and ends with all servers at R. A right-to-left phase is completely symmetric, i.e., we replace R by L and R_i by L_i.

Left-to-right phase:

– *Step 1:* Apply strategy $S(R_1)$ until WFA moves h servers to R_1.

– *Step 2:* Repeat $S(R_2)$ and $S(R_1)$ until WFA moves all its k servers to R.

We now state two basic lemmas for counting the online and offline cost during each step. Proofs of those lemmas, require a characterization of work function values, which comes later on. Here we just give the high-level idea behind the proofs. Let ALG and ADV denote the online and offline cost respectively.

Lemma 4.1. *During Step 1, ALG$= h^2$ and ADV$= h$.*

Intuitively, we will exploit the fact that the WFA moves its servers too slowly towards R as long as $\ell < h$. In particular, we show that whenever the WFA has ℓ servers in R, it incurs a cost of 2ℓ inside R before it moves its $(\ell+1)$th server there. This way we get that the cost of the algorithm is $\sum_{\ell=1}^{h-1} 2\ell + h = h^2$. The fact that adversary pays h is trivial; the adversary just moves its h servers by distance 1 and then serves all requests at zero cost.

Lemma 4.2. *During Step 2, $ALG \geq (2-2\epsilon)h^2 + h$ and $ADV \leq (2+\epsilon)h$.*

Roughly speaking, to prove this lemma we make sure that for almost the whole Step 2, the WFA has h servers in R and they incur a cost h times higher than the adversary. The additional servers move to R almost at the end of the phase.

The theorem follows from lemmata 4.1 and 4.2. Summing up for the whole phase, the offline cost is $(3+\epsilon)\cdot h$ and the online cost is $3(3-2\epsilon)h^2 + h$. We get that

$$\frac{ALG}{ADV} \geq \frac{h^2 + (2-2\epsilon)h^2 + h}{h + (2+\epsilon)h} = \frac{(3-2\epsilon)h^2 + h}{(3+\epsilon)h} \to h + \frac{1}{3}.$$

□

Work Function values We now give a detailed characterization of how the work function evolves during left-to-right phase. For right-to-left phases the structure is completely symmetric.

Basic Properties: Recall that for any two configurations X, Y at distance $d(X,Y)$ the work function w satisfies,

$$w(X) \leq w(Y) + d(X,Y).$$

Also, let w and w' the work function before and after a request r respectively. For a configuration X such that $r \in X$, we have that $w'(X) = w(X)$. In the rest of this section, we use these properties of work functions without further explanation.

Notation: Let (L^i, R^{k-i}) denote a configuration which has i servers at L and $(k-i)$ servers at R. Let $w(L^i, R^{k-i})$ the minimum work function value of a configuration (L^i, R^{k-i}). If we need to make precise how many servers are in each elementary subtree, we denote by (L^i, r_1, r_2) a configuration which has i servers at L, r_1 servers at R_1 and r_2 servers at R_2. Same as before, $w(L^i, r_1, r_2)$ denotes the minimum work function value of those configurations.

Initialization: For convenience we assume that at the beginning of the phase the minimum work function value of a configuration is zero. Note that this is without loss of generality: If $m = \min_X w(X)$ when the phase starts, we just subtract m from the work function values of all configurations. We require the invariant that at the beginning of the phase, for $0 \le i \le k$:

$$w(L^{k-i}, R^i) = i. \tag{4.4}$$

This is clearly true for the beginning of the first phase, as all online servers are initially at L. We are going to make sure, that the symmetric condition holds at the end of the phase, so a right-to-left phase may start.

First Step: We now focus on the first step of the phase, i.e. the first execution of $S(R_1)$. The next lemma characterizes the structure of the work function each time the WFA decides to move a new server to R.

LEMMA 4.3. *At the moment when the WFA moves its ℓth server from L to R, for any $1 \le \ell < h$, the following two things hold:*

(i) For all $0 \le i < \ell$, $w(L^{k-i}, R^i) = w(L^{k-\ell}, R^\ell) + (\ell - i)$

(ii) For all $\ell \le i \le k$, $w(L^{k-i}, R^i) = w(L^{k-\ell}, R^\ell) + (i - \ell)$

In other words, having ℓ servers in R is the lowest state, and all other states are tight with respect to it.

Proof. We use induction on ℓ.

Induction Base: In the base case when $\ell = 0$, the first part of the lemma is vacuously true. The second part of the Lemma holds by the assumed invariant (4.4).

Induction Step: Assume that this is true for some $0 \le \ell < h - 1$. We are going to show that the lemma holds for $\ell + 1$. Let w be the work function at the time when the ℓth server arrives at R and w' be the work function at the time when the $(\ell+1)$th server arrives at R.

(i) We first show that the lemma holds for $i = \ell$: By construction of the WFA, at the moment it moves its $(\ell + 1)$th server at R, the following equation holds:

$$w'(L^{k-\ell}, R^\ell) = w'(L^{k-(\ell+1)}, R^{\ell+1}) + 1, \tag{4.5}$$

We now focus on $i < \ell$. By induction hypothesis, we have that $w(L^{k-i}, R^i) = w(L^{k-\ell}, R^\ell) + (\ell - i)$. During the time when WFA has ℓ servers at R, all requests are located in R. Thus, $w(L^{k-i}, R^i)$ increases at least as much as $w(L^{k-\ell}, R^\ell)$, i.e.

$$w'(L^{k-i}, R^i) - w(L^{k-i}, R^i) \ge$$
$$w'(L^{k-\ell}, R^\ell) - w(L^{k-\ell}, R^\ell).$$

So, we have that

$$\begin{gathered} w'(L^{k-i}, R^i) \ge \\ w'(L^{k-\ell}, R^\ell) - w(L^{k-\ell}, R^\ell) + w(L^{k-i}, R^i) \\ = w'(L^{k-\ell}, R^\ell) + (\ell - i) \end{gathered}$$

Clearly, as $w'(L^{k-i}, R^i) \le w'(L^{k-\ell}, R^\ell) + (\ell - i)$. This implies that equality holds, and we get that

$$\begin{aligned} w'(L^{k-i}, R^i) &= w'(L^{k-\ell}, R^\ell) + (\ell - i) \\ &= w(L^{k-\ell-1}, R^{\ell+1}) + (\ell + 1 - i), \end{aligned}$$

where the last equality holds due to (4.5).

(ii) For $i \ge (\ell + 1)$: Since the beginning of the phase there are $(\ell + 1)$ points requested, so $w(L^{k-i}, R^i)$ does not increase. By (4.4), we get that

$$\begin{aligned} w'(L^{k-i}, R^i) &= i = \ell + 1 + (i - (\ell + 1)) \\ &= w'(L^{k-\ell-1}, R^{\ell+1}) + i - (\ell + 1). \end{aligned}$$

□

Second step: By lemma 4.3, at the beginning of the second step, the work function values satisfy:

$$w(L^{2h-i}, R^i) = w(L^h, R^h) + (h - i), \text{ for } 0 \le i \le h, \tag{4.6}$$

$$w(L^{2h-i}, R^i) = w(L^h, R^h) + (i - h), \text{ for } h \le i \le 2h, \tag{4.7}$$

We first claim that (4.6) holds for the entire second step. This follows as all the requests arrive at R, so the optimal way to serve those requests using $i < h$ servers, can only have larger cost than using h servers.

We now consider executions of S during the time that WFA has h servers in R. We describe the changes in work function values of configurations (L^h, r_1, r_2) during an execution of $S(R_2)$ and for $S(R_1)$ the situation will be completely symmetric. We require that initially for all $i \le h$,

$$w(L^h, h - i, i) = w(L^h, h, 0) + \epsilon \cdot i. \tag{4.8}$$

This is true at the beginning of the second step, and we are going to make sure that at the end of $S(R_2)$ the symmetric is true, so an execution of $S(R_1)$ can start. Similarly to lemma 4.3 we can show the following:

LEMMA 4.4. *Consider an execution of $S(R_2)$ such that WFA has exactly h servers in R. At the moment when the ℓth server moves to R_2, the following two things hold:*

(i) For all $0 \le i < \ell$, $w(L^h, h-i, i) = w(L^h, h-\ell, \ell) + \epsilon(\ell - i)$

(ii) For all $\ell \le i \le h$, $w(L^h, h-i, i) = w(L^h, h-\ell, \ell) + \epsilon(i - \ell)$

Proof. The proof is exactly the same as the proof of lemma 4.3, we just need to replace (L^{k-i}, R^i) by $(L^h, h-i, i)$ and scale all distances by ϵ. $\square$

We get that when hth server arrives to R_2, $w(L^h, i, h-i) = w(L^h, 0, h) + \epsilon \cdot i$ for all $i \le h$. Observe that this is the symmetric version of (4.8). That means, the work function satisfies the initial requirement for $S(R_1)$ to start.

Now, we come to harder part where we want to argue that the WFA does not bring in extra servers. To this end, we will investigate the changes in $w(L^{h-i}, R^{h+i})$ during executions of second step.

Lemma 4.5. *Consider an execution of S such that for all its duration WFA has at most $h+i$ servers in R, for $0 \le i \le (h-1)$. During such an execution, $\Delta w(L^{h-i}, R^{h+i}) = \epsilon(h-i)$.*

Proof. We prove the lemma for executions of $S(R_2)$. We focus on configurations (L^{h-i}, R^{h+i}). Initially, among those configurations, a configuration (L^{h-i}, h, i) has the minimum work function value. This is clearly true for the first execution of second step and we will show that it holds for all subsequent executions. Moreover for all $j \le h$, the following equation holds:

$$w(L^{h-i}, h-j, i+j) = w(L^{h-i}, h, i) + j \cdot \epsilon \tag{4.9}$$

Up to the time that at most i points of R_2 are requested, $w(L^{h-i}, h, i)$ does not increase. After the time that ith WFA server arrives at R_2, we can apply lemma 4.4 for $\ell = i$ to h. In particular, $w(L^{h-i}, h-\ell+i, \ell)$ increases in the same way as $w(L^h, h-\ell, \ell)$.

At the end, a configuration (L^{h-i}, i, h) has the minimum work function value. Moreover, we have that $w'(L^{h-i}, i, h) = w(L^{h-i}, i, h)$, because exactly h points of R_2 are requested. For $w(L^{h-i}, i, h)$ we can apply equation (4.9) for $j = h-i$. We get that

$$w'(L^{h-i}, R^{h+i}) = w'(L^{h-i}, i, h) = w(L^{h-i}, i, h) =$$
$$= w(L^{h-i}, R^{h+i}) + \epsilon \cdot (h-i).$$

$\square$

Next two lemmata show that we can force the algorithm use only h servers in R for a long time and that additional servers arrive roughly at the same time.

Let $N = 2/\epsilon$.

Lemma 4.6. *After $N-2$ executions of S in Step 2, the WFA still has h servers in R*

Proof. Using lemma 4.5 we get that after $(N-2)$ executions of S in Step 2,

$$w(L^h, R^h) = h + (N-2) \cdot \epsilon \cdot h = 3h - 2\epsilon \cdot h, \tag{4.10}$$

$$\begin{aligned} \text{and } w(L^{h-1}, R^{h+1}) &= h + 1 + (N-2) \cdot \epsilon(h-1) = \\ &= 3h - 1 - 2\epsilon(h-1). \end{aligned} \tag{4.11}$$

By (4.10) and (4.11) we get that $w(L^{h-1}, R^{h+1}) + 1 > w(L^h, R^h) + \epsilon h$. That means, that up to the end of $(N-2)$th execution of S, the WFA would always prefer to move a server inside R by distance ϵ rather than moving a server from L by distance 1. This clearly shows that the WFA is still in some configuration (L^h, R^h). $\square$

Lemma 4.7. *Step 2 consists of at most $N+1$ executions of S.*

Proof. Assume that after N executions of S in Step 2, the WFA has $h+i$ servers in R, for some $0 \le i \le (h-1)$. In other words, it is in some configuration (L^{h-i}, R^{h+i}). We can apply lemma 4.5 for (L^{h-i}, R^{h+i}) for all N executions. After N executions of S in Step 2

$$\Delta w(L^{h-i}, R^{h+i}) = 2/\epsilon \cdot \epsilon \cdot (h-i) = 2(h-i) \tag{4.12}$$

By adding (4.7) and (4.12) we get that

$$\begin{aligned} w'(L^{h-i}, R^{h+i}) &= w(L^{h-i}, R^{h+i}) + 2(h-i) \\ &= h + i + 2h - 2i = 2h + (h-i). \end{aligned} \tag{4.13}$$

Since the beginning of the phase, exactly $2h$ points in R are requested. Thus, $w'(L^0, R^{2h}) = w(L^0, R^{2h}) = 2h$; during the whole phase. This way (4.13) is equivalent to $w'(L^{h-i}, R^{h+i}) = w'(L^0, R^{2h}) + (h-i)$. That implies that during the next execution of S, the WFA moves to the configuration (L^0, R^{2h}), i.e brings all its servers at R, so the phase ends. $\square$

End of the phase: By (4.6) and (4.13) we get that in the end of the phase $w(L^i, R^{2h-i}) = 2h + i$ for all $0 \le i \le 2h$. So a new right-to-left phase may start, satisfying the initial assumption about the structure of work function values.

Bounding Costs Now, we are ready to bound online and offline costs during the phase.

LEMMA 4.8. *During the time when the WFA uses ℓ servers in R, it incurs a cost of 2ℓ.*

Proof. Let w the work function at time when ℓth server arrives at R and w' the work function at time when $(\ell+1)$th arrives there. From lemma 4.3 we get that

$$w'(L^{k-\ell}, R^{\ell}) - w(L^{k-\ell}, R^{\ell}) =$$
$$= w'(L^{k-\ell-1}, R^{\ell+1}) + 1 - w(L^{k-\ell-1}, R^{\ell+1}) = 2.$$

That means, the optimal way to serve all requests during this time using ℓ servers costs 2 (this holds because this configuration is in the support of work function). All those requests are placed into an elementary subtree, which is equivalent to the paging problem. By [11] we get that for that request sequence, any online algorithm using ℓ servers incurs a cost at least ℓ. Thus, WFA pays at least 2ℓ inside R during the time it has ℓ servers there. $\square$

LEMMA 4.1 (RESTATED). *During Step 1, ALG= h^2 and ADV= h.*

Proof. Clearly, ADV= h; the adversary moves h servers by distance 1 and then serves all requests at zero cost. By lemma 4.8 we get that WFA incurs a cost $\sum_{\ell=1}^{h-1} 2\ell$ inside R. Moreover, it pays a movement cost of h to move its h servers from L to R. Overall, we get that the online cost during Step 1 is $\sum_{\ell=1}^{h-1} 2\ell + h = h^2$.

$\square$

LEMMA 4.9. *Consider an execution of S during Step 2, where the WFA uses only the h servers in R to serve the requests. For such an execution, ALG= ϵh^2 and ADV= ϵh.*

Proof. Same as proof of lemma 4.1 , just scale everything by ϵ. $\square$

LEMMA 4.2 (RESTATED). *During Step 2, ALG$\geq (2 - 2\epsilon)h^2 + h$ and ADV$\leq (2+\epsilon)h$.*

Proof. Second step consists of at most $2/\epsilon+1$ executions of S, where in each one of them the adversary incurs a cost $\epsilon \cdot h$. Thus the offline cost is at most $(2+\epsilon)h$.

Let us now count the online cost during Step 2. By lemma 4.6, there are $N-2 = \frac{2}{\epsilon} - 2$ executions of S such that WFA has h servers in R. By lemma 4.9 we get that during each one of those executions, the online cost is $\epsilon \cdot h^2$. For the rest of Step 2, the WFA incurs a cost of at least h, as it moves h servers from L to R. We get that overall, during Step 2, the cost of WFA is at least

$$(\frac{2}{\epsilon} - 2) \cdot \epsilon \cdot h^2 + h = (2-2\epsilon)h^2 + h.$$

$\square$

5 Concluding Remarks

Several intriguing open questions remain, and we list some of them here.

- Is the dependence on d in Theorem 1.3 necessary? While Theorem 1.4 gives a separation between depth-1 and depth-2 HSTs, it is unclear to us whether a lower bound which increases substantially with depth is possible. Note that a lower bound of $g(d)$ for depth d, where $g(d) \to \infty$ as $d \to \infty$ (provided that h is large enough), would be very surprising. This would imply that there is no $O(1)$-competitive algorithm for general metric spaces.

- Can we get an $o(h)$-competitive algorithm for other metric spaces? An interesting metric is the line: Both DC and WFA have competitive ratio $\Omega(h)$ in the line, and we do not even know any good candidate algorithm. Designing an algorithm with such a guarantee would be very interesting. Also, the only lower bound known for the line is 2 for $h = 2$. It would be interesting to get a non-trivial lower bound for arbitrary h.

References

[1] Nikhil Bansal, Marek Eliáš, Łukasz Jeż, Grigorios Koumoutsos, and Kirk Pruhs. Tight bounds for double coverage against weak adversaries. In *Approximation and Online Algorithms - 13th International Workshop (WAOA)*, pages 47–58, 2015.

[2] Yair Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *37th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 184–193, 1996.

[3] Yair Bartal and Elias Koutsoupias. On the competitive ratio of the work function algorithm for the k-server problem. *Theor. Comput. Sci.*, 324(2–3):337–345, 2004.

[4] Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis.* Cambridge University Press, 1998.

[5] Marek Chrobak, Howard J. Karloff, Thomas H. Payne, and Sundar Vishwanathan. New results on server problems. *SIAM J. Discrete Math.*, 4(2):172–181, 1991.

[6] Marek Chrobak and Lawrence L. Larmore. An optimal on-line algorithm for k-servers on trees. *SIAM J. Comput.*, 20(1):144–148, 1991.

[7] Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *J. ACM*, 47(4):617–643, July 2000.

[8] Elias Koutsoupias. Weak adversaries for the k-server problem. In *Proc. of the 40th Symp. on Foundations of Computer Science (FOCS)*, pages 444–449, 1999.

[9] Elias Koutsoupias and Christos H. Papadimitriou. On the k-server conjecture. *J. ACM*, 42(5):971–983, 1995.

[10] Mark S. Manasse, Lyle A. McGeoch, and Daniel D. Sleator. Competitive algorithms for server problems. *J. ACM*, 11(2):208–230, 1990.

[11] Daniel Dominic Sleator and Robert Endre Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985.

[12] Neal E. Young. The k-server dual and loose competitiveness for paging. *Algorithmica*, 11(6):525–541, 1994.

A Algorithm

A.1 Proofs of lemmas from Section 3

Proof. During the Phase 1, each server of the algorithm resides either in T_u or in T_s for some $s \in E \cup D$; therefore $k = k_u + \sum_{s\in E} k_s + \sum_{s\in D} k_s$. However, for each $s \in D$, we have $k_s \leq \lfloor \beta_{\ell(s)} \cdot h_s \rfloor$, otherwise E_e would be positive for the edge e containing s. Therefore we have

$$\sum_{s\in D} k_s \leq \sum_{s\in D} \lfloor \beta_{\ell(s)} h_s \rfloor \leq \sum_{s\in D} \lfloor \beta_d h_s \rfloor \leq$$

$$\leq \sum_{s\in D} \beta_d \cdot h_s \leq \beta_d (h - h_u).$$

To prove the second inequality, observe that

$$\sum_{s\in E} k_s = k - k_u - \sum_{s\in D} k_s \geq k - k_u - \beta_d (h - h_u).$$

However, we assumed that $k_u \leq \lfloor \beta_2 h_u \rfloor \leq \beta_d h_u$, and therefore $\sum_{s\in E} k_s \geq k - \beta_d \cdot h$. □

Proof. [Proof of Lemma 3.2] In this lemma, we crucially use the fact that $\beta_\ell = \beta \cdot \beta_{\ell-1}$. Similarly to the previous proof, we have $k_s \leq \lfloor \beta_{\ell(s)} \cdot h_s \rfloor \leq \beta_{\ell-1} \cdot h_s$ for each server $s \in D$, since s can be located in a level at most $\ell - 1$. However, in this case we claim that

$$\sum_{s\in D} k_s \leq \sum_{s\in D} \beta_{\ell-1} \cdot h_s \leq \beta_{\ell-1}(h_q^- - 1).$$

This is because we assume that adversary has already served the request and some server a, one of his h_q^- servers in T_q^-, is already at the requested point. Since no online servers reside in the path between q and the requested point, a does not belong to T_s for any $s \in D \cup E$. Therefore we have $\sum_{s\in D} h_s \leq \sum_{s\in(D\cup E)} h_s \leq h_q^- - 1$. To finish the proof of the first inequality, observe that our assumption implies

$$k_q^- \geq \lfloor \beta\beta_{\ell-1} h_q^- \rfloor \geq \beta\beta_{\ell-1} h_q^- - 1 \geq \beta(\beta_{\ell-1} h_q^- - 1).$$

Therefore we have $\beta_{\ell-1}(h_q^- - 1) \leq \beta_{\ell-1} h_q^- - 1 \leq k_q^- / \beta$.

For the second inequality, note that $\frac{1}{\gamma} = (1 - \frac{1}{\beta})$. Since $k_q^- = \sum_{s\in D} k_s + \sum_{s\in E} k_s$, we have

$$\sum_{s\in E} k_s \geq k_q^- - \frac{1}{\beta} k_q^- = \frac{1}{\gamma} k_q^-.$$

□

A.2 Algorithm for bounded-diameter trees

Since Algorithm 1 works for depth-d trees with arbitrary edge lengths, we can embed any diameter-d tree into a depth-d tree with a very small distortion by adding fake paths of short edges to all nodes.

More precisely, let T be a tree of diameter d with arbitrary edge lengths, and let α be the length of the shortest edge of T (for any finite T such α exists). We fix $\epsilon > 0$ a small constant. We create an embedding T' of T as follows. We choose the root r arbitrarily, and to each node $v \in T$ such that the path from r to v contains ℓ edges, we attach a path containing $d - \ell$ edges of total length $\epsilon\alpha/2$. The leaf at the end of this path we denote v'. We run Algorithm 1 in T' and each request at $v \in T$ we translate to $v' \in T'$. We maintain the correspondence between the servers in T and the servers in T', and the same server which is finally moved to v' by Algorithm 1, we also use to serve the request $v \in T$.

For the optimal solutions on T and T' we have $(1+\epsilon)OPT(T) \geq OPT(T')$, since any feasible solution in T we can be converted to a solution in T' with cost at most $(1+\epsilon)$ times higher. By Theorem 3.1, we know that the cost of Algorithm 1 in T' is at most $R \cdot OPT(T')$, for $R = \Theta(d \cdot \gamma^{d+1})$, and therefore we have $ALG(T') \leq (1+\epsilon)R \cdot OPT(T)$.

B Lower bound for WFA on the line

The lower bound of Section 4.2 for the WFA can also be applied on the line. The lower bound strategy is the same as the one described for depth-3 HSTs, we just need to replace subtrees by line segments.

More precisely, the lower bound strategy is applied in an interval I of length 1. Let L and R be the leftmost and rightmost regions of I, of length $\epsilon/2$, for $\epsilon \ll 1$. Similarly, L_1, L_2 and R_1, R_2 are smaller regions inside L and R respectively. Again, the distance between L_1 and L_2 (R_1 and R_2 resp.) is much larger than their length. Whenever the adversary moves to such a region, it places its servers in h equally spaced points inside it. Similarly to the proof of Theorem 1.2, we can get a lower bound $h + 1/3$ on the competitive ratio of the WFA on the line when $k = 2h$.

Interestingly, the lower bound obtained by this construction for the line has a matching upper bound. As it was observed in [1], results from [3], [8] imply that for the line the cost of WFA with k servers WFA_k is at most $(h+1)\mathrm{OPT}_h - \mathrm{OPT}_k + \mathrm{const}$, where OPT_i is the optimal cost using i servers. Briefly, this upper bound holds for the following reason: In [8] it was shown that in general metrics, $\mathrm{WFA}_k \leq 2h\mathrm{OPT}_h - \mathrm{OPT}_k + \mathrm{const}$. However, if we restrict our attention to the line, using the result of [3], we can get that

$$\mathrm{WFA}_k \leq (h+1)\mathrm{OPT}_h - \mathrm{OPT}_k + \mathrm{const}\ . \tag{B.1}$$

See [1] for more details.

In theorem 1.2 we showed a lower bound $(h + 1/3)\mathrm{OPT}_h$ on WFA_k. We now show that this construction matches the upper bound of (B.1). In particular, it suffices to show that $(h + 1/3)\mathrm{OPT}_h$ goes arbitrary close to $(h+1)\mathrm{OPT}_h - \mathrm{OPT}_k$, or equivalently

$$\frac{2\mathrm{OPT}_h}{3} \to \mathrm{OPT}_k \tag{B.2}$$

As we showed in the proof of theorem 1.2, for every phase of the lower bound strategy, $\mathrm{OPT}_h \leq (3+\epsilon)\cdot h$. Moreover it is clear that $\mathrm{OPT}_k = 2h$; during a phase the minimum work function value using k servers increases by exactly $2h$. We get that

$$\frac{2\mathrm{OPT}_h}{3} \leq \frac{2\cdot(3+\epsilon)\cdot h}{3} = 2h\frac{3+\epsilon}{3} \to 2h = \mathrm{OPT}_k.$$