A Hitting Set Relaxation for k-Server and an Extension to Time-Windows

Anupam Gupta
Computer Science Department
Carnegie Mellon University
anupamg@cs.cmu.edu

Amit Kumar
Dept. of Computer Science and Engg.
IIT Delhi
amitk@cse.iitd.ac.in

Debmalya Panigrahi
Department of Computer Science
Duke University
debmalya@cs.duke.edu

2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS) | 978-1-6654-2055-6/22/$31.00 ©2022 IEEE | DOI: 10.1109/FOCS52979.2021.00057

***Abstract*—We study the k-server problem with time-windows. In this problem, each request i arrives at some point v_i of an n-point metric space at time b_i and comes with a deadline e_i. One of the k servers must be moved to v_i at some time in the interval $[b_i, e_i]$ to satisfy this request. We give an online algorithm for this problem with a competitive ratio of $\text{poly}\log(n, \Delta)$, where Δ is the aspect ratio of the metric space. Prior to our work, the best competitive ratio known for this problem was $O(k \,\text{poly}\log(n))$ given by Azar et al. (STOC 2017).**

Our algorithm is based on a new covering linear program relaxation for k-server on HSTs. This LP naturally corresponds to the min-cost flow formulation of k-server, and easily extends to the case of time-windows. We give an online algorithm for obtaining a feasible fractional solution for this LP, and a primal dual analysis framework for accounting the cost of the solution. Together, they yield a new k-server algorithm with poly-logarithmic competitive ratio, and extend to the time-windows case as well. Our principal technical contribution lies in thinking of the covering LP as yielding a *truncated* covering LP at each internal node of the tree, which allows us to keep account of server movements across subtrees. We hope that this LP relaxation and the algorithm/analysis will be a useful tool for addressing k-server and related problems.

I. Introduction

The k-Server problem, originally proposed by Manasse, McGeoch, and Sleator [17], is perhaps the most well-studied problem in online algorithms. Given an n-point metric space and an online sequence of requests at various locations, the goal is to coordinate k servers so that each request is served by moving a server to the corresponding location. The objective of the algorithm is to minimize the total distance moved by the servers (i.e., the movement cost). It has been known for more than two decades that the best deterministic competitive ratio for this problem is between k [17] and $2k-1$ [16], although determining the exact constant remains open. For randomized algorithms, even obtaining a tight asymptotic bound is still open, although there has been tremendous progress in the last decade culminating in a poly-logarithmic competitive ratio [4], [8], [9].

We focus on the *k-server with time-windows* (k-ServerTW) problem, where each request arrives at a location in the metric space at some time b with a deadline $e \geq b$. The algorithm must satisfy the request by moving a server to that location at any point during this time interval $[b, e]$. (If $e = b$ for every request, this reduces to k-Server.) The techniques used to solve the standard k-Server problem seem to break down in the case of time-windows. Nonetheless, an $O(k \,\text{poly}\log n)$-competitive deterministic algorithm was given for the case where the underlying metric space is a tree [2]; this gives an $O(k \,\text{poly}\log n)$-competitive *randomized* algorithm for arbitrary metric spaces using metric embedding results.

For the special case of k-ServerTW on an unweighted star, [2] obtained competitive ratios of $O(k)$ and $O(\log k)$ using deterministic and randomized algorithms respectively. The deterministic competitive ratio of $O(k)$ extended to weighted stars as well (which is same as Weighted Paging), but a randomized (poly)-logarithmic bound already turned out to be more challenging; a bound of $\text{poly}\log(n)$ was obtained only recently [14]. This raises the natural question: can we obtain a poly-logarithmic competitive ratio for the k-ServerTW problem on general metric spaces? The technical gap between Weighted Paging and k-Server is substantial and bridging this gap for randomized algorithms was the preeminent challenge in online algorithms for some time. Moreover, the approaches eventually used to bridge this gap do not seem to extend to time-windows, so we have to devise a new algorithm for k-Server as well in solving k-ServerTW. We successfully answer this question.

Theorem I.1 (Randomized Algorithm)**.** *There is an $O(\text{poly}\log(n\Delta))$-competitive randomized algorithm for k-ServerTW on any n-point metric space with aspect ratio Δ.*

Theorem I.1 follows from our main technical result Theorem I.2 below. Indeed, since any n-point metric space can be probabilistically approximated using λ-HSTs with height $H = O(\log_\lambda \Delta)$ and expected stretch $O(\lambda \log_\lambda n)$ [11], we can set $\lambda = O(\log \Delta)$ and use the rounding algorithm from [4], [8] to complete the reduction.

Theorem I.2 (Fractional Algorithm for HSTs)**.** *Fix $\delta' \leq 1/n^2$. There is an $O(\text{poly}(H, \lambda, \log n))$-competitive fractional*

AG supported in part by NSF awards CCF-1907820, CCF-1955785, and CCF-2006953. DP supported in part by NSF awards CCF-1750140 (CAREER) and CCF-1955703, and ARO award W911NF2110230.

2575-8454/21/$31.00 ©2021 IEEE
DOI 10.1109/FOCS52979.2021.00057

algorithm for k-ServerTW using $\frac{k}{1-\delta'}$ servers such that for any instance on a λ-HST with height H and $\lambda \geq 10H$, and for each request interval $R = [b, e]$ at some leaf ℓ in this instance, there is a time in this interval at which the number of servers at ℓ is at least 1.

Apart from the result itself, a key contribution of our paper is an approach to solve a new covering linear program for k-Server. Previous results in k-Server (e.g., [8]) used a very different LP relaxation, and it remains unclear how to extend that relaxation to the case of time-windows. The covering LP in this paper is easy to describe and flexible. It is quite natural, following from the min-cost LP formulation for k-Server (see the full version for this connection).

A. Our Techniques

The basis of our approach is a restatement of k-Server (and thence k-ServerTW) as a covering LP without box constraints. This LP has variables $x(v, t)$ that try to capture the event that a server leaves the subtree rooted at v at some time t. There are several complications with this LP: apart from having an exponential number of constraints, it is too unstructured to directly tell us how to move servers. E.g., the variable for a node may increase but that for its parent or child edges may not. Or the online LP solver may increase variables for timesteps in the past, which then need to be translated to server movements at the present timestep.

Our principal technical contribution is to view this new LP as yielding "truncated" LPs, one for each internal node v of the tree. This "local" LP for v restricts the original LP to inequalities and variables corresponding to the subtree below v. This truncation is contingent on prior decisions taken by the algorithm, and so the constraints obtained may not be implied by those for the original LP. However, we show how the primal—and just as importantly—the dual solutions to local LPs can be composed to give primal/dual solutions to the original LP. These are then crucial for our accounting.

The algorithm for k-Server proceeds as follows. Suppose a request comes at some leaf ℓ, and suppose ℓ has less than $1 - \delta'$ amounts of server at it (else we deem it satisfied):

(i) Consider a vertex v_i on the *backbone* (i.e., the path $\ell = v_0, v_1, \ldots, v_H = \mathbb{r}$ from leaf ℓ to the root $\mathbb{r}$). If v_i has off-backbone children whose descendant leaves contain non-trivial amounts of server, we move servers from these descendants to ℓ until the total server movement has cost roughly some small quantity γ. Since the cost of server movement grows exponentially up the tree, and the movement cost is roughly the same for each v_i, more server mass is moved from closer locations. Since there are H levels in the HST, the total movement cost is roughly $H\gamma$. This concludes one "round" of server movement. This server movement is now repeated over multiple rounds until ℓ has $1 - \delta'$ amount of server at it. (This can be thought of as a discretization of a continuous process.)

(ii) To account for server movement at node v_i, we raise both primal and dual variables of the local LP at v_i. The primal increase tells us which children of v_i to move the servers from. The dual increase allows us to account for the server movement. Indeed, we ensure that the total dual increase for the local LP at each v_i—and hence by our composition operations, the dual increase for the global LP—is also approximately γ in each round. Moreover, we show this dual scaled down by $\beta \approx O(\log n)$ is feasible. This means that the $O(H\gamma)$ cost of server movement in each round can be approximately charged to this increase of the global LP dual, giving us $H\beta = O(H \log n)$-competitiveness.

(iii) The choice of dual variables to raise for the local LP at node v is dictated by the corresponding dual variables for the children of v. Each constraint in the local LP at v is composed from the local constraints at some of its children. It is possible that there are several constraints at v that are composed using the same constraint at a child u of v. We maintain the invariant that the total dual values of the former is bounded by the dual value of the latter. Now, we can only raise those dual variables at v where there is some slack in this invariant condition.

Finally, to extend our results to k-ServerTW, we say that a request $(\ell, I = [b, q])$ becomes critical (at time q) if the amount of server mass at ℓ at any time during I was at most $1 - \delta'$. We proceed as above to move server mass to ℓ. However, after servicing ℓ, we also service active request intervals at nearby leaves: we service these *piggybacked requests* according to (a variation of) the earliest deadline rule while ensuring that the total cost incurred remains bounded by (a factor times) the cost incurred to service ℓ. We use ideas from [2] (for the case of $k = 1$) to find this tour, but we need a new dual-fitting-based analysis of this algorithm. Moreover, new technical insights are needed to fit this dual-fitting analysis (which works only for $k = 1$) with the rest of our analytical framework. Indeed, the power of our LP relaxation for k-Server lies in the ease with which it extends to k-ServerTW.

B. Related Work

The k-Server problem is arguably the most prominent problem in online algorithms. Early work focused on deterministic algorithms [12], [16], and on combinatorial randomized algorithms [13], [6]. k-Server has also been studied for special metric spaces, such as lines, (weighted) stars, trees: [7] gives more background on the k-Server problem. Works obtaining poly-logarithmic competitive ratio are more recent, starting with [5], and more recently, by [8]. ([9] gives an alternate projection-based perspective on [8].) A new LP relaxation was introduced by [8], who then use a mirror

descent strategy with a multi-level entropy regularizer to obtain the online dynamics. However, it is unclear how to extend their LP when there are time-windows, even for the case of star metrics. Our competitive ratio for k-Server on HSTs is $\text{poly}\log(n\Delta)$ as against just $\text{poly}\log(k)$ in their work, but this weaker bound is in exchange for a more flexible algorithm/analysis that extends to time-windows.

Online algorithms where requests can be served within some time-window (or more generally, with delay penalties) have recently been given for several problems (see [2], [3] and references therein). The work closest to ours is that of [2] who show $O(k\log^3 n)$-competitiveness for k-Server with general delay functions, and leave open the problem of getting poly-logarithmic competitiveness. Another related work is [14] who show $O(\log k \log n)$-competitiveness for Weighted Paging, which is the same as k-Server with delays for weighted star metrics. This work also used a hitting-set LP: this was based on two different kinds of extensions of the request intervals and was very tailored to the star metric, and is unclear how to extend it even to 2-level trees. Our new LP relaxation is more natural, being implied by the min-cost flow relaxation for k-Server, and extends to time-windows.

Algorithms for the online set cover problem were first given by [1]: this led to the general primal-dual approach for covering linear programs (and sparse set-cover instances) [10], and to sparse CIPs [15]. Our algorithm also uses a similar primal-dual approach for the local LPs defined at each node of the tree; we also need to crucially use the sparsity properties of the corresponding set-cover-like constraints.

II. A Covering LP Relaxation

For the rest of the paper, we consider the k-Server problem on hierarchically well-separated trees (HSTs) with n leaves, rooted at node $\mathbb{r}$ and having height H. (The standard extension to general metrics via tree embeddings was outlined in §I.) Define the *level* of a node as its combinatorial height, with the leaves at level 0, and the root at level H. For a non-root node v, the length of the edge $(v, p(v))$ going to its parent $p(v)$ is $c_v := \lambda^{\text{level}(v)}$. So leaf edges have length 1, and edges between the root and its children have length λ^{H-1}. We assume that $\lambda \geq 10H$. For a vertex v, let χ_v be its children, T_v be the subtree rooted at v, and L_v be the leaves in this subtree. Let $n_v := |T_v|$. For a subset A of nodes of a tree T, let T^A denote the minimal subtree of T containing the root node and set A, i.e., the subtree consisting of all nodes in A and their ancestors.

Request Times and Timesteps.: Let the request sequence be $\mathfrak{R} := r_1, r_2, \ldots$. For k-Server, each request $r_i \in \mathfrak{R}$ is a tuple (ℓ_{q_i}, q_i) for some leaf ℓ_{q_i} and distinct *request time* $q_i \in \mathbb{Z}_+$, such that $q_{i-1} < q_i$ for all i. In k-ServerTW each request r_i is a tuple $(\ell_{e_i}, I_i = [b_i, e_i])$ for a leaf ℓ_i and (request) interval $I_i = [b_i, e_i]$ with *arrival/start time* b_i and *end time* e_i. The algorithm sees this request r_i at time b_i; again $b_{i-1} < b_i$ for all i. A solution must ensure that a server visit ℓ_i during interval I_i. The set of all starting and ending times of intervals are called *request times*; we assume these are distinct integers. [1]

Between any two *request times* q and $q+1$, we define a large collection of *timesteps* (denoted by τ or t)—these timesteps take on values $\{q + i\eta\}$ for some small value $\eta \in (0, 1)$. (Each request arrival time is also a timestep). We use $\mathfrak{T}$ to denote the set of timesteps. Our fractional algorithm moves a small amount of server to the request location r_q at some of the timesteps $t \in [q, q+1)$. Given a timestep τ, let $\lfloor \tau \rfloor$ refer to the request time q such that $\tau \in [q, q+1)$.

A. The Covering LP Relaxation

We first give a covering LP relaxation for k-Server, and then generalize it to k-ServerTW. Consider an instance of k-Server specified by an HST and a request sequence $r_1, r_2, \ldots$. Our LP relaxation $\mathfrak{M}$ has variables $x(v,t)$ for every non-root node v and timestep t, where $x(v,t)$ indicates the amount of server traversing the edge from v to its parent $p(v)$ at timestep t. The objective function is

$$\sum_{v \neq \mathbb{r}} \sum_{t} c_v \, x(v,t). \tag{II.1}$$

There are exponentially many constraints. Let A be a subset of leaves. Let $\boldsymbol{\tau} := \{\tau_u\}_{u \in T^A}$ be a set of timesteps for each node in T^A, i.e., nodes in A and their ancestors.[2] These timesteps must satisfy two conditions: (i) each (leaf) $\ell \in A$ has a request at time $\lfloor \tau_\ell \rfloor$, and (ii) for each internal node $u \in T^A$, $\tau_u = \max_{\ell \in A \cap T_u} \tau_\ell$; i.e., τ_u is the latest timestep assigned to a leaf in u's subtree by $\boldsymbol{\tau}$. For the tuple $(A, \boldsymbol{\tau})$, the LP relaxation contains the constraint $\varphi_{A,\boldsymbol{\tau}}$:

$$\sum_{v \in T^A, v \neq \mathbb{r}} x(v, (\tau_v, \tau_{p(v)}]) \geq |A| - k. \tag{II.2}$$

Define $x(v, I) := \sum_{t \in I} x(v,t)$ for any interval I. We now prove validity of these constraints. (In the full version of the paper, we show these constraints are implied by the usual min-cost flow formulation for k-Server, giving another proof of validity.)

Claim II.1. *The linear program $\mathfrak{M}$ is a valid relaxation for the k-Server problem.*

Proof: Consider a solution to the k-server instance that ensures that for a request at a leaf ℓ at time q, there is a server at ℓ at time q. We assume that this solution has the *eagerness* property—if leaves ℓ and ℓ', requested at times q and q' respectively, are two consecutive locations visited by a server, the server moves from q to q' at timestep $q + \eta$ (which is less than q').

[1] k-Server (without time-windows) can be modeled by time-intervals of length 1, where each $e_i = b_i + 1$.

[2] We use boldface $\boldsymbol{\tau}$ to denote a vector of timesteps, and τ_u to be the value of this vector for a vertex u.

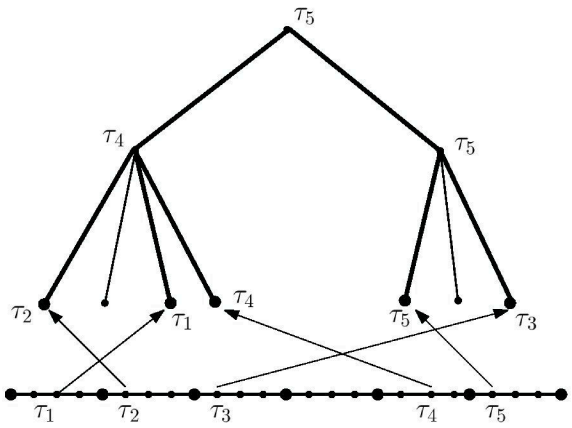

Figure 1: Example of a tuple $(A, \boldsymbol{\tau})$: the set A is given by the leaves in bold, the tree T^A by the bold edges, and $\boldsymbol{\tau}$ is shown against each vertex in T^A. Bold circles on the timeline denote request arrival times, the dots denote timesteps. Each shown timestep has the corresponding request arriving at the shown leaf at the arrival time (bold dot) preceding it.

Now for a constraint of the form (II.2), let $A_1, A_2, \ldots, A_k$ be the subsets of A that are served by the different servers (some of these sets may be empty). Define $x_i(v,t) \leftarrow 1$ if server i crosses the edge $(v, p(v))$ *from v to $p(v)$ (i.e., upwards)* at timestep t, and 0 otherwise. We show that

$$\sum_{v \in T(A_i), v \neq \mathfrak{r}} x_i(v, (\tau_v, \tau_{p(v)}]) \geq |A_i| - 1.$$

Defining $x(v,t) := \sum_i x_i(v,t)$ by summing over all i gives (II.2). For any server i and set A_i, define E' to be the edges $(v, p(v))$ for which $x_i(v, (\tau_v, \tau_{p(v)}]) = 1$. If $|E'| < |A_i| - 1$, then deleting the edges in E' from the tree leaves a connected component C with at least two vertices from A_i. Server i serves at least two leaf vertices $C \cap A_i$, say v, w, requested at times $q_v = \lfloor \tau_v \rfloor, q_w = \lfloor \tau_w \rfloor$ respectively. Say $q_v < q_w$, and let u be the least common ancestor of v, w. Notice that $\tau_u \geq \tau_v$, and if the path from v to u is labeled $v_0 = v, v_1, \ldots, v_h = u$, then the intervals $(\tau_{v_i}, \tau_{v_{i+1}}]$ partition $(\tau_v, \tau_u]$. Since the server is at v at timestep τ_v (by the construction above) and is at w at time $q_w \leq \tau_w$, there must be an edge (v_i, v_{i+1}) such that it crosses this edge upwards during $(\tau_{v_i}, \tau_{v_{i+1}}]$. Then this edge should be in E', a contradiction. ■

Remark II.2. We could have replaced the constraint (II.2) by its simpler version involving $x_i(v, (q_v, q_{p(v)}])$, where $q_v := \lfloor \tau_v \rfloor$: that would be valid and sufficient. However, since our algorithm works at the level of timesteps, it is convenient to use (II.2).

Extension to Time-Windows: We now extend these ideas to k-ServerTW. In constraint (II.2) for a pair $(A, \boldsymbol{\tau})$, the timesteps for ancestors of (a leaf in) A could be inferred from the values assigned by τ to A. We now generalize this by (i) allowing A to contain non-leaf nodes, as long as they are independent

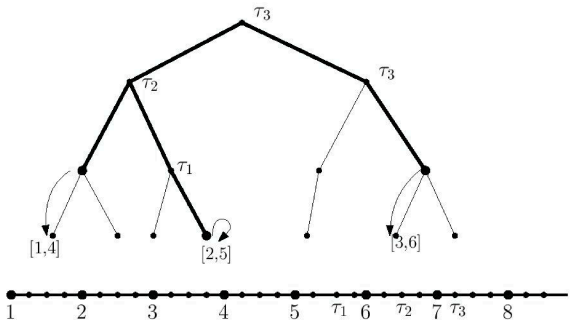

Figure 2: Example of a tuple $(A, f, \boldsymbol{\tau})$ for k-ServerTW: the set A is given by the bold nodes, the tree T^A by the bold edges, and $\boldsymbol{\tau}$ is shown against each internal vertex in T^A. Arrows indicate the mapping f to a leaf request interval.

(in terms of the ancestor-descendant relationship), and (ii) the timestep assigned to an internal node is *at least* that of each of its descendants in A. Formally, consider a tuple $(A, f, \boldsymbol{\tau})$, where A is a subset of tree nodes such that no two of them have an ancestor-descendant relationship, the function $f : A \to \mathfrak{R}$ maps each node $v \in A$ to a request $(\ell_v, [b_v, e_v])$ given by a leaf $\ell_v \in T_v$ and an interval $[b_v, e_v]$ at ℓ_v, and the assignment $\boldsymbol{\tau}$ maps each node $u \in T^A$ to a timestep τ_u satisfying the following two (monotonicity) properties:

(a) For each node $v \in T^A$, $\tau_v \geq \max_{u \in A \cap T_v} e_u$.
(b) If v_1, v_2 are two nodes in T^A with v_1 being the ancestor of v_2, then $\tau_{v_1} \geq \tau_{v_2}$.

Given such a tuple $(A, f, \boldsymbol{\tau})$, define the constraint $\varphi_{A,f,\boldsymbol{\tau}}$[3]

$$\sum_{v \in A, v \neq \mathfrak{r}} x(v, (b_v, \tau_{p(v)}]) + \sum_{v \in T^A \setminus A, v \neq \mathfrak{r}} x(v, (\tau_v, \tau_{p(v)}]) \geq |A| - k. \quad \text{(II.3)}$$

Note the differences with constraint (II.2): the LHS for a node $v \in A$ has a longer interval starting from b_v instead of from τ_v. Also, (II.3) does not use the timesteps $\{\tau_v\}_{v \in A}$: these will be useful later in defining the truncated constraints. In the special case of k-Server where $e_v = b_v + 1$, the above constraint is similar to (II.2), though the terms for nodes in A differ slightly. The objective function is the same as (II.1). We denote this LP by $\mathfrak{M}_{TW}$. The proof of the following result is given in the full version.

Claim II.3. *The linear program $\mathfrak{M}_{TW}$ is a valid relaxation for k-ServerTW.*

III. The Local LPs: Truncation and Composition

We maintain a collection of *local* LPs $\mathfrak{L}^v$, one for each internal vertex v of the tree. While the constraints of local LPs for the non-root nodes are not necessarily valid for

[3]The condition $v \neq \mathfrak{r}$ in the first summation is invoked only when $A = \{\mathfrak{r}\}$, in which case the LHS is empty.

the original k-Server instance, those in the local LP $\mathfrak{L}^{\mathbb{r}}$ are implied by constraints of $\mathfrak{M}$ or $\mathfrak{M}_{TW}$. This gives us a handle on the optimal cost. The constraints in the local LP at a node are related to those in its children's local LPs, allowing us to relate their primal/dual solutions, and their costs.

To define the local LPs, we need some notation. Our (fractional) algorithm $\mathcal{A}$ moves server mass around over timesteps. In the local LPs, we define constraints based on the state of our algorithm $\mathcal{A}$. Let $k_{v,t}$ be the server mass that $\mathcal{A}$ has in v's subtree T_v at timestep t (when v is a leaf, this is the amount of server mass at v at timestep t). We choose three non-negative parameters δ, δ', γ. The first two help define lower and upper bounds on the amount of (fractional) servers at any leaf, and γ denotes the granularity at which movement of server mass happens. We ensure $\delta' \gg \delta \gg \gamma$, and set $\delta' = \frac{1}{n^2}, \delta = \frac{1}{10n^3}, \gamma = \frac{1}{n^4}$.

Definition III.1 (Active and Saturated Leaves)**.** Given an algorithm $\mathcal{A}$, a leaf ℓ is *active* if it has at least δ amount of server (and *inactive* otherwise). The leaf is *saturated* if ℓ has more than $1-\delta'$ amount of server (and *unsaturated* otherwise).

The server mass at each location should ideally lie in the interval $[\delta, 1-\delta']$, but since we move servers in discrete steps, we maintain the following (slightly weaker) invariant:

> **Invariant (I1).** *The server mass at each leaf lies in the interval* $[\delta/2, 1-\delta'/2]$.

Constraints of $\mathfrak{L}^v$ are defined using *truncations* of the constraints $\varphi_{A,\boldsymbol{\tau}}$. For a node v and subset of nodes A in T, let the subtree T_v^A be the minimal subtree of T_v containing v and all the nodes in $A \cap T_v$.

Definition III.2 (Truncated Constraints)**.** Consider a node v, a subset A of leaves in T and a set $\boldsymbol{\tau} := \{\tau_u\}_{u \in T_v^A}$ of timesteps satisfying the conditions: (i) each (leaf) $\ell \in A$ has a request at time $\lfloor \tau_\ell \rfloor$, and (ii) for each internal node $u \in T_v^A, \tau_u = \max_{\ell \in A \cap T_v}$. The *truncated constraint* $\varphi_{A,\boldsymbol{\tau},v}$ is defined as:

$$\sum_{u: u \neq v, u \in T_v^A} y^v(u, (\tau_u, \tau_{p(u)}]) \geq |A \cap T_v| - k_{v,\tau_v} - 2\delta(n - n_v); \qquad \text{(III.1)}$$

recall that k_{v,τ_v} is the amount of server mass in T_v *at the end of* timestep τ_v. We say that the truncated constraint $\varphi_{A,\boldsymbol{\tau},v}$ *ends at* τ_v.

The truncated constraint $\varphi_{A,\boldsymbol{\tau},v}$ can be thought of as truncating an actual LP constraint of the form (II.2) for the nodes in T_v^A only. One subtle difference is the last term that weakens the constraint slightly; we will see in Lemma III.5 that this weakening is crucial. The truncated constraint $\varphi_{A,f,\boldsymbol{\tau},v}$ in case of k-ServerTW is defined analogously: given a node v, a tuple $(A, f, \boldsymbol{\tau})$ satisfying the conditions stated above (II.3) with the restriction that A lies in T_v and $\boldsymbol{\tau}$ is defined for nodes in T_v^A only, the truncated constraint $\varphi_{A,f,\boldsymbol{\tau},v}$ (ending at τ_v) is defined analogously as

$$\sum_{u \in A \cap T_v, u \neq v} y^v(u, (b_u, \tau_{p(u)}]) + \sum_{u \in T_v^A \setminus A, u \neq v} y^v(u, (\tau_u, \tau_{p(u)}]) \geq |A \cap T_v| - k_{v,\tau_v} - 2\delta(n - n_v) \qquad \text{(III.2)}$$

A few remarks about the truncation: first, this truncated constraint uses *local variables* y^v that are "private" for the node v instead of the global variables x. In fact, we can think of x as denoting variables $y^{\mathbb{r}}$ local to the root, and therefore $\varphi_{A,\boldsymbol{\tau},\mathbb{r}} = \varphi_{A,\boldsymbol{\tau}}$ (or $\varphi_{A,f,\boldsymbol{\tau},\mathbb{r}} = \varphi_{A,f,\boldsymbol{\tau}}$). Second, a truncated constraint is *not necessarily implied* by the LP relaxation $\mathfrak{M}$ (or $\mathfrak{M}_{TW}$) even when we replace y^v by x, since a generic algorithm is not constrained to maintain k_{v,τ_v} servers in subtree T_v after timestep τ_v. But, at the root (i.e., when $v = \mathbb{r}$), we always have $k_{v,\tau_v} = k$ and the last term is 0, so replacing $y^{\mathbb{r}}$ by x in its constraints gives us constraints of the form (II.2) from the actual LP.

Definition III.3 ($\perp$-constraints)**.** A truncated constraint where $|A| = 1$ is called a $\perp$-*constraint*.

Such $\perp$-constraints play a special role when a subtree has only one active leaf, namely the requested leaf. In the case of k-Server, if $|A| = 1$ then the constraint (III.1) has no terms on the LHS but a positive RHS, so it can never be satisfied. Nevertheless, such constraints will be useful when forming new constraints by composition.

Composing Truncated Constraints: The next concept is that of *constraint composition*: a truncated constraint $\varphi_{A,\boldsymbol{\tau},v}$ can be obtained from the corresponding truncated constraints for the children of v. Consider a subset X of v's children. For $u \in X$, let $C_u := \varphi_{A(u),\boldsymbol{\tau}(u),u}$ be a constraint in $\mathfrak{L}^u$ ending at $\tau_u := \tau(u)_u$, given by some linear inequality $\langle a^{C_u}, y^u \rangle \geq b^{C_u}$. Then defining $A := \cup_{u \in X} A(u)$ and $\tau : T^A \to \mathfrak{T}$ obtained by extending maps $\boldsymbol{\tau}(u)$ and setting $\tau_v = \max_{u \in X} \tau_u$, the constraint $\varphi_{A,\boldsymbol{\tau},v}$ is written as: [4]

$$\sum_{u \in X} \Big(y^v(u, (\tau_u, \tau_v]) + \langle a^{C_u}, y^v \rangle \Big) \geq \sum_{u \in X} b^{C_u} - \Big(k_{v,\tau_v} - \sum_{u \in X} k_{u,\tau_u} \Big) + 2\delta \Big(n_v - \sum_{u \in X} n_u \Big). \qquad \text{(III.3)}$$

The constraints $\varphi_{A(u),\boldsymbol{\tau}(u),u}$ used their local variables y^u, whereas this new constraint uses y^v. Every constraint in $\mathfrak{L}^v$ can be obtained this way, and so the constraints of $\mathfrak{L}^{\mathbb{r}}$

[4]The vector a^{C_u} has one coordinate for every node in T_u^A, whereas y^v has one coordinate for each node in $T_v^A \supseteq T_u^A$. We define the inner product $\langle a^{C_u}, y^v \rangle$ by adding extra coordinates (set to 0) in the vector a^{C_u}.

(which are implied by $\mathfrak{M}$) can be obtained by recursively composing truncated constraints for its children's local LPs. In case of k-ServerTW, the composition operation holds for the constraints $\varphi_{A,f,\boldsymbol{\tau},v}$: a minor change is that the terms in LHS involving a vertex $u \in A$ have $y^v(u,(b_u,\tau_v])$, where b_u is the starting time of the request corresponding to $f(u)$.

A. Constraints in Terms of Local Changes

The local constraints (III.1) and the composition rule (III.3) are written in terms of k_{u,τ_u}, the amount of server that our algorithm $\mathcal{A}$ places at various locations and times. It will be more convenient to rewrite them in terms of server movements in $\mathcal{A}$.

Definition III.4 (g,r,D)**.** For a vertex v and timestep t, let the *give* $g(v,t)$ and the *receive* $r(v,t)$ denote the total (fractional) server movement *out of* and *into* the subtree T_v on the edge $(v,p(v))$ at timestep t. For interval I, let $g(v,I) := \sum_{t\in I} g(v,t)$ and define $r(v,I)$ similarly, and define the "difference" $D(v,I) := g(v,I) - r(v,I)$.

Restating the composition rule in terms of the quantities D defined above shows the utility of the extra term on the RHS of the truncated constraint.

Lemma III.5. *Consider a vertex v, a timestep τ and a subset X of children of v such that at timestep τ all active leaves in T_v are descendants of the nodes in X. For each $u \in X$, consider a truncated constraint $C_u := \varphi_{A(u),\boldsymbol{\tau}(u),u}$ given by some linear inequality $\langle a^{C_u}, y^u\rangle \geq b^{C_u}$. Define $(A,\boldsymbol{\tau})$ as in (III.3) with $\tau := \tau_v$, and assume Invariant (I1) holds. Then the truncated constraint $\varphi_{A,\boldsymbol{\tau},v}$ from (III.3) implies the inequality:*[5]

$$\sum_{u\in X}\left(y^v(u,(\tau_u,\tau_v]) + \langle a^{C_u}, y^v\rangle\right) \geq \sum_{u\in X}\left(D(u,(\tau_u,\tau_v]) + b^{C_u}\right) + \Big(n_v - \sum_{u\in X} n_u\Big)\delta, \quad \text{(III.4)}$$

We call this the composition rule. *An analogous statement holds for a tuple $(A,f,\boldsymbol{\tau})$ for a vertex v in the case of k-ServerTW, except that τ_u is replaced by b_u for every vertex $u \in A$ on the LHS (details in the full version).*

B. Timesteps and Constraint Sets

Recall that $\mathfrak{T}$ is the set of all timesteps. For each vertex v we define a subset $\mathcal{R}(v) \subseteq \mathfrak{T}$ of *relevant* timesteps, such that the local LP $\mathfrak{L}^v$ contains a non-empty set of constraints $\mathfrak{L}^v(\tau)$ for each $\tau \in \mathcal{R}(v)$. The variables in this local LP are $x(u,\tau)$ where $u \in T_v$ and τ is a timestep. Each constraint in $\mathfrak{L}^v(\tau)$ is of the form $\varphi_{A,\boldsymbol{\tau},v}$ for a tuple $(A,\boldsymbol{\tau})$ ending at τ. Overloading notation, let $\mathfrak{L}^v := \bigcup_{\tau\in\mathcal{R}(v)} \mathfrak{L}^v(\tau)$ denote the set of all constraints in the local LP at v. The objective function of this local LP is $\sum_{u\in T_v,\tau} c_u\, y^v(u,\tau)$.

[5]When $y \geq 0$, a constraint $\langle a,y\rangle \geq b$ is said to *imply* a constraint $\langle a',y\rangle \geq b'$ if $a \leq a'$ (componentwise) and $b \geq b'$.

The timesteps in $\mathcal{R}(v)$ are partitioned into $\mathcal{R}^s(v)$ and $\mathcal{R}^{ns}(v)$, the *solitary* and *non-solitary* timesteps for v. The decision whether a timestep belongs to $\mathcal{R}(v)$ is made by our algorithm. and is encoded by adding τ to either $\mathcal{R}^s(v)$ or $\mathcal{R}^{ns}(v)$. For each timestep $\tau \in \mathcal{R}^s(v)$, the algorithm creates a constraint set $\mathfrak{L}^v(\tau)$ consisting of a single $\perp$-constraint (recall Definition III.3); for each timestep $\tau \in \mathcal{R}^{ns}(v)$ it creates a constraint set $\mathfrak{L}^v(\tau)$ containing only non-$\perp$-constraints obtained by composing constraints from $\mathfrak{L}^w(\tau_w)$ for some children w of v and timesteps $\tau_w \in \mathcal{R}(w)$, where $\tau_w \leq \tau$.

For each τ, a constraint $C \in \mathfrak{L}^v(\tau)$ corresponds to a dual variable z_C, which is raised only at timestep τ. We ensure the following invariant.

Invariant (I2). *At the end of each timestep $\tau \in \mathcal{R}^{ns}(v)$, the objective function value of the dual variables corresponding to constraints in $\mathfrak{L}^v(\tau)$ equals γ. I.e., if a generic constraint C is given by $\langle a^C \cdot y^v\rangle \geq b^C$, then*

$$\sum_{C\in\mathfrak{L}^v(\tau)} b^C \cdot z_C = \gamma \qquad \forall \tau \in \mathcal{R}^{ns}(v). \quad \text{(I2)}$$

Furthermore, $b^C > 0$ for all $C \in \mathfrak{L}^v(\tau)$ and $\tau \in \mathcal{R}(v)$.

No dual variables z_C are defined for $\perp$-constraints, and (the first statement of) Invariant (I2) does not apply to timesteps $\tau \in \mathcal{R}^s(v)$. In the following sections, we show how to maintain a dual solution that is feasible for $\mathfrak{D}^v$ (the dual LP for $\mathfrak{L}^v$) when scaled down by some factor $\beta = \text{poly}\log(n\lambda)$.

Awake Timesteps: For a vertex v, we maintain a subset $\mathsf{Awake}(v)$ of *awake* timesteps. The set $\mathsf{Awake}(v)$ has the property that it contains all the solitary timesteps, i.e., $\mathcal{R}^s(v)$, and some non-solitary ones. Hence $\mathcal{R}^s(v) \subseteq \mathsf{Awake}(v) \subseteq \mathcal{R}^s \cup \mathcal{R}^{ns}(v) = \mathcal{R}(v)$. Whenever we add a timestep to $\mathcal{R}(v)$, we initially add it to $\mathsf{Awake}(v)$; some of the non-solitary ones subsequently get removed. A timestep τ is awake for vertex v at some moment in the algorithm if it belongs to $\mathsf{Awake}(v)$ at that moment. For any vertex v, define

$$\mathsf{prev}(v,\tau) := \arg\max\{\tau' \in \mathsf{Awake}(v) \mid \tau' \leq \tau\} \quad \text{(III.5)}$$

Note that as the set $\mathsf{Awake}(v)$ evolves over time, so does the identity of $\mathsf{prev}(v,\tau)$. We show in the full version that prev is well-defined for all relevant (v,τ) pairs.

Starting configuration: At the beginning of the algorithm, assume that the root has $2k$ "dummy" leaves as children, each of which has server mass $1/2$ at time $q = 0$. All other leaves of the tree have mass $\delta/2$. (This ensures Invariant (I1) holds.) No requests arrive at any dummy leaf v; moreover, we add a $\perp$-constraint $\varphi_{A,\boldsymbol{\tau},v}$, where $A = \{v\}$ and $\tau_v = 0$. Assuming this starting configuration only changes the cost of our solution by at most an additive term of $O(k\Delta)$, where Δ is the aspect ratio of the metric space.

IV. Algorithm for k-Server

We now describe our algorithm for k-Server. At request time q, the request arrives at a leaf ℓ_q. The *main procedure* calls *local update* procedures for each ancestor of ℓ_q. Each such local update possibly moves servers to ℓ_q, and also adds constraints to the local LPs and raises the primal/dual values to account for this movement. We use $\mathsf{ReqLoc}(\tau)$ to denote the location of request with deadline at time $\lfloor\tau\rfloor$, i.e., $\ell_{\lfloor\tau\rfloor}$.

A. The Main Procedure

In the main procedure of Algorithm 1, let the *backbone* be the leaf-root path $\ell_q = v_0, v_1, \ldots, v_H = \mathtt{r}$. We move servers to ℓ_q from other leaves until it is saturated: this server movement happens in small discrete increments over several timesteps. Each iteration of the **while** loop in line (1.4) corresponds to a distinct timestep τ. Let $\mathsf{activesib}(v, \tau)$ be the siblings v' of v with active leaves in their subtrees $T_{v'}$ (at timestep τ). Let i_0 be the smallest index with non-empty $\mathsf{activesib}(v_{i_0}, \tau)$. The procedure SimpleUpdate adds a $\perp$-constraint to each of the sets $\mathfrak{L}^{v_i}(\tau)$ for $i = 0, \ldots, i_0$. For $i > i_0$, the procedure FullUpdate adds (non-$\perp$) constraints to $\mathfrak{L}^{v_i}(\tau)$. If $\mathsf{activesib}(v_i, \tau)$ is non-empty, it also transfers some servers from the subtrees below $\mathsf{activesib}(v_i, \tau)$ to ℓ_q.

Algorithm 1: Main Procedure

1.1 **foreach** $q = 1, 2, \ldots$ **do**
1.2 get request r_q; let the path from r_q to the root be $\ell_q = v_0, v_1, \ldots, v_H = \mathtt{r}$.
1.3 $\tau \leftarrow q + \eta$, the first timestep after q.
1.4 **while** $k_{v_0,\tau} \le 1 - \delta'$ **do**
1.5 let $i_0 \leftarrow$ smallest index such that $\mathsf{activesib}(v_{i_0}, \tau) \neq \varnothing$.
1.6 **for** $i = 0, \ldots, i_0$ **do** call SimpleUpdate(v_i, τ).
1.7 **for** $i = i_0 + 1, \ldots, H$ **do** call FullUpdate(v_i, τ).
1.8 $\tau \leftarrow \tau + \eta$. // move to the next timestep

B. The Simple Update Procedure

This procedure adds timestep τ to both $\mathcal{R}^s(v)$ and $\mathsf{Awake}(v)$, and creates a $\perp$-constraint in the LP $\mathfrak{L}^v$.

Algorithm 2: SimpleUpdate(v, τ)

2.1 let $v_0 \leftarrow \mathsf{ReqLoc}(\tau)$.
2.2 add timestep τ to event set $\mathcal{R}^s(v)$ and to $\mathsf{Awake}(v)$. // "solitary" timestep for v
2.3 $\mathfrak{L}^v(\tau) \leftarrow$ the $\perp$-constraint $\varphi_{A,\boldsymbol{\tau},v}$, where $A = \{v_0\}$ and $\tau_w = \tau$ for nodes w on the v_0-v path.

C. The Full Update Procedure

The FullUpdate(v, τ) procedure is called for backbone nodes v that are above v_{i_0} (using the notation of Algorithm 1). It has two objectives. First, it transfers servers to the requested leaf node v_0 from the subtrees of the off-backbone children of v, incurring a total cost of at most γ. Second, it defines the constraints $\mathfrak{L}^v(\tau)$ and runs a primal-dual update on these constraints until the total dual value raised is exactly γ. This dual increase is at least the server transfer cost, which we use to bound the algorithm's cost. We now explain the steps of Algorithm 3 in more detail. (The notions of slack and depleted constraints are in Definition IV.1.)

Algorithm 3: FullUpdate(v, τ)

3.1 let $h \leftarrow \mathsf{level}(v) - 1$ and $u_0 \in \chi_v$ be child containing the current request $v_0 := \mathsf{ReqLoc}(\tau)$.
3.2 let $U \leftarrow \{u_0\} \cup \mathsf{activesib}(u_0, \tau)$; say $U = \{u_0, u_1, \ldots, u_\ell\}$, $L_U \leftarrow$ active leaves below $U \setminus \{u_0\}$.
3.3 add timestep τ to event set $\mathcal{R}^{ns}(v)$ and to $\mathsf{Awake}(v)$. // "non-solitary" timestep for v
3.4 set timer $s \leftarrow 0$.
3.5 **repeat**
3.6 **for** $u \in U$ **do**
3.7 let $\tau_u \leftarrow \mathsf{prev}(u, \tau)$ and $I_u = (\tau_u, \tau]$.
3.8 let C_u be a slack constraint in $\mathfrak{L}^u(\tau_u)$. // slack constraint exists since $\mathsf{prev}(u, \tau)$ is awake
3.9 let $\sigma \leftarrow (C_{u_0}, C_{u_1}, \ldots, C_{u_\ell})$ be the resulting tuple of constraints.
3.10 add new constraint $C(v, \sigma, \tau)$ to constraint set $\mathfrak{L}^v(\tau)$.
3.11 **while** *all constraints C_{u_j} in σ are slack* ***and*** *dual objective for $\mathfrak{L}^v(\tau)$ less than γ* **do**
3.12 increase timer s at uniform rate.
3.13 increase $z_{C(v,\sigma,\tau)}$ at the same rate as s.
3.14 for all $u \in U$, define $S_u := I_u \cap (\mathcal{R}^{ns}(u) \cup \{\tau_u + \eta\})$.
3.15 increase $y^v(u, t)$ for $u \in U, t \in S_u$ according to $\frac{dy^v(u,t)}{ds} = \frac{y^v(u,t)}{\lambda^h} + \frac{\gamma}{Mn \cdot \lambda^h}$.
3.16 **transfer** server mass from T_u into v_0 at rate $\frac{dy^v(u,I_u)}{ds} + \frac{b^{C_u}}{\lambda^h}$ using the leaves in $L_U \cap T_u$, for each $u \in U \setminus \{u_0\}$
3.17 **foreach** *constraint C_{u_j} that is depleted* **do**
3.18 **if** *all the constraints in $\mathfrak{L}^{u_j}(\tau_{u_j})$ are depleted* **then** remove τ_{u_j} from $\mathsf{Awake}(u_j)$.
3.19 **until** *the dual objective corresponding to constraints in $\mathfrak{L}^v(\tau)$ becomes γ.*

Consider a call to FullUpdate(v, τ) with u_0 being the child of v on the path to the request v_0 (See Figure 3). Each iteration of the **repeat** loop adds a constraint C to

$\mathfrak{L}^v(\tau)$ and raises the dual variable z_C corresponding to it. For each node u in $U := \{u_0\} \cup \mathsf{activesib}(u_0, \tau)$, define $\tau_u := \mathsf{prev}(u, \tau)$ to be the most recent timestep currently in $\mathsf{Awake}(u)$. This timestep τ_u may move backwards over the iterations as nodes are removed from $\mathsf{Awake}(u)$ in line (3.17). One exception is the node u_0: we will show that τ_{u_0} stays equal to τ for the entire run of FULLUPDATE. Indeed, we add τ to $\mathsf{Awake}(u_0)$ during SIMPLEUPDATE(u_0, τ) or FULLUPDATE(u_0, τ) before calling FULLUPDATE(v, τ). We will prove in the full version that τ stays awake in $\mathcal{R}(u_0)$ during FULLUPDATE(v, τ).

1) We add a constraint $C(v, \sigma, \tau)$ to $\mathfrak{L}^v(\tau)$ by taking one constraint $C_u \in \mathfrak{L}^u(\tau_u)$ for each $u \in U$ and setting $\sigma := (C_1, \ldots, C_{|U|})$. (The choice of constraint from $\mathfrak{L}^u(\tau_u)$ is described in item 3 below.) Each C_u has form $\varphi_{A(u),\boldsymbol{\tau}(u),u}$ ending at $\tau_u := \boldsymbol{\tau}(u)_u$ for some tuple $(A(u), \boldsymbol{\tau}(u))$. The new constraint $C(v, \sigma, \tau)$ is the composition $\varphi_{A,\boldsymbol{\tau},v}$ as in (III.3), where $I_u := (\tau_u, \tau]$. Since U contains all the children of v whose subtrees contain active leaves at τ, the set $A = \cup_u A(u)$ and the $\boldsymbol{\tau}$ obtained by extending the $\boldsymbol{\tau}(u)$ functions both satisfy the conditions of Lemma III.5, which shows that $\varphi_{A(u),\boldsymbol{\tau}(u),u}$ implies:

$$\underbrace{\sum_{u \in U} \left(y^v(u, I_u) + a^{C_u} \cdot y^v \right)}_{a^{C(v,\sigma,\tau)} \cdot y^v} \geq \underbrace{\sum_{u \in U} \left(D(u, I_u) + b^{C_u} \right) + (n_v - \sum_{u \in U} n_u)\delta}_{\leq b^{C(v,\sigma,\tau)}}. \quad \text{(IV.1)}$$

2) Having added constraint $C(v, \sigma, \tau)$, we raise the new dual variable $z_{C(v,\sigma,\tau)}$ at a constant rate in line (3.13), and the primal variables $y^v(u, t)$ for each $u \in U$ and any t in some index set S_u using an exponential update rule in line (3.15). The index set S_u consists of all timesteps in $I_u \cap \mathcal{R}^{ns}(u)$ and the first timestep of I_u—which is $\tau_u + \eta$ if I_u is non-empty.[6] We will soon show that S_u is not too large, yet captures all the "necessary" variables that should be raised (see Figure 3). Moreover, we transfer servers from active leaves in T_u into $\mathsf{ReqLoc}(q)$ in line (3.16). This transfer is done arbitrarily, i.e., we move servers out of any of the leaf nodes that were active at the beginning of this procedure. Our definition of $\mathsf{activesib}(u_0, \tau)$ means that T_u has at least one active leaf and hence at least δ servers to begin with. Since we move at most $\gamma \ll \delta$ amounts of server, we maintain Invariant (I1). The case of u_0 is special: since $\tau_{u_0} = \tau$, the interval I_{u_0} is empty so no variables $y^v(u_0, t)$ are raised.

[6]This timestep may not belong to $\mathcal{R}(u)$, but all other timesteps in S_u lie in $\mathcal{R}(u)$; see also Figure 3.

Somewhat unusually for an online primal-dual algorithm, both the primal and dual variables are used to account for our algorithm's cost, and not for actual algorithmic decisions (i.e., the server movements). This allows us to increase primal variables from the past, even though the corresponding server movements are always executed at the current timestep.

To describe the stopping condition for this process, we need to explain the relationships between these local LPs, and define the notions of *slack* and *depleted* constraints. We use the fact that we have an almost-feasible dual solution $\{z_C\}_{C \in \mathfrak{L}^u(\tau_u)}$ for each $u \in U$. This in turn corresponds to an increase in primal values for variables $y^u(u', \tau')$ in $\mathfrak{L}^u$. It will suffice for our proof to ensure that when we raise $z_{C(v,\sigma,\tau)}$, we constrain it as follows:

Invariant (I3). *For every $u \in \chi_v, t \in \mathcal{R}^{ns}(u)$, and every constraint $C \in \mathfrak{L}^u(t)$ (which by definition of $\mathcal{R}^{ns}(u)$ is not a $\perp$-constraint):*

$$\left(1 + \frac{1}{H}\right) z_C \geq \sum_{\tau' \geq t} \sum_{\sigma : C \in \sigma} z_{C(v,\sigma,\tau')}. \quad \text{(I3)}$$

Definition IV.1 (Slack and Depleted Local Constraints)**.** A non-$\perp$ constraint $C \in \mathfrak{L}^u$ is *slack* if (I3) is satisfied with a strict inequality, else it is *depleted*. By convention, $\perp$-constraints are always slack.

We can now explain the remainder of the local update.

3) The choice of the constraint in line (3.8) is now easy: C_u is chosen to be any slack constraint in $\mathfrak{L}^u(\tau_u)$. If $\tau_u \in \mathcal{R}^s(u)$, this is the unique $\perp$-constraint in $\mathfrak{L}^u(\tau_u)$. The primal-dual update in the **while** loop proceeds as long as all constraints C_u in σ are slack: once a constraint becomes tight, some other slack constraint $C_u \in \mathfrak{L}^u(\tau_u)$ is chosen to be in σ. If there are no more slack constraints in $\mathfrak{L}^u(\tau_u)$, the timestep τ_u is removed from the awake set (in line (3.17)). In the next iteration, τ_u gets redefined to be the most recent awake timestep before τ (in line (3.7)). In the full version, we show that there is always an awake timestep on the timeline of every vertex.
4) The dual objective corresponding to constraints in $\mathfrak{L}^v(\tau)$ is $\sum_{C \in \mathfrak{L}^v(\tau)} b^C z_C$, where C is of the form $\langle a^C, y^v \rangle \geq b^C$. The local update process ends when the increase in this dual objective due to raising variables $\{z_C \mid C \in \mathfrak{L}^v(\tau)\}$ equals γ.

For a constraint $C \in \mathfrak{L}^u(t)$, the variable z_C is only raised in the call FULLUPDATE(u, t). Subsequently, only the right side of (I3) can be raised. Hence, once a constraint C becomes depleted, it stays depleted. It is worth discussing the special

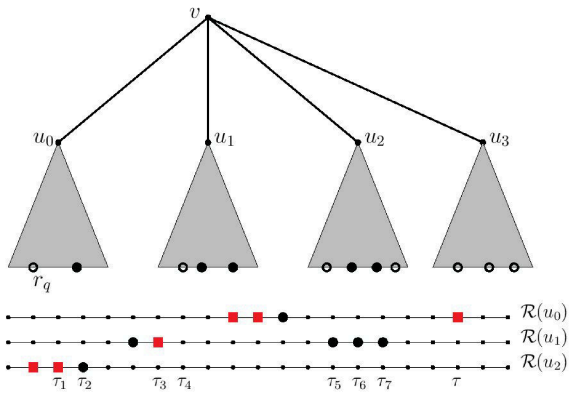

Figure 3: Illustration of FULLUPDATE(v, τ): leaves with filled dots are active and open dots are inactive, so $\mathsf{activesib}(u_0, \tau) = \{u_1, u_2\}$. The bold red squares or black circles denote timesteps in $\mathcal{R}(u) = \mathcal{R}^s(u) \cup \mathcal{R}^{ns}(u)$, with the red squares being awake at timestep τ. Hence, $I_{u_0} = S_{u_0} = \varnothing$, $I_{u_1} = (\tau_3, \tau], S_{u_1} = \{\tau_4, \tau_5, \tau_6, \tau_7\}, I_{u_2} = (\tau_1, \tau], S_{u_2} = \{\tau_2\}$. Timesteps in $\mathcal{R}^s(u)$ always remain awake.

case when $\mathsf{activesib}(u_0, \tau)$ is empty, so that $U = \{u_0\}$. In this case, no server transfer can happen, and the constraint $C(v, \sigma, \tau)$ is same as a slack constraint of $\mathfrak{L}^{u_0}(\tau)$, but with an additive term of $(n_v - n_{u_0})\delta$ on the RHS, as in (IV.1). We still raise the dual variable $z_{C(v,\sigma,\tau)}$, and prove that the dual objective value rises by γ.

There is a parameter M in line (3.15) that specifies the rate of change of y^v. This value M should be an upper bound on the size of the index set S_u over all calls to FULLUPDATE, and over all $u \in U$. We show that $M \le \frac{5H\lambda^H k}{4\gamma} + 1$, independent of the trivial bound $M \le T$, where T is the length of the input sequence.

V. ANALYSIS DETAILS

The proof rests on two lemmas: the first (proved in §V-A) bounds the movement cost in terms of the increase in dual value, and the second (proved in §V-B) shows near-feasibility of the dual solutions.

Lemma V.1 (Server Movement)**.** *The total movement cost during an execution of the procedure* FULLUPDATE *is at most 2γ, and the objective value of the dual $\mathfrak{D}^v$ increases by exactly γ.*

Lemma V.2 (Dual Feasibility)**.** *For each vertex v, the dual solution to $\mathfrak{L}^v$ is feasible if scaled down by a factor of β, where $\beta = O(\log \frac{nMk}{\gamma}) = O(H \log(n\lambda))$.*

Theorem V.3 (Competitiveness for k-server)**.** *Given any instance of the k-server problem on a λ-HST with height $H \le \lambda/10$, Algorithm 1 ensures that each request location ℓ_q is saturated at some timestep in $[q, q+1)$. The total cost of (fractional) server movement is $O(\beta H) = O(H^2 \log(n\lambda))$ times the cost of the optimal solution.*

Proof: All the server movement happens within calls to FULLUPDATE. By Lemma V.1, each iteration of the **while** loop of line (1.4) in Algorithm 1 incurs a total movement cost of $O(H\gamma)$ over at most H vertices on the backbone. Moreover, the call FULLUPDATE$(\mathbb{r}, \tau)$ corresponding to the root vertex $\mathbb{r}$ increases the value of the dual solution to the LP $\mathfrak{L}^{\mathbb{r}}$ by γ. This means the total movement cost is at most $O(H)$ times the dual solution value. Since all constraints of $\mathfrak{L}^{\mathbb{r}}$ are implied by the relaxation $\mathfrak{M}$, any feasible dual solution gives a lower-bound on the optimal solution to $\mathfrak{M}$. By Lemma V.2, the dual solution is feasible when scaled down by β, and so the (fractional) algorithm is $O(\beta H) = O(H^2 \log(n\lambda))$-competitive. ■

As mentioned in the introduction, using λ-HSTs with $\lambda = O(\log \Delta)$ allows us to extend this result to general metrics with a further loss of $O(\log^2 \Delta)$.

A. Bounds on Server Transfer and Dual Increase

The dual increase of γ claimed by Lemma V.1 will follow from the proof of Invariant (I2). The upper bound on the server movement will follow from a new invariant, which we state below. Then in §V-A we show both invariants are indeed maintained throughout the algorithm.

We first define the notion of the "lost" dual increase. Consider a call FULLUPDATE(v, τ). Let u be v's child such that request location v_0 lies in T_u. We say that u is v's *principal* child at timestep τ. We can prove that $\tau \in \mathcal{R}(u)$ remains in the awake set and hence $\tau_u = \tau$ throughout this procedure call. The dual update raises $z_{C(v,\sigma,\tau)}$ in line (3.13) and transfers servers from subtrees $T_{u'}$ for $u' \in \mathsf{activesib}(u, \tau)$ into subtree T_u in line (3.16). This transfer has two components, which we consider separately. The first is the *local component* $\frac{dy^v(u,t)}{ds}$, and the second is the *inherited component* $\frac{b^{C_u}}{\lambda^h}$. In a sense, the inherited component matches the dual increase corresponding to the term $\sum_{u' \in \mathsf{activesib}(u,\tau)} b^{C_{u'}}$ on the RHS of (IV.1). The only term without a corresponding server transfer is b^{C_u} itself, where $C_u \in \mathfrak{L}^u(\tau)$ is the constraint in σ corresponding to the principal child u. Motivated by this, we give the following definition.

Definition V.4 (Loss)**.** For vertex u with parent v, consider a timestep $\tau \in \mathcal{R}^{ns}(v)$ such that $\tau \in \mathcal{R}(u)$ as well. If $\tau \in \mathcal{R}^s(u)$, define $\mathsf{loss}(u, \tau) := 0$. Else $\tau \in \mathcal{R}^{ns}(u)$, in which case.

$$\mathsf{loss}(u, \tau) := \sum_{C \in \mathfrak{L}^u(\tau)} \sum_{C(v,\sigma,\tau) : C \in \sigma} b^C \, z_{C(v,\sigma,\tau)} \quad . \tag{V.1}$$

Invariant (I4). *For node v and timestep $\tau \in \mathcal{R}^{ns}(v)$, let u be v's principal child at timestep τ. The server mass entering subtree T_u during the procedure*

FULLUPDATE(v,τ) *is at most*

$$\frac{\gamma - \mathsf{loss}(u,\tau)}{\lambda^{\mathsf{level}(u)}}. \tag{I4}$$

Moreover, timestep $\tau \in \mathcal{R}(u)$ *stays awake during the call* FULLUPDATE(v,τ).

Multiplying the amount of transfer by the cost of this transfer, we get that the total movement cost is at most $O(\gamma)$. Invariants (I2) and (I4) prove Lemma V.1. In order to prove these invariants, we define a total order on the pairs (v,τ), with $\tau \in \mathcal{R}(v)$, and proceed by induction on this ordering. Details are deferred to the full version.

B. Approximate Dual Feasibility

For $\beta \geq 1$, a dual solution z is *β-feasible* if z/β satisfies satisfies the dual constraints. We now show that the dual variables raised during the calls to FULLUPDATE(v,τ) for various timesteps τ remain β-feasible for $\beta = O(\ln \frac{nMk}{\gamma})$. Invariant (I1) can be shown easily. We now give bounds on variables $y^v(u,t)$.

Claim V.5. *For any vertex v, any child u of v, and timestep τ, the variable $y^v(u,\tau) \leq 4\gamma M + k$, and $M \leq \frac{5H\lambda^H k}{4\gamma} + 1$.*

The following key claim shows that each constraint in the local LP $\mathfrak{L}^v$ is essentially sparse in the sense that variables in the constraint are dominated by a much smaller subsets of variables.

Claim V.6. *Let t be any timestep in $\mathcal{R}(u)$, and v be the parent of u. Define t_1 to be the last timestep in $\mathcal{R}(u) \cap [0,t]$, and t_2 to be the next timestep, i.e., $t_1 + \eta$. Let C be a constraint in $\mathfrak{L}^v$ containing the variable $y^v(u,t)$ on the LHS. Then C contains at least one of $y^v(u,t_1)$ and $y^v(u,t_2)$. Moreover, whenever we raise $z(C)$ in line (3.13) of the* FULLUPDATE *procedure, we also raise either $y^v(u,t_1)$ or $y^v(u,t_2)$ according to line (3.15).*

We now show the approximate dual feasibility. Recall that the constraints added to $\mathfrak{L}^v(\tau)$ are of the form $C(v,\sigma,\tau)$ given in (IV.1), and we raise the corresponding dual variable $z_{C(v,\sigma,\tau)}$ only during the procedure FULLUPDATE(v,τ) and never again.

Lemma V.7 (Approximate Dual Feasibility)**.** *For a node v at height $h+1$, the dual variables z_C are β_h-feasible for the dual program $\mathfrak{D}^v$, where $\beta_h = (1 + 1/H)^h\, O(\ln n + \ln M + \ln(k/\gamma))$.*

Proof: We prove the claim by induction on the height of v. For a leaf node, this follows vacuously, since the primal/dual programs are empty. Suppose the claim is true for all nodes of height at most h. For a node v at height $h+1 > 0$ with children χ_v, the variables in $\mathfrak{L}^v$ are of two types: (i) $y^v(u,t)$ for some timestep t and child $u \in \chi_v$, and (ii) $y^v(u',t)$ for some timestep t and non-child descendant $u' \in T_v \setminus \chi_v$. We consider these cases separately:

I. Suppose the dual constraint corresponds to variable $y^v(u,t)$ for some child $u \in \chi_v$. Let $\mathfrak{L}'$ be the set of constraints in $\mathfrak{L}^v$ containing $y^v(u,t)$ on the LHS. The dual constraint is:

$$\sum_{C \in \mathfrak{L}'} z_C \leq c_u = \lambda^h. \tag{V.2}$$

Let t_1, t_2 be as in the statement of Claim V.6. When we raise z_C for a constraint $C \in \mathfrak{L}'$ in line (3.13) at unit rate, we raise either $y^v(u,\tau_1)$ or $y^v(u,t_2)$ at the rate given by line (3.15). Therefore, if we raise the LHS of the dual constraint (V.2) for a total of Γ units of the timer, we would have raised one of the two variables, say $y^v(u,\tau_1)$, for at least $\Gamma/2$ units of the timer. Therefore, the value of $y^v(u,\tau_1)$ variable due to this exponential update is at least

$$\frac{\gamma}{Mn}(e^{\Gamma/2\lambda^h} - 1).$$

By Claim V.5, this is at most $4\gamma M + k$, so we get

$$\Gamma = \lambda^h \cdot O\left(\ln n + \ln M + \ln(k/\gamma)\right) = \beta_0 c_u,$$

hence showing that (V.2) is satisfied up to β_0 factor.

II. Suppose the dual constraint corresponds to some variable $y^v(u',\tau)$ with $u' \in T_u$, and $u \in \chi_v$. Suppose u' is a node at height $h' < h$. Now let $\mathfrak{L}'$ be the constraints in $\mathfrak{L}^u$ (the LP for the child u) which contain $y^u(u',\tau)$. By the induction hypothesis:

$$\sum_{C \in \mathfrak{L}'} z_C \leq \beta_{h-1}\, c_{u'}. \tag{V.3}$$

Let $\mathfrak{L}''$ denote the set of constraints in $\mathfrak{L}^v$ (the LP for the parent v) which contain $y^v(u',\tau)$. Each constraint $C(v,\sigma,\tau)$ in this set $\mathfrak{L}''$ has the coordinate σ_u corresponding to the child u being a constraint in $\mathfrak{L}'$, which implies:

$$\begin{aligned}\sum_{C(v,\sigma,\tau) \in \mathfrak{L}''} & z_{C(v,\sigma,\tau)} \\ &= \sum_{C \in \mathfrak{L}'} \sum_{C(v,\sigma,\tau) \in \mathfrak{L}'' : \sigma_u = C} z_{C(v,\sigma,\tau)} \\ &\leq (1 + 1/H) \sum_{C \in \mathfrak{L}'} z_C, \end{aligned} \tag{V.4}$$

where the last inequality uses Invariant (I3). Now the induction hypothesis (V.3) and the fact that $\beta_h = (1 + 1/H)\,\beta_{h-1}$ completes the proof.

■

Lemma V.7 means that the dual solution for $\mathfrak{L}^{\mathrm{r}}$ is β_H-feasible, where $\beta_H = O(\ln \frac{nMk}{\gamma})$. This proves Lemma V.2 and completes the proof of our fractional k-server algorithm.

VI. Algorithm for k-ServerTW

In this section, we give an overview of the online algorithm for k-ServerTW. The structure of the algorithm remains similar to that for k-Server. Again, we have a main procedure (Algorithm 4) which considers the backbone consisting of the path from the requested leaf node to the root node. It calls a suitable subroutine for each node on this backbone to add local LP constraints and/or transfer servers to v_0. We say that a request interval $R_q = [b, q]$ at a leaf node ℓ_q becomes *critical* (at time q) if it has deadline q, and it has not been served until time q, i.e., if $k_{\ell_q,t} < 1 - 2\delta'$ for all timesteps $t \in [b, q)$: for technical reasons we allow a gap of up to $2\delta'$ instead of δ'. In case this node becomes critical at q, the algorithm ensures that ℓ_q receives at least $1 - \delta'$ amount of server at time q. This ensures that we move at least δ' amount of server mass when a request becomes critical. The parameters δ, δ' remain unchanged, but we set γ to $\frac{1}{n^4\Delta}$. We extend the definition of ReqLoc from §IV in the natural way: $\mathsf{ReqLoc}(\tau)$ is location of request with deadline at time $\lfloor\tau\rfloor$, and $\mathsf{ReqInt}(\tau)$ is the request interval with deadline at time $\lfloor\tau\rfloor$.

Algorithm 4: Main Procedure for Time-Windows

4.1 **foreach** $q = 1, 2, \ldots$ **do**
4.2 **if** *ReqInt*(q) *exists and is critical* **then**
4.3 let the path from $\ell_q := \mathsf{ReqLoc}(q)$ to the root be $\ell_q = v_0, v_1, \ldots, v_H = \mathbb{r}$.
4.4 let $Z_q, \{F_{v,q} \mid v \in Z_q\} \leftarrow$ BuildTree(q)
4.5 $\tau \leftarrow q + \eta$, the first timestep after q
4.6 **while** $k_{v_0,\tau} \le 1 - \delta'$ **do**
4.7 let $i_0 \leftarrow$ smallest index such that $\mathsf{activesib}(v_{i_0}, \tau) \neq \varnothing$.
4.8 **for** $i = 0, \ldots, i_0$ **do** call SimpleUpdate$(v_i, \tau, \lambda^i \cdot \gamma/\lambda^{i_0})$.
4.9 **for** $i = i_0 + 1, \ldots, H$ **do** call FullUpdate(v_i, τ).
4.10 $\tau \leftarrow \tau + \eta$. // create a new timestep
4.11 serve requests at leaves in $\{F_{v_i,q} \mid v_i \in Z_q\}$ using server mass at v_0.

We now describe the main differences with respect to Algorithm 1 and a brief description of the subroutines called in Algorithm 4:

(i) When we service a critical request at a leaf ℓ_q, we would like to also serve active requests at nearby nodes. The procedure BuildTree(q) returns a set of backbone nodes $Z_q \subseteq \{v_0, \ldots, v_H\}$, and a tree $F_{v_i,q}$ rooted at each node $v_i \in Z_q$. In line (4.11), we service all the outstanding requests at the leaf nodes of these subtrees $\{F_{v_i,q} \mid v_i \in Z_q\}$ using the server at v_0. (These are called *piggybacked* requests.)

(ii) For a node v_i with $i \le i_0$, the previous SimpleUpdate procedure in §IV-B would define the set $\mathfrak{L}^{v_i}(\tau)$ in the local LP $\mathfrak{L}^{v_i}$ to contain just one $\perp$-constraint. For the case of time-windows, we give a new SimpleUpdate procedure, which defines a richer set of constraints based on a *charging forest* $\mathcal{F}^{ch}(v_i)$. This procedure also raises some local dual variables; this dual increase was not previously needed in the case of the $\perp$-constraint. Finally, the procedure constructs the tree $F_{v_i,q}$ rooted at v_i which is used for piggybacking requests. Although this construction of the charging tree is based on ideas used by [2] for the single-server case, we need a new dual-fitting analysis in keeping with our analysis framework.

(iii) We need a finer control over the amount of dual raised in the call SimpleUpdate in line (4.8). Fix a call to SimpleUpdate(v_i, τ, ξ); hence $i \le i_0$ at this timestep. To prove dual feasibility, we want the increase in the dual objective function value to match the cost (with respect to vertex v_i) of the server movement into v_i during this iteration of the **while** loop. This server mass entering v_i is dominated by the server mass transferred to the request location v_0 by FullUpdate(v_{i_0+1}, τ), which is roughly γ/λ^{i_0}. The cost of transferring this server mass to v_i from its parent is $\lambda^i \cdot \gamma/\lambda^{i_0}$. We pass this value as an argument ξ to SimpleUpdate in line (4.8), indicating the extent to which we should raise dual variables in this procedure.
Moreover, we need to remember these values: for each node v and timestep $\tau \in \mathcal{R}^{ns}(v)$, we maintain a quantity $\Gamma(v, \tau)$, which denotes the total dual objective value raised for the constraints in $\mathfrak{L}^v(\tau)$. If these constraints were added by SimpleUpdate(v, τ, ξ), we define it as ξ; and finally, if they were added by FullUpdate(v, τ) procedure, this stays equal to the usual amount γ (as in the algorithm for k-Server). In case $\tau \in \mathcal{R}^s(v)$, this quantity is undefined.

(iv) The procedure is essentially the same as the version in §IV-C, with one change. Previously, if $\mathsf{activesib}(, \tau)$ was not empty, we could have had very little server movement, in case most of the dual increase was because of b^C. To avoid this, we now force a non-trivial amount of server movement. When the dual growth reaches γ, we stop the dual growth, but if there has been very little server movement, we transfer servers from active leaves below $\mathsf{activesib}(v_i, \tau)$.
The intuition for this step is as follows: in the SimpleUpdate(v_i, v_0, ξ) procedure for v_i below v, we need to match the dual increase (given by ξ) by the amount of server that actually moves into v_i. This matching is based on the assumption that at least γ/λ^h transfer happens during the FullUpdate procedure. By adding this extra step to FullUpdate, we ensure

that a roughly comparable amount of transfer always happens.

Again, most of the details are deferred to the full version of this paper.

VII. Closing Remarks

Our work suggests several interesting directions for future research. Can our LP extend to variants and generalizations of k-Server in the literature? One natural candidate is the hard version of the k-taxi problem. Another interesting direction is to exploit the fact that our LP easily extends to time-windows. The special case of k-ServerTW where $k = 1$ is known as *online service with delay*. While poly-logarithmic competitive ratios are known for this problem (and also follow from our current work), no super-constant lower bound on its competitive ratio bound is known. On the other hand, a sub-logarithmic competitive ratio is not known even for simple metrics like the line. Can our LP (or a variant) bridge this gap?

More immediate technical questions concern the k-ServerTW problem itself. For instance, can the competitive ratio of the k-ServerTW problem be improved from $\operatorname{poly}\log(n, \Delta)$ to $\operatorname{poly}\log(k)$? Another direction is to extend k-ServerTW to general delay penalties. Often, techniques for time-windows extend to general delay functions by reducing the latter to a prize-collecting version of the time-windows problem. Exploring this direction for k-ServerTW would be a useful extension of the results presented in this paper.

References

[1] N. Alon, B. Awerbuch, Y. Azar, N. Buchbinder, and J. Naor. The online set cover problem. *SIAM J. Comput.*, 39(2):361–370, 2009.

[2] Y. Azar, A. Ganesh, R. Ge, and D. Panigrahi. Online service with delay. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 551–563. ACM, 2017.

[3] Y. Azar and N. Touitou. General framework for metric optimization problems with delay or with deadlines. In D. Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 60–71. IEEE Computer Society, 2019.

[4] N. Bansal, N. Buchbinder, A. Madry, and J. Naor. A polylogarithmic-competitive algorithm for the k-server problem. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 267–276, 2011.

[5] N. Bansal, N. Buchbinder, A. Madry, and J. Naor. A polylogarithmic-competitive algorithm for the k-server problem. *J. ACM*, 62(5):40, 2015.

[6] Y. Bartal and E. Grove. The harmonic k-server algorithm is competitive. *J. ACM*, 47(1):1–15, 2000.

[7] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, New York, NY, USA, 1998.

[8] S. Bubeck, M. B. Cohen, Y. T. Lee, J. R. Lee, and A. Madry. k-server via multiscale entropic regularization. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 3–16. ACM, 2018.

[9] N. Buchbinder, A. Gupta, M. Molinaro, and J. S. Naor. k-servers with a smile: Online algorithms via projections. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 98–116. SIAM, 2019.

[10] N. Buchbinder and J. S. Naor. Online primal-dual algorithms for covering and packing. *Math. Oper. Res.*, 34(2):270–286, 2009.

[11] J. Fakcharoenphol, S. Rao, and K. Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *J. Comput. Syst. Sci.*, 69(3):485–497, 2004.

[12] A. Fiat, Y. Rabani, and Y. Ravid. Competitive k-server algorithms. *J. Comput. Syst. Sci.*, 48(3):410–428, 1994.

[13] E. F. Grove. The harmonic online k-server algorithm is competitive. In C. Koutsougeras and J. S. Vitter, editors, *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA*, pages 260–266. ACM, 1991.

[14] A. Gupta, A. Kumar, and D. Panigrahi. Caching with time windows. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 1125–1138. ACM, 2020.

[15] A. Gupta and V. Nagarajan. Approximating sparse covering integer programs online. *Mathematics of Operations Research*, 39(4):998–1011, 2014.

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

[17] M. S. Manasse, L. A. McGeoch, and D. D. Sleator. Competitive algorithms for server problems. *J. Algorithms*, 11(2):208–230, 1990.