The Infinite Server Problem*

Christian Coester[1], Elias Koutsoupias[1], and Philip Lazos[1]

[1]Department of Computer Science, University of Oxford

April 27, 2017

Abstract

We study a variant of the k-server problem, the infinite server problem, in which infinitely many servers reside initially at a particular point of the metric space and serve a sequence of requests. In the framework of competitive analysis, we show a surprisingly tight connection between this problem and the (h, k)-server problem, in which an online algorithm with k servers competes against an offline algorithm with h servers. Specifically, we show that the infinite server problem has bounded competitive ratio if and only if the (h, k)-server problem has bounded competitive ratio for some $k = O(h)$. We give a lower bound of 3.146 for the competitive ratio of the infinite server problem, which implies the same lower bound for the (h, k)-server problem even when $k/h \to \infty$ and holds also for the line metric; the previous known bounds were 2.4 for general metric spaces and 2 for the line. For weighted trees and layered graphs we obtain upper bounds, although they depend on the depth. Of particular interest is the infinite server problem on the line, which we show to be equivalent to the seemingly easier case in which all requests are in a fixed bounded interval away from the original position of the servers. This is a special case of a more general reduction from arbitrary metric spaces to bounded subspaces. Unfortunately, classical approaches (double coverage and generalizations, work function algorithm, balancing algorithms) fail even for this special case.

1 Introduction

The k-server problem is a fundamental well-studied online problem [19, 16]. In this problem k servers serve a sequence of requests. The servers reside at k points of a metric space M and requests are simply points of M. Serving a request entails moving one of the servers to the request. The objective is to minimize the total distance traveled by the servers. The most interesting variant of the problem is its online version, in which the requests appear one-by-one and the online algorithm must decide how to serve a request without knowing the future requests. It is known that the deterministic k-server problem has competitive ratio between k and $2k - 1$ for every metric space with at least $k + 1$ distinct points [19, 18].

In this paper, we study the *infinite server problem*, the variant of the k-server problem in which there are infinitely many servers, all of them initially residing at a given point, the *source*[1]. At first glance it may appear that the lower bound of k for the k-server problem would imply

*Supported by the ERC Advanced Grant 321171 (ALGAME) and by EPSRC.

[1]We first learned about this problem from Kamal Jain [14].

an unbounded competitive ratio for the infinite server problem. But consider, for example, the version of the k-server problem on uniform metric spaces (i. e. the distance between any two points is 1), and observe that the infinite server problem has competitive ratio 1 for this case.

The infinite server problem is closely related to the (h, k)-server problem, the resource augmentation version of the k-server problem in which the online algorithm has k servers and competes against an offline algorithm for $h \leq k$ servers. This model is also known as *weak adversaries* [2, 15]. One major open problem in competitive analysis is whether the (h, k)-server problem has bounded competitive ratio when $k \gg h$. Here we show a, perhaps surprising, tight connection between the infinite server problem and the (h, k)-server problem, which also allows us to improve lower bounds for the latter.

The infinite server problem is also a considerable generalization of the ski-rental problem, since the ski-rental problem is essentially a special case of the infinite server problem when the metric space is an isosceles triangle.

Besides its theoretical appeal, our three main reasons for investigating the infinite server problem are the following. First, the competitive ratio of the k-server problem goes to infinity as $k \to \infty$, but for $k = \infty$ it goes back to a small constant at least on some metric spaces. This suggests that the high competitive ratio of the k-server problem is somewhat artificial. Second, the problem allows to model applications where the number of servers is so high that it is not a limitation in practice, or where more servers can be bought. A price for buying new servers can be modeled easily by appropriate placement of the source in the metric space. Third, the relationship between the infinite server problem and the (h, k)-server problem allows for new ways to tackle the latter.

1.1 Previous Work

The k-server problem was first formulated by Manasse et al. [19], to generalize a variety of online settings whose stepwise cost had a ‘metric’-like structure. They built on previous work by Sleator and Tarjan [20], the genesis of competitive analysis, on the paging problem. This problem can be easily recast as a k-server instance for the uniform metric and was already known to be k-competitive.

Manasse et al. [19] also showed that the competitive ratio of the k-server problem is at least k on any metric space with more than k points. They then proposed the renowned k-server conjecture, stating that this bound is tight. This has been shown to be true for $k = 2$ [19] and for several special metric spaces [6, 7, 17, 19, 20]. A stream of refinements [12, 3] lead to better competitive ratios for general metric spaces until [18] showed that a competitive ratio of $2k - 1$ can be achieved on any metric space. Chasing the competitive ratio for the deterministic (and randomized) k-server problem has been pivotal for the development of competitive analysis. For a more in depth view on the history of the k-server problem and further related work, we refer to [16].

In the weak adversaries setting, significantly less is known. For the (h, k)-server problem, the exact competitive ratio is $\frac{k}{k-h+1}$ on uniform metrics (equivalent to paging) [20] and weighted stars (equivalent to weighted paging) [21]. Bansal et al. [1] showed recently for weighted trees that the competitive ratio as $k/h \to \infty$ is bounded by a constant depending on the depth of the tree. On general metrics, the (h, k)-server problem is still poorly understood. No algorithm is known for general metrics that performs better than disabling the $k - h$ extra servers and using h servers only. In fact, for the line it was shown [2, 1] that the Double Coverage Algorithm and the Work Function Algorithm – despite achieving the optimal competitive ratio of h if $k = h$

[6, 4] – perform strictly worse in the resource augmentation setting than disabling the $k-h$ extra servers and applying the same algorithm to h servers only. For the case that h is not fixed, the Work Function Algorithm was shown to be $2h$-competitive simultaneously against any number $h \leq k$ of offline servers [15].

In terms of lower bounds, it is known that unlike for (weighted) paging, the competitive ratio does not converge to 1 on general metrics even as $k/h \to \infty$. Prior to this work, the best known lower bounds were 2 on the line, due to Bar-Noy and Schieber (see [5, p. 175]), and 2.4 for general metrics as shown recently by Bansal et al. [1].

The closest publication to this work is by Csirik et al. [10], which studies a problem that is essentially the special case of the infinite server problem on the uniform metric space augmented by a far away source. It is cast as a paging problem where new cache slots can be bought at a fixed price per unit and gives matching upper and lower bounds of ≈ 3.146 on the competitive ratio.

1.2 Our Results

Our main result is an equivalence theorem between the infinite server problem and the (h, k)-server problem, presented in Section 2. It states that the infinite server problem is competitive on every metric space if and only if the (h, k)-server problem is $O(1)$-competitive on every metric space as $k/h \to \infty$. We show further that it is not even necessary to let k/h tend to infinity because in the positive case, there must also exist some $k = O(h)$. The theorem holds also if "every metric space" is replaced by "the real line".

In Section 3 we present upper and lower bounds on the competitive ratio of the infinite server problem on a variety of metric spaces. Extending the work in [10], we present a tight lower bound for non-discrete spaces, which is then turned into a 3.146 lower bound for the (h, k) setting. To our knowledge, this is the largest bound on the weak adversaries setting for any metric space, as $k/h \to \infty$. We show how recent work by Bansal et al. [1] can be adapted to give an upper bound on the competitive ratio of the infinite server problem on bounded-depth weighted trees. We also consider layered graph metrics, which are equivalent (up to a factor of 2) to general graph metrics. We have not settled the case for their competitive ratio, but we present a natural algorithm with tight analysis and pose challenges for further research. The main open problem is whether there exists a metric space on which the infinite server problem is not competitive.

In Section 4 we show how a variety of known algorithms such as the work function and balancing algorithms fail for the infinite server problem, even on the real line. We focus in particular on a class of speed-adjusted variants of the well-known double coverage algorithm.

Finally, we present a useful reduction from arbitrary metric spaces to bounded subspaces in Section 5. In particular, the infinite server problem on the line is competitive if and only if it is competitive for the special case where requests are restricted to some bounded interval further away from the source.

1.3 Preliminaries

Let $M = (M, d)$ be a metric space and let s be a point of M. In the *infinite server problem on* (M, s), an unbounded number of servers starts at point s and serves a finite sequence $\sigma = (\sigma_0 = s, \sigma_1, \sigma_2, \ldots, \sigma_m)$ of requests $\sigma_i \in M$. Serving a request entails moving one of the servers to it. The goal is to minimize the total distance traveled by the servers.

We drop s in the notation if the location of the source is not relevant or understood. We refer to the action of moving a server from the source to another point as *spawning*. Throughout this work we use the letter d for the metric associated with the metric space.

In the online setting, the requests are revealed one by one and need to be served immediately without knowledge of future requests. All algorithms considered in this paper are deterministic. An algorithm is called *lazy* if it moves only one server to serve a request at an unoccupied point and moves no server if the requested point is already covered. An algorithm is called *local* [9] if it moves a server from a to b only if there is no server at some other point c on a shortest path from a to b, i. e. with $d(a,b) = d(a,c) + d(c,b)$. It is easy to see that any algorithm can be turned into a lazy and local algorithm without increasing its cost (i. e. the total distance traveled by all servers).

For an algorithm ALG, we denote by $ALG(\sigma)$ its cost on the request sequence σ. Similarly, we write $OPT(\sigma)$ for the optimal (offline) cost.

An online algorithm ALG is *ρ-competitive* for $\rho \geq 1$ if $ALG(\sigma) \leq \rho OPT(\sigma) + c$ for all σ, where c is a constant independent of σ. The *competitive ratio* of an algorithm is the infimum of all such ρ. We say that an algorithm is *competitive* if it is ρ-competitive for some ρ. We also call an online problem itself (ρ-)competitive if it admits such an algorithm. If the additive term c in the definition is 0, then the algorithm is also called *strictly ρ-competitive* [11].

The *(h,k)-server problem on M* is defined like the infinite server problem except that the number of servers is k for the online algorithm and h for the optimal (offline) algorithm against whom it is compared in the definition of competitiveness. For this problem, the servers are not required to start at the same point, although a different initial configuration would only affect the additive term c. The problem is interesting only when $k \geq h$. The case $h = k$ is the standard k-server problem and the case $k \geq h$ is known as the *weak adversaries* model. One major open problem is to determine the competitive ratio of the (h,k)-server problem as k tends to infinity.

We will sometimes write OPT_h and OPT_∞ for the optimal offline algorithm, where the index specifies the number of servers available.

The following two propositions will be useful later in the paper.

Proposition 1. *If for every metric space there exists a competitive algorithm for the infinite server problem, then there exists a universal competitive ratio ρ such that the infinite server problem is* strictly *ρ-competitive on every metric space.*

Proof. We first show the existence of ρ such that the infinite server problem is ρ-competitive (strictly or not) on every metric space. Suppose such ρ does not exist, then for every $n \in \mathbb{N}$ we can find a metric space M_n containing some point s_n such that the infinite server problem on (M_n, s_n) is not n-competitive. Consider the metric space obtained by taking the disjoint union of all spaces M_n and gluing all the points s_n together. The infinite server problem would not be competitive on this metric space, in contradiction to the assumption.

Analogously we can also find a universal constant c that works for all metric spaces as additive constant in the definition of ρ-competitiveness. A scaling argument shows that also $c = 0$ works. □

With a very similar argument we get:

Proposition 2. *Let $k = k(h)$ be a function of h. Suppose that for every metric space M and for all h there exists an $O(1)$-competitive algorithm for the (h,k)-server problem on M. Then there exists a universal competitive ratio ρ such that the (h,k)-server problem is strictly ρ-competitive on every metric space if all servers start at the same point.*

2 Equivalence of Infinite Servers and Weak Adversaries

The main result of this section is the following tight connection between the infinite server problem and the weak adversaries model.

Theorem 3. *The following are equivalent:*

(a) The infinite server problem is competitive.

(b) The (h, k)-server problem is $O(1)$-competitive as $k/h \to \infty$.

(c) For each h there exists $k = O(h)$ so that the (h, k)-server problem is $O(1)$-competitive.

The three statements above are also equivalent if we fix the metric space to be the real line.

The implication "(c) $\Longrightarrow$ (b)" is trivial. The proof of the equivalence theorem consists in its core of two reductions. Theorem 4 contains the easier of the two reductions, which is from the infinite server problem to the k-server problem against weak adversaries ("(b) $\Longrightarrow$ (a)"). By Propositions 1 and 2, it suffices to consider only strictly competitive algorithms. Theorem 5 proves essentially the inverse for general metric spaces, and Theorem 6 specializes it to the line ("(a) $\Longrightarrow$ (c)").

As a corollary of the theorem we get the non-trivial implication "(b) $\Longrightarrow$ (c)", a potentially useful statement towards resolving the major open problem about weak adversaries: "Is Statement (b) true?" This highlights the importance of the infinite server problem.

Theorem 4. *Fix a metric space M and consider algorithms with all servers starting at some $s \in M$. If for every h there exists $k = k(h)$ such that the (h, k)-server problem on M is strictly ρ-competitive, for some constant ρ, then there exists a strictly ρ-competitive online strategy for the infinite server problem on M.*

Proof. Let $ALG_{k(h)}$ denote an online algorithm with $k(h)$ servers that is strictly ρ-competitive against an optimal algorithm OPT_h for h servers, i. e.

$$ALG_{k(h)}(\sigma) \leq \rho OPT_h(\sigma) \tag{1}$$

for every request sequence σ. Without loss of generality, algorithm $ALG_{k(h)}$ is lazy.

For every request sequence σ, consider the equivalence relation $\equiv_\sigma$ on natural numbers in which $h \equiv_\sigma h'$ if and only if $ALG_{k(h)}(\sigma)$ and $ALG_{k(h')}(\sigma)$ serve σ in exactly the same way (i. e., make exactly the same moves). To every σ, we associate an equivalence class $H(\sigma)$ of $\equiv_\sigma$ that satisfies

- $H(\sigma)$ is infinite,
- $H(\sigma r) \subseteq H(\sigma)$, for every request r.

This is done inductively in the length of σ (in a manner reminiscent of König's lemma) as follows: For the base case when σ is the empty request sequence, $H(\sigma) = \mathbb{N}$. For the induction step, suppose that we have defined $H(\sigma)$. Consider the equivalence classes of $\equiv_{\sigma r}$, a refinement of the equivalence classes of $\equiv_\sigma$. Since there are only finitely many possible ways to serve r, they partition $H(\sigma)$ into finitely many parts. At least one of these parts is infinite and we select it to be $H(\sigma r)$; if there is more than one such sets, we select one arbitrarily, say the lexicographically first.

Given such a mapping H, we define the online algorithm ALG_∞ which serves every σ in the same way as all the online algorithms $ALG_{k(h)}$ for $h \in H(\sigma)$. The second property of H guarantees that ALG_∞ is a well-defined online algorithm.

By construction, $ALG_\infty(\sigma) = ALG_{k(h)}(\sigma)$ for every $h \in H(\sigma)$. To finish the proof, observe that since $H(\sigma)$ is infinite, it contains some h greater than the length of σ, and for such an h we have $OPT_\infty(\sigma) = OPT_h(\sigma)$. Substituting these to (1), we see that ALG_∞ is strictly ρ-competitive. □

We now show the reduction from the k-server problem against weak adversaries to the infinite server problem on general metric spaces.

Theorem 5. *If the infinite server problem on general metric spaces is strictly $\tilde{\rho}$-competitive, then there exists a constant ρ such that the (h,k)-server problem is ρ-competitive, for $k = O(h)$. In particular, for every $\epsilon > 0$, we can take $\rho = (3+\epsilon)\tilde{\rho}$ and any $k \geq (1+1/\epsilon)\tilde{\rho}h$.*

Proof. Fix some metric space M and a point $s \in M$. We will describe a strictly ρ-competitive algorithm for the (h,k)-server problem on M for the case that all servers start at s. This implies a (not necessarily strictly) ρ-competitive algorithm for any initial configuration.

The idea is to simulate a strictly $\tilde{\rho}$-competitive infinite server algorithm, but whenever it would spawn a $(k+1)$-st server, we bring all servers back to the origin and restart the algorithm. The problem is that the overhead cost for returning the servers to the origin, may be very high. To compensate for this, we assume that every time the servers return to the origin, they pretend to start from a different point further away from the origin. This motivates the following notation:

Definition 1. *Given a metric M, a point $s \in M$, and a value $w \geq 0$, we will use the notation $M_{s \oplus w}$ to denote the metric derived from M when we increase the distance of s from every other point by w; we will also denote the relocated point by $s \oplus w$.*

Let ALG_∞ denote a strictly $\tilde{\rho}$-competitive online algorithm for the infinite server problem. We now define an online algorithm ALG_k for k servers (all starting at s). We will make use of the notation $A(\sigma; s)$ to denote the cost of algorithm A to serve the request sequence σ when all servers start at s.

Definition 2 (ALG_k derived from ALG_∞). *Algorithm ALG_k runs in phases with the initial phase being the 0th phase. At the beginning of every phase, all servers of ALG_k are at s. In every phase i, the algorithm simulates the infinite server algorithm ALG_∞, whose servers start at $s \oplus w_i$ for some $w_i \geq 0$. The parameters w_i are determined online, and initially $w_0 = 0$. Whenever ALG_∞ spawns a server from $s \oplus w_i$, algorithm ALG_k spawns a server from s.*

The phase ends just before ALG_∞ spawns its $(k+1)$-st server or when the request sequence ends. In the former case, all servers of ALG_k return to s to start the $(i+1)$-st phase. To determine the starting point of the simulated algorithm of the next phase, we set

$$w_{i+1} = \epsilon \, \frac{OPT_h(\sigma_i; s)}{h}\,, \tag{2}$$

where σ_i is the sequence of requests during phase i.

Let n be the number of phases. The cost of ALG_k for the requests in phase $i < n$ is $ALG_\infty(\sigma_i; s \oplus w_i) - kw_i$; the last term is subtracted because the k servers do not have to actually travel the distance between $s \oplus w_i$ and s. However for the last phase no such term can be subtracted since we do not know how many servers are spawned during the phase, and we can only bound the cost from above by $ALG_\infty(\sigma_n; s \oplus w_n)$. The cost of returning the servers to s at the end of a phase can at most double the cost during the phase.

From this, we see that the total cost of ALG_k in phase i is

$$cost_i \leq \begin{cases} 2\left(ALG_\infty(\sigma_i; s \oplus w_i) - kw_i\right) & \text{for } i < n \\ ALG_\infty(\sigma_n; s \oplus w_n) & \text{for } i = n\,. \end{cases}$$

Since ALG_∞ is strictly $\tilde{\rho}$-competitive, we have

$$\begin{aligned} ALG_\infty(\sigma_i; s \oplus w_i) &\leq \tilde{\rho}\, OPT_\infty(\sigma_i; s \oplus w_i) \\ &\leq \tilde{\rho}\, OPT_h(\sigma_i; s \oplus w_i) \\ &\leq \tilde{\rho}\left(OPT_h(\sigma_i; s) + hw_i\right) \end{aligned}$$

and substituting this in the expression for the cost, we can bound the total cost by

$$\begin{aligned} ALG_k(\sigma; s) = \sum_{i=0}^{n} cost_i &\leq 2\sum_{i=0}^{n-1}\left(\tilde{\rho}(OPT_h(\sigma_i; s) + hw_i) - kw_i\right) + \tilde{\rho}(OPT_h(\sigma_n; s) + hw_n) \\ &= 2\sum_{i=0}^{n-1}\left(\tilde{\rho}OPT_h(\sigma_i; s) - (k - \tilde{\rho}h)w_i\right) + \tilde{\rho}OPT_h(\sigma_n; s) + \tilde{\rho}hw_n\,. \end{aligned}$$

The parameters w_i and k were selected so that the summation telescopes, and we are left with

$$\begin{aligned} ALG_k(\sigma; s) &\leq 2\,\tilde{\rho}\, OPT_h(\sigma_{n-1}; s) + \tilde{\rho}\, OPT_h(\sigma_n; s) + \tilde{\rho}\,\epsilon\, OPT_h(\sigma_{n-1}; s) \\ &\leq (3 + \epsilon)\,\tilde{\rho}\, OPT_h(\sigma; s)\,. \end{aligned}$$ □

The previous reduction requires the infinite server problem to be competitive on *every* metric space. The following variant only requires the infinite server problem to be competitive on the line.

Theorem 6. *If the infinite server problem on the line is ρ-competitive, then for every $h \in \mathbb{N}$ and $\epsilon > 0$, the (h,k)-server problem on the line is $(3+\epsilon)\rho$-competitive, when $k \geq 2\lceil(1+1/\epsilon)\rho h\rceil$.*

Proof. A straightforward adaptation of the proof of the previous lemma, shows the existence of a $(3 + \epsilon)\rho$-competitive algorithm for the interval $[0, \infty)$, when $k \geq 2(1 + 1/\epsilon)\rho h$. By doubling the number of online servers so that half of them are used in each half-line, we get a $(3 + \epsilon)\rho$-competitive algorithm for the entire line, when $k \geq 2\lceil(1 + 1/\epsilon)\rho h\rceil$.

Note that the proof assumes strictly competitive algorithms. But, by a straightforward scaling argument, if the infinite server problem on the line is ρ-competitive, then it is also strictly ρ-competitive. This in turn implies a strictly ρ-competitive online algorithm for $M_{0\oplus w}$, since this space is isometric to the subspace $\{-w\} \cup (0, \infty)$ of the line. □

In the next section we look at some particular metric spaces and give upper and lower bounds on the competitive ratio.

3 Upper and Lower Bounds

Unlike the k-server problem, which is 1-competitive if and only if the metric spaces has at most k points and conjectured k-competitive otherwise, the situation is more diverse for the infinite server problem. For example, on uniform metric spaces (where all distances are the same) the problem is trivially 1-competitive even if the metric space consists of uncountably many points. This is because an optimal strategy in this case is to spawn a server to every requested point. More generally, this strategy achieves a finite competitive ratio on any metric space where distances are bounded from below and above by positive constants. This suggests that statements about the competitive ratio for the infinite server problem cannot be as simple as the (conjectured) dichotomy for the k-server problem, which depends only on the number of points of the metric space. In this section we derive bounds on the competitive ratio for particular classes of metric spaces.

3.1 Weighted Trees

We consider the infinite server problem on metric spaces that can be modeled by edge-weighted trees. The points of the metric space are the nodes of the tree, and the distance between two nodes is the sum of edge weights along their connecting path. We choose the source of the metric space as the root of the tree, and define the depth of the tree as the maximal number of edges from the root to a leaf. The number of nodes can be infinite (otherwise the infinite server problem is trivially 1-competitive), but we assume the depth to be finite.

An upper bound on the competitive ratio of such trees follows easily from an upper bound for the (h, k)-server on such trees [1] and the equivalence theorem:

Theorem 7. *The competitive ratio of the infinite server problem on trees of depth d is at most $O(2^d \cdot d)$.*

Proof. Bansal et al. [1, Theorem 1.3] showed that the competitive ratio of the (h, k)-server problem on trees of depth d is at most $O(2^d \cdot d)$ provided that k/h is large enough. Inspection of the proof in [1] shows that if all servers start at the root, it is in fact strictly $O(2^d \cdot d)$-competitive. Thus, Theorem 4 implies the result for the infinite server problem. □

3.2 Non-Discrete Spaces and Spaces with Small Infinite Subspaces

The following theorem gives a lower bound of 3.146 on the competitive ratio of the infinite server problem on any metric space containing an infinite subspace of a diameter that is small compared to the subspace's distance from the source. For example, every non-discrete metric space has this property (unless the source is the only non-discrete point), since non-discrete metric spaces contain infinite subspaces of arbitrarily small diameter. The theorem is a generalization of such a lower bound established in [10] for a variant of the paging problem where cache cells can be bought. Crucial parts of the subsequent proof are as in [10].

Theorem 8. *Let M be a metric space containing an infinite subspace $M_0 \subset M$ of finite diameter δ and a point $s \in M \setminus M_0$ such that the infimum Δ of distances between s and points in M_0 is positive. Let $\lambda > 3.146$ be the largest real solution to*

$$\lambda = 2 + \ln \lambda . \tag{3}$$

The competitive ratio of any deterministic online algorithm for the infinite server problem on (M, s) is bounded from below by a value that converges to λ as $\Delta/\delta \to \infty$. In particular, the competitive ratio is at least λ if $M \setminus \{s\}$ contains a non-discrete part.

Proof. By scaling the metric, we can assume that $\delta = 1$. Let $p_1, p_2, p_3, \ldots$ be infinitely many distinct points in M_0.

Fix some lazy deterministic online algorithm ALG. We consider the request sequence that always requests the point p_i with i minimal such that p_i is not occupied by a server of ALG. We call a move of a server between two points in M_0 *local* (i. e. every move that does not spawn is local). Let f_j be the cumulative cost of local moves incurred to ALG until it spawns its jth server. Let σ_k be this request sequence that is stopped right after ALG spawns its kth server, for some large k. The total online cost is

$$ALG(\sigma_k) \geq k\Delta + f_k \,. \tag{4}$$

Let $h = \lceil k/\lambda \rceil$. We consider several offline algorithms that start behaving the same way, so we think of it as one algorithm initially that is forked into several algorithms later. The offline algorithms make use of only h servers and they begin by spawning them to the points $p_1, \ldots, p_h$. They do not need to move any servers until ALG spawns its hth server. Whenever ALG spawns its jth server for some $j \geq h$, every offline algorithm is forked to h distinct algorithms: Each of them moves a different server to p_{j+1} (to prepare for the next request, which will be at p_{j+1}). We will keep the invariant that each offline algorithm already has a server at the next request. To this end, whenever ALG does a local move from p to p', every offline algorithm that does not have a server at p moves a server from p' to p; note that the algorithm had a server at p' by the invariant, and the next request will be at p.

When ALG has j spawned servers ($j \geq h$), the offline algorithms are in $\binom{j}{h-1}$ different configurations, each of which occurs equally often among them. If ALG does a local move from p to p', there are $\binom{j-1}{h-1}$ different offline configurations for which a local move is made in the opposite direction. Thus, for each local move by ALG while having j servers in total, a portion $\binom{j-1}{h-1}/\binom{j}{h-1} = \frac{j-h+1}{j}$ of the offline algorithms move a server in the opposite direction for the same cost.

We use the average cost of all offline algorithms we considered as an upper bound on the optimal cost. The cost of spawning h servers is at most $h(\Delta + 1)$, and the average cost while ALG has j spawned servers (for $j = h, \ldots, k-1$) is at most $\frac{j-h+1}{j}(f_{j+1} - f_j) + 1$ (with the "+1" coming from the move when offline algorithms fork). Hence,

$$\begin{aligned} OPT(\sigma_k) &\leq h(\Delta + 1) + k - h + \sum_{j=h}^{k-1} \frac{j-h+1}{j}(f_{j+1} - f_j)\,, \\ &\leq h\Delta + k + \frac{k-h}{k-1} f_k - \frac{f_h}{h} - \sum_{j=h+1}^{k-1} \frac{h-1}{j(j-1)} f_j\,, \end{aligned}$$

Note that $\frac{f_k}{k}$ is bounded from above because otherwise ALG would not be competitive, and it is bounded from below by 0. Thus, $L = \liminf_{k\to\infty} \frac{f_k}{k}$ exists. In the following we use the asymptotic notation $o(1)$ for terms that disappear as $k \to \infty$. We can choose arbitrarily large values of k such that $\frac{f_k}{k} = L + o(1)$. Since $h = \lceil k/\lambda \rceil$, we have $\frac{f_j}{j} \geq L + o(1)$ for all $j \geq h$.

Moreover, $\sum_{j=h+1}^{k-1} \frac{1}{j-1} = \ln(\lambda) + o(1)$. This allows us to simplify the previous bound to

$$\begin{aligned} OPT(\sigma_k) &\leq \frac{k}{\lambda}\Big(\Delta + \lambda + \big(\lambda - 1 - \ln(\lambda)\big)L + o(1)\Big) \\ &= \frac{k}{\lambda}\big(\Delta + L + \lambda + o(1)\big)\,, \end{aligned}$$

where the last step uses equation (3).

The competitive ratio is at least

$$\begin{aligned} \frac{ALG(\sigma_k) + O(1)}{OPT(\sigma_k)} &\geq \frac{k\Delta + f_k + O(1)}{\frac{k}{\lambda}\big(\Delta + L + \lambda + o(1)\big)} \\ &= \lambda \cdot \frac{\Delta + L}{\Delta + L + \lambda} + o(1)\,. \end{aligned}$$

The fraction in the last term tends to 1 as $\Delta \to \infty$. □

This bound is tight due to a matching upper bound in [10] that shows (translated to the terminology of the infinite server problem) that a competitive ratio of λ can be achieved on metric spaces where all pairwise distances are 1 except that the source is at some larger distance Δ from the other points.

The previous theorem together with the equivalence theorem also allows us to obtain a new lower bound for the k-server problem against weak adversaries.

Corollary 9. *For sufficiently large h, there is no 3.146-competitive algorithm for the (h, k)-server problem on the line, even if $k \to \infty$.*

Proof. By a scaling argument it is easy to see that if the infinite server problem on the line is ρ-competitive, then it is also *strictly* ρ-competitive. Thus, the statement follows from Theorems 4 and 8. □

This improves upon both the previous best known lower bounds of 2 for this problem on the line [5, p. 175] and 2.4 on general metric spaces [1].

3.3 Layered Graphs

A *layered graph of depth D* is a graph whose (potentially infinitely many) nodes can be arranged in layers $0, 1, \ldots, D$ so that all edges run between adjacent layers and each node – except for a single node in layer 0 – is connected to at least one node of the previous layer. The induced metric space is the set of nodes with the distance being the minimal number of edges of a connecting path. For the purposes of the infinite server problem, the single node in layer 0 is the source. We assume $D \geq 2$ to avoid trivial cases.

Note that a connected graph is layered if and only if it is bipartite. Moreover, any graph can be embedded into a bipartite graph by adding a new node in the middle of each edge. So essentially, layered graphs capture *all* graph metrics.

Let *Move Only Outwards (MOO)* be some lazy and local algorithm for the infinite server problem on layered graphs that moves servers along edges only in the direction away from the source. Not surprisingly, the competitive ratio of this simple algorithm is quite bad and we show that it is exactly $D - 1/2$. Nonetheless, at least for $D \leq 3$ this is actually the optimal competitive ratio.

Theorem 10. *The competitive ratio of MOO is exactly $D - \frac{1}{2}$.*

Proof.
Upper bound:
Consider some final configuration of the algorithm. Let n_j be the number of servers in the jth layer. Then the cost of MOO is

$$cost = \sum_{j=1}^{D} j n_j .$$

To obtain an upper bound on OPT, observe that every node occupied by MOO in the final configuration must have been visited by an offline server at least once. We account an offline cost of 1 for each visit of a node on layers $1, \dots, D-2$ and an offline cost of 2 for each visit of a node on layer D. This cost of 2 covers the last two edge-traversals before visiting the layer-D-node, so this may include serving a request on layer $D-1$. If $n_{D-1} > n_D$, then we can account another $n_{D-1} - n_D$ cost for visiting the remaining at least $n_{D-1} - n_D$ requested nodes on layer $D-1$. In summary,

$$OPT \geq \sum_{j=1}^{D-2} n_j + 2n_D + (n_{D-1} - n_D)^+$$

where $(n_{D-1} - n_D)^+ := \max\{0, n_{D-1} - n_D)$. The upper bound on the competitive ratio follows since

$$\begin{aligned}\frac{cost}{OPT} &\leq \frac{\sum_{j=1}^{D} j n_j}{\sum_{j=1}^{D-2} n_j + 2n_D + (n_{D-1} - n_D)^+} \\ &\leq \frac{(D-2)\sum_{j=1}^{D-2} n_j + (2D-1)n_D + (D-1)(n_{D-1} - n_D)^+}{\sum_{j=1}^{D-2} n_j + 2n_D + (n_{D-1} - n_D)^+} \\ &\leq D - \frac{1}{2} .\end{aligned}$$

Lower bound:
Let $k, n \in \mathbb{N}$ be some large integers. We construct the following graph: Layers $0, \dots, D-2$ consist of one node each and layers $D-1$ and D consist of infinitely many nodes each, denoted $a_0, a_1, a_2, \dots$ and $b_0, b_1, b_2, \dots$ respectively. For each $i \in \mathbb{N}_0$, the k nodes $b_{ik}, b_{ik+1} \dots, b_{(i+1)k-1}$ are adjacent to each of the $2k$ nodes $a_{ik}, a_{ik+1}, a_{(i+2)k-1}$ and to no other nodes. The set of remaining edges is uniquely determined by the fact that this is a layered graph of depth D.

The request sequence consists of n rounds $0, 1, \dots, n-1$, where each request in round i is at a node from the list $a_{ik}, a_{ik+1}, \dots, a_{(i+1)k-1}, b_{ik}, b_{ik+1}, \dots, b_{(i+1)k-1}$. Round i starts with requests on the nodes $a_{ik}, a_{ik+1}, \dots, a_{(i+1)k-1}$. Then, for $j = 0, \dots, k-1$, the adversary first requests b_{ik+j} and then requests whichever node from $a_{ik}, a_{ik+1}, \dots, a_{(i+1)k-1}$ has been left by an MOO-server to serve the request at b_{ik+j}. Note that by definition of MOO and the graph, the server it moves to b_{ik+j} does indeed come from $a_{ik}, a_{ik+1}, \dots, a_{(i+1)k-1}$.

In round i, MOO first pays $k(D-1)$ to move k servers to $a_{ik}, a_{ik+1}, \dots, a_{(i+1)k-1}$ and then, for each $j = 0, \dots, k-1$, it pays 1 to move to b_{ik+j} and $D-1$ to spawn a new server at the group

$a_{ik}, a_{ik+1}, \dots, a_{(i+1)k-1}$. Over n rounds this makes a total cost of $n(k(D-1)+k(1+D-1)) = nk(2D-1)$.

The offline algorithm can serve requests as follows: The requests at $a_{ik}, \dots, a_{(i+1)k-1}$ at the beginning of round i are served by spawning if $i = 0$ (for cost $(d-1)k$) and by sending servers from $b_{(i-1)k}, \dots, b_{ik-1}$ if $i \geq 1$ (for cost k). The request at b_{ik} is served by spawning a server (cost D) and the requests at $b_{ik+1}, \dots, b_{ik+k-1}$ are served by sending a server from a node in $a_{ik}, \dots, a_{(i+1)k-1}$ that will not be requested any more (cost 1 each, so $k-1$ per round). Over n rounds, this adds up to an offline cost of $(D-1)k + (n-1)k + n(D+k-1) = 2nk + (D-2)k + n(D-1)$. The ratio of online and offline cost is

$$\frac{nk(2D-1)}{2nk + (D-2)k + n(D-1)} = \frac{2D-1}{2 + \frac{D-2}{n} + \frac{D-1}{k}},$$

which gets arbitrarily close to $D - \frac{1}{2}$ for n and k large enough. □

Theorem 11. *The competitive ratio of the infinite server problem on layered graphs of depth D is exactly 1.5 for $D = 2$, exactly 2.5 for $D = 3$ and at least 3 for $D \geq 4$.*

Proof. For $D = 2$, the only possibility to move a server closer to the source is from layer 2 to layer 1. But since spawning to layer 1 is at least as good, we can restrict our attention to algorithms of the type MOO. The result follows from Theorem 10.

For $D = 3$, the upper bound follows from Theorem 10. It remains to show the lower bounds for $D \geq 3$.

Fix some large integers $k, n \in \mathbb{N}$. Consider the following layered graph of depth D. For $i = 0, \dots, D-1$ there exists a node v_i in layer i. The remaining nodes are defined inductively as all nodes obtained by the following two rules:

- There exist a set S_0 of $2k$ nodes and sets A^{S_0} and B^{S_0} of k nodes.
- Let S be a set of $2k$ nodes such that A^S and B^S exist. Then for each $S' \subset S \cup B^S$ of size $2k$ there are sets $A^{S'}$ and $B^{S'}$ of k nodes.

The nodes in the sets A^S are in layer $D-1$, the nodes in S_0 and in the sets B^S are in layer D. For a node in some set A^S, the set of adjacent nodes in layer D is $S \cup B^S$. The remaining edges are so that this is a layered graph with the layers as specified.

For purposes of the analysis below, we further define a *generation* of a node as follows: The nodes $v_0, \dots, v_{D-1}$ and the nodes in S_0 have generation 1. The generation of nodes in A^S and B^S is the maximal generation of any node in S plus 1.

Let ALG be some online algorithm. We assume without loss of generality that ALG is lazy and local.

The adversary chooses the following request sequence against ALG. First, request the nodes in S_0 until ALG has a server at each of them. The adversary also moves $2k$ servers towards these nodes. The adversary uses only these $2k$ servers for the entire sequence of requests. The remainder of the requests consists of several rounds. We will keep the invariant that at the beginning of the ith round, the $2k$ adversary servers occupy a set S for which A^S and B^S (with nodes of generation $i+1$) exist, and the online servers occupy nodes of generation at most i. Clearly this holds before the first round. Let $A^S = \{a_1, \dots, a_k\}$ and $B^S = \{b_1, \dots, b_k\}$.

The requests of the ith round are divided into part a and part b, consisting of *steps* a.1,...,a.k, b.1,...,b.k that are executed in this order. Step a.j consists of the following one or two requests:

First request a_j. If ALG moves a server from some $b \in S$ towards a_j, immediately request b. We can assume that online servers cover A^S after the end of part a (otherwise request nodes in A^S again at the end of part a until this is the case). Step b.j consists of the following two or three requests: First request b_j. Note that any path from a node of generation at most i to b_j contains a node in S, and from any node in S, the shortest paths to b_j include the ones along the nodes in A_S. Thus, since ALG is local, it will move a server from some $a \in A^S$ towards b_j. The second request of step b.j is at this node a and, if ALG moves a server from some $b \in S \cup B^S$ towards a, then the step contains a third request at b.

The adversary cost per round is at most $2k+2$: For each $j = 1, \dots, k$, there are at least j nodes in S that will *not* be requested during steps a.j, …, a.k, b.1, …, b.$(k-1)$. Hence, the adversary can serve all requests of part a for cost k by moving k servers from S towards A^S whilst keeping servers at all nodes of S that will be requested during the steps b.1,…,b.$(k-1)$. Similarly, it can serve the steps b.1,…,b.$(k-1)$ for cost $k-1$ by moving $k-1$ servers from A^S to B^S. The final step b.k of the round can be served at cost 3 using the last server in A^S to serve the requests and finish with all $2k$ offline servers in some set $S' \subseteq S \cup B^S$.

We analyze the online cost for the cases $D = 3$ and $D \geq 4$ separately.

If $D = 3$, then the cost for each step a.j is at least 2 and the cost for each step b.j is at least 3. Thus, the cost per round is at least $5k$. As k goes to infinity, the ratio of online and offline cost in each round converges to 2.5. As the number of rounds goes to infinity, the online and offline costs before the first round become negligible, which proves the lower bound of 2.5 for $D = 3$.

For $D \geq 4$, we use a potential Φ equal to the number of online servers in layer $D-1$. During step a.j, either Φ does not change and the cost is at least 2, or Φ increases by 1 and the cost is at least 3. Thus, during step a.j we have $\Delta cost \geq 2 + \Delta\Phi$ and hence during part a we have $\Delta cost \geq 2k + \Delta\Phi$. During step b.$j$, either Φ decreases by 1 and the cost is at least 3, or Φ does not change and the cost is at least 4. Thus, during part b we have $\Delta cost \geq 4k + \Delta\Phi$. In total, this adds up to $\Delta cost \geq 6k + \Delta\Phi$ during the round. Over n rounds, this makes $\Delta cost \geq 6nk + \Delta\Phi \geq 6nk$ since Φ starts at 0 before the first round and remains nonnegative. As k and n go to infinity, the ratio of our bounds on online and offline cost converges to 3. $\square$

It remains an open problem to close the gap between the lower bound of 3 and the upper bound of 3.5 for $D = 4$. More importantly, we are interested in the question whether an algorithm better than MOO exists for large D, achieving a competitive ratio of less than $D - 1/2$ on any layered graph of depth D. Note that if no algorithm with a competitive ratio of $O(1)$ as $D \to \infty$ exists, then the infinite server problem on general metric spaces would not be competitive.

For large D, the lower bound of 3 is certainly not tight: Consider a layered graph where each layer contains one node except that the bottom layer contains infinitely many nodes. By Theorem 8 (and a matching upper bound shown in [10]), the competitive ratio on this graph converges to $\lambda \approx 3.146$ as $D \to \infty$.

4 Algorithms with Unbounded Competitive Ratio

We examine the performance of classical algorithms known for the k-server problem when applied to the infinite server problem. The main focus of this section is a generalization of the Double Coverage algorithm for the line with adjusted server speeds. This idea has proved successful for the (h,k)-server problem (and hence the infinite server problem) on weighted trees [1]. However, neither of these algorithms is competitive for the infinite server problem even on the line.

4.1 Work Function Algorithm

The Work Function Algorithm (WFA, [8]) for the k-server problem achieves a competitive ratio of at most $2k-1$, which is the best known upper bound for general metric spaces [18]. Given a sequence of requests $r_1, r_2, \dots$ and a configuration C (i. e. a multiset of server positions), the work function $w_t(C)$ is defined as the minimal cost of serving the first t requests and ending up in configuration C. If C_{t-1} is the server configuration before the tth request, the algorithm moves to a configuration C_t that contains r_t and minimizes the quantity

$$w_t(C_t) + d(C_{t-1}, C_t)\,, \tag{5}$$

where $d(C_{t-1}, C_t)$ is the cost of moving from C_{t-1} to C_t.

Proposition 12. *The WFA is not competitive for the infinite server problem on the line.*

Proof. Let the source be at 0 and, for some small $\delta > 0$, let $p_1, p_2, \dots \in [1, 1+\delta]$ be infinitely many distinct points. Consider the request sequence that always requests the point p_i with i minimal such that p_i is not occupied by an online server. Let σ_k be the prefix of this request sequence until the WFA spawns its kth server. It is easy to see that the WFA does not spawn faster than the optimal algorithm. More precisely, the WFA spawns its kth server only if the optimal way of serving the already seen requests is to bring k servers to the points $p_1, \dots, p_k$. In particular, $OPT(\sigma_k) = \sum_{i=1}^{k} p_i = k + o(1)$ as $\delta \to 0$. Thus, the optimal offline cost increases by $1 + o(1)$ during the period when the WFA has k spawned servers, and it increases by at least as much for an offline algorithm that is restricted to using k servers only. Let $cost_k$ be the cost incurred to the WFA during this period. Due to the lower bound of k on the competitive ratio of the k-server problem, $cost_k$ is at least k times this increase of the optimal cost (up to an additive error of $o(1)$ as $\delta \to 0$), i. e. $cost_k \geq k + o(1)$. Thus, the total cost of the WFA given the request sequence σ_n is at least

$$\sum_{k=1}^{n-1} cost_k = \Omega(n^2).$$

Meanwhile, the optimal cost is $OPT(\sigma_n) = n + o(1)$. Letting n tend to infinity we obtain an unbounded competitive ratio. □

4.2 Balance and Balance2

The algorithm *Balance* serves a request r by sending a server x that minimizes the quantity $D_x + d(x,r)$, where D_x is the cumulative distance traveled by x so far and $d(x,r)$ is the distance between x and r. For the k-server problem, Balance is k-competitive on metric spaces with $k+1$ points [19] and for weighted paging [6]. Young showed that for weighted paging against a weak adversary with h servers the competitive ratio of Balance is $k/(k-h+1)$ [21]. On general metric spaces however, Balance has unbounded competitive ratio, even if $k = 2$ [19]. It is therefore unsurprising that it is also not competitive for the infinite server problem.

Proposition 13. *Balance is not competitive for the infinite server problem on the line.*

Proof. Suppose all servers start at source 0 and consider the request sequence $r_0, r_1, r_2, \dots, r_n$ where $r_i = 1 - i\epsilon$. As $\epsilon \to 0$, the optimal cost tends to 1 whereas the cost of Balance tends to $n+1$. Since n can be arbitrarily high, this shows an unbounded competitive ratio. □

The intuitive problem of Balance is that it is not greedy enough. The algorithm *Balance2* by Irani and Rubinfeld [13] compensates for this weakness by giving more weight to the distance between the server and the request: To serve request r, Balance2 sends a server x that minimizes the quantity $D_x + 2d(x, r)$. Irani and Rubinfeld showed that, unlike Balance, Balance2 is competitive for two servers (achieving a competitive ratio of at most 10) and they conjectured that it is also competitive for any other finite number of servers [13].

However, for the infinite server problem this algorithm is also not competitive:

Proposition 14. *Balance2 is not competitive for the infinite server problem on the line.*

Proof. Suppose the source is at 0 and fix some small constant $\epsilon > 0$. The request sequence consists of several phases, starting with phase 0. Phase i consists of alternating requests at $1 - 2i\epsilon$ and $1 - (2i + 1)\epsilon$. We will ensure that all requests of a phase are served by the same online server, and we call this the active server. As soon as the cumulative distance traveled by the active server exceeds $2 - (4i + 5)\epsilon$, the phase ends and a new phase begins. Note that this means that the active server of a phase will not be used to serve any request of a subsequent phase because, by definition of Balance2, the algorithm would rather spawn a new server. Thus, the first request of each phase is served by spawning a new server, which becomes the active server of that phase. While the cumulative distance of the active server is at most $2 - (4i + 5)\epsilon$ and since its distance from the next request of the phase is always exactly ϵ, its associated quantity $D_s + 2d(s, r)$ is at most $2 - (4i + 3)\epsilon$. Hence, Balance2 rather uses this server during the phase instead of spawning a new server. Thus, it is indeed the active server that serves *all* requests of its phase.

Let n be the number of phases and choose ϵ small enough so that all requests are in the interval $[1/2, 1]$. Thus, the cost of Balance2 is $\Omega(n)$.

An offline algorithm could serve all requests with two servers only that move to 1 and $1 - \epsilon$ initially and then back towards $1/2$, always covering the two points that are requested during a phase, resulting in an offline cost of less than 3. As n goes to infinity, the ratio between online and offline cost becomes arbitrarily large. $\square$

4.3 Double Coverage Variants

Perhaps more surprising than for WFA and balancing algorithms is that a class of algorithms extending the Double Coverage (DC) algorithm [6] is also not competitive for the infinite server problem. The basic DC algorithm on the line serves each request by an adjacent server. If the request lies between two servers, both servers move towards it at equal speed until one of them reaches the request. A sensible extension of this algorithm seems to be to give different speeds to servers, so that they move away from the source faster than towards it.

We consider here only the half-line $[0, \infty)$ with the source at the left border 0. Let x_i be the position of the ith server from the right. We use the notation x_i both for its position and for the server itself. As servers do not overtake each other, x_i is the ith spawned server. Let $\mathcal{S} = \{s_i \geq 1 \mid i \in \mathbb{N} \text{ and } i \geq 2\}$ for a monotonic (non-decreasing or non-increasing) sequence of speeds s_i. The algorithm $\mathcal{S}$-DC is defined as follows:

- If there exist servers x_{i+1} and x_i to the left and right of the request, move them towards it with speeds s_{i+1} and 1 respectively until one of the two reaches it.
- If a request does not have a server to its right, move the rightmost server to the request.

If $s_i = 1$ for all i, this is precisely the original DC algorithm.

We will prove that $\mathcal{S}$-DC is not competitive. The intuitive reason is that servers move to the right either too slowly or too quickly: Imagine repeatedly requesting the same n points in some small interval away from the source, until $\mathcal{S}$-DC covers all n points. One case is that $\mathcal{S}$-DC spawns too slowly and is therefore defeated by an adversary covering these n positions immediately with n servers. In the other case, the adversary will also use n servers to cover the initial group of requests and then shift its group of servers slowly towards the source, always making requests at the new positions of these offline servers. As $\mathcal{S}$-DC tries to cover the new requests, it is tricked into spawning too many servers. Both cases lead to an unbounded competitive ratio.

The proof consists of several lemmas. The lemmas hold also for non-monotonic speeds and we use monotonicity only to easily combine the lemmas in the end.

A useful property of $\mathcal{S}$-DC is that its cost can be calculated using only the final positions of the servers.

Lemma 15. *Let $x_1 \geq x_2 \geq \dots$ be the server positions of $\mathcal{S}$-DC after serving a sequence of requests. Then the cost paid is $\sum_{i=1}^{\infty} z_i x_i$ where*

$$z_1 = 1 \tag{6}$$

$$z_i = \frac{z_{i-1}}{s_i} + 1 + \frac{1}{s_i}\,. \tag{7}$$

Proof. The movement x_i of each server can be written as $x_i = r_i - l_i$ where r_i and l_i are the cumulative distances traveled by that server while moving to the right and left respectively. By definition of $\mathcal{S}$-DC, for all i we have

$$l_i = \frac{r_{i+1}}{s_{i+1}},$$

since any right move (apart from the rightmost server) is accompanied by a left move of another server. Observe that the online cost is

$$\begin{aligned} cost = \sum_{i=1}^{\infty}(r_i + l_i) &= \sum_{i=1}^{\infty}(r_i + \frac{r_{i+1}}{s_{i+1}}) \\ &= \sum_{i=1}^{\infty} r_i + \sum_{i=2}^{\infty} \frac{r_i}{s_i} = r_1 + \sum_{i=2}^{\infty} r_i(1 + \frac{1}{s_i})\,. \end{aligned} \tag{8}$$

Similarly,

$$\begin{aligned} \sum_{i=1}^{\infty} z_i x_i &= \sum_{i=1}^{\infty} z_i(r_i - l_i) \\ &= \sum_{i=1}^{\infty} z_i r_i - \sum_{i=1}^{\infty} z_i \frac{r_{i+1}}{s_{i+1}} \\ &= z_1 r_1 + \sum_{i=2}^{\infty} r_i \left(z_i - \frac{z_{i-1}}{s_i} \right)\,. \end{aligned} \tag{9}$$

By equating (8) and (9) term by term, we get the desired recurrence for z_i. $\square$

The next lemma takes care of the case when online servers spawn too slowly.

Lemma 16. *If the speeds in $\mathcal{S}$ satisfy* $\liminf_{n\to\infty} \sqrt[n]{\prod_{i=2}^{n} s_i} = 1$ *then $\mathcal{S}$-DC is not competitive.*

Proof. For this lower bound we have requests on n arbitrary positions in the interval $[1,2]$, until $\mathcal{S}$-DC covers them all.

The optimal cost is at most $2n$. This can be achieved by spawning a fresh server for each requested position.

Since for every spawned online server we have $x_i \geq 1$, by Lemma 15 the online cost is $\mathit{cost} = \sum_{i=1}^{n} z_i x_i \geq \sum_{i=1}^{n} z_i$. Unraveling the recurrence we get that $z_i = 1 + \frac{2}{s_i} + \frac{2}{s_i s_{i-1}} + \ldots + \frac{2}{s_i \cdot \ldots \cdot s_2}$. Thus,

$$\begin{aligned} \mathit{cost} &\geq n + \sum_{i=1}^{n-1} \sum_{j=1}^{n-i} \frac{2}{\prod_{k=j+1}^{j+i} s_k} \\ &\geq \sum_{i=1}^{f(n)} \sum_{j=1}^{n-i} \frac{2}{\prod_{k=j+1}^{j+i} s_k} \,. \end{aligned} \tag{10}$$

where

$$f(n) = \left\lfloor \frac{n}{2 + 2\log_2 \prod_{i=2}^{n} s_i} \right\rfloor \leq \frac{n}{2} \,.$$

We argue that for each $i = 1, \ldots, f(n)$, it holds for at least half of the values of $j = 1, \ldots, n-i$ that $\prod_{k=j+1}^{j+i} s_k \leq 2$. Indeed, suppose this were not the case for some i. Let us partition the set $J = \{1, \ldots, n-i\}$ of j-values into subsets $J_0, \ldots, J_{i-1}$, where J_m contains precisely those numbers from J that are congruent to m modulo i. By assumption, we have $\prod_{k=j+1}^{j+i} s_k > 2$ for at least half the values $j \in J$, so this must also be true for at least half the values $j \in J_m$ for some m. However, this would mean that

$$\prod_{k=2}^{n} s_k \geq \prod_{j \in J_m} \prod_{k=j+1}^{j+i} s_k > 2^{|J_m|/2} \geq 2^{\lfloor \frac{n-i}{i} \rfloor/2} \geq 2^{\frac{n}{2f(n)} - 1} \geq \prod_{i=2}^{n} s_i \,,$$

a contradiction because the second inequality is strict.

Thus, continuing from (10) we can further bound the online cost as

$$\mathit{cost} \geq f(n) \frac{n - f(n)}{2} \geq \frac{n f(n)}{4} \,.$$

Since the optimal cost is at most $2n$, the competitive ratio is at least $f(n)/8$. However, $f(n)$ is unbounded because

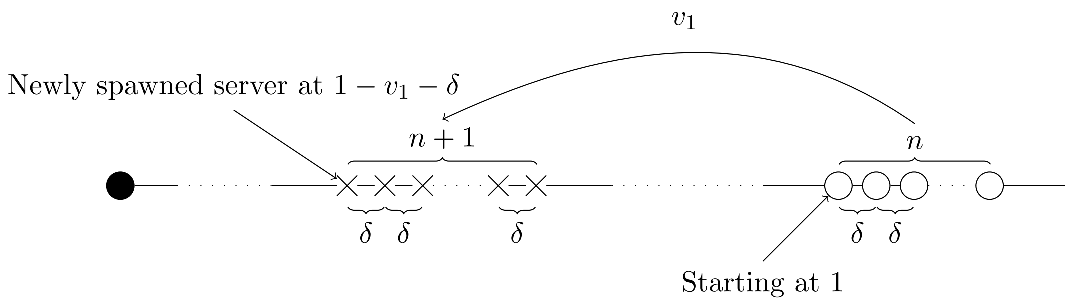
$$\frac{n}{2 + 2\log_2 \prod_{i=2}^{n} s_i} = \frac{1}{\frac{2}{n} + 2\log_2 \sqrt[n]{\prod_{i=2}^{n} s_i}}$$

and the denominator in the last term gets arbitrarily close to 0. □

The case of servers being spawned too aggressively is handled by the following lemma.

Lemma 17. *If there exists an unbounded function $f(n)$ such that for each $k \in \mathbb{N}$ we have $\prod_{i=k}^{k+n} s_i \geq f(n)$, then $\mathcal{S}$-DC is not competitive. In particular, if* $\liminf_{i\to\infty} s_i > 1$ *then $\mathcal{S}$-DC is not competitive.*

Proof. Consider the following configuration of online positions and requests, denoted by circles and crosses respectively.



v_1
Newly spawned server at $1 - v_1 - \delta$
$n+1$
n
δ δ δ
δ δ
Starting at 1

We start by spawning n online servers grouped tightly, with the leftmost being at distance 1 from the source and a very small gap δ between them. This is easily accomplished by repeating several requests on those points. Afterwards, we shift this group of n servers (by means of requests on new $n+1$ points) to the left by v_1, chosen so that the $n+1$ points are covered exactly by the n old servers plus a newly spawned one, which occupies the leftmost requested position $1 - v_1 - \delta$.

This is repeated again and again, shifting each time the leftmost n spawned servers a new v_k to the left via multiple requests on $n+1$ positions. The goal each time is to pull a new server from the source *and* leave one behind forever, thus achieving an arbitrarily high competitive ratio for $\mathcal{S}$-DC variants that spawn servers too fast.

The offline cost can be calculated easily. The offline algorithm uses n servers to cover the first group of n requested points in the interval $[1, 1+n\delta]$. Then it adds one more server and moves the group of $n+1$ servers to the left to satisfy all of the following requests. At most, the group of offline servers will return close to the source, yielding an optimal cost of

$$OPT \leq 2(n+1)(1+n\delta) = O(n) \tag{11}$$

since δ is very small.

To bound the online cost, we need to compute the values v_k first. Let ℓ_i^k and r_i^k denote the cumulative distance to the left and right respectively traveled by x_i during the left shift by v_k of the group $x_k, x_{k+1}, \ldots, x_{k+n-1}$. The nonzero values among these are

$$\begin{aligned}
\ell_k^k &= v_k \\
r_{k+1}^k &= v_k s_{k+1} \\
\ell_{k+1}^k &= v_k(1 + s_{k+1}) \\
r_{k+2}^k &= v_k(s_{k+2} + s_{k+1}s_{k+2}) \\
\ell_{k+2}^k &= v_k(1 + s_{k+2} + s_{k+1}s_{k+2}) \\
&\vdots \\
r_{k+n}^k &= v_k(s_{k+n} + s_{k+n-1}s_{k+n} + \ldots + \prod_{j=k+1}^{k+n} s_j) = v_k \sum_{i=k+1}^{k+n} \prod_{j=i}^{k+n} s_j\,.
\end{aligned} \tag{12}$$

On the other hand, the new position of the server x_{k+n} pulled from the source during these moves is $1 - \sum_{i=1}^{k} v_i - k\delta$. Equating this with (12) and solving for v_k yields (and assuming that

n is even)

$$v_k = \frac{1 - \sum_{i=1}^{k-1} v_i - k\delta}{1 + \sum_{i=k+1}^{k+n} \prod_{j=i}^{k+n} s_j} \le \frac{1}{\frac{n}{2} \prod_{j=k+\frac{n}{2}}^{k+n} s_j} \le \frac{2}{nf(\frac{n}{2})}\,.$$

We will calculate the number of repetitions before the left border of the group of servers (just) passes $\frac{1}{2}$. If l is the number of repetitions, we have

$$\frac{1}{2} \le \sum_{k=1}^{l} (v_k + \delta) \le \frac{2l}{nf(\frac{n}{2})} + l\delta$$

and for sufficiently small δ this means that

$$l \ge \frac{n}{5} f\left(\frac{n}{2}\right)$$

If we do $l-1$ repetitions, then each of them will pull a new server at least $1/2$ away from the source, resulting in an online cost of $\Omega(n) \cdot f(\frac{n}{2})$. As the offline cost is $O(n)$ and $f(n)$ is unbounded, the algorithm is not competitive. □

Since the sequence of speeds s_i is monotonic and bounded from below by 1, we have either $\lim_{i\to\infty} s_i = 1$, in which case Lemma 16 applies, or otherwise $\liminf_{i\to\infty} s_i > 1$ and Lemma 17 applies. In any case, the competitive ratio is unbounded:

Theorem 18. *Algorithm $\mathcal{S}$-DC is not competitive for any $\mathcal{S}$.*

5 Reduction to Bounded Spaces

In this section we show a reduction from the infinite server problem on general metric spaces to bounded subspaces. Specifically, a metric space can be partitioned into "rings" of points whose distance from the source is between r^n and r^{n+1}, where $r > 1$ is fixed and $n \in \mathbb{Z}$. We show that if the infinite server problem is strictly ρ-competitive on each ring, then it is competitive on the entire metric space.

Theorem 19. *Let M be a metric space and $s \in M$ and let $r > 1$. For $n \in \mathbb{Z}$ let $M_n = \{s\} \cup \{p \in M \mid d(s,p) \in [r^n, r^{n+1})\}$. If for each n the infinite server problem on (M_n, s) is strictly ρ-competitive, then on (M, s) it is strictly $\frac{4r-1}{r-1}\rho$-competitive.*

Proof. Let ALG_n be a ρ-competitive algorithm for the infinite server problem on (M_n, s).

For a request sequence σ, let σ_n be the subsequence of requests in M_n. Let ALG be the algorithm for (M, s) that uses different servers for each of the subsequences σ_n and serves them independently according to ALG_n.

The total online cost is $ALG(\sigma) = \sum_n ALG_n(\sigma_n) \le \rho \sum_n OPT(\sigma_n)$. To finish the proof, it suffices to show that

$$\sum_n OPT(\sigma_n) \le \frac{4r-1}{r-1} OPT(\sigma)\,. \tag{13}$$

Thus, we only need to analyze the offline cost. We do this for each offline server separately. Fix some offline server x. Let N_0 and N_1 be the minimal and maximal values of n such that x visits

M_n. We can assume without loss of generality (by adding virtual points to the metric space) that whenever x moves from M_n to $M_{n'}$ for some $n < n'$, it travels across points $p_{n+1}, p_{n+2}, \ldots, p_{n'}$ with $d(s, p_i) = r^i$, and similarly for $n > n'$.

The movements of server x can be tracked by many servers, one server x_n in every set M_n for $N_0 \leq n \leq N_1$. When server x is in M_n, server x_n is exactly at the same position tracking the movement of x. When server x exits M_n at some point p at the boundary to M_{n-1} or M_{n+1}, server x_n freezes at p. The movement cost of x_n can be partitioned into the cost of deploying x_n at the first point visited in M_n, the tracking cost within M_n, and the cost of of relocating x_n whenever x re-enters M_n at a location different from the last exiting location.

The total tracking cost of all servers x_n is bounded by the distance traveled by x. The cost of deploying all servers x_n is $\sum_{n=N_0}^{N_1} r^n \leq \sum_{n=-\infty}^{N_1} r^n = r^{N_1+1}/(r-1)$, which is at most $\frac{r}{r-1}$ times the total movement of server x, because the latter is at least r^{N_1}.

To bound the relocating cost, say x exits M_n at p and re-enters it at p'. Then p and p' are at the boundary of M_n and M_{n+u} for $u \in \{-1, +1\}$. Let b be the distance traveled by x in M_{n+u} between the times when it is entered at p and when it is next exited. If this exiting is at p', then the relocating cost $d(p, p')$ is at most b by the triangle inequality. Otherwise, x exits M_{n+u} at a point p'' at the boundary of M_{n+u} and M_{n+2u}. If $u = 1$, then $d(p, p') \leq d(s, p) + d(s, p') = 2r^{n+1}$ and $b \geq d(p, p'') \geq d(s, p'') - d(s, p) = r^{n+2} - r^{n+1} = (r-1)r^{n+1}$. If $u = -1$, then $d(p, p') \leq d(s, p) + d(s, p') = 2r^n$ and $b \geq d(p, p'') \geq d(s, p) - d(s, p'') = r^n - r^{n-1} = \frac{r-1}{r} r^n$. In both cases, the relocating cost $d(p, p')$ is at most $\frac{2r}{r-1} b$. Thus, the total relocating cost of all servers x_n is at most $\frac{2r}{r-1}$ times the total distance traveled by x.

Thus, the sum of deployment, tracking and relocating cost of the servers x_n is at most $\frac{4r-1}{r-1}$ times the distance traveled by x. This shows (13), giving the statement of the theorem. □

The last theorem can also be slightly generalized to the case where instead of *strict* ρ-competitiveness, an additive term proportional to r^n is allowed. It is not difficult to show the following specialization for the line, where the premise can be weakened to require competitiveness only on a single interval:

Corollary 20. *Let* $0 < a < b$. *The infinite server problem is competitive on the line if and only if it is competitive on* $(\{0\} \cup [a, b], 0)$.

Another consequence of Theorem 19 is a reduction to spaces where the source is at a uniform distance from all other points. This models the case of a fixed cost for "buying" new servers.

Corollary 21. *Suppose there exists* ρ *so that the infinite server problem is strictly* ρ*-competitive on any metric space where the distance from the source to any other point is the same. Then the infinite server problem on general metric spaces is competitive.*

Proof. Follows from Theorem 19 by increasing the distance from s to the other points in M_n to r^{n+1}, making a multiplicative error of at most r. □

6 Open Problems

The most obvious open problem is whether the infinite server problem is competitive on general metric spaces. A challenging special case is to resolve the question for the real line. Similarly, improving the MOO algorithm and settling the question for layered graphs remains open. It would also be interesting to find a metric space with a competitive ratio greater than 3.146 for the infinite server problem or the (h, k)-server problem when $k \gg h$. Another possible line of research is to consider randomized algorithms.

References

[1] Nikhil Bansal, Marek Eliáš, Łukasz Jeż, and Grigorios Koumoutsos. The (h, k)-server problem on bounded depth trees. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, 2017*, pages 1022–1037. SIAM, 2017.

[2] Nikhil Bansal, Marek Eliáš, Łukasz Jeż, Grigorios Koumoutsos, and Kirk Pruhs. Tight bounds for double coverage against weak adversaries. In *International Workshop on Approximation and Online Algorithms*, pages 47–58. Springer, 2015.

[3] Yair Bartal and Eddie Grove. The harmonic k-server algorithm is competitive. *Journal of the ACM (JACM)*, 47(1):1–15, 2000.

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

[5] Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis.* Cambridge University Press, 2005.

[6] Marek Chrobak, Howard Karloff, Tom Payne, and Sundar Vishwanathan. New results on server problems. *SIAM Journal on Discrete Mathematics*, 4(2):172–181, 1991.

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

[8] Marek Chrobak and Lawrence L Larmore. The server problem and on-line games. In *On-line Algorithms, volume 7 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science.* Citeseer, 1992.

[9] Ilan R. Cohen, Alon Eden, Amos Fiat, and Łukasz Jeż. Pricing online decisions: Beyond auctions. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on discrete algorithms*, pages 73–91. SIAM, 2014.

[10] János Csirik, Csanád Imreh, John Noga, Steve S. Seiden, and Gerhard J. Woeginger. Buying a constant competitive ratio for paging. In *European Symposium on Algorithms*, pages 98–108. Springer, 2001.

[11] Yuval Emek, Pierre Fraigniaud, Amos Korman, and Adi Rosén. On the additive constant of the k-server work function algorithm. In *International Workshop on Approximation and Online Algorithms*, pages 128–134. Springer, 2009.

[12] Amos Fiat, Yuval Rabani, and Yiftach Ravid. Competitive k-server algorithms. In *Foundations of Computer Science, 1990. Proceedings., 31st Annual Symposium on*, pages 454–463. IEEE, 1990.

[13] Sandy Irani and Ronitt Rubinfeld. A competitive 2-server algorithm. *Information Processing Letters*, 39(2):85–91, 1991.

[14] Kamal Jain. Personal Communication.

[15] Elias Koutsoupias. Weak adversaries for the k-server problem. In *Foundations of Computer Science, 1999. 40th Annual Symposium on*, pages 444–449. IEEE, 1999.

[16] Elias Koutsoupias. The k-server problem. *Computer Science Review*, 3(2):105–118, 2009.

[17] Elias Koutsoupias and Christos Papadimitriou. The 2-evader problem. *Information Processing Letters*, 57(5):249–252, 1996.

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

[19] Mark Manasse, Lyle McGeoch, and Daniel Sleator. Competitive algorithms for on-line problems. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 322–333. ACM, 1988.

[20] Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

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