Fairness in the k-Server Problem

Mohammadreza Daneshvaramoli
University of Massachusetts Amherst, MA, USA

Mohammad Hajiesmaili
University of Massachusetts Amherst, MA, USA

Shahin Kamali
York University, Toronto, Canada

Helia Karisani
University of Massachusetts Amherst, MA, USA

Cameron Musco
University of Massachusetts Amherst, MA, USA

Abstract

We initiate a formal study of fairness for the k-server problem, where the objective is not only to minimize the total movement cost, but also to distribute the cost equitably among servers. We first define a general notion of (α, β)-fairness, where, for parameters $\alpha \geq 1$ and $\beta \geq 0$, no server incurs more than an α/k-fraction of the total cost plus an additive term β. We then show that fairness can be achieved without a loss in competitiveness in both the offline and online settings. In the offline setting, we give a deterministic algorithm that, for any $\varepsilon > 0$, transforms any optimal solution into an (α, β)-fair solution for $\alpha = 1 + \varepsilon$ and $\beta = O(\text{diam} \cdot \log k/\varepsilon)$, while increasing the cost of the solution by just an additive $O(\text{diam} \cdot k \log k/\varepsilon)$ term. Here diam is the diameter of the underlying metric space. We give a similar result in the online setting, showing that any competitive algorithm can be transformed into a randomized online algorithm that is fair with high probability against an oblivious adversary and still competitive up to a small loss.

The above results leave open a significant question: can fairness be achieved in the online setting, either with a deterministic algorithm or a randomized algorithm, against a fully adaptive adversary? We make progress towards answering this question, showing that the classic deterministic Double Coverage Algorithm (DCA) is fair on line metrics and on tree metrics when $k = 2$. However, we also show a negative result: DCA fails to be fair for any non-vacuous parameters on general tree metrics. We further show that on uniform metrics (i.e., the paging problem), the deterministic First-In First-Out (FIFO) algorithm is fair. We show that any "marking algorithm", including the Least Recently Used (LRU) algorithm, also satisfies a weaker, but still meaningful notion of fairness.

2012 ACM Subject Classification Theory of computation → Online algorithms; Theory of computation → Design and analysis of algorithms

Keywords and phrases k-server problem, online algorithms, fairness, competitive analysis

Digital Object Identifier 10.4230/LIPIcs.ITCS.2026.45

Related Version *Full Version*: `https://arxiv.org/abs/2512.20960`

Funding This research is supported by National Science Foundation grants 2045641, 2325956, 2512128, and 2533814. And support of the Natural Sciences and Engineering Research Council of Canada (NSERC) funding reference number RGPIN-2025-07295.

1 Introduction

The k-server problem, introduced by Manasse, McGeoch, and Sleator [50], is central to the theory of online algorithms. In this problem, k servers move within a metric space to serve a sequence of requests, and the goal is to minimize the total movement cost. In particular, the algorithm is presented with an input sequence $\sigma = (\sigma_1, \ldots, \sigma_T)$ consisting of T requests,

each corresponding to a point in the metric space. At time $t \in \{1, \ldots, T\}$, it must move some server to request σ_t, incurring a cost equal to the distance moved by that server. The challenge is to decide which server to move at each step in order to minimize the total movement cost. In the online setting, such a decision is made without prior knowledge of forthcoming requests.

Over the past three decades, the k-server problem has served as a fundamental benchmark for studying online decision-making and competitive analysis, with deep connections to paging, metrical task systems, and beyond [32, 17, 16, 43, 33, 46]. Many variants on the problem, including those where server costs are weighted or determined by different metric spaces [12, 7, 8, 45, 23, 12], or where requests have both pick-up and drop-off locations [28, 20, 24] have been studied. The problem has also been investigated under advice complexity, where the algorithm receives some information about the input in the form of some error-free bits of advice [31, 55, 14, 35], and more recently under learning-augmented frameworks with predictions, where additional information is provided, but may be erroneous [48, 6, 5, 25].

Fairness for k-server. The k-server problem can be motivated by scenarios where a service provider dispatches mobile agents to client locations, aiming to minimize the total travel distance. In such settings, it is also important to consider a notion of fairness: ideally, the workload should be balanced to avoid overburdening any single server. Similarly, in distributed systems and cloud platforms, k virtual machines (servers) offering identical services may need to migrate between nodes to handle incoming requests [21, 4, 29]. In such domains, in addition to minimizing the total migration cost, it is important to ensure that this cost is distributed fairly among servers.

Fairness has been studied in a wide range of offline and online decision-making problems. Examples include various forms of clustering [1, 49, 40, 30], matching [36, 37, 2], scheduling [3, 52, 38, 15], selection problems like knapsack [54, 47, 34], and the secretary problem [41, 10]. A common interpretation of fairness involves ensuring proportional balance across different groups or agents. For instance, fair clustering methods may consider colored points and seek to minimize the standard clustering cost while ensuring that all colors are approximately equally represented in each cluster [1, 40]. Similarly, fair class matching considers settings where vertices are associated with agents, aiming to maximize the number of matched vertices while ensuring that each agent is proportionally represented in the matching [37, 36, 58].

More closely related to k-server, Singh et al. [56] study the *k-FOOD problem*, where requests are defined by source-destination pairs and associated pickup time windows, effectively enforcing deadlines for serving a request. In the standard version of the problem, the objective is to minimize the total distance traveled by all servers, while in the *fair* variant, the goal is to minimize the maximum distance traveled by any server. [56] establishes NP-hardness of the offline version of both the standard and fair versions – hardness stems primarily from additional constraints such as time windows and deadlines. Note that these constraints are not present in the classical k-server formulation, and the offline version of the problem is solvable in polynomial time [16].

Martinez-Sykora et al. [51] study a food delivery problem with fairness constraints that aim to minimize the range of waiting times across servers – they formulate this problem as an ILP and demonstrate that it can be solved on small input instances, but do not show that it is tractable in general. Ong et al. [53] study algorithms for ride scheduling that aim to ensure fairness to customers (i.e., requests rather than servers) – they focus primarily on experimental evaluation rather than theoretical guarantees. Chiplunkar et al. [22] study online competitive algorithms for a min-max fairness objective for k-paging, which is a special case of k-server, again focusing on fairness across requests rather than servers.

Despite the vast literature on fairness in algorithm design, including on problems related to k-server, perhaps surprisingly, little work has directly addressed fairness in the k-server problem itself. The original work by Manasse, McGeoch, and Sleator [50] proposed the *Balance Algorithm (BAL)*, which selects a server to move at each step by greedily minimizing the maximum cost incurred across all servers after that step. This algorithm is explicitly motivated by fairness and was shown to be k-competitive on metric spaces with $k+1$ points [50]. Unfortunately, however, BAL fails to achieve a bounded competitive ratio on general metrics. Thus, to the best of our knowledge, the question of designing algorithms for the k-server problem that are both fair and competitive has remained entirely open.

The well-studied *paging problem* is a classical special case of the k-server problem in which the underlying metric space is uniform – i.e., all pairwise distances are equal [57, 39]. Paging models the behavior of caches in operating systems, memory hierarchies, and web caching. A *fault* occurs when a request arrives for a page (vertex) that is not currently stored in one of the k cache slots (servers). In the k-server interpretation, each cache slot corresponds to a server, and each fault corresponds to moving a server to the requested vertex at unit cost. Classical deterministic paging algorithms include First-In-First-Out (FIFO) and Least-Recently-Used (LRU). In this interpretation, FIFO moves the server that has not moved for the longest time, while LRU moves the server whose most recent movement occurred farthest in the past. Both FIFO and LRU achieve the optimal deterministic competitive ratio of k for paging [57]. This close connection between paging and the k-server problem motivates our later analysis of the fairness properties of FIFO, LRU, and marking algorithms on uniform metrics.

Full version

Due to the ITCS page limit, the conference submission omits several technical proofs and additional discussion. A full version of this paper, containing all appendices and complete proofs, is available on arXiv.

1.1 Our Contributions

In this work, we initiate a formal study of fairness as an objective for the k-server problem. Our contributions are summarized below.

1.1.1 (α, β)-Fairness Definition

We start by formalizing a natural notion of fairness for the k-server problem: we say that a deterministic algorithm A is (α, β)-fair on request sequence σ if no server pays cost greater than $\alpha \cdot \frac{\text{cost}(A,\sigma)}{k} + \beta$, where $\text{cost}(A, \sigma)$ is the total cost paid by all servers to serve σ. A perfectly fair algorithm, in which all servers pay identical costs, would have $\alpha = 1$ and $\beta = 0$.

For randomized algorithms, we distinguish between *ex-ante* and *ex-post* fairness [9]. Ex-ante fairness means *fairness in expectation* – i.e., a randomized algorithm A is (α, β)-ex-ante fair on request sequence σ if no server has an *expected cost* greater than $\alpha \cdot \frac{\mathbb{E}[\text{cost}(A,\sigma)]}{k} + \beta$. Via a simple random swapping strategy, it is easy to achieve $(1, \text{diam})$-ex-ante fairness while maintaining competitiveness. Here diam denotes the diameter of the underlying metric space. Suppose we start with an algorithm over the given metric space, with expected cost $\mathbb{E}[\text{cost}(A, \sigma)]$ on input σ. At the beginning of the algorithm, we randomly permute the server identities, swap the server positions according to the permutation, and reassign the workloads of the servers going forward according to the permutation as well. Then, after the additive $\leq \text{diam}$ cost paid per server for the initial swap, by symmetry, all servers have exactly the same *expected cost*, equal to $\frac{\mathbb{E}[\text{cost}(A,\sigma)]}{k}$.

The above approach, however, is not very compelling. While the servers have the same expected cost up to a small additive factor after the initial random swap, the cost distribution of the servers may still be very unfair for any given choice of swap. Thus, throughout the paper, we adopt an *ex-post* notion of fairness for randomized algorithms. Specifically, we say that a randomized algorithm A is (α, β)-fair on a request sequence σ if, with high probability over its random choices, every server incurs cost no larger than $\alpha \cdot \frac{\text{cost}(A,\sigma)}{k} + \beta$. The exact probability for which the bounds holds will naturally impact the achievable α and β.

While we focus on (α, β)-fairness in this work, we show that it either encompasses or can be easily reduced to other natural notions of fairness, e.g., multiplicative fairness in which all servers pay cost within a fixed multiplicative factor of each other, or additive fairness, in which all servers pay cost within a fixed additive factor of each other. We discuss such alternative definitions along with reductions and connections between them in the full version of the paper.

1.1.2 Offline Fairness

Our first contribution is to show that, in the offline setting, (α, β)-fairness is achievable without significant loss in competitiveness. In particular, in Section 3 we prove:

▶ **Theorem 1** (Theorem 11 Restated). *For any k-server input sequence σ over a metric space with diameter* diam *and any $\varepsilon > 0$, there is an offline solution for σ that is (α, β)-fair for $\alpha = 1 + \varepsilon$ and $\beta = O\left(\frac{\log k \cdot \text{diam}}{\varepsilon}\right)$, and has total cost bounded by $\textsc{Opt}(\sigma) + O\left(\frac{k \log k \cdot \text{diam}}{\varepsilon}\right)$, where $\textsc{Opt}(\sigma)$ is the offline optimal cost. Further, this fair solution can be computed in polynomial time.*

That is, we can achieve fairness up to an $\alpha = 1 + \varepsilon$ factor while paying just a small additive factor β and a small increase in the optimal offline cost, which is a function of k, diam and $1/\varepsilon$, but importantly, independent of the input length and the offline optimal cost.

Proof overview. To prove Theorem 1, we introduce an algorithm (Algorithm 1) that takes any optimal solution for an input sequence σ and transforms it into a fair solution through a sequence of pairwise server *swaps*. A pairwise swap is a modification to a solution sequence in which, at a given time z, two servers exchange positions in the metric space, and at all times after z, they serve the requests previously served by the other server.

Our algorithm proceeds in rounds, iteratively improving the fairness of the initial solution. At each round, it identifies the server H that currently pays the highest cost and the server L that currently pays the lowest cost. It further identifies a time step such that, if the servers are swapped at this point, their loads will be nearly balanced. We prove that, as long as the solution is not yet (α, β)-fair, after any server participates in at least two swaps, the maximum server load is decreased by a multiplicative factor $\approx 1 + \varepsilon$. Thus, after $O(\log_{1+\varepsilon} k) = O(\frac{\log k}{\varepsilon})$ swaps per server, the maximum load is decreased from potentially as high as k times the average load to at most $(1 + \varepsilon)$ times the average load. Since each swap itself incurs additive cost $\leq 2 \cdot \text{diam}$, this leads to our bound of $\alpha = 1 + \varepsilon$, $\beta = O\left(\frac{\log k \cdot \text{diam}}{\varepsilon}\right)$ and our increase in cost over $\textsc{Opt}(\sigma)$ of $O\left(\frac{k \log k \cdot \text{diam}}{\varepsilon}\right)$.

It is not hard to see that our swapping algorithm runs in polynomial time, and thus, since an offline optimal solution for k-server can be computed in polynomial time [16], (α, β)-fairness is achievable in polynomial time. This contrasts with recent work on related fair scheduling problems, which have proven to be computationally hard [56, 51].

1.1.3 Online Fairness using Randomization

Our second contribution shows that (α, β)-fairness can also be achieved for online k-server using a randomized algorithm against an oblivious adversary –i.e., where the request sequence σ is chosen ahead of time and may not depend on the decision of the algorithm (this is the standard setting that randomized algorithms for online k-server are studied). In particular, in Section 4 we prove:

▶ **Theorem 2** (Theorem 12 Restated). *For any fixed $\gamma > 0$ and $\varepsilon > 0$, there is a randomized online k-server algorithm that, on any fixed input sequence σ over a metric space with diameter* diam *achieves (α, β)-fairness for $\alpha = 1 + \varepsilon$ and $\beta = O(c(\sigma)^{1/(1+\gamma)} \cdot \text{diam})$ with probability $1 - k \cdot \exp\left(-\Omega\left(\frac{\varepsilon^2}{\gamma \cdot k} \cdot c(\sigma)^{1/(\gamma+1)}\right)\right)$. Further, the total cost incurred by the algorithm is at most $c(\sigma) + O(k \cdot c(\sigma)^{1/(1+\gamma)} \cdot \text{diam})$. Here, $c(\sigma)$ is the best-known cost achievable by a randomized algorithm over the given metric space, e.g., $c(\sigma) = \text{polylog}(k, n) \cdot \text{OPT}(\sigma)$ for n-point metrics [11, 18].*

To interpret Theorem 2, consider the simplified setting with $\gamma = 1$ and $\varepsilon = 1$. The theorem shows that (α, β)-fairness is achievable with $\alpha = 2$, $\beta = O(c(\sigma)^{1/2} \cdot \text{diam})$, and cost increase $O(k \cdot c(\sigma)^{1/2} \cdot \text{diam})$ with probability $1 - k \cdot \exp(-\Omega(c(\sigma)^{1/2})$. For sequences where $c(\sigma)$ is large, this probability is very close to 1, and both β and the cost increase are $o(c(\sigma))$, and thus negligible compared to the total cost. Thus, Theorem 2 is analogous to Theorem 1, establishing that in the online setting, strong fairness is achievable without significantly impacting competitiveness. The parameter γ, which does not appear in the offline result, balances a tradeoff between β and the cost increase against the success probability.

Proof overview. To prove Theorem 2, we use the randomized server-swapping technique, in a similar way to the approach described in Section 1.1.1 for achieving ex-ante (i.e., expected) fairness. However, to give an algorithm that is fair with high probability on any input sequence, we make a series of swaps instead of one, randomly permuting the servers at each swap. We argue via concentration inequalities that the servers have similar total loads with high probability.

The difficulty is determining how often to swap – too many swaps increase the cost of the original algorithm too much, but too few swaps mean that fairness is not guaranteed with high probability. To balance this trade-off, we introduce a phase-based algorithm (Algorithm 2) that swaps anytime the total cost incurred increases by a large enough factor. As an example, in the case when $\gamma = 1$, the algorithm swaps after incurring a total cost of 1, then after incurring an additional total cost of 2, then after incurring an additional total cost of 3, and so on. It makes s swaps after incurring total cost roughly $\sum_{i=1}^{s} s = \Omega(s^2)$. Thus, after incurring cost $c(\sigma)$, the algorithm has made $O(c(\sigma)^{1/2})$ swaps, and incurs total cost at most $O(c(\sigma)^{1/2})$ between each swap. This enables us to argue that both the additional additive cost associated with the swaps and the imbalance in server loads are bounded by $O(c(\sigma)^{1/2})$ with high probability and thus negligible in comparison to the total cost $c(\sigma)$. We trade off the success probability and the cost increase through the parameter γ in Theorem 2 – roughly choosing phase i with cost increase i^γ so that the total cost incurred after s swaps is $\Omega(s^{1+\gamma})$. Increasing γ leads to fewer swaps, but a lower probably of successfully satisfying the fairness guarantee.

Table 1 Summary of deterministic fair online algorithms for different metrics.

Metric	Algorithm	Competitive Ratio	Fairness	Reference
Line	DCA	k [26]	$(1, O(k \cdot \text{diam}))$	Theorem 3
Tree with $k = 2$	DCA	$k(= 2)$ [26]	$(1.5, 1.75 \cdot \text{diam})$	Theorem 4
Uniform Metrics (paging)	FIFO	k [57]	(1,1)	Theorem 6
Metrics with $k + 1$ points	BALANCE	k [50]	$(1, \text{diam})$	Proposition 8

1.1.4 Towards Deterministic Online Fairness

Theorems 1 and 2 establish that fairness is achievable for the k-server problem in both the offline setting and the online setting using randomization against an oblivious adversary. It is straightforward to verify that Theorem 2 also holds for an adaptive online adversary that can change its input based on the server locations at each round but is unable to distinguish servers based on their IDs (and thus, cannot tell if a random swap of the servers has occurred). However, it remains open if fairness can be achieved against stronger online adversaries or, potentially, deterministically. For example, the following is open:

Is there a deterministic online k-server algorithm that is $O(k)$-competitive on general metrics while also being (α, β)-fair for $\alpha = O(1)$ and β depending only on k and diam*?*

Towards progress on the above question, in Section 5, we investigate fairness properties of several widely studied deterministic online k-server algorithms that are competitive on certain classes of metrics. In particular, we consider fairness of the Double Coverage Algorithm (DCA), which is known to be k-competitive on tree metrics. We also study the fairness of several k-competitive deterministic algorithms that apply to uniform metrics – i.e., to the classic paging problem. These include the First-In First-Out (FIFO) algorithm, and any *marking algorithm*, encompassing e.g., the Least Recently Used (LRU) algorithm.

1.1.4.1 Tree Metrics

We first show two positive results for DCA. We prove that the algorithm is strongly fair for line metrics (i.e., tree metrics whose underlying graph is a line). In particular:

▶ **Theorem 3** (Theorem 14 Restated). *The DCA algorithm run on any input sequence σ over a line metric with diameter* diam *gives a solution in which the absolute difference between the cost paid by any two servers is $O(k \cdot \text{diam})$.*

Theorem 3 implies that DCA is $(1, O(k \cdot \text{diam}))$-fair on line metrics. The proof is fairly straightforward: on a line, servers can be ordered from left to right. We can readily establish that no algorithm aiming to minimize total cost (including DCA) will ever change their relative ordering. Thus, each server moves left and right, always serving requests that land between itself and its adjacent left and right neighbors (or one of the endpoints of the line). DCA couples the movements of adjacent servers: whenever a request lands between two servers, both move towards it. Due to this coupling, we can argue that the cost difference between neighboring servers depends only on their net movement, which is bounded by diam. Telescoping across all k-servers yields a maximum overall gap in costs of $O(k \cdot \text{diam})$.

We also prove that DCA is (α, β)-fair for the 2-server problem on general tree metrics, for $\alpha = O(1)$ and $\beta = O(\text{diam})$. In particular:

▶ **Theorem 4** (Theorem 15 Restated). *The DCA algorithm run on any input sequence σ over a tree metric with diameter* diam *for the 2-server problem is $(1.5, 1.75 \cdot \text{diam})$-fair.*

We prove Theorem 4 by splitting the total cost paid by each server into the *diverging cost*, when the server is moving away from the other server, and the *converging cost*, when the server is moving towards the other. We show in the complete version of the paper that these costs both comprise a constant fraction of the total cost paid by the server. Since DCA couples converging costs (when one server moves towards the other, the other server moves as well), this allows us to argue that the servers pay the same costs up to roughly a constant multiplicative factor.

Despite the above positive results, we show that, surprisingly, DCA can exhibit substantial unfairness on general tree metrics, forcing one server to do a majority of the total work.

▶ **Theorem 5** (Theorem 16 Restated). *There is a tree metric and an input sequence σ over this metric on which DCA outputs a solution in which a single server pays cost $\Omega(k \cdot \textsc{Opt}(\sigma))$, where $\textsc{Opt}(\sigma)$ is the offline optimal cost for request sequence σ.*

Note that DCA is known to be k-competitive on general tree metrics and thus it incurs total cost $\leq k \cdot \textsc{Opt}(\sigma)$. Theorem 5 shows that a constant fraction of this cost can be incurred by just a single server, and thus DCA is not (α, β)-fair for any $\alpha = o(k)$ and fixed β.

The hard case is a tree metric that is similar to a line: the tree consists of a line of *major* nodes, each of which connects to a very close by *minor* leaf node. Requests alternate between midpoints on the line and nearby minor nodes, repeatedly forcing the leftmost server to traverse the entire structure, past other servers which have been moved to minor nodes so that they can be "passed". This causes the leftmost server to incur cost $\Omega(k \cdot \textsc{Opt})$, despite DCA being k-competitive overall.

1.1.4.2 Uniform Metrics

We next turn our attention to uniform metrics – where all points have the same distance from each other. This variant of k-server is also know as the *paging problem*. We prove several positive results for existing algorithms. For simplicity, assume that the metric is scaled so all points are at distance 1 from each other. First, we prove that the classic First-In First-Out (FIFO) algorithm, which is known to be k-competitive on uniform metrics [57], achieves essentially the strongest possible notion of fairness: since the algorithm simply "cycles through" servers (i.e., cache positions) to serve any incoming request that is not already covered (i.e., to cover any cache miss), all servers pay the same cost, up to an additive factor of one. Formally,

▶ **Theorem 6** (Theorem 17 Restated). *FIFO is $(1, 1)$-fair.*

We next observe (Theorem 18) that the popular Least Recently Used (LRU) algorithm does not enjoy such a strong fairness guarantee: there are input instances where one server pays essentially all of the cost of the algorithm (i.e., one cache slot is reloaded for essentially all cache misses seen). However, LRU, and in fact any of the broader class of *marking algorithms*, still achieves an interesting notion of fairness: no server pays cost greater than $\textsc{Opt}(\sigma)$ on request sequence σ. I.e., no server pays more than what the average cost per server might be on a worst-case input sequence with competitive ratio k. Formally:

▶ **Theorem 7** (Theorem 19 Restated). *For any marking algorithm for the k-server problem on uniform metrics, the cost incurred by any given server on input σ is at most $\textsc{Opt}(\sigma)$.*

This result contrasts e.g., with Theorem 5, which shows that for DCA on general tree metrics, one server may pay up to k times the optimal offline cost. The proof follows a similar approach to how marking algorithms are proven to be k-competitive: we argue that between

any two cache misses that cause the same cache position to be reloaded, at least k unique files must be requested. Thus, the sequence containing these two misses must cause at least one miss even for the optimal offline solution. Iterating this argument, no cache position (i.e., server) has more faults than $\textsc{Opt}(\sigma)$ over the full input sequence.

1.1.4.3 Metrics with $k+1$ Points

Finally, we consider metrics with $k+1$ points, which have been widely studied in the literature on the k-server problem. It is known that no deterministic algorithm can achieve a competitive ratio better than k on any metric with $k+1$ points [50]. It is also well known that the Balance algorithm matches this optimal competitive ratio [50]. Recall that this algorithm is inherently fair: it always moves the server whose total cost, after potentially serving the current request, is minimum among all servers. We can therefore conclude the following:

▶ **Proposition 8.** *Balance is a* $(1, \text{diam})$*-fair algorithm for metrics with* $k+1$ *points.*

1.2 Roadmap

The remainder of the paper is organized as follows. Section 2 introduces the k-server problem and formalizes our (α, β)-fairness notion. We explore alternative notions of fairness in a longer version of this paper. Section 3 presents our offline (α, β)-fair algorithm and Section 4 presents our randomized online algorithm. Section 5 evaluates the fairness of deterministic algorithms for particular metric spaces, such as DCA, FIFO, and LRU. We conclude the paper with open questions in Section 6.

2 Problem Setup and (α, β)-Fairness

In this section, we formally define the k-server problem and introduce notation used throughout the paper. We then define the central fairness notion that we consider.

The k-server problem. The k-server problem is defined over a metric space (M, d) with k servers, labeled $1, \ldots, k$, initially located at positions $s_1(0), \ldots, s_k(0) \in M$. At each time step $t = 1, 2, \ldots, T$, a request $\sigma_t \in M$ arrives. The algorithm must choose one server to move to the requested location, incurring a cost equal to the distance moved – i.e., $d(s_i(t-1), \sigma_t)$ if server i is moved and was previously at position $s_i(t-1)$. The total cost is the sum of the costs over the full sequence of requests $\sigma = (\sigma_1, \ldots, \sigma_T)$.

Cost notation. Let A be a k-server algorithm. For a fixed request sequence σ, let $c_i(A, \sigma, t)$ denote the distance that server i moves at time t under algorithm A, and define $c_i(A, \sigma) = \sum_{t=1}^{T} c_i(A, \sigma, t)$ as the total cost incurred by server i. The total cost of the algorithm is $\text{cost}(A, \sigma) = \sum_{i=1}^{k} c_i(A, \sigma)$.

We denote by $\textsc{Opt}(\sigma)$ the minimum total cost achievable by any offline algorithm on input σ. When A and σ are clear from context, we will sometimes drop them as arguments, using $c_i(t)$, c_i, cost, and OPT to refer to the above quantities.

Competitive ratio. An algorithm A is said to be *c-competitive* if there exists a constant $C \geq 0$ (possibly depending on k and on the underlying metric space) such that for all input sequences σ, $\text{cost}(A, \sigma) \leq c \cdot \textsc{Opt}(\sigma) + C$. For randomized algorithms, we consider the *expected* cost and say that A is *c-competitive in expectation* if $\mathbb{E}[\text{cost}(A, \sigma)] \leq c \cdot \textsc{Opt}(\sigma) + C$.

The k-server problem is typically studied in the *online setting*, where an algorithm must make its decision at time t after seeing request σ_t, but before seeing any later requests. It is well known that no deterministic online algorithm can achieve competitive ratio $< k$ for general metrics [44]. This lower bound is matched on tree metrics by the well-known Double Coverage Algorithm (DCA) [27], and up to a constant factor by the Work Function Algorithm (WFA), which is $2k-1$ competitive on general metric spaces [44, 42], and k-competitive on line metrics, star metrics, and metric spaces with $k+2$ points [13]. For randomized algorithms against an oblivious adversary, the best known upper bounds are polylogarithmic in k and n [11, 18]. Recently, the long-standing conjecture that $O(\log k)$ is achievable was disproved; the randomized competitive ratio is now known to be $\Omega(\log^2 k)$ in some metrics [19].

(α, β)-Fairness

We now define our central fairness notion, which requires that no individual server incurs significantly more than its proportional share of the total cost.

▶ **Definition 9** ((α,β)-Fairness). *Let A be a k-server algorithm, and let $w(\sigma)$ be a reference cost baseline associated with request sequence σ. We say that A is* (α,β)-fair with respect to *w if for all input sequences σ and all servers $i \in [k]$,*

$$c_i(A,\sigma) \leq \frac{\alpha \cdot w(\sigma)}{k} + \beta, \tag{1}$$

where $c_i(A,\sigma)$ denotes the total cost incurred by server i under algorithm A on input σ.

Randomized case. *A randomized algorithm A is* (α,β)-fair with probability $1-\delta$ with respect to *w if for every input sequence σ,*

$$\Pr\left[\forall i \in [k],\ c_i(A,\sigma) \leq \frac{\alpha \cdot w(\sigma)}{k} + \beta\right] \geq 1-\delta. \tag{2}$$

When $\delta \leq 1/\mathrm{poly}(k,|\sigma|)$, we simply say that A is (α,β)-fair with high probability*.*

Note that our general definition allows fairness to be expressed relative to a chosen baseline w, such as the optimal offline cost $\textsc{Opt}(\sigma)$ or the algorithm's own total cost, $\mathrm{cost}(A,\sigma)$.

▶ Remark 10 (Fairness v.s. Competitiveness). We remark that when $w(\sigma) = \mathrm{cost}(A,\sigma)$, (α,β)-fairness is not an interesting notion unless the algorithm A is also required to have a bounded competitive ratio. Otherwise, we could have all servers move in synchrony and achieve perfect fairness, but at the cost of competitiveness. Roughly speaking, in the online setting, given an algorithm A that is α competitive, and assuming that each time a server moves to serve request σ_t we move all other servers an equal amount (but do not change their positions), then we have a perfectly fair algorithm that is $\alpha \cdot k$-competitive. Thus, we will always look for fair algorithms that beat this trivial baseline.

3 A Fair Offline Algorithm

We first show that in the offline setting, we can obtain a solution that is both near-optimal in cost and provably fair across servers. In particular, we design a transformation (Algorithm 1) that takes any (possibly unfair) optimal offline solution and converts it into a solution that is $(1+\varepsilon,\beta)$-fair for $\beta = O(\mathrm{diam} \cdot \log k/\varepsilon)$. The conversion increases the total cost by, at most an additive constant. The key idea is to redistribute cost among servers using a sequence of pairwise swaps that gradually balance the load. The transformation is simple, deterministic, and black-box – it does not require any specific structure of the underlying optimal solution.

▶ **Theorem 11** ($(1+\varepsilon,\beta)$-Fair Transformation for Offline k-Server)**.** *Consider any k-server input sequence σ of length T over a metric space of diameter* diam*. Let $\{s_i(t)\}_{i\in[k],t\in[T]}$ be a sequence of server positions that achieves optimal cost on this input sequence.*

For any $\varepsilon > 0$, Algorithm 1 takes $\{s_i(t)\}_{i\in[k],t\in[T]}$ as input and returns a modified sequence of positions $\{\hat{s}_i(t)\}_{i\in[k],t\in[T]}$ that is $(1+\varepsilon,\beta)$-fair with respect to $\textsc{Opt}(\sigma)$, for $\beta = O\left(\frac{\log k\cdot\text{diam}}{\varepsilon}\right)$. Moreover, the total cost of the output solution is $\textsc{Opt}(\sigma) + O\left(\frac{k\log k\cdot\text{diam}}{\varepsilon}\right)$.

Proof Sketch. Algorithm 1 iteratively reduces the maximum server cost in the input solution by performing a series of swaps between the maximum and minimum cost servers. Each swap exchanges the servers' remaining work from a chosen time z onward. Using a balancing lemma (as proved in the full version), we show that there is always a choice of a swap that reduces the cost difference between the swapped servers to at most an additive diam factor. These swaps each add at most $2\cdot\text{diam}$ to the total cost of the new algorithm. Further, note that if the solution is not yet fair, the maximum server load is at least $\approx 1+\varepsilon$ times as large as the minimum. This allows us to argue that after a swap, both servers' costs are less than the previous maximum cost by roughly a $1/(1+O(\varepsilon))$ multiplicative factor (details in the full version). We also observe that the maximum server load always decreases with a swap and that non-swapped servers retain their cost (details in the full version). We proceed by considering any window of $k+1$ swaps. By the pigeonhole principle, at least one server must appear twice as the maximum-cost server in a swap. This server's cost must have decreased by a $\approx 1/(1+O(\varepsilon))$ factor between these two swaps. Thus, we can argue that the maximum cost drops by a factor of at least $(1+\varepsilon)$ every $k+1$ rounds (details in the full version). Repeating this process for $O\left(k\cdot\log_{1+\varepsilon}k\right) = O\left(\frac{k\log k}{\varepsilon}\right)$ swaps to reduce a potential initial multiplicative imbalance for k to $(1+\varepsilon)$ yields a solution satisfying $(1+\varepsilon,\beta)$-fairness. Full details and formal proofs of all lemmas and claims appear in the full version on arXiv. ◀

4 A Fair Randomized Online Algorithm

We next show how to achieve fairness with minimal loss in competitiveness in the online setting with a randomized algorithm. Similar to Algorithm 1, we assume access to some existing online k-server algorithm A with bounded competitive ratio, but potentially with no fairness guarantees. We use a series of random server swaps to balance the servers' loads and make the algorithm fair.

At each swap point, we randomly permute all server identifiers, swap the server positions according to the permutation, and reassign the workloads of the servers going forward according to the permutation as well. Thus, between swaps, by symmetry, the expected cost incurred by all servers is the same. Via concentration inequalities, as long as we swap sufficiently often, we can prove that all servers pay similar costs with high probability, establishing (α,β)-fairness. The key challenge is determining how often to swap – too many swaps will increase the cost of the original algorithm too much, but too few swaps will mean that fairness is not guaranteed with high probability. To balance this trade-off, we introduce a phase-based algorithm (Algorithm 2) that swaps anytime the total cost incurred has increased by a large enough factor. We prove the following:

Algorithm 1 Offline Fair k-server via Pairwise Swapping.

Input: Optimal server positions $\{s_i(t)\}_{i\in[k],t\in[T]}$ with corresponding cost allocation $\{c_i(t)\}_{i\in[k],t\in[T]}$ and parameter ε

Output: Modified sequence of positions $\{\hat{s}_i(t)\}_{i\in[k],t\in[T]}$ satisfying $(1+\varepsilon,\beta)$-fairness.

1 Set round counter $r \leftarrow 0$

2 Initialize $\hat{s}_i^{(0)}(t) \leftarrow s_i(t)$ and $\hat{c}_i^{(0)}(t) \leftarrow c_i(t)$ for all $i \in [k], t \in [T]$

3 $\beta \leftarrow 2\cdot(1+\varepsilon)\cdot diam\cdot(\frac{3}{2}+\log_{\frac{2+2\varepsilon}{2+\varepsilon}} k)$

4 **while** $\max_{i\in[k]} \sum_{t=1}^{T} \hat{c}_i^{(r)}(t) > \frac{(1+\varepsilon)\cdot OPT(\sigma)}{k} + \beta$ **do**

5 Let $H_r \leftarrow \arg\max_{i\in[k]} \sum_{t=1}^{T} \hat{c}_i^{(r)}(t)$; `// Heaviest-loaded server`

6 Let $L_r \leftarrow \arg\min_{i\in[k]} \sum_{t=1}^{T} \hat{c}_i^{(r)}(t)$; `// Lightest-loaded server`

7 Let $z \in [T]$ be the any index such that the difference in the total costs of H_r and L_r after a swap at time z become less than diam ; `// Such a z always exists (check the full paper for more details)`

8 Initiate current cost/locations for all i, t: $\hat{c}_i^{(r+1)}(t) \leftarrow \hat{c}_i^{(r)}(t), \hat{s}_i^{(r+1)}(t) \leftarrow \hat{s}_i^{(r)}(t)$

9 **for** $t > z$ **do**

10 $\hat{s}_{H_r}^{(r+1)}(t) \leftarrow \hat{s}_{L_r}^{(r)}(t)$, $\hat{s}_{L_r}^{(r+1)}(t) \leftarrow \hat{s}_{H_r}^{(r)}(t)$; `// swap positions after z`

11 $\hat{c}_{H_r}^{(r+1)}(t) \leftarrow \hat{c}_{L_r}^{(r)}(t)$, $\hat{c}_{L_r}^{(r+1)}(t) \leftarrow \hat{c}_{H_r}^{(r)}(t)$; `// swap costs after z`

12 $\hat{c}_{H_r}^{(r+1)}(z+1) \leftarrow d(\hat{s}_{H_r}^{(r+1)}(z+1), \hat{s}_{H_r}^{(r+1)}(z))$; `// swap penalty`

13 $\hat{c}_{L_r}^{(r+1)}(z+1) \leftarrow d(\hat{s}_{L_r}^{(r+1)}(z+1), \hat{s}_{L_r}^{(r+1)}(z))$; `// swap penalty`

14 Update round: $r \leftarrow r+1$

15 **return** $\{\hat{c}_i^{(r)}(t)\}$ *and* $\{\hat{s}_i^{(r)}(t)\}$

▶ **Theorem 12** (Randomized Online Fairness Guarantee). *Let A be any online k-server algorithm and let σ be any request sequence. Then, Algorithm 2 with input A and parameter $\gamma > 0$ satisfies: for any $\varepsilon > 0$, with probability at least $1 - k\cdot\exp\left(-\Omega\left(\frac{\varepsilon^2}{\gamma\cdot k}\cdot cost(A,\sigma)^{1/(\gamma+1)}\right)\right)$, each server incurs cost at most*

$$(1+\varepsilon)\cdot\frac{cost(A,\sigma)}{k} + O\left(cost(A,\sigma)^{1/(\gamma+1)}\cdot \mathrm{diam}\right).$$

Further, the total cost across all servers is bounded by $cost(A,\sigma) + O\left(cost(A,\sigma)^{1/(\gamma+1)}\cdot k\cdot \mathrm{diam}\right)$.

In terms of (α,β)-fairness, Theorem 12 implies that Algorithm 2 is (α,β)-fair with respect to $\mathrm{cost}(A,\sigma)$ for $\alpha = 1+\varepsilon$, $\beta = O\left(\mathrm{cost}(A,\sigma)^{1/(\gamma+1)}\cdot\mathrm{diam}\right)$ with high probability. It pays total cost penalty $O\left(\mathrm{cost}(A,\sigma)^{1/(\gamma+1)}\cdot k\cdot\mathrm{diam}\right)$. Both β and this total cost increase grow sublinearly in $\mathrm{cost}(A,\sigma)$, and can be regarded as lower order. The success probability depends on k, γ, and ε, but tends to one as $\mathrm{cost}(A,\sigma)$ tends to infinity.

Proof Sketch. Algorithm 2 divides the input sequence into phases, with each phase ending once the base algorithm A accumulates cost at least $\phi_\ell = \ell^\gamma$ (line 7). At the start of each phase (line 8), the algorithm applies a random permutation to the server identities and exchanges their positions. This guarantees that the expected share of the phase cost is uniformly distributed across servers, regardless of their prior history.

Algorithm 2 Online Fair k-Server via Phased Random Swapping.

Input: Online request sequence $\sigma = (\sigma_1, \sigma_2, \dots)$, k-server algorithm A, initial server configuration $(s_1(0), \dots, s_k(0))$, phase exponent $\gamma > 0$

Initialize: phase number $\ell \leftarrow 1$, phase budget $\phi_\ell \leftarrow \ell^\gamma$, and cumulative cost $U_\ell \leftarrow 0$.
Reposition servers via a random permutation π_ℓ: for all $i \in [k]$, set $s_i(0) \leftarrow s_{\pi_\ell(i)}(0)$ where π_ℓ is chosen uniformly at random.
foreach *request* σ_t **do**
    Serve σ_t using algorithm A on servers labeled before repoistion.$\{s_{\pi_\ell^{-1}(i)}(t)\}$.
    Update $U_\ell \leftarrow U_\ell + cost(A, t)$.
    if $U_\ell \geq \phi_\ell$ **then**
        $\ell \leftarrow \ell + 1$, $\phi_\ell \leftarrow \ell^\gamma$, and $U_\ell \leftarrow 0$.
        Reposition servers via a random permutation π_ℓ: for all $i \in [k]$, set $s_i(t) \leftarrow s_{\pi_\ell(i)}(t)$ where π_ℓ is chosen uniformly at random .

Let m denote the number of phases. Since each phase contributes $\phi_\ell = \ell^\gamma$ cost and $\sum_{\ell=1}^{m} \ell^\gamma = \Theta(m^{\gamma+1})$, we have: $m = \Theta\left(\text{cost}(A)^{1/(\gamma+1)}\right)$. Thus, each server's expected cost is:

$$\mathbb{E}[c_i] = \frac{\text{cost}(A)}{k} + O\left(m \cdot \text{diam}\right) = \frac{\text{cost}(A)}{k} + O\left(\text{cost}(A)^{1/(\gamma+1)} \cdot \text{diam}\right).$$

To show high-probability fairness, we apply a concentration inequality to the random variables equal to each server's cost per phase. Each of these variables is upper-bounded by ϕ_ℓ, with maximum phase cost $\phi_m = m^\gamma = \Theta\left(\text{cost}(A)^{\gamma/(\gamma+1)}\right)$. Thus, we can bound the total variance of these random variables by $\sigma^2 = O\left(\sum \phi_\ell^2 / k\right)$.

Applying Bernstein's inequality with these upper bounds gives that each server's total cost remains $(1+\varepsilon)$ close to its expectation with probability at least $1 - \exp\left(-\Omega\left(\frac{\varepsilon^2}{\gamma \cdot k} \cdot \text{cost}(A, \sigma)^{1/(\gamma+1)}\right)\right)$. A union bound over all k servers gives the final high-probability guarantee: with probability at least $1 - k \cdot \exp\left(-\frac{\varepsilon^2}{\gamma \cdot k} \cdot \Omega\left(\text{cost}(A, \sigma)^{1/(\gamma+1)}\right)\right)$, for all $i \in [k]$,

$$c_i \leq (1+\varepsilon) \cdot \frac{\text{cost}(A)}{k} + O\left(\text{cost}(A)^{1/(\gamma+1)} \cdot \text{diam}\right).$$

Full details of the proof appear in the full version of the paper. ◀

▶ Remark 13 (Extension to ID-Oblivious Adversary). We remark that Theorem 12 also holds when σ is generated by an *ID-oblivious* adaptive adversary – i.e., an adversary that can pick σ_t based on the positions of the servers at rounds $t' < t$, but not on their identities. At the start of each phase, Algorithm 2 applies a random permutation to the server IDs without changing their overall set of positions. As a result, an ID-oblivious adversary observes the same sequence of positions under Algorithm 2 as under the base algorithm A, and thus, the analysis for the oblivious adversary case carries through unchanged.

5 Towards Deterministic Online Fairness

A central open question left by our randomized fairness result is whether fairness can be achieved by *deterministic* online algorithms, or against fully adaptive adversaries that observe server identities. In this section we study the fairness properties of classical deterministic

k-server algorithms on specific metric classes. Our focus is on the Double Coverage Algorithm (DCA) on tree metrics and on deterministic paging algorithms (FIFO, LRU, and marking algorithms) on uniform metrics. These algorithms are well-studied from a competitive-analysis perspective, and analyzing their fairness properties gives insight into what kinds of fairness may be achievable deterministically.

5.1 Fairness of the DCA on Special Cases

The Double Coverage Algorithm (DCA) works on a tree metric and moves all "unblocked" servers that do not have another server on their path to the request point, each at a uniform speed towards the request until the request is served.

We first show that, owing to the inherent fairness of moving all unblocked servers at the same time, the algorithm is fair on line metrics for any k and general tree metrics for $k = 2$. However, we also show a strong negative result: DCA can be highly unfair on general tree metrics for general k.

We begin with our positive result showing that DCA is $(1, O(k \cdot \text{diam}))$-fair with respect to Opt on line metrics for any value of k.

▶ **Theorem 14.** *Let σ be any request sequence over a line metric with diameter* diam*, and let c_i denote the total cost incurred by server i when serving σ using DCA. We have* $\max_{i,j\in[k]} |c_i - c_j| = O(k \cdot \text{diam})$*. Thus, DCA is* $(1, O(k \cdot \text{diam}))$*-fair on line metrics.*

Proof Sketch. Since DCA never moves servers past each other on the line, we can fix a unique labeling of the servers as $1, \ldots, k$ based on their initial positions from left to right. The double coverage approach ensures that, for each request, the two closest servers move toward the request at equal speed until one of them reaches it. Therefore, every time server i moves to the right, it is matched by an equal leftward movement by server $i + 1$, and vice versa. This gives the recurrence:

$$R_i = L_{i+1}, \quad \text{where } R_i \text{ is server } i\text{'s rightward movement and } L_i \text{ is its leftward movement.}$$

The total cost for server i is $c_i = R_i + L_i$. Letting $d_i = R_i - L_i$ be the net movement of server i,

$$c_{i+1} - c_i = L_{i+1} + R_{i+1} - (L_i + R_i) = R_{i+1} - L_i = d_{i+1} + d_i.$$

Since all movement is along a line of diameter diam, we have $|d_i| \leq \text{diam}$ for all i. Therefore, the cost difference between adjacent servers is bounded by:

$$|c_{i+1} - c_i| \leq |d_i| + |d_{i+1}| \leq 2 \cdot \text{diam}.$$

By summing the pairwise differences across all $k - 1$ server pairs, we conclude that:

$$\text{for any } i, j \in [k] \ |c_i - c_j| \leq 2(k-1) \cdot \text{diam},$$

as claimed. See the full version for complete proofs. ◀

We next show that DCA is (α, β)-fair for $\alpha = O(1)$ and $\beta = O(\text{diam})$ for any tree metric in the special case of $k = 2$.

▶ **Theorem 15.** *DCA is* $(1.5, 1.75 \cdot \text{diam})$*-fair for any tree metric when $k = 2$.*

Proof. Fix an input σ. For each server, let $con_i(\sigma)$ denote the total distance moved by i while "converging" towards the other server (moving i closer to the other server). Similarly, let $div_i(\sigma)$ denote the total distance moved by i while "diverging" away from the other server. For example, when a request is made on the shortest path between i and i', the distance moved by any of the two servers contributes to $con_i(\sigma)$. When i moves away from the other server (and the other server does not move), the distance moved by i contributes to $div_i(\sigma)$.

Intuitively, for two servers the divergence and convergence terms directly reflect how their mutual distance changes over time: when they diverge the servers move farther apart, and when they converge they move closer together. Since their separation can never exceed the diameter of the metric (nor become negative), the total amount by which divergence can exceed convergence, or vice versa, is necessarily bounded by diam.

Formally, for any input sequence σ, with $k = 2$ and any server i, the following inequality holds (see more details on the proof in the full version of the paper).

$$-\text{diam} \leq div_i(\sigma) + div_{i'}(\sigma) - 2con_i(\sigma) \leq \text{diam}. \tag{3}$$

Since each unit move of a server i either increases or decreases its distance to the other server, we can write: $c_i = con_i(\sigma) + div_i(\sigma)$. Furthermore, by the definition of DCA and the nature of converging moves, we have $con_i(\sigma) = con_{i'}(\sigma)$ since both servers move towards each other at the same pace during converging moves (this argument fails for $k > 2$, when one server can be blocked from moving by some third server). Therefore, using the inequalities in (3) we can conclude that:

$$\begin{aligned} c_i = con_i(\sigma) + div_i(\sigma) &\leq 3 \cdot con_i(\sigma) - div_{i'}(\sigma) + \text{diam} \\ &\leq 3 \cdot con_i(\sigma) + \text{diam} \\ &= 1.5 \cdot con_i(\sigma) + 1.5 \cdot con_i(\sigma) + \text{diam} \\ &\leq 1.5 \cdot con_i(\sigma) + 0.75 \cdot (div_i(\sigma) + div_{i'}(\sigma)) + 1.75 \cdot \text{diam}. \end{aligned}$$

Substituting the definitions of c_i and $c_{i'}$ and the fact that $con_i(\sigma) = con_{i'}(\sigma)$ into the previous bound yields:

$$c_i \leq 0.75 \cdot c_i + 0.75 \cdot c_{i'} + 1.75 \cdot \text{diam} \leq 1.5 \cdot \frac{cost(DCA, \sigma)}{2} + 1.75 \cdot \text{diam}. \qquad \blacktriangleleft$$

5.2 Unfairness of DCA on General Tree Metrices

We next show that, unfortunately, the positive results of Theorems 14 and 15 cannot be extended to the full range of settings where DCA is k-competitive. In particular, DCA can fail to satisfying any reasonable notion of fairness on general tree metrics for general k, assigning the majoring of work – in fact, $\Omega(k \cdot \textsc{Opt}(\sigma))$ work – to just a single server.

▶ **Theorem 16.** *There exists a tree metric and a request sequence σ on that metric such that, when the Double Coverage Algorithm (DCA) is run on σ, some server incurs cost $\Omega(k \cdot \textsc{Opt}(\sigma))$, where $\textsc{Opt}(\sigma)$ denotes the cost of the offline optimal solution.*

Proof Sketch. We construct a tree where $k + 1$ major nodes form a path (the "spine"), and each major node has an attached minor leaf at distance ε, for arbitrarily small ε. Label the spine nodes $\{1, 2, \ldots, k + 1\}$ and place servers, labeled $s_1, s_2, \ldots, s_k$, initially at positions $1, 3, 4, \ldots, k + 1$ respectively, skipping 2 to create a gap. The goal is to force s_1 to move from the start to the end of the spine, incurring cost $\Omega(k)$, while all other servers (and the offline optimal solution) incur cost just $O(1)$. The hard request sequence proceeds in repeated steps:

- First, issue a request at position 2, causing both servers s_1 and s_2 at positions 1 and 3 to move toward it, perturbing the distances by an arbitrarily small amount so that s_2 arrives first and actually serves the request.
- Next, request the minor leaf connected to 2. This forces s_2 to move off the spine, freeing up position 2. s_1 is "blocked" by s_2 and does not move.
- Now, issue a request at position 3; s_1 at position 2 and s_3 at position 4 move toward it, and again the other server (now s_3) serves the request and is sent off the spine.

This pattern repeats: server s_1 slowly moves right, passing each server one by one while the others are diverted off the main path. After k steps, s_1 incurs cost $\Omega(k)$ while all other servers incur cost just $O(1+\varepsilon)$. See Figure 1 for an illustration.

Further, the offline optimum can serve all spine requests using just one server move of length 1 (s_1 moves to position 2 in the first step), and handle minor leaf requests with ε-moves. Thus, $\mathrm{OPT}(\sigma) = O(1+\varepsilon k)$, and so, setting ε arbitrarily small, s_1 pays costs $\Omega(k \cdot \mathrm{OPT}(\sigma))$, as desired. See the full version for full details, and an argument that this hard sequence can be repeated to apply to settings where $\textsc{Opt}(\sigma)$ is arbitrarily large in comparison to k and the metric space diameter. ◀

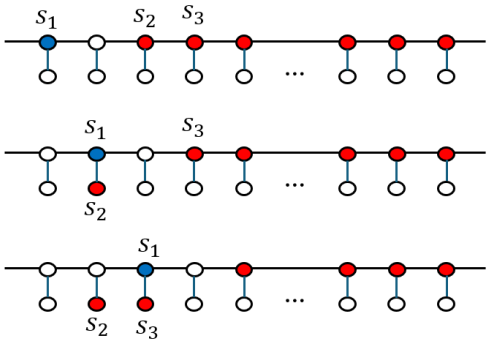

Figure 1 Hard instance for Theorem 12. *Top:* Initialization of server positions on the spine. *Middle:* Requests 1–2 move s_1 one step to the right and push s_2 off the spine via a minor leaf. *Bottom:* Requests 3–4 repeat the process with s_3, and so on, forcing s_1 to traverse the entire spine while all other servers incur $O(1)$ additional cost.

5.3 Uniform Metrics and Paging

Finally, we consider the fairness of deterministic k-server algorithms on uniform metrics, where we scale all distances to be 1 for simplicity. This setting is equivalent to the classical paging problem, where the k servers correspond to the k memory cells of a cache. A *fault* occurs when a requested page is not currently in the cache, equivalently, when a request is made to a node (page) without a server located on it. The objective of minimizing total server movements thus becomes equivalent to minimizing the number of faults.

Given this equivalence, we adopt the terminology of paging. To simplify the discussion, we restrict our attention to *lazy paging algorithms*, which evict a page only when a fault occurs and the cache is full. Non-lazy algorithms, such as Flush-When-Full, are also relevant in the k-server context,

We start with a strong positive result concerning the First-In-First-Out (FIFO) algorithm, which evicts the page that has been in the cache the longest. It is well known that FIFO is k-competitive [57]. We now show that FIFO also guarantees the strongest notion of fairness.

▶ **Theorem 17.** *FIFO is* $(1, 1)$*-fair on uniform metrics.*

Proof. Let A and B be two servers (memory cells), and let a_i, b_i denote the indices of the ith faults in the input at A and B, respectively. Suppose the first fault at A occurs before the first fault at B. By the FIFO rule, for any i, we have

$$a_i < b_i < a_{i+1} < b_{i+1}.$$

This implies that the number of faults at A exceeds that of B by at most one. Applying this logic to all pairs of servers yields the $(1, 1)$-fairness guarantee. ◀

Thus, FIFO is not only optimally competitive (among deterministic algorithms) but also optimally fair. One may ask whether other paging algorithms, such as Least-Recently-Used (LRU), which is also k-competitive [57], share this fairness property. The following theorem shows that they do not. Recall that LRU evicts the page in the cache that has been least recently used.

▶ **Theorem 18.** *There exists a request sequence* σ *on a uniform metric such that under LRU, one server incurs cost* $\Omega(\text{cost}(\text{LRU}, \sigma))$ *while all other servers incur only* $O(1)$*. Consequently, LRU does not satisfy* (α, β)*-fairness with respect to its own total cost for any* $\alpha = o(k)$ *and any* $\beta = o(\text{cost}(\text{LRU}, \sigma))$*.*

Proof. We construct an input where a single server incurs (almost) all the cost incurred by LRU. Consider the input sequence

$$(\alpha, 1, 2, \ldots, k-1, \beta, 1, 2, \ldots, k-1)^m.$$

By the LRU rule, pages $1, 2, \ldots, k-1$ remain in the cache throughout at positions $2, \ldots, k$ (i.e., covered by servers $s_2, \ldots, s_k$). The first cache location (server s_1) experiences faults at requests to β and after repeating the sequence at requests to α. Thus, all of the algorithm's cost beyond the initial $k-1$ "cold misses" (the faults before the cache becomes full) is borne by this single server. ◀

Despite the above negative result, one can show that LRU, and more generally all *marking algorithms*, guarantee a weaker but still meaningful fairness property: each server incurs a cost no more than the optimal offline solution. That is, they are $(1, 0)$-fair with respect to $k \cdot \textsc{Opt}(\sigma)$, the potential highest cost paid by the algorithm on a worst-case input where it is k-competitive. We explore this notion further in the full version of paper where we call it *acceptable fairness*.

A marking algorithm works as follows On a hit, the requested page is simply marked. On a fault, if all pages are already marked, all marks are cleared; then an unmarked page is evicted and the requested page is inserted and marked. LRU fits this framework because its notion of "least-recently-used" exactly corresponds to the unmarked pages: a page loses its mark precisely when it has not been requested among the most recent k distinct requests, matching the classical marking definition.

▶ **Theorem 19.** *For any marking algorithm on a uniform metric, the cost incurred by each server on input sequence* σ *is at most* $\text{OPT}(\sigma)$*.*

Proof. We use the standard phase-partitioning technique. Fixing the input sequence σ, define the first phase as the maximal prefix of σ containing k distinct pages. Define subsequent phases recursively on the remainder of the sequence. Let m_σ denote the number of phases.

Then $\mathrm{OPT}(\sigma) \geq m_\sigma$, since each phase introduces $k+1$ distinct pages and thus forces at least one fault for OPT. On the other hand, in each phase, any server of a marking algorithm incurs at most one fault: after its first fault in a phase, its page is marked and remains protected for the rest of the phase. Thus, each server pays cost at most $\textsc{Opt}(\sigma)$. ◀

6 Conclusion

In this work, we have introduced a formal definition of (α, β)-fairness for the k-server problem, and shown that it is achievable without significantly compromising competitiveness in both the offline setting and the online setting with a randomized algorithm against an oblivious adversary.

The central open question left by our work is whether fairness can be achieved on general metrics via a deterministic online algorithm with a competitive ratio of $O(k)$. Or with a randomized algorithm against an adaptive adversary. Even a partial resolution of this question, e.g., on just tree metrics or with an algorithm that is just $o(k^2)$ competitive rather than $O(k)$-competitive, would be interesting. In the full version of the paper, we discuss some reductions between different notions of fairness that may be helpful in tackling this open problem.

Our results naturally extend to the *egalitarian cost* model, where the cost of an algorithm is defined as the *maximum* distance traveled by any server, instead of the *sum* of distances traveled (the utilitarian model). In this setting, the optimal egalitarian cost satisfies $\textsc{Opt}_e = \frac{\textsc{Opt}}{k}$.

Let A be an (a, b)-fair algorithm that is c-competitive under the usual (utilitarian) cost model. For its egalitarian cost, we can write $A_e \leq \frac{a}{k} A + b$. Since $A \leq c \cdot \textsc{Opt}$, it follows that

$$A_e \leq \frac{a}{k} \cdot (c \cdot \textsc{Opt}) + b = ac \cdot \textsc{Opt}_e + b.$$

Therefore, an (a, b)-fair algorithm that is c-competitive under the utilitarian cost model becomes an (ac)-competitive algorithm in the egalitarian model.

A final remark concerns the appearance of the parameter diam in some of our additive fairness bounds. In our analysis, diam denotes the maximum separation ever created between any two servers during the algorithm's execution – not the diameter of the metric space or of the request set. Under the standard assumption that all servers start at the same location, this quantity is always bounded by the algorithm's own movement cost. Thus the dependence on diam is not fundamental: it can be removed entirely by absorbing diam into the multiplicative term at the expense of a slightly larger constant factor. We present the cleaner additive–multiplicative form in the main statements, but both formulations could be equivalent up to constant factors.

Beyond the classic k-server problem, it would also be interesting to consider (α, β)-fairness for other important generalizations, e.g., metrical task systems [17].

References

1 Sara Ahmadian, Alessandro Epasto, Ravi Kumar, and Mohammad Mahdian. Clustering without over-representation. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2019)*, pages 267–275, 2019. doi:10.1145/3292500.3330987.

2 Elad Aigner-Horev and Erel Segal-Halevi. Envy-free matchings in bipartite graphs and their applications to fair division. *Information Sciences*, 587:164–187, 2022. doi:10.1016/J.INS.2021.11.059.

3 Miklós Ajtai, James Aspnes, Moni Naor, Yuval Rabani, Leonard J. Schulman, and Orli Waarts. Fairness in scheduling. *Journal of Algorithms*, 29(2):306–357, 1998. doi:10.1006/JAGM.1998.0953.

4 M. Ananthanarayanan et al. The distributed k-server problem: A competitive distributed translator for k-server algorithms. *Journal of Algorithms*, 23(2):241–264, 1997. doi:10.1006/JAGM.1996.0826.

5 Antonios Antoniadis, Christian Coester, Marek Elias, Adam Polak, and Bertrand Simon. Mixing predictions for online metric algorithms. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, volume 202 of *Proceedings of Machine Learning Research*, pages 969–983, 2023. URL: https://proceedings.mlr.press/v202/antoniadis23b.html.

6 Antonios Antoniadis, Christian Coester, Marek Elias, Adam Polak, and Bertrand Simon. Online metric algorithms with untrusted predictions. *ACM Transactions on Algorithms*, 19(2):19:1–19:34, 2023. doi:10.1145/3582689.

7 Nikhil Ayyadevara and Ashish Chiplunkar. The randomized competitive ratio of weighted k-server is at least exponential. In *Proceedings of the 29th European Symposium on Algorithms (ESA)*, volume 204 of *LIPIcs*, pages 9:1–9:11, 2021. doi:10.4230/LIPIcs.ESA.2021.9.

8 Nikhil Ayyadevara, Ashish Chiplunkar, and Amatya Sharma. A decomposition approach to the weighted k-server problem. In *Proceedings of the 44th International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 323 of *LIPIcs*, 2024. doi:10.4230/LIPIcs.FSTTCS.2024.6.

9 Haris Aziz, Rupert Freeman, Nisarg Shah, and Rohit Vaish. Best of both worlds: Ex ante and ex post fairness in resource allocation. *Operations Research*, 72(4):1674–1688, 2024. doi:10.1287/OPRE.2022.2432.

10 Eric Balkanski, Will Ma, and Andreas Maggiori. Fair secretaries with unfair predictions. In *Advances in Neural Information Processing Systems 37 (NeurIPS)*, 2024.

11 Nikhil Bansal, Niv Buchbinder, Aleksander Madry, and Joseph Naor. A polylogarithmic-competitive algorithm for the k-server problem. *Journal of the ACM*, 62(5):1–49, 2015. doi:10.1145/2783434.

12 Nikhil Bansal, Marek Elias, Grigorios Koumoutsos, and Jesper Nederlof. Competitive algorithms for generalized k-server in uniform metrics. *ACM Transactions on Algorithms*, 19(1):8:1–8:15, 2023. doi:10.1145/3568677.

13 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, 2004. doi:10.1016/J.TCS.2004.06.001.

14 Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královic, and Richard Královic. On the advice complexity of the k-server problem. *Journal of Computer and System Sciences*, 86:159–170, 2017. doi:10.1016/J.JCSS.2017.01.001.

15 Vincenzo Bonifaci and Leen Stougie. Online k-server routing problems. In *Proceedings of the 4th International Conference on Approximation and Online Algorithms (WAOA 2006)*, pages 83–94, 2006. doi:10.1007/11970125_7.

16 Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998. Chapter 10.

17 Allan Borodin, Nathan Linial, and Michael E. Saks. An optimal online algorithm for metrical task systems. *Journal of the ACM*, 39(4):745–763, 1992. doi:10.1145/146585.146588.

18 Sebastien Bubeck, Michael B. Cohen, Yin Tat Lee, James R. Lee, and Aleksander Madry. The k-server problem via multiscale entropic regularization. In *Proceedings of the 50th Annual ACM Symposium on Theory of Computing (STOC)*, pages 3–16, 2018.

19 Sébastien Bubeck, Yuval Rabani, and Sergei Vassilvitskii. The randomized k-server conjecture is false. *Journal of the ACM*, 70(1):1–45, 2023.

20 Niv Buchbinder, Christian Coester, and Joseph Naor. Online k-taxi via double coverage and time-reverse primal-dual. *Mathematical Programming*, 197(2):499–527, 2022. doi:10.1007/S10107-022-01815-6.

21 Jannik Castenow, Björn Feldkord, Till Knollmann, Manuel Malatyali, and Friedhelm Meyer auf der Heide. The k-server with preferences problem. In *Proceedings of the 34th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 345–356, 2022. doi:10.1145/3490148.3538595.

22 Ashish Chiplunkar, Monika Henzinger, Sagar Sudhir Kale, and Maximilian Vötsch. Online min-max paging. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1545–1565, 2023. doi:10.1137/1.9781611977554.CH57.

23 Ashish Chiplunkar and Sundar Vishwanathan. Randomized memoryless algorithms for the weighted and the generalized k-server problems. *ACM Transactions on Algorithms*, 16(1):14:1–14:28, 2020. doi:10.1145/3365002.

24 Tze-Yang Poon Christian Coester. Online 3-taxi on general metrics. In *Proceedings of the 37th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2026.

25 Nicolas Christianson, Junxuan Shen, and Adam Wierman. Optimal robustness-consistency tradeoffs for learning-augmented metrical task systems. In *Proceedings of the 26th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 206, pages 9377–9399, 2023. URL: https://proceedings.mlr.press/v206/christianson23a.html.

26 Marek Chrobak and Lawrence L. Larmore. An optimal online algorithm for k-servers on trees. *SIAM Journal on Computing*, 20(1):144–148, 1991. doi:10.1137/0220008.

27 Marek Chrobak and Lawrence L. Larmore. The server problem and online games. In *Online Algorithms: The State of the Art*, volume 1442 of *Lecture Notes in Computer Science*, pages 64–96. 1998.

28 Christian Coester and Elias Koutsoupias. The online k-taxi problem. In *Proceedings of the 51st Annual ACM Symposium on Theory of Computing (STOC)*, pages 1136–1147, 2019. doi:10.1145/3313276.3316370.

29 Christina Delimitrou and Christos Kozyrakis. Quasar: Resource-efficient and qos-aware cluster management. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 127–144, 2014. doi:10.1145/2541940.2541941.

30 John Dickerson, Seyed A. Esmaeili, Jamie Morgenstern, and Claire Jie Zhang. Fair clustering: Critique, caveats, and future directions. *arXiv preprint arXiv:2406.15960*, 2024. doi:10.48550/arXiv.2406.15960.

31 Yuval Emek, Pierre Fraigniaud, Amos Korman, and Adi Rosén. Online computation with advice. *Theoretical Computer Science*, 412(24):2642–2656, 2011. doi:10.1016/J.TCS.2010.08.007.

32 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. doi:10.1016/0196-6774(91)90041-V.

33 Amos Fiat, Yuval Rabani, and Yiftach Ravid. Competitive k-server algorithms. *Journal of Computer and System Sciences*, 48(3):410–428, 1994. doi:10.1016/S0022-0000(05)80060-1.

34 Till Fluschnik, Piotr Skowron, Mervin Triphaus, and Kai Wilker. Fair knapsack. In *Proceedings of the AAAI Conference on Artificial (AAAI)*, pages 1941–1948, 2019. doi:10.1609/AAAI.V33I01.33011941.

35 Sushmita Gupta, Shahin Kamali, and Alejandro López-Ortiz. On the advice complexity of the k-server problem under sparse metrics. *Theory of Computing Systems*, 59(3):476–499, 2016. doi:10.1007/S00224-015-9649-X.

36 MohammadTaghi Hajiaghayi, Shayan Chashm Jahan, Mohammad Sharifi, Suho Shin, and Max Springer. Fairness and efficiency in online class matching. In *Advances in Neural Information Processing Systems 37 (NeurIPS)*, 2024.

37 Hadi Hosseini, Zhiyi Huang, Ayumi Igarashi, and Nisarg Shah. Class fairness in online matching. *Artificial Intelligence*, 2023. A preliminary version appeared in AAAI 2023.

38 Sven Jäger, Alexander Lindermayr, and Nicole Megow. The power of proportional fairness for non-clairvoyant scheduling under polyhedral constraints. In *Proceedings of the 36th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3901–3930, 2025. doi:10.1137/1.9781611978322.132.

39 Anna R. Karlin, Mark S. Manasse, Lyle A. McGeoch, and Susan Owicki. Competitive snoopy caching. *Algorithmica*, 3:77–119, 1988. doi:10.1007/BF01762111.

40 Marina Knittel, Max Springer, John P. Dickerson, and MohammadTaghi Hajiaghayi. Fair, polylog-approximate low-cost hierarchical clustering. In *Advances in Neural Information Processing Systems 36 (NeurIPS)*, 2023.

41 Mario Köppen, Rodrigo Verschae, and Masato Tsuru. Masp: A multi-attribute secretary problem approach to multi-objective optimization with a fair decision maker. In *Proceedings of the 6th International Conference on Genetic and Evolutionary Computing (ICGEC 2012)*, pages 324–327, 2012. doi:10.1109/ICGEC.2012.108.

42 Elias Koutsoupias. *On-line algorithms and the k-server conjecture*. PhD thesis, University of California, San Diego, June 1994.

43 Elias Koutsoupias. The k-server problem. *Theoretical Computer Science*, 2009:105–118, 2009. doi:10.1016/J.COSREV.2009.04.002.

44 Elias Koutsoupias and Christos H. Papadimitriou. On the k-server conjecture. *Journal of the ACM*, 42(5):971–983, 1995. doi:10.1145/210118.210128.

45 Elias Koutsoupias and David Scot Taylor. The cnn problem and other k-server variants. *Theoretical Computer Science*, 324(2–3):347–359, 2004. doi:10.1016/J.TCS.2004.06.002.

46 Predrag Krnetic, Darya Melnyk, Yuyi Wang, and Roger Wattenhofer. The k-server problem with delays on the uniform metric space. In *Proceedings of the 31st International Symposium on Algorithms and Computation (ISAAC)*, volume 181 of *LIPIcs*, pages 61:1–61:13, 2020. doi:10.4230/LIPIcs.ISAAC.2020.61.

47 Adam Lechowicz, Rik Sengupta, Bo Sun, Shahin Kamali, and Mohammad Hajiesmaili. Time fairness in online knapsack problems. In *Proceedings of the 12th International Conference on Learning Representations (ICLR)*, 2024.

48 Alexander Lindermayr, Nicole Megow, and Bertrand Simon. Double coverage with machine-learned advice. In *Proceedings of the 13th Conference on Innovations in Theoretical Computer Science (ITCS)*, volume 215, pages 99:1–99:18, 2022. doi:10.4230/LIPIcs.ITCS.2022.99.

49 Yury Makarychev and Ali Vakilian. Approximation algorithms for socially fair clustering. In *Proceedings of the 34th Annual Conference on Computational Learning Theory (COLT)*, volume 134 of *Proceedings of Machine Learning Research*, 2021. URL: http://proceedings.mlr.press/v134/makarychev21a.html.

50 Mark S. Manasse, Lyle A. McGeoch, and Daniel D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11(2):208–230, 1990. doi:10.1016/0196-6774(90)90003-W.

51 Antonio Martinez-Sykora, Fraser N. McLeod, Tom J. Cherrett, and Adrian Friday. Exploring fairness in food delivery routing and scheduling problems. *Expert Systems with Applications*, 240:122488, 2024. doi:10.1016/J.ESWA.2023.122488.

52 April Niu, Agnes Totschnig, and Adrian Vetta. Fair algorithm design: Fair and efficacious machine scheduling. In *Proceedings of the 16th International Symposium on Algorithmic Game Theory (SAGT 2023)*, volume 14238 of *Lecture Notes in Computer Science*, pages 239–256, 2023. doi:10.1007/978-3-031-43254-5_14.

53 Yi Cheng Ong, Nicos Protopapas, Vahid Yazdanpanah, Enrico H. Gerding, and Sebastian Stein. Fair and efficient ride-scheduling: A preference-driven approach. *Journal of Simulation*, 18(6):940–956, 2024. doi:10.1080/17477778.2024.2334826.

54 Deval Patel, Arindam Khan, and Anand Louis. Group fairness for knapsack problems. In *Proceedings of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021)*, pages 1001–1009, 2021. doi:10.5555/3463952.3464069.

55 Marc P. Renault and Adi Rosén. On online algorithms with advice for the k-server problem. *Theory of Computing Systems*, 56(1):3–21, 2015. doi:10.1007/S00224-012-9434-Z.

56 Daman Deep Singh, Amit Kumar, and Abhijnan Chakraborty. Towards fairness in online service with k servers and its application on fair food delivery. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 5856–5864, 2023.

57 Daniel Dominic Sleator and Robert Endre Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985. doi:10.1145/2786.2793.

58 Robust Fair Clustering with Noisy Memberships. Optimal robustness-consistency tradeoffs for learning-augmented metrical task systems. In *Proceedings of the 28th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2025.