The k-Server Problem with Delays on the Uniform Metric Space

Predrag Krnetić
Distributed Computing Group, ETH Zurich, Switzerland
pkrnetic@student.ethz.ch

Darya Melnyk
Distributed Computing Group, ETH Zurich, Switzerland
dmelnyk@ethz.ch

Yuyi Wang
Distributed Computing Group, ETH Zurich, Switzerland
yuwang@ethz.ch

Roger Wattenhofer
Distributed Computing Group, ETH Zurich, Switzerland
wattenhofer@ethz.ch

Abstract

In this paper, we present tight bounds for the k-server problem with delays in the uniform metric space. The problem is defined on $n + k$ nodes in the uniform metric space which can issue requests over time. These requests can be served directly or with some delay using k servers, by moving a server to the corresponding node with an open request. The task is to find an online algorithm that can serve the requests while minimizing the total moving and delay costs. We first provide a lower bound by showing that the competitive ratio of any deterministic online algorithm cannot be better than $(2k + 1)$ in the clairvoyant setting. We will then show that conservative algorithms (without delay) can be equipped with an accumulative delay function such that all such algorithms become $(2k + 1)$-competitive in the non-clairvoyant setting. Together, the two bounds establish a tight result for both, the clairvoyant and the non-clairvoyant settings.

2012 ACM Subject Classification Theory of computation → K-server algorithms; Theory of computation → Caching and paging algorithms; Theory of computation → Online algorithms

Keywords and phrases Online k-Server, Paging, Delayed Service, Conservative Algorithms

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2020.27

1 Introduction

The k-server problem is a classic problem in the realm of competitive online analysis. Given a metric space with $n + k$ nodes, an online algorithm must move one of k available servers to a node that is requesting a service. The goal of the algorithm is to minimize the total distance of all server movements. Since 1988, there is an unanswered conjecture that the k-server problem is k-competitive [16]. This conjecture is known as the *k-server conjecture*.

In this paper, we study a variation of the k-server problem proposed by Azar et al. [1]. In the classic k-server problem, requests arrive at discrete time points and a server must be *immediately* moved to serve an open request. In the variant of this paper, known as the k-server problem *with delays* (k-OSD), requests may arrive at any time point, and an algorithm may decide not to move a server immediately. Instead, it may let the request wait and incur time costs before being served. The goal of an online algorithm is then to not only minimize the total distance of all server movements, but also the total waiting time of the requests.

Note that the competitive ratio of the k-OSD problem cannot be smaller than the one

31st International Symposium on Algorithms and Computation (ISAAC 2020).
Editors: Yixin Cao, Siu-Wing Cheng, and Minming Li; Article No. 27; pp. 27:1–27:13
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

of the classical k-server problem, i.e., the deterministic k-OSD problem is also at least k-competitive. This can be shown by letting the time between any two appearing requests in the sequence be sufficiently long such that the offline algorithm covers every newly arrived request immediately. This way, even if the adversary tells the online algorithm designer the fact that the time gap between two consecutive requests is very large, the online algorithm still needs to solve a classical k-server problem.

While allowing the algorithm to incur delays, we also simplify the generalized setting in two ways: we restrict ourselves to deterministic algorithms only. We also consider a special case where the nodes are chosen from a uniform metric, i.e., where any two of the $n+k$ nodes have the same distance (but our lower bound works for general metrics). We thus have to pay a constant cost, plus possibly the waiting delay, if no server is present on a node with an open request. If a server is already present at the requesting node, serving a request is free.

The k-server problem on uniform metrics is often called the *paging problem*. In paging, serving a page request comes at the cost of evicting an arbitrary other page from cache, i.e., moving a server from one page to another in a uniform metric space. However, the k-server problem *with delays* is not exactly the paging problem, as it would allow to delay loading a page when needed, at a cost. Since not serving a page when needed probably goes along with scheduling a different process, it is questionable whether just summing up waiting times does model paging accurately. We will therefore refer to our setting as the uniform k-server problem with delays, while sometimes referring to known algorithms and lower bounds for the paging problem.

1.1 Related Work

More than 30 years ago, Tarjan and Sleator [19] were the first to study online algorithms using the concept of competitive analysis. They investigated deterministic algorithms for the paging and the list update problems. A couple of years later, Manasse et al. [16] defined the online k-server problem. The introduction of this new problem has widened horizons for further research in online algorithms. In their paper, the authors showed that no online algorithm can have a lower competitive ratio than k compared to an omnipotent adversary, independent of the metric space, for an arbitrary input sequence. This lower bound motivated the k-server conjecture, which stated that there is an online algorithm for the k-server problem with competitive ratio k. Soon the conjecture was proven for the special cases of $k = 2$ by Manasse et al. [17], line metrics by Chrobak et al. [6], tree metrics by Chrobak and Larmore [7], metrics with $k+1$ nodes by Manasse et al. [17] and for metrics with $k + 2$ nodes by Koutsoupias and Papadimitriou [13]. In this matter, a conceivable result was obtained by Koutsoupias and Papadimitriou [12], who applied the work function algorithm to the k-server problem and achieved a competitive ratio of $2k - 1$. Since then, there has been only slight progress on the k-server conjecture and the competitiveness of the work function algorithm remains the best known bound.

The k-server problem has also been studied by using randomized algorithms. Their beginnings go back to the 1980s: Raghavan and Snir [18] introduced the harmonic algorithm, where servers are moved based on a probability distribution which is inversely proportional to their distance. They achieved a competitive ratio of $\frac{k(k+1)}{2}$ on metric spaces with $k+1$ nodes. Randomness also enabled algorithms with a competitive ratio sublinear in k. Randomized algorithms for the unweighted and weighted paging problems, both have been shown to be $\mathcal{O}(\log k)$-competitive in [10] and [3] respectively. The first poly-logarithmic-competitive randomized online algorithm was introduced by Bansal et al. [2] on an arbitrary finite metric space for the k-server problem with n nodes, which has a competitive ratio of $\tilde{\mathcal{O}}(\log^3 n \log^2 k)$.

It improves upon the work function algorithm of Koutsoupias and Papadimitriou whenever n is subexponential in k. Similar to the deterministic k-server problem, a randomized k-server conjecture was presented, which states that there is a randomized algorithm for arbitrary metrics with a competitive ratio of $O(\log k)$ (see e.g., [2]). A more detailed illustration of the history of the k-server problem with deterministic and randomized algorithms can be found in [11]. In [15], Lee claimed that there is an $\mathcal{O}(\log^6 k)$-competitive randomized algorithm for the k-server problem on any metric space.

The idea of using delays was first considered by Emek et al. [9], where requests of the min-cost matching problem arrive online at points of a finite metric space and the algorithm may decide to match them later at a cost. Azar et al. transferred this idea to the well-known k-server problem and showed a poly-logarithmic competitive ratio of their preemptive service algorithm in [1]. It has a competitive ratio of $\mathcal{O}(kh^4)$ on a hierarchically separated tree (HST) of depth h, which implies a competitive ratio of $\mathcal{O}(k \log^5 n)$ on general metrics with n points. The same authors also considered the special case of the uniform and the star metrics. For the uniform metric, they proposed an $\mathcal{O}(k)$-competitive algorithm. Their algorithm takes any deterministic algorithm for the online paging problem and incorporates the delay by letting the algorithm only serve a node when its requests accumulate a delay penalty of 1. Note that our analysis suggests that this choice of the delay threshold is not optimal. Also for the star metric, the authors provide an $\mathcal{O}(k)$-competitive algorithm without specifying the exact constants. Another special case was recently considered by Bienkowski et al. [4], who provided a deterministic $\mathcal{O}(\log n)$-competitive algorithm for line metrics consisting of n equidistant points.

1.2 Model

In this section, we formally define the online service with delays (OSD) problem considered in this paper and explain the necessary concepts that are needed for the analysis. We consider the k-server problem with delays on a uniform metric space $M = (X, d)$, which we will refer to as the uniform k-OSD problem. The problem is defined on $k + n$ nodes in $X = \{x_1, ..., x_{n+k}\}$, where the distance between any pair of nodes x_i and x_j is set to be $d(x_i, x_j) = 1, \ \forall i \neq j, \ x_i, x_j \in X$. Further, an input sequence of requests $\sigma = (\sigma_1 \ldots \sigma_m) = (x_1, t_1), ..., (x_m, t_m)$ is given to an algorithm, where $x_1, \ldots, x_m \in X$ denote the source of the request and $t_1 \leq \ldots \leq t_m \in \mathbb{R}$ denote the timestamp at which the corresponding request is issued. An algorithm for the uniform k-OSD problem is given k servers, $s_1, \ldots, s_k$ to serve all requests from the input sequence. The algorithm serves a request on node x_j by moving a server s_i to this node. The goal of the algorithm is to minimize the sum of the moving and the delay costs, denoted C_{moving} and C_{delay} respectively. Every time the algorithm moves a server from one node to another, it pays a moving cost of one. If a request appears on a node where a server is already located, the request is served directly without any additional costs. Moreover, an algorithm does not have to serve all requests immediately. Instead, it is allowed to incur delay cost for each request. The delay cost is defined to be a non-negative monotonic function in the serving delay, i.e., serving a request which was issued at time t_i at time t_j will result at a delay cost of $f(t_j - t_i)$, where f is a non-decreasing function. Note that there can be several requests issued at one node which all incur delay costs. The delay cost of a node is then accumulated over all its unserved requests by summing up the corresponding delay costs. The total cost of the algorithm (C_{total}) is calculated as the sum of the moving and the delay cost over the whole input sequence σ. We will differentiate between two settings for the delay cost function f of the online algorithm: in the *clairvoyant* setting, the online algorithm is assumed to know the delay cost function f_i of request σ_i

when this request appears. In the *non-clairvoyant* setting, the online algorithm does not know the function f_i in advance, instead, the algorithm is given the accumulated delay costs of σ_i at any time point t. Note that the clairvoyant setting gives the online algorithm more power. In this paper, we will call requests which are unserved for a node *open requests* of the corresponding algorithm. We are interested in finding an online algorithm ALG which minimizes the total moving and delay cost. ALG will receive the input sequence of requests in an online fashion, where each σ_i will be presented at time t_i to the algorithm. We compare ALG to an omniscent offline algorithm OPT which is given the whole input sequence σ at the beginning of the algorithm. The quality of ALG is measured using the competitive ratio, i.e., an α, for which the inequality $C_{total}^{ALG} \leq \alpha \cdot C_{total}^{OPT} + c$ holds for all request sequences σ with some constant c.

1.3 Our Contribution

In this paper, we present tight bounds for the uniform k-OSD problem in the clairvoyant and the non-clairvoyant settings. We start by presenting a lower bound of $2k+1$ for the clairvoyant setting by comparing a worst-case request sequence designed for any online algorithm to a combination of $2k+1$ offline strategies - $k+1$ strategies covering the overall moving and k strategies covering the overall delay cost of any online algorithm. In [1], Azar et al. presented an $O(k)$-competitive algorithm for uniform metrics, but the tight constants remained unknown. In this paper, we define main properties that an online algorithm has to satisfy in order for it to match our lower bound. We show that conservative algorithms which are popular for solving the paging problem can all be equipped with an accumulative delay cost function such that they become $2k+1$-competitive when delays are allowed. Unlike in the paging problem, the analysis of conservative algorithms with delay requires a more involved partition of the request sequence which also depends on the actions of the optimal offline algorithm. The presented algorithms with delays are designed in the non-clairvoyant setting and thus show that clairvoyance does not give any advantage for the k-OSD problem with delays on uniform metrics.

2 Lower Bound for Deterministic Algorithms

In this section, we show a lower bound of $2k+1$ for the uniform k-OSD problem. We focus on uniform metrics on $k+1$ nodes. In such spaces, we can assume that at any time point, k nodes are each covered by a server and there is exactly one node that is not covered by a server. We define this uncovered node to be a *hole* of the corresponding algorithm. Note that in a uniform metric space with more than $k+1$ nodes, the sequence of requests can always be issued on a subset of $k+1$ nodes, and the presented lower bounds are still valid. We will prove the lower bound using an averaging technique similar to the paging problem [5].

▶ **Theorem 1.** *There exists no deterministic online algorithm for the uniform k-OSD problem with a competitive ratio lower than $2k+1-\varepsilon$ in the clairvoyant setting, where $\varepsilon > 0$ is an arbitrarily small constant.*

Proof. We will compare any online algorithm ALG to $2k+1$ offline algorithms whose actions depend on the actions of the online algorithm. We will therefore first define the input sequence σ dependent on the actions of a considered online algorithm as follows: as soon as the online algorithm ALG moves a server away from a node, a new request is issued on the resulting hole. This way, ALG will incur a unit moving costs for each request from the input sequence in addition to the incurred delay cost. We further define the delay cost function f

for the requests to be linear in the serving delay, i.e., serving a request σ_i which was issued at time t_1 at time t_2 will result at a delay cost of $c(\sigma_i) \cdot (t_2 - t_1)$, where $c(\sigma_i)$ is a predefined slope for this request. Let σ_i be the i-th request in the input sequence. Then, the slope of the delay cost function of σ_i is defined as $c(\sigma_i) = c^i$, where $c \gg 2$ is a large constant.[1]

The $2k+1$ offline algorithms are defined with respect to the actions of the online algorithms. We first associate each node with one offline algorithm that has a hole on this node from the beginning until the end of the input sequence σ. At the end of the input sequence, each such algorithm moves one server to the hole in order to serve all unserved requests. We will call these algorithms *static* and denote the cost of all $k+1$ static algorithms C_{total}^{static}. We also define k *dynamic* algorithms which make movements during the input sequence σ. The cost of all k dynamic algorithms we denote $C_{total}^{dynamic}$. Assume that the online algorithm ALG has a hole at x_i. Then, each of the dynamic offline algorithms is assumed to respectively have a hole at one of the nodes $\{x_1, \ldots, x_{k+1}\} \setminus x_i$. Once ALG moves a server from x_j to x_i, the offline algorithm with a hole at x_j moves a server from x_i to x_j.

We next evaluate the total cost of all $2k+1$ offline algorithms. We start by analyzing the sum of the delay costs of all static algorithms. Note that all static algorithms together incur the same delay costs as ALG, and, in addition, costs of all previous open requests. We will therefore show the following statement: For all $\bar{\varepsilon} > 0$ there exists an initial slope c, such that for the delay cost of all static algorithms holds $C_{delay}^{static} < (1+\bar{\varepsilon})C_{delay}^{ALG}$.

In order to show this statement we will choose $c := 2/\bar{\varepsilon}$. We can divide the input sequence into phases as follows: a phase p_i starts when a new requests σ_i is issued at the hole of ALG at time t_i. The phase ends with the appearance of the next request. We defined the input sequence σ such that ALG always incurs delay costs at exactly one node. We will show that the delay costs of all $k+1$ static algorithms on phase p, denoted $C_{delay}^{static}(p)$, can be bounded by $(1+\bar{\varepsilon})C_{delay}^{ALG}(p)$. Observe that there is always one static algorithm that incurs costs for the current open request σ_i of ALG. Besides these costs, the static algorithms also incur costs for the requests $\sigma_1, \ldots, \sigma_{i-1}$. The ratio of the costs in phase p_i can be computed as follows:

$$\frac{C_{delay}^{static}(p)}{C_{delay}^{ALG}(p)} = \frac{\sum_{j=1}^{i-1} c^j + c^i}{c^i} = 1 + \frac{\sum_{j=1}^{i-1} c^j}{c^i} < 1 + \frac{2}{c} = 1 + \bar{\varepsilon}$$

Note that the total moving cost of all static algorithms is $C_{moving}^{static} = k+1$, as each algorithm will move a server only once and since we consider a uniform metric space. The total moving cost of the k dynamic algorithms is equal to $C_{moving}^{dynamic} = C_{moving}^{ALG}$, as each movement by ALG makes only one of the dynamic algorithms move over the same edge. Further, the delay cost of all dynamic algorithms is 0, because we can assume that each offline algorithm first serves the newly arrived request at x_i before moving the server away from this node. For all $2k+1$ offline strategies together we receive the following costs:

$$\frac{C_{total}^{ALG}}{C_{total}^{static} + C_{total}^{dynamic}} = \frac{C_{moving}^{ALG} + C_{delay}^{ALG}}{C_{moving}^{static} + C_{delay}^{static} + C_{moving}^{dynamic}} \geq \frac{C_{moving}^{ALG} + C_{delay}^{ALG}}{k+1+(1+\bar{\varepsilon}) \cdot C_{delay}^{ALG} + C_{moving}^{ALG}}$$

For large m, the moving costs of ALG will be significantly larger than $k+1$, and thus the above ratio will converge to 1. On average, the $2k+1$ offline algorithms therefore have the same costs as ALG for $m \to \infty$. That is, there exists an offline algorithm for which ALG has a competitive ratio $\geq 2k+1-\varepsilon$, where $\varepsilon > 0$ is arbitrarily small.

[1] Note that our proof also works if we assume that all requests have the same linear delay cost function. Then, instead of increasing the slope $c(\sigma_i)$ of a request, we can set the number of requests appearing simultaneously on this node to $c(\sigma_i)$.

■

Observe that the above lower bound can be extended to an arbitrary bounded metric space by adjusting the analysis: only the moving costs of the static algorithms will change, namely C_{moving}^{static} will be the sum of all shortest incoming edges of each node (each static algorithm has to move one server to serve the requests on its only hole). Since we can choose m to be arbitrarily large, the competitive ratio will also be $2k + 1$ in the limit. Note that there is also a simpler lower bound proof for the k-OSD problem with delays, if the online algorithm is assumed to be non-clairvoyant.

3 Upper Bounds on Uniform Metrics

In this section, we present a non-clairvoyant algorithm for the k-OSD problem on uniform metrics with a competitive ratio of $2k+1$, which we will refer to as ALG. This result will show that the lower bound from the previous section is tight. We assume that in the considered uniform metric space all nodes have a distance of one to each other. The corresponding metric is defined on $k + n$ nodes and the algorithm has k servers for serving the requests, where $n, k \geq 1$. We therefore can assume that at any point in the algorithm execution there are always k nodes covered by a server and n nodes with a hole.

The presented algorithm will make use of accumulated delays on each node that has a hole. The idea is that as soon as the accumulated delay of a node reaches a certain threshold, the algorithm needs to move a server to the corresponding hole. We will refer to the nodes whose requests have reached this certain threshold, but have not been served by ALG yet, as *critical* nodes. Let s_1,...,s_k denote the servers of ALG. We assign a history counter to each server s_i that remembers the ordering in which the servers were moved. Note that the history counter therefore will need to remember the last time when s_i was used to serve a request and that the counters can be updated by ALG with every new request σ_i. Using the history counter, ALG is able to deterministically choose the server with the smallest history counter to be moved as we will discuss in Section 4.

The Online Algorithm ALG

The considered online algorithm ALG starts out with the same server constellation as its offline adversary OPT. At the beginning of the algorithm, the history counter of all servers is set to 0. Once a new requests appears, the algorithm ALG executes the following steps:

1. If a new request appears on a node with a server of ALG, such a request is served immediately and the history counter of each server is updated.
2. If a new request appears on a node with a hole, an accumulative delay counter is started at this node or incremented if already existing.
3. Once some accumulative delay counter reaches a predefined threshold δ, where $\delta > 0$ is a constant, ALG moves a server according to Properties 1 and 2 defined below to the corresponding hole and updates the history counters of all the servers.

Note that we will determine the optimal value for δ later in the proof.

In the case of concurrent requests, ties are broken arbitrarily. We will next define two properties that ALG needs to satisfy in order to be $2k + 1$-competitive with respect to our analysis. The first property is *conservativeness*, which is often used for the paging problem, and the second property is the so-called *perfect-usefulness*. Both properties aim at the fact that an algorithm should reuse each server as few times as possible. These properties are necessary for the analysis and they will help us to derive tight bounds for some well-known paging algorithms that are equipped with delays in Section 4.

▶ **Property 1.** *(Conservativeness) An online algorithm ALG is called conservative if, for every subsequence of requests σ' that contains requests on k or fewer critical nodes, ALG incurs a moving cost of at most k in order to serve the requests of σ'.*

▶ **Property 2.** *(Perfect-usefulness) An online algorithm ALG is called perfectly-useful if it moves exactly one server for every critical node.*

Phase Partitioning

The partitioning of the input sequence that we will introduce here is different from the analysis of the paging problem. In the paging problem, the phase partitioning is only dependent on the input sequence, since it aims at minimizing the moving cost of any online algorithm. When delays are added, such a static partitioning cannot be used anymore, as an optimal algorithm can incur arbitrarily much delay for some requests, thus making its own actions independent of the input sequence.

In order to analyze the competitive ratio of the previously presented algorithm, we first define phases on every node with respect to the actions of the offline algorithm OPT. Consider therefore k servers and a fixed request sequence σ. We define the phases on the holes of OPT, i.e., the nodes that are not covered by a server of OPT. Let x_j be such a hole of OPT and σ_i be the first unserved request on x_j after x_j has become a hole. Let σ_i appear at time t_i and let $t_{i'}$ be the point in time when OPT serves the request σ_i and potentially other request that appeared on x_j within the time interval $[t_i, t_{i'}]$. We call the time interval $[t_i, t_{i'}]$ a *phase* p of OPT. We further associate each phase with a delay and a moving cost, denoted $C^{OPT}_{delay}(p)$ and $C^{OPT}_{moving}(p)$ respectively. $C^{OPT}_{delay}(p)$ is defined to be the delay cost incurred by all open requests on x_j during the time interval $[t_i, t_{i'}]$. $C^{OPT}_{moving}(p)$ only consists of a moving cost of 1 when OPT moves a server to x_j at the end of phase p. Note that the phases defined with respect to the same node do not overlap, but the phases defined on different nodes may do so. We will order the phases on all nodes with respect to their starting point and enumerate them. We will further associate each i-th phase p_i with a total delay cost denoted δ_i, where $\delta_i := C^{OPT}_{delay}(p_i)$. In contrast to OPT, we will define the delay and the moving costs of ALG to be the accumulated delay and moving cost over all nodes during the time interval $p_i = [t_i, t_{i'}]$. We denote these costs $C^{ALG}_{delay}(p)$ and $C^{ALG}_{moving}(p)$ respectively.

Our basic proof idea to show the competitiveness of ALG will be to partition the input sequence σ into phases and analyze the competitive ratio of ALG on each phase separately. If ALG is strictly α-competitive for every subsequence of requests of a phase, then the whole sequence σ is also α-competitive, as the following lemma states:

▶ **Lemma 2.** *Let there be two phases p_1, p_2 and let $p = p_1 \cup p_2$ be the union of the phases defined as the time interval between the start of phase p_1 and the end of longest of the two phases. If the subsequences of requests p_1 and p_2 are strictly α-competitive, then p is also strictly α-competitive.*

Proof. Let $C_{ALG}(p)$, $C_{ALG}(p_1)$, $C_{ALG}(p_2)$ be the costs of ALG and let $C_{OPT}(p)$, $C_{OPT}(p_1)$, $C_{OPT}(p_2)$ be the costs of OPT during the phase p, p_1, p_2 described by the corresponding time interval respectively. Note that the following holds:

$$\begin{aligned} C_{ALG}(p) &\le C_{ALG}(p_1) + C_{ALG}(p_2) \\ C_{OPT}(p) &= C_{OPT}(p_1) + C_{OPT}(p_2) \end{aligned} \tag{1}$$

Note that the equation for OPT holds because the delay and the space costs of OPT are defined with respect to a single node and that consecutive phases that take place on the same node do not intersect. In contrast to this, we defined the costs of ALG to be the costs over all nodes, i.e., in the case of overlapping phases on different nodes, some of the costs might be counted for both intervals. By the assumption of the lemma statement we have

$$\frac{C_{ALG}(p_1)}{C_{OPT}(p_1)} \leq \alpha, \quad \frac{C_{ALG}(p_2)}{C_{OPT}(p_2)} \leq \alpha \tag{2}$$

Now we show for phase $p = p_1 \cup p_2$ that

$$\begin{aligned} \frac{C_{ALG}(p)}{C_{OPT}(p)} &\overset{(1)}{\leq} \frac{C_{ALG}(p_1) + C_{ALG}(p_2)}{C_{OPT}(p_1) + C_{OPT}(p_2)} = \frac{\frac{C_{ALG}(p_1)}{C_{OPT}(p_1)\cdot C_{OPT}(p_2)} + \frac{C_{ALG}(p_2)}{C_{OPT}(p_1)\cdot C_{OPT}(p_2)}}{\frac{1}{C_{OPT}(p_2)} + \frac{1}{C_{OPT}(p_1)}} \\ &\overset{(2)}{\leq} \frac{\frac{1}{C_{OPT}(p_2)}\alpha + \frac{1}{C_{OPT}(p_1)}\alpha}{\frac{1}{C_{OPT}(p_2)} + \frac{1}{C_{OPT}(p_1)}} = \alpha \end{aligned} \tag{3}$$

■

3.1 Algorithm Analysis

Let ALG be an algorithm from the set of online algorithms defined in Section 3 and let OPT be any optimal offline algorithm, both equipped with k servers. We need to handle the first phase p_1 together with the last phase, since requests in the last phase might only be open for ALG, but not for OPT. In the next lemma we will present the competitiveness analysis for any middle phase, i.e., not the first or the last phase, of the partitioning:

▶ **Theorem 3.** *The presented deterministic online algorithm based on conservativeness and perfect-usefulness in Section 3 is $2k+1$-competitive for the uniform k-OSD problem.*

Proof. We will start this proof by showing that the considered deterministic algorithm is $\max\{k \cdot (1+\delta), 2k+1, \frac{1}{\delta} \cdot (k+1) \cdot (1+\delta)\}$-competitive on each phase p_i. By setting $\delta := \frac{k+1}{k}$ at the end of this proof, we will achieve the optimal competitive ratio of $2k+1$ for ALG, as stated in the theorem statement. In order to show the above formula for the competitive ratio, we fix any request sequence σ and consider its phase partition. For each phase, we make a case distinction based on the size of δ_i where δ_i is the delay cost of OPT in p_i defined in the previous section.

We claim that for any phase p_i with a total delay cost of δ_i, ALG incurs costs for at most $k + \lfloor \frac{\delta_i}{\delta} \rfloor \cdot (1+k)$ requests on holes, each of which has a delay cost of δ. The main idea for this proof is that every time OPT incurs a delay cost of δ, ALG can afford to move a server. Throughout the proof, we will assume that a phase of OPT starts at x_j with a request σ_i for which both algorithms, OPT and ALG, have a hole on x_j. This assumption is justified since all requests that can be immediately served by ALG but not by OPT do not contribute to a larger competitive ratio and can therefore be omitted in the analysis. Note that all requests that can be immediately served by OPT but not by ALG do increase the competitive ratio and will be taken care of in so-called post-phases that will be defined later.

Let the phase p_i begin with an open request on node x_j which is a hole for both algorithms. Assume at first that $\delta_i > \delta$. At the beginning of the request sequence, both algorithms have the same server constellation. Since we assumed $\delta_i > \delta$, ALG will have to serve the node x_j at least once before OPT does. In the worst case, due to the definition of the accumulative delay cost of ALG, ALG will have to serve x_j at most $\lfloor \frac{\delta_i}{\delta} \rfloor$ times before OPT moves a server to x_j. After ALG serves x_j for the first time, there can be requests on at most k distinct nodes that are covered by the servers of OPT but not by the servers of ALG. These requests are served by OPT immediately whereas ALG incurs delay and moving costs for each of the requests. Note that there cannot be more than k such requests due to Property 1. Moreover, ALG would end up with the same server constellation as OPT after k such requests, as a request would appear on a hole for ALG and OPT otherwise. Observe that each request is served after time δ and only one server is moved to serve that request according to Property 2. Therefore, ALG will incur costs of at most $k \cdot (1+\delta)$ for these k requests. This situation can appear every time after ALG serves a request on x_j. Thus, ALG would incur a cost of $\lfloor \frac{\delta_i}{\delta} \rfloor \cdot (k+1) \cdot (1+\delta)$.

A phase might end with or only consist of a small delay interval of length $\delta_i - \lfloor \frac{\delta_i}{\delta} \rfloor \delta$ just before OPT serves all open requests on x_j. Note that the k requests that can be issued on the nodes which are covered by the servers of OPT are already accounted for in the previous analysis, if $\delta_i > \delta$. The costs of ALG for serving the last requests on this time interval are however not accounted for so far. Neither are the costs for the time before ALG serves x_j for the first time, which corresponds to the whole phase if $\delta_i < \delta$. In fact, also during this time requests on the k nodes which are covered by the servers of OPT but not by the servers of ALG might appear. In order to account for these k requests and for the moving costs of 1 for serving x_j after OPT has done so, we will introduce the concept of a *post-phase*. A post-phase consists of requests on at most k nodes after the end of a phase. Since OPT has a server at x_j at the end of phase p_i, x_j is one of the k nodes covered by a server of OPT in the post-phase. Therefore, the delay and the moving costs for serving the last requests on x_j will also be covered in the post-phase. As the post-phase consists of requests on at most k nodes, the total cost for ALG during a post-phase is at most $k \cdot (1+\delta)$. This part of the analysis explains how the costs between two non-overlapping phases are covered.

In the case when the phases overlap, or even when the next phase ends before the previous, the upper bound on the number of requests still holds. The serving times of requests of any number of overlapping phases can be ordered by the point in time when ALG moves a server to the corresponding node. Each movement thereby accounts for the requests on k nodes that are covered by the servers of OPT and which may appear after every served request by ALG. Since each phase can have at most one post-phase, it is also calculated in the above costs. Therefore, overlapping phases cannot increase the competitive ratio of ALG. An example of the phase partition on a small example with long, short, and overlapping phases as well as the corresponding post-phases is visualized in Figure 1.

Finally, we can summarize the costs incurred by each of the algorithms for any phase. OPT always incurs costs of $C^{OPT}_{total}(p_i) = 1 + \delta_i$. The costs of ALG are calculated as the sum of the costs during a phase and the costs of the post-phase. These costs are equal to

$$C^{ALG}_{total}(p_i) = \left(\left\lfloor \frac{\delta_i}{\delta} \right\rfloor \cdot (k+1) + k \right) \cdot (1+\delta)$$

The competitive ratio for each phase can be calculated as the quotient of the two terms. In order to estimate the competitive ratio for the algorithm, we need to make a case distinction on δ_i - the delay cost of a phase p_i:

$\delta_i < \delta$: In this case we have $\lfloor \frac{\delta_i}{\delta} \rfloor = 0$. Therefore the competitive ratio becomes

$$C^{ALG}_{total}(p_i) / C^{OPT}_{total}(p_i) \leq k \cdot (1+\delta)$$

$\delta_i = \delta$: In this case we have $\lfloor \frac{\delta_i}{\delta} \rfloor = 1$ and the term $1 + \delta_i = 1 + \delta$. The competitive ratio is

$$C^{ALG}_{total}(p_i) / C^{OPT}_{total}(p_i) \leq k + (k+1) = 2k + 1$$

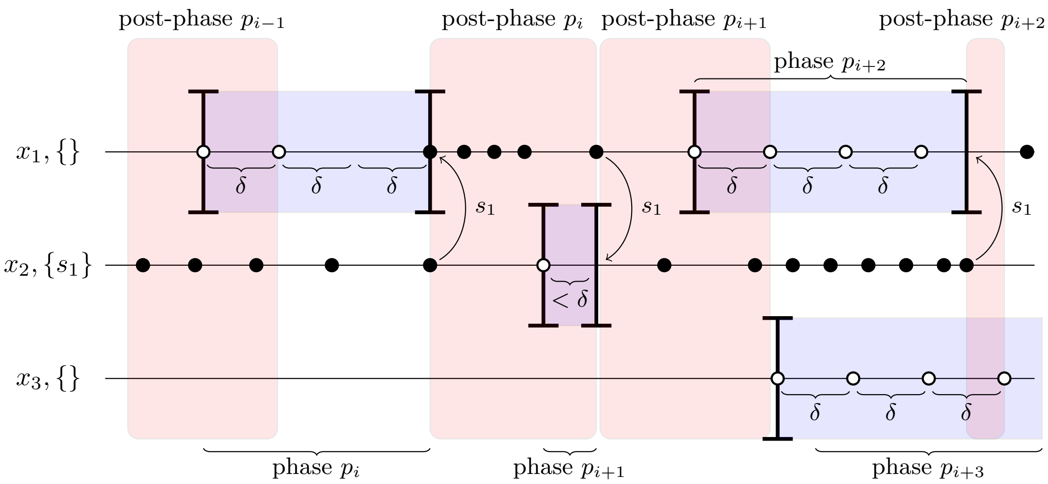

Figure 1 An example of the partitioning of the phases into a phase and a post-phase on three nodes, where $k = 1$. The server of OPT is located on the node x_2 at the beginning of the visualized sequence and the uncovered nodes are denoted with empty braces. OPT incurs delay for requests on its holes, represented as unfilled circles, and moves the server at the end of the phase. Solid circles represent requests that can be served immediately by OPT, possibly incurring a moving cost. For visual purposes, we assume that all requests have the same delay function. Each phase starts with an open request and ends once OPT serves this request, which is visualized by arrows between the nodes. Moreover, a post-phase is appended at the end of any phase, where requests on the two covered nodes by OPT may appear. Also between any two open requests, possibly from different phases, requests may appear on the two uncovered nodes. The figure depicts three different situations considered in the analysis: $\delta_i \geq \delta$, $\delta_i < \delta$, and when two phases overlap.

$\delta_i \to \infty$: In this case the delay cost for an interval δ_i is maximized. We can divide the numerator and the denominator of the competitive ratio by δ_i and consider the limit for $\delta_i \to \infty$. This results in a competitive ratio of

$$C_{total}^{ALG}(p_i)/C_{total}^{OPT}(p_i) \leq \frac{1}{\delta} \cdot (1+k) \cdot (1+\delta)$$

For all other values of δ_i, the competitive ratio lies between $k\cdot(1+\delta)$, $2k+1$ and $\frac{1}{\delta}\cdot(k+1)\cdot(1+\delta)$. This holds, since the competitive ratio as a function of δ_i is minimized for the values of δ_i considered in the three cases of the case distinction. The competitive ratio in this case can be defined as $\max\{k \cdot (1+\delta), 2k+1, \frac{1}{\delta} \cdot (1+k) \cdot (1+\delta)\}$, just as in the statement of this lemma. Note that we omitted the case $\delta \to \infty$, as it can be easily shown that ALG cannot be competitive in this case.

In order to show that ALG is $2k+1$-competitive, we will consider the maximum competitive ratio achieved in the above case distinction. As a function of δ, the competitive ratio of Case 1 is a strictly monotonically increasing function, while the one of Case 3 is strictly monotonically decreasing. The global minimum can be found by looking at their intersection point. Setting these functions equal to each other, we obtain

$$\frac{1}{\delta} \cdot (k+1) \cdot (1+\delta) = k \cdot (1+\delta)$$

which is equal if and only if $\delta = \frac{k+1}{k}$. This means that ALG is optimal, if δ is set to the value $\frac{k+1}{k}$. Applying this δ to Case 1 or Case 3, we get a competitive ratio of $2k+1$, which is equal to the competitive ratio of Case 2. ∎

4 Deterministic Paging Algorithms with Delays

In our analysis of the upper bound, we considered an online algorithm with the properties of conservativeness and perfect-usefulness, where we move the server with the smallest history counter in case of a critical node based on its replacement policy. In this section, we will draw a connection between our ideas in the analysis and the well-known deterministic paging algorithms. In the paging problem, online algorithms are often divided into two kinds of algorithms - the marking and the conservative algorithms, a broad analysis of which can be found in the book of Borodin and El-Yaniv [5]. We will restrict ourselves to three main conservative algorithms that we will equip with delays according to the definition of our k-OSD algorithm in Section 3. We will then show that these generalized algorithms satisfy Properties 1 and 2 and thus are $2k+1$ competitive. We will further address the flush-when-full algorithm, a marking algorithm, whose competitive ratio does not directly follow from our analysis in Section 3.1. The following paging algorithms are considered in this section:

- **(i)** LRU (least recently used): In the case where a server has to leave a node in order to cover a new node that has become critical, it chooses the server on the node whose most recent request was earliest. That is, the history counter of each server is updated with each served request, and the server with the smallest history counter is moved.
- **(ii)** CLOCK (clock replacement): This algorithm is an approximation of the LRU where a single "use" bit represents the implicit timestamp of LRU.
- **(iii)** FIFO (first-in/first-out): In case of a critical node, let the server from the node that has been covered by a server longest be moved to the critical node. This strategy corresponds to updating history timers whenever a server was moved last.
- **(iv)** FWF (flush-when-full): Here, the idea is to only let servers with a history counter 0 serve critical nodes and update the history counter to the time when the server was moved. Once all history counters are non-zero and a new critical node appears, all k counters are reset to 0, i.e., they are "flushed".

Note at first that all presented algorithms are paging algorithms without delay and therefore only define which servers will be moved and not at which point in time. The time at which a server is moved is defined in the definition of the generalized k-OSD algorithm in Section 3. This algorithm demands that for each critical node only one server movement is performed using one of the above strategies. Therefore, all these algorithms equipped with delays satisfy Property 2.

▶ **Lemma 4.** *The LRU, CLOCK and FIFO algorithms equipped with the delay function from Section 3 are $2k+1$-competitive for the k-OSD problem.*

Proof. It has been shown in the literature that the LRU, CLOCK, and FIFO algorithms are conservative, see for example [5]. Since we defined the algorithms and the conservativeness property with respect to server movements and their history counters, we will use the LRU algorithm to show that it satisfies conservativeness with respect to the definition of Property 1. Assume that there is a subsequence σ with requests on k distinct holes. Since there exists a well-defined order of the history counters, for any critical node, the LRU algorithm chooses the server on the node whose most recent request was the earliest. The history counter of that server is updated and never used again in σ, since the other $k-1$ history counters are smaller. That means LRU moves exactly one server for each critical node, i.e., k movements in total for the whole sequence σ. This fulfills the conditions of Property 1. ■

▶ **Lemma 5.** *The FWF algorithm equipped with the delay function from Section 3 is* $2k+1$*-competitive for the* k*-OSD problem.*

Proof. Observe that FWF is not a conservative algorithm. This is because, within a subsequence of k critical nodes, FWF can reset all its history counters and reuse some of its servers for this subsequence, thus moving servers more than k times. Between two flush operations, the algorithm however does satisfy Property 1. We will not give the full proof for the competitiveness of FWF here, but instead the idea of how the reused servers can be accounted for in the analysis. Consider therefore consecutive phases defined as in Section 3: At the beginning of the sequence, the servers of OPT and FWF start on the same nodes. Assume that the first phase is of the kind $\delta_i < \delta$. In this case, after a movement of OPT, FWF has to serve at most k additional requests on k nodes that are already served by OPT. That is, FWF would update all k history counters and have a critical node at the end of phase p_1 in the worst case. In addition, its servers will reach the same constellation as the servers of OPT. With the critical node, all counters will be reset to 0 and a new phase will begin. In this case, the competitive ratio of $2k+1$ holds. The interesting cases are when the phases overlap or when requests are not issued on all k nodes covered by OPT. This way, OPT and FWF will not have their servers on the same nodes when entering the next phase p_2 and not all history counters of FWF will be equal to zero. In this case, FWF might have to reset its history counters during the next phase and reuse some of the already used servers in that phase. Note that the number of reused servers can be upper bounded by the number of nodes that are covered by the servers of OPT but not by the servers of FWF at the beginning of p_2. Therefore, we can always count the delay and moving costs of reused servers to the post-phase of the previous phase. In the case when $\delta_i > \delta$, a similar argument can be derived for all subphases with delay cost δ. ◼

From the above analysis, it follows that conservative algorithms can be used to extend our analysis and obtain tight results for the uniform k-OSD problem. The same observations were shown in the paging problem without delays with a competitive ratio of k in [5] by Borodin and El-Yaniv. There are of course also some well-known deterministic paging algorithms that are not competitive for the paging problem and therefore also for the uniform k-OSD problem. Such algorithms are for example the LIFO (last-in/first-out) or the LFU (least frequently used).

5 Discussion

In this paper, we have considered a special case of the k-server problem with delays, where the distances between all nodes in the system are equal. We devised deterministic online algorithms with competitive ratio $2k+1$ by equipping known conservative algorithms with a carefully chosen delay function. We have also shown that this bound is tight. The tight bound for the k-OSD problem on general finite metric spaces is not known yet. Our results, however, inspire the following question which is analog to the classical k-server conjecture: *Is there a* $2k+1$*-competitive deterministic algorithm for the* k*-server problem with delays on any finite metric space?* This question has been answered negatively by Azar et al. [1] in the non-clairvoyant setting, where they show that the lower bound for a weighted start depends on the aspect ratio.

On the other hand, our results can be positive evidence that there are many variants of the k-server problem for which the classic k-server conjecture does not generalize, i.e., the deterministic competitive ratio in uniform metrics is different than arbitrary metric spaces.

For example, in the weighted k-server problem, even for $k = 2$, there exists a 5-competitive deterministic algorithm on uniform metrics [8] but it is known that for the line metric the competitive ratio is more than 10 [14].

Another natural extension to our work would be to consider randomized algorithms of uniform k-OSD problems. Without delays it is only possible to choose a random distribution of the servers which are to be moved, thus achieving a well-known tight bound of H_k for k servers. When delays are allowed, a new possibility is given to algorithms, namely to also make use of a randomized delay function. This opens the question of what is the tight bound can be achieved by a randomized algorithm for the uniform k-OSD problem.

References

1. Yossi Azar, Arun Ganesh, Rong Ge, and Debmalya Panigrahi. Online service with delay. In *Proceedings of the 49th Annual ACM SIGACT STOC*, pages 551–563. ACM, 2017.
2. Nikhil Bansal, Niv Buchbinder, Aleksander Madry, and Joseph S. Naor. A polylogarithmic-competitive algorithm for the k-server problem. In *2011 IEEE 52nd Annual FOCS*, pages 267–276, 2011.
3. Nikhil Bansal, Niv Buchbinder, and Joseph S. Naor. A primal-dual randomized algorithm for weighted paging. *Journal of the ACM (JACM)*, 59(4):19, 2012.
4. Marcin Bienkowski, Artur Kraska, and Paweł Schmidt. Online service with delay on a line. In *SIROCCO*, pages 237–248. Springer, 2018.
5. Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*, chapter 1, 3, 10, pages 1–22, 32–43, 150–181. Cambridge University Press, 2005.
6. Marek Chrobak, H. Karloff, Tom Payne, and Sundar Vishwnathan. 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 Jiří Sgall. The weighted 2-server problem. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 593–604. Springer, 2000.
9. Yuval Emek, Shay Kutten, and Roger Wattenhofer. Online Matching: Haste Makes Waste! STOC '16, pages 333–344, New York, NY, USA, 2016. ACM.
10. Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel D. Sleator, and Neal E. Young. Competitive paging algorithms. *Journal of Algorithms*, 12(4):685–699, 1991.
11. Elias Koutsoupias. The k-server problem. *Computer Science Review*, 3(2):105–118, 2009.
12. Elias Koutsoupias and Christos H. Papadimitriou. On the k-server conjecture. *Journal of the ACM (JACM)*, 42(5):971–983, 1995.
13. Elias Koutsoupias and Christos H. Papadimitriou. The 2-evader problem. *Information Processing Letters*, 57(5):249–252, 1996.
14. Elias Koutsoupias and David Scot Taylor. The cnn problem and other k-server variants. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 581–592. Springer, 2000.
15. J. R. Lee. Fusible hsts and the randomized k-server conjecture. In *2018 IEEE 59th Annual Symposium on FOCS*, pages 438–449, Oct 2018.
16. Mark S. Manasse, Lyle A. McGeoch, and Daniel D. Sleator. Competitive algorithms for online problems. In *Proceedings of the twentieth annual ACM STOC*, pages 322–333. ACM, 1988.
17. Mark S. Manasse, Lyle A. McGeoch, and Daniel D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11(2):208–230, 1990.
18. Prabhakar Raghavan and Marc Snir. Memory versus randomization in on-line algorithms. In *International Colloquium on Automata, Languages, and Programming*, pages 687–703. Springer, 1989.
19. Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.