Polylogarithmic Guarantees for Generalized Reordering Buffer Management

Matthias Englert
DIMAP and Department of Computer Science
University of Warwick
Coventry, UK
m.englert@warwick.ac.uk

Harald Räcke
Department of Informatics
Technical University of Munich
Munich, Germany
raecke@in.tum.de

Richard Stotz
Department of Informatics
Technical University of Munich
Munich, Germany
stotz@in.tum.de

***Abstract*— In the Generalized Reordering Buffer Management Problem (GRBM) a sequence of items located in a metric space arrives online, and has to be processed by a set of k servers moving within the space. In a single step the first b still unprocessed items from the sequence are accessible, and a scheduling strategy has to select an item and a server. Then the chosen item is processed by moving the chosen server to its location. The goal is to process all items while minimizing the total distance travelled by the servers.**

This problem was introduced in [Chan, Megow, Sitters, van Stee TCS 12] and has been subsequently studied in an online setting by [Azar, Englert, Gamzu, Kidron STACS 14]. The problem is a natural generalization of two very well-studied problems: the k-server problem for $b = 1$ and the Reordering Buffer Management Problem (RBM) for $k = 1$. In this paper we consider the GRBM problem on a uniform metric in the online version. We show how to obtain a competitive ratio of $\mathcal{O}(\log k(\log k + \log\log b))$ for this problem. Our result is a drastic improvement in the dependency on b compared to the previous best bound of $\mathcal{O}(\sqrt{b}\log k)$, and is asymptotically optimal for constant k, because $\Omega(\log k + \log\log b)$ is a lower bound for GRBM on uniform metrics.

I. Introduction

In the Generalized Reordering Buffer Management Problem (GRBM) a sequence of items located in a metric space arrive online, and have to be processed by a set of k servers moving within the space. In a single step the first b still unprocessed items from the sequence are accessible, and a scheduling strategy has to select an item and a server. Then the chosen item is processed by moving the chosen server to its location. The goal is to process all items while minimizing the total distance travelled by the servers.

This problem was introduced by Chan et al. [18] and has subsequently been studied in an online setting by Azar et al. [10]. It is a natural generalization of two very well-studied problems: the k-server problem for $b = 1$ [32] and the Reordering Buffer Management Problem (RBM) for $k = 1$ [33]. RBM and GRBM can be used for modeling context switching cost that occur in applications in many different areas ranging from production engineering through computer graphics to information retrieval (see e.g. [8], [10], [14], [30]). In this paper we consider the GRBM problem on a uniform metric in the online version in which the scheduling strategy does not know future items in the input stream but has to make its decisions only depending on the sequence of items it has seen so far.

Since the RBM problem is just the GRBM problem with $k = 1$ a lower bound of $\Omega(\log\log b)$ on the competitive ratio of RBM in uniform metrics (see [2]) applies also to GRBM. Another lower bound of $\Omega(\log k)$ stems from the fact that for $b = 1$ the GRBM problem is just the well known k-server problem which, for uniform metrics (the paging problem), has a well-known lower bound of $\Omega(\log k)$ [23].

The best upper bound on the competitive ratio for uniform metrics is due to Azar et al. [10] who gave a guarantee of $\mathcal{O}(\sqrt{b}\log k)$. In this paper we drastically reduce the dependency on b in the upper bound and present an algorithm that obtains a competitive ratio of $\mathcal{O}(\log k(\log k + \log\log b))$.

One potential approach for obtaining an algorithm for GRBM would be to extend an algorithm for RBM (e.g. the optimal algorithm for RBM by Avigdor-Elgrabli and Rabani [8]) to several servers. However, due to the complexity of this algorithm this approach seems challenging. Hence, we proceed differently. A major building block of our algorithm is a buffer management strategy for block-devices as introduced in [3]. The block device model is closely related to RBM on a uniform metric and can be described as follows. The metric space is formed by a (uniform) star and the items appear at the leaf vertices of this star, while the server is located at the center. In a single step the server may visit a leaf vertex v, process all items located at v, and then return to the center. Such a step is called a block operation (on vertex v) and induces a cost of 1. The difference between both models is that in RBM the server could stay at v and also process subsequent items that appear there. This seemingly subtle difference between the models means that the optimum cost for serving a request sequence may differ widely between the block-device model and RBM. An input consisting of a sequence of ℓ requests for a single vertex for example could be served with cost 1 in the RBM model, but would result in cost of ℓ/b in the block-device model.

Adamaszek et al. [3] gave an optimal $\mathcal{O}(\log\log b)$-

2575-8454/19/$31.00 ©2019 IEEE
DOI 10.1109/FOCS.2019.00012

competitive algorithm for scheduling block-devices. In contrast to the RBM problem, the block-device version of the problem can be easily formulated as a covering linear program, which allows a fairly straightforward application of the standard online primal-dual framework by Buchbinder and Naor [17].

The rough idea behind our approach is to allow the algorithm to perform block operations in addition to using the k servers. Note that in a uniform metric, we can simulate a block operation with constant cost (pick any of the k servers, move it to the desired vertex to process all items there, and immediately move it back to its original position) and therefore this addition does not really change the model. We then perform block operations according to the algorithm given in [3].

This actually means that we only apply the online primal-dual scheme to a part of the LP (some of the variables and some of the constraints). We then complement this with a second procedure which handles the remaining parts. This procedure solves the sub-problem of placing the k servers and will be a variant of the paging problem.

In other words, we solve our problem by splitting it into the part that is amenable to be dealt with by an online primal-dual scheme and the remaining part which is then tackled by other means. This way we are able to solve a problem which otherwise seems out of reach for the online primal-dual framework.

This split into two parts turns out to be useful for another reason as well. For one part we can utilize prior insights from paging and for the other part we can rely on prior insights from block-device scheduling. However, note that in particular the paging related sub-problem is still significantly more challenging than a standard paging problem (see e.g. [23], [34]) and requires new techniques. We briefly discuss this as part of Section III-B1.

Finally, we note that, as a special case (with $k = 1$), we obtain a new *optimal* algorithm for RBM which is very different from the one by Avigdor-Elgrabli and Rabani [8] further contributing to our understanding of this well-studied problem. Indeed, if we were only interested to obtain this result for $k = 1$, our algorithm and proof could be simplified significantly. For example, Section V is not needed at all for this case.

Further Related Work

There is a vast literature on the k-server and RBM problems, see for example [1], [12], [16], [29], [31] and [4], [13], [19], [20], [22], [24], [25], [26], [27], [28] respectively. We limit our discussion to the results most closely related to ours. The RBM problem for uniform metrics was introduced by Räcke et al. [33] who gave a deterministic algorithm with competitive ratio $\mathcal{O}(\log^2 b)$. This was subsequently improved by Englert and Westermann [21] to $\mathcal{O}(\log b)$, by Avigdor-Elgrabli and Rabani [9] to $\mathcal{O}(\log b/\log\log b)$ and by Adamaszek et al. [2] to $\mathcal{O}(\sqrt{\log b})$. The latter result is close to optimal due to a lower bound of $\Omega(\sqrt{\log b/\log\log b})$ for deterministic algorithms shown in the same paper.

For the randomized case Avigdor-Elgrabli and Rabani [8] developed an online algorithm with optimum competitive ratio $\mathcal{O}(\log\log b)$. Avigdor-Elgrabli et al. [6] extended this to the case of a star-metric space, i.e., where the points in the metric space are leaves of a star with arbitrary edge-length. For this scenario they obtain a competitive ratio of $\mathcal{O}((\log\log(b\Delta))^2)$, where Δ is the ratio between the length of the longest and shortest edge.

For the offline variant Asahiro et al. [5] and Chan et al. [18] independently established the NP-hardness of the problem on uniform metrics. Avigdor-Elgrabli and Rabani [7] designed a constant factor approximation algorithm for this offline setting.

A related problem is the Online Service with Delay problem (OSD-problem) introduced by Azar et al. [11]. In this problem items appear in a metric space but there is no upper bound on the number of unprocessed items at any given time (like the buffer-size b in GRBM). Instead the items come with delay penalty functions that ensure that eventually items have to be processed as otherwise the accumulated delay penalty would become too large. The goal is to minimize the total distance traveled by the servers plus the total delay penalty. For infinite delay penalties this problem reduces to paging (for a uniform metric) and to k-server (general metric), respectively.

Azar et al. give a competitive ratio of $\mathcal{O}(k\,\mathrm{polylog}\,n)$ for arbitrary metrics and show that for uniform metrics the problem is the paging problem, which gives a competitive ratio of $\mathcal{O}(\log k)$.

II. Model

In this section we give the precise definition of our model. As is common in the RBM problem we refer to the individual locations within the uniform metric space as *colors*. For notational convenience we assume that at any point in time the buffer can store $b + 1$ items (the difference between b and $b+1$ does not affect our asymptotic results). At the start of a step, a new item appears and is brought into the buffer. If this item has a color that currently has a server assigned to it we directly remove the item again and proceed to the next step. Otherwise, it might happen that now the buffer contains $b + 1$ items. Then we cannot proceed to the next time step but we first have to remove at least one item from the buffer. We can do this by selecting a color c and either perform a block operation for c or re-assign one of the k servers to c. In both cases all items of color c are removed from the buffer. The goal is to minimize the total number of operations. Our LP formulation is Primal as shown in Figure 1 with $\delta_c(0) = 0$ for all colors c.

We introduce variables $\delta_c(t)$, and $y_c(t)$ for every color c and every time-step t. Having $\delta_c(t) = 1$ means that there

$$
\begin{array}{lrcll}
\min & \sum_{t,c} y_c(t) + \sum_{t,c} x_c(t) & & & \\
\text{s.t.} & \sum_c \delta_c(t) & \leq & k & Z(t) \\
& \delta_c(t) & \leq & 1 & p_c(t) \\
& \delta_c(t-1) - \delta_c(t) & \leq & x_c(t) & B_c(t-1) \\
& \sum_c \big(\sum_{\tau \leq t} |E_c(\vec{v},\tau)|_t \cdot y_c(\tau) + |E_c(\vec{v},t)|_t \cdot \delta_c(t) \big) & \geq & |E(\vec{v},t)| - b & \alpha(\vec{v},t) \\
& y_c(t), x_c(t), \delta_c(t) & \geq & 0 &
\end{array}
\qquad \text{(Primal)}
$$

$$
\begin{array}{lrcll}
\max & \sum_{\vec{v},t} \big(|E(\vec{v},t)| - b \big) \alpha(\vec{v},t) - k \sum_t Z(t) - \sum_{c,t} p_c(t) & & & \\
\text{s.t.} & \sum_{\vec{v}, t \geq \tau} |E_c(\vec{v},\tau)|_t \cdot \alpha(\vec{v},t) & \leq & 1 & y_c(\tau) \\
& B_c(t-1) & \leq & 1 & x_c(t) \\
& -p_c(t) - Z(t) - B_c(t) + B_c(t-1) + \sum_{\vec{v}} |E_c(\vec{v},t)|_t \cdot \alpha(\vec{v},t) & \leq & 0 & \delta_c(t) \\
& \alpha(\vec{v},t), p_c(t), B_c(t), Z(t) & \geq & 0 &
\end{array}
\qquad \text{(Dual)}
$$

Figure 1. Primal and dual linear program used by our algorithm.

is a server at color c at the end of step t and $y_c(t) = 1$ means that we perform a block operation for c in step t. The $Z(t)$-constraints ensures that we use at most k servers while the $p_c(t)$-constraint ensures that we do not move more than one server to any location. If we want to move a server away from a color c, i.e., if $\delta_c(t-1) - \delta_c(t) = 1$, the $B_c(t-1)$-constraint ensures that we have to pay for this move by setting $x_c(t) = 1$. Note that in our uniform model we do not pay the distance that a server travels, but we simply pay 1 for every server move.

It remains to model the buffer constraint. For this we introduce the following notation. For two vectors $\vec{u}, \vec{v}$ of time steps (one entry for every color) we use $E_c(\vec{u}, \vec{v})$ to denote the items of color c that arrived in time-interval $(u_c, v_c]$ (we also allow $u_c = 0$ in the vector so that also the first item can be in such a set). We define $E(\vec{u}, \vec{v}) := \bigcup_c E_c(\vec{u}, \vec{v})$. We extend this notation to the case where the second parameter is a scalar with the meaning that all entries of the corresponding vector are equal to this scalar: $E_c(\vec{u}, t) := E_c(\vec{u}, \vec{v})$ with $v_c = t$ for all c (similarly for $E(\vec{u}, t)$).

LP Primal is not a faithful representation of the problem as an integral solution does not necessarily satisfy all constraints. However, we can modify any solution in such a way that the cost increases by a factor of at most 2 and the modified solution does satisfy the LP. Specifically, take any solution and whenever one of the k servers is moved away from color c (i.e. $\delta_c(t-1) = 1$ and $\delta_c(t) = 0$), we perform a block operation for color c (i.e., $y_c(t) = 1$). Note that this increases the cost by a factor of at most 2. Concretely, it may increase $\sum_{t,c} y_c(t)$ in the objective function, but it increases it by at most $\sum_{t,c} x_c(t)$.

A solution modified in this way, now satisfies $\alpha(\vec{v}, t)$ constraints. Consider the items in a set $E(\vec{v}, t)$. How many of these items are we removing by time t? Clearly, a block operation for color c at time τ removes at most $|E_c(\vec{v}, \tau)|$ of these items. A server that leaves at time $\tau < t$ also can remove at most $|E_c(\vec{v}, \tau)|$ of these items. However, since a leave event is co-located with a block operation we actually do not need to count the items that are removed by servers that left before time t as these are counted via the corresponding block operation. In case a server is located at c at time t this server can remove at most $|E_c(\vec{v}, t)|$ items. Hence,

$$\sum_c \big(\sum_{\tau \leq t} |E_c(\vec{v},\tau)| \cdot y_c(\tau) + |E_c(\vec{v},t)| \cdot \delta_c(t) \big) \geq |E(\vec{v},t)| - b$$

is a valid constraint as the left hand side is an upper bound on the number of items that we removed and the right hand side is a lower bound on the number of items from $E(\vec{v}, t)$ that we need to remove in order to fulfill the buffer constraint at time t. The $\alpha(\vec{v}, t)$ constraint in the primal linear program is a strengthening of this constraint where we define $|E_c(\vec{v}, \tau)|_t := \max\{0, \min\{|E_c(\vec{v}, \tau)|, |E(\vec{v}, t)| - b\}\}$. For integral solutions this does not change the feasibility. This is because for all y or δ variables that are 0, a change of the coefficient of that variable is immaterial. For a variable that is equal to 1, we may decrease the coefficient and therefore the left-hand side of the constraint. However, we at most decrease the coefficient to a value of $|E(\vec{v}, t)| - b$. Hence, if the corresponding variable is 1, the constraint remains satisfied since the left hand side is still at least $|E(\vec{v}, t)| - b$. On the other hand, we never increase any coefficient and therefore do not introduce new feasible solutions either. The

dual problem is Dual shown in Figure 1.

A. Modifying the Linear Program

A crucial ingredient for the analysis of the block-device scenario [3] and the RBM setting [8] is that a slight change in the buffer size only changes the competitive ratio by a constant factor. An analogous property holds for the GRBM problem and is proven in the appendix.

Theorem 1. *For any input sequence, the cost* $\mathrm{OPT}_{b'}$ *of an optimal offline solution utilizing a buffer of size* $b' = (1-\varepsilon)\cdot b$ *is at most a factor of* $(2+\varepsilon \ln b')/(1-\varepsilon)$ *larger than the cost* OPT_b *of an optimal offline solution utilizing a buffer of size* b*.*

This theorem allows us to change b into $b' = (1 - 1/\log b) \cdot b$, in LP Primal while only increasing the optimum solution value by a constant factor. In the new LP the capping is done w.r.t. b', i.e., $|E_c(\vec{v},\tau)|_t := \max\{0, \min\{|E_c(\vec{v},\tau)|, |E(\vec{v},t)| - b'\}\}$.

We then remove all $\alpha(\vec{v},t)$-constraints for pairs $(\vec{v},t)$ with $|E(\vec{v},t)| \leq b$, which cannot increase the optimum solution value. This means that now we have $|E(\vec{v},t)| - b' \geq b/\log b$ for every $\alpha(\vec{v},t)$-constraint.

Corollary 2. *The value of the modified LP is at most a constant factor larger than the cost* OPT_b *of an optimal offline algorithm using a buffer of size* b*.*

B. Introducing Dummy Steps

Purely to simplify the presentation of our algorithm and our proofs we introduce dummy time steps. These are time steps in which no new item arrives. We assume that between any two real time steps (in which an item arrives) there are an arbitrarily large number of dummy time steps. In fact, for ease of presentation, we allow the algorithm to make infinitesimal changes to variables during a time step. Alternatively, this can be thought of as sufficiently small discrete steps.

Our algorithm has to ensure that at the start of each real time step, there are at most b items stored in the buffer, so that the new item arriving in the real time step can be accommodated.

III. The Algorithm

Our algorithm consists of two main parts. The first part is the *base procedure* and only performs block operations. The base procedure can run on its own and produce a feasible solution to the problem. However, because it does not utilize the k servers and exclusively performs block operations, the cost of such a solution may be too large.

The second part of our algorithm is the *cost control procedure*. This procedure schedules additional block operations and, crucially, determines the placement of the k servers in such a way that the base procedure can produce a feasible solution with fewer block operations. The goal is to do this in such a way that, overall, our cost is greatly reduced.

The base procedure will (fractionally) increase y-variables in the primal in conjunction with producing assignments for the dual α-variables. It also has a randomized rounding procedure for the y-variables which produces the final schedule of block operations that ensure the buffer constraint (i.e. ensure that at the start of each real time step t no more than b items are stored in the buffer).

The cost control procedure integrally sets primal y-variables to 1 at certain points in time (in other words, performs a block operation) and assigns integer values to the δ- and x-variables (in other words, places the k servers). While the cost control procedure only sets variables in the LP integrally, and therefore does not require a rounding routine, the procedure itself is randomized and vaguely inspired by the well known randomized marking algorithm for paging.

The base procedure largely follows [3]. The only material difference is that we have to take the δ-variables of the LP into account. In [3], the last time step at which a block operation to a color occurred, played a special role because we know that all items of that color arriving before that operation are not stored in the buffer anymore. Now, we need to change this to say "last time step at which either a block operation to color c occurred, or a server was located at c".[1] This is reflected in the inclusion of δ-variables in the definition of a *proper constraint* in the following section.

Our main contribution is the cost control procedure as well as the insight that the problem can be nicely separated into these two parts which, arguably, simplifies the algorithm design and analysis.

A. The Base Procedure

We now start by discussing the base procedure. Before we describe the procedure in more detail, we introduce the notion of a *proper* constraint. This uses a constant scaling factor $\beta := 40$.

Definition 3. *For a given variable assignment, an* $\alpha(\vec{z},t)$ *constraint in the primal is called* proper *if the following holds for every color* c*:*

(i) $\sum_{z_c \leq \tau \leq t}(y_c(\tau) + \delta_c(\tau)) \geq 1/\beta$*,*
(ii) $\sum_{z_c < \tau \leq t}(y_c(\tau) + \delta_c(\tau)) < 2/\beta$*, and*
(iii) $y_c(\tau) + \delta_c(\tau) < 1/\beta$ *for all* $\tau \in \{z_c + 1, \ldots, t\}$*.*

Note that in particular, because of (iii), if $\delta_c(\tau) = 1$ or $y_c(\tau) = 1$, then for a proper constraint $\alpha(\vec{z},t)$ we must have $z_c \geq \tau$.

The base procedure maintains a vector $\vec{z}$ such that the primal $\alpha(\vec{z},t)$ constraint is proper for the current time step t. If the current $\alpha(\vec{z},t)$ constraint is violated, the procedure increases y-variables of the primal using the standard online

[1] This is only an intuition, because we actually have to deal with fractional block operations, but it gives an accurate idea of what the only difference to [3] is.

primal-dual approach by Buchbinder and Naor [17]. This continues until the constraint is not proper anymore (in which case $\vec{z}$ is updated) or until the constraint is satisfied (in which case the procedure progresses to the next real time step and receives the next item from the input). More precisely, if at any time, $\alpha(\vec{z}, t)$ is not proper (due to previous increases of δ- or y-variables), we recompute $\vec{z}$ by setting, for each color c for which one of the conditions is violated, z_c to the largest possible value (less or equal to t) such that $\sum_{z_c \leq \tau \leq t}(y_c(\tau) + \delta_c(\tau)) \geq 1/\beta$. While not exactly true, intuitively, z_c can be thought of as (roughly) the largest time such that every item of color c which arrived before that time has been completely removed from the buffer.

The base procedure also maintains a second vector $\vec{s}$. The default is that $s_c = z_c$ for every color c. However, at certain points, the cost control procedure in our algorithm can decide to set $s_c < z_c$ for some specific color and then "freeze" that value of s_c. That means that later updates of z_c are ignored for s_c and s_c remains fixed until we "unfreeze" it. To describe this state or event we say that color c is or becomes *active*. Once we unfreeze a color, s_c is set to be equal to z_c again. We then say that the color is *deactivated*. When we decide to make a color active, the precise value of s_c is chosen such that $|E_c(\vec{s}, \vec{z})| = \lceil f_{\max} \rceil$, with $f_{\max} := (b - b')/2k = b/(2k \log b)$. The precise way in which colors are activated and deactivated is described later.

After the base procedure updates values of the (fractional) y-variables, it uses an (online) randomized rounding procedure to determine which block operations to execute. For this, the procedure updates a probabilistic distribution μ over deterministic block operation schedules.

A complete summary of the base procedure for a time step t is shown in Procedure 1. Note that the procedure maintains two sets of the dual α-variables: α_1 and α_2. However it only uses α_1. The variables α_2 are only generated so they can be used in the cost control procedure to be described later. Accordingly, the base procedure maintains vector $\vec{s}$ in addition to $\vec{z}$, but again, this is only needed to set α_2-variables and is only used in the cost control procedure.

The rounding routine of the base procedure and its analysis are described in more detail in Sections B1 and B2. This is mostly identical to [3], but is included for completeness. Note that the rounding procedure only requires that the vector z increase monotonically and that before the next real time step the $\alpha(\vec{z}, t)$-constraint is not violated. In particular there is no need to satisfy all primal constraints in order for the rounding procedure to be successful.

B. The Cost Control Procedure

The cost control procedure reacts when items arrive or when dual α-variables are increased in Line 4 of the base procedure (Procedure 1).

The cost control procedure generates a sequence of what we call *explicit requests* for colors. An explicit request for a color c may be handled in one of two ways: either the cost control procedure ensures that one of the k servers is located at color c or the procedure performs a block operation for color c.

Procedure 1 base procedure for a time step t

1: **if** primal constraint $\alpha(\vec{z}, t)$ is not proper **then**
2: recompute $\vec{z}$ and $\vec{s}$
3: **if** primal constraint $\alpha(\vec{z}, t)$ is violated **then**
4: Increase $\alpha_1(\vec{z}, t)$ and $\alpha_2(\vec{s}, t)$ by the same infinitesimal amount $\mathrm{d}\alpha$
5: **for** each variable $y_c(\tau)$, $z_c < \tau \leq t$ **do**
6: $\Delta(\tau, c) := 0$
7: **if** $\sum_{\vec{v}, t' \geq \tau} |E_c(\vec{v}, \tau)|_{t'} \cdot \alpha_1(\vec{v}, t') == 1$ **then**
8: // dual constraint $y_c(\tau)$ is tight
9: **if** $(\sum_{\tau \leq i \leq t} y_c(i) < \frac{1}{\log^3 b})$ **then**
10: $\hat{y}_c(t) := \frac{1}{\log^3 b}$
11: **if** $\sum_{\vec{v}, t' \geq \tau} |E_c(\vec{v}, \tau)|_{t'} \cdot \alpha_1(\vec{v}, t') > 1$ **then**
12: // dual constraint $y_c(\tau)$ is violated
13: $\Delta(\tau, c) := \sum_{\tau \leq i \leq t} y_c(i) \cdot |E_c(\vec{z}, \tau)|_t \cdot \mathrm{d}\alpha$
14: $\forall c : \mathrm{d}y_c(t) := \hat{y}_c(t) + \max_{\tau : z_c < \tau \leq t} \Delta(\tau, c)$
15: $\forall c : y_c(t) := y_c(t) + \mathrm{d}y_c(t)$
16: adjust distribution μ to reflect new y-values
17: **else** // proper primal constraint $\alpha(\vec{z}, t)$ is satisfied
18: // rounding guarantees $\leq b$ items stored in buffer
19: proceed to the next real time step

Explicit requests are generated according to the following rules. We call $E_c(\vec{z}, t)$ the set of *live items* of color c. The rounding part of the base procedure ensures that all elements outside this set have been removed from the buffer. The set $E_c(\vec{s}, \vec{z})$ is called the set of *extra items* of color c (recall that $\vec{s} \leq \vec{z}$).

- If a color c is not active we issue an explicit request if at least $f_{\max}$ items of color c arrived since the last explicit request for color c.
  This rule guarantees that for inactive colors the number $|E_c(\vec{z}, t)|$ of live items of that color is at most $f_{\max}$. Otherwise, an explicit request would be issued which means either $\delta_c(t)$ or $y_c(t)$ is set to 1 to serve the request. In order for the $\alpha(\vec{z}, t)$-constraint to remain proper z_c is then increased to t giving $|E_c(\vec{z}, t)| = 0$.
- Suppose a color c is active, and let t' denote the time step at which the last explicit request for c was issued. Note that $z_c \geq t'$ because when the request appears, z_c is set to t' in order for the $\alpha(z, t')$-constraint to remain proper. We issue a new request if $|E_c(\vec{s}, t)| \geq (1+\gamma) \cdot |E_c(\vec{s}, t')|$. This means that since the most recent explicit request $\lceil \gamma \cdot |E_c(\vec{s}, t')| \rceil$ new items of color c appeared. Here, $\gamma := 1/128$.

Once a color received $D := \lceil \log_{1+\gamma}((2b+2)/(\gamma f_{\max}))\rceil + 1 = \Theta(\log k + \log\log b)$ requests within the current activity period, we do not issue further requests to this color.

The following claim shows that by defining the request generation as above we are guaranteed that for active colors c nearly all items in the set $E_c(\vec{s},t)$ are extra items.

Claim 4. *For active colors, the number $|E_c(\vec{z},t)|$ of live items of color c is at most a γ-fraction of the number of extra items $|E_c(\vec{s},\vec{z})|$.*

Proof: Suppose the color has not yet seen D requests within its current activity period. Then $|E_c(\vec{z},t)|$ ¿ $\gamma|E_c(\vec{s},\vec{z})|$ implies $|E_c(\vec{s},t)| = |E_c(\vec{s},\vec{z})| + |E_c(\vec{z},t)| > (1+\gamma)|E_c(\vec{s},\vec{z})| \geq (1+\gamma)|E_c(\vec{s},t')|$, which triggers a request and restores the property.

Right after the i-th request we have $|E_c(\vec{s},t)| = |E_c(\vec{s},\vec{z})| \geq (1+\gamma)^{i-1} f_{\max}$. For $i = D$, this means at least $(2b+2)/\gamma$ elements. Therefore in order to violate the claim we would require $|E_c(\vec{z},t)| \geq 2b+2$. Now consider the $\alpha(\vec{z},t)$-constraint in LP Primal:

$$\sum_c \Big(\sum_{\tau\leq t} |E_c(\vec{z},\tau)|_t \cdot y_c(\tau) + |E_c(\vec{z},t)|_t \cdot \delta_c(t)\Big) \geq |E(\vec{z},t)| - b \ .$$

This constraint may at most be violated by an additive 1, because it is the proper constraint used by the base procedure. Using $|E_c(\vec{z},\tau)|_t \leq |E_c(\vec{z},t)|$ gives

$$|E(\vec{z},t)| - b - 1 \leq \sum_c |E_c(\vec{z},t)| \cdot \Big(\sum_{\tau=z_c+1}^{t} y_c(\tau) + \delta_c(t)\Big) \leq \frac{2}{\beta}|E(\vec{z},t)| \ ,$$

where the last step follows from the second condition for proper constraints. This gives $|E(\vec{z},t)| \leq \frac{\beta}{\beta-2}(b+1)$, which gives a contradiction as $\beta > 4$. ■

To complete the description of our algorithm, we have to specify when colors are activated and deactivated and how exactly explicit requests are handled. We postpone the exact details to later sections and only give an overview here.

The algorithm is loosely inspired by the randomized marking algorithm for the paging problem [23]. It is based on partitioning the sequence of explicit requests, which it generates according to the above rules, into *phases* during which colors are *marked*. Each phase starts with all colors being unmarked and it ends once the k-th color has been marked. Then a new phase starts.

The marking of colors is guided by the activation/deactivation routine. At the start of the algorithm all colors are inactive. A request to an unmarked inactive color activates the color. When the color is deactivated later the color becomes marked *if* activation and deactivation lie in the same phase (a phase could start with a color in an active state).

1) Sketch of the analysis: The rough intuition behind the request generation and the process of activating and deactivating colors is as follows. First suppose that no requests are generated during the runtime of the algorithm. This means that the k servers are not used at all but all the work is done by the base procedure.

Intuitively this is fine because it means $|E_c(\vec{z},t)| \leq f_{\max}$ always holds (otherwise a request is generated) and therefore it is not possible that more than $f_{\max}$ items of the same color accumulate in the buffer at any time. Then using the k servers is not of much help anyway as they can only reduce the number of items by $kf_{\max}$, which is fairly small.

Now, suppose a request to a color c is issued. The $\lceil f_{\max}\rceil$ items that caused this request will be removed at a cost of 1 by either performing a block operation or moving a server to c. We activate the color (setting $s_c = z_c - \lceil f_{\max}\rceil$). Because of this, the following increases of $\alpha_2(\vec{s},t)$ give more dual profit (for color c) than the corresponding increases of $\alpha_1(\vec{z},t)$. We will use this *extra profit* to help pay for serving the explicit request that made c active (or indeed all explicit requests for c that appear within the same activity period).

While on the one hand freezing s_c increases the dual profit it also increases the violation of the dual LP. The exact rule for deactivating colors is postponed to later sections but essentially a color is deactivated once this additional violation reaches a certain threshold.

If there is only one server, serving the explicit requests is straightforward. Otherwise, we have a paging problem as we have to decide which server to move in case we do not perform a block operation. For a phase, we call the marked colors that were unmarked in the previous phase *new* colors. Colors (regardless whether marked or not) that were marked in the previous phase are called *stale colors*. Note that while our marking scheme is motivated by the analysis for marking algorithms in paging, there are slight differences, e.g., there could be phases with no new colors.

In Section V, we give an algorithm for placing the servers that always assigns a server to a marked color and analyze the performance of this algorithm. We essentially show that the algorithm has a cost of $\mathcal{O}(D\log k\sum_i n_i)$, where n_i is the number of new colors that appear in the i-th phase. This will then lead to a competitive ratio of $\mathcal{O}(D\log k) = \mathcal{O}(\log k(\log k + \log\log b))$ for our problem. The analysis draws insights from the analysis of paging algorithms but there is one crucial difference: the request sequence that we generate partially depends on the placement of servers. This means we have to carefully analyze this dependency, and design our algorithm accordingly. If it were not for this dependency we could adapt the paging algorithm by Blum et al. [15] to guarantee a cost of $\mathcal{O}((D+\log k)\sum_i n_i)$, which would then lead to an optimal competitive ratio of

$\mathcal{O}(D + \log k) = \mathcal{O}(\log k + \log\log b)$.

Another important ingredient for our analysis is to show a lower bound on the optimum cost of (very roughly) $\Omega(\sum n_i + \mathrm{cost}_{\mathrm{base}}/W_{\mathrm{base}})$, where $\mathrm{cost}_{\mathrm{base}}$ is the cost of the base procedure, and W_{base} is the violation of the $y_c(\tau)$-constraints in the dual solution using $\alpha_1(\vec{z},t)$-variables. This violation is only $\mathcal{O}(\log\log b)$ due to the analysis of the block-device problem in [3]. For completeness, the proof of this is also included as Lemma 28 in the appendix. We show that the violation generated by the $\alpha_2(\vec{s},t)$-variables is not much larger and then we show how to extend the assignment of $\alpha_2(\vec{s},t)$-variables to assignments to all variables in the dual such that the resulting dual profit is large. This is challenging as the online primal dual framework is mostly applied to packing or covering LPs and obtaining results for other LPs usually requires a very intricate analysis.

IV. Analysis

We view the base procedure as working in infinitesimal steps, where each step increases the sum of α_1-variables (and also the sum of α_2-variables) by $\mathrm{d}\alpha$. We use α to denote the sum of all α_1-variables for a particular point in time. Since the base procedure only increases a single α_1-variable and a single α_2-variable at a time we can view $\vec{s}$, $\vec{z}$, and t as functions in α, i.e., we can say that in step α the algorithm increases dual variables $\alpha_1(\vec{z}(\alpha),t(\alpha))$ and $\alpha_2(\vec{s}(\alpha),t(\alpha))$, or we can write that the total profit of the dual solution, when using α_2, is $\int_0^{\alpha_{\max}}(|E(\vec{s}(\alpha),t(\alpha)|-b')\mathrm{d}\alpha - k\sum_t Z(t) - \sum_{c,t} p_c(t)$, where $\alpha_{\max}$ is the sum of all α_2-variables in the end. However, in order to avoid notational clutter we just write $|E(\vec{s},t)|$ instead of $|E(\vec{s}(\alpha),t(\alpha)|)$ if the dependency on α is clear.

The crucial part of the analysis is to come up with a lower bound on the cost of an optimum solution. For this we construct a (nearly feasible) assignment to LP Dual with a large profit. The profit of the dual solution when using α_2-variables is $\int_0^{\alpha_{\max}}(|E(\vec{s},t)|-b')\mathrm{d}\alpha - \sum_{c,t} p_c(t) - k\sum_t Z(t)$. Since our algorithm consists of two parts it is natural to also split this profit into two parts:

$$\text{base budget: } \frac{1}{2}\int_0^{\alpha_{\max}} \Big(|E(\vec{z},t)| - b'\Big)\mathrm{d}\alpha$$

$$\text{cost control budget: } \int_0^{\alpha_{\max}} |E(\vec{s},\vec{z})|\mathrm{d}\alpha - \sum_{c,t} p_c(t) - k\sum_t Z(t) + k f_{\max}\cdot\alpha_{\max}$$

Recall that $|E(\vec{z},t)| - b' \geq b/\log b$ for an $\alpha(\vec{z},t)$-variable (See Section II-A). Therefore, we have $k f_{\max}\alpha_{\max} \leq \frac{1}{2}\int_0^{\alpha_{\max}}(|E(\vec{z},t)|-b')\mathrm{d}\alpha$, which shows that the two budgets sum up to at most the dual profit.

The total expected cost of our algorithm is proportional to the final objective value of the primal, i.e., $\mathcal{O}(\sum_{t,c} y_c(t) + \sum_{t,c} x_c(t))$. The reason is that $x_c(t)$ are set integrally and exactly express the cost for moving the k servers. The $y_c(t)$ may be fractional, but our randomized rounding routine ensures that the expected cost for block operations is $\mathcal{O}(\sum_{t,c} y_c(t))$. See Section B1.

The base procedure (Procedure 1) increases y-variables by a total of $\sum_{c,t}\mathrm{d}y_c(t)$ (see Lines 14 and 15). In Section B2 in the appendix we analyze this quantity and show that $\sum_{c,t}\mathrm{d}y_c(t) \leq 2\sum_{\vec{v},t}\big(|E(\vec{v},t)| - b'\big)\alpha_1(\vec{v},t)$ (Lemma 33). This means, up to a factor of 4, it is bounded by the base budget defined above. This analysis follows [3].

In the following the main focus is to show that the cost of the scheduling algorithm which deals with explicit requests (and which we have not completely defined yet) can be bounded by the optimal cost. It is therefore crucial to obtain a good lower bound on the dual profit. For this, obtaining a lower bound on the cost control budget is critical. We have to show how to assign values to $p_c(t)$'s, $Z(t)$'s and $B_c(t)$'s such that all dual constraints are fulfilled (or only violated by a small factor) and such that the budget becomes large. In particular, it is important that we assign values to dual variables such that the cost control budget is non-negative.

A. The Lower Bound

In this section, we develop our general lower bound technique that will allow us to show that our online algorithm is close to optimal. We first introduce some notation. Let for a subset $I \subseteq \{1,\ldots,T\}$ of (consecutive) time-steps $\alpha^-(I) := \inf\{\alpha \mid t(\alpha) \in I\}$ and $\alpha^+(I) := \sup\{\alpha \mid t(\alpha) \in I\}$. The profit collected for the cost control budget during some time-interval I is

$$\int_{\alpha^-(I)}^{\alpha^+(I)} |E(\vec{s},\vec{z})|\mathrm{d}\alpha - \sum_{t\in I}\sum_c p_c(t) - k\sum_{t\in I} Z(t) + k(\alpha^+(I) - \alpha^-(I))f_{\max} \ .$$

We use $\mathrm{act}_c(I)$ and $\overline{\mathrm{act}}_c(I)$ to denote the subset of the interval $[\alpha^-(I),\alpha^+(I)]$ during which color c is active and inactive, respectively. We define the *extra volume* collected during a time interval $I \subseteq \{1,\ldots,T\}$ for color c by

$$\mathrm{vol}_c^E(I) = \int_{\alpha^-(I)}^{\alpha^+(I)} |E_c(\vec{s},\vec{z})|\mathrm{d}\alpha = \int_{\mathrm{act}_c(I)} |E_c(\vec{s},\vec{z})|\mathrm{d}\alpha \ .$$

The equality follows because s_c and z_c are equal whenever c is not active. We define the *total volume* of a color c collected during interval I by

$$\mathrm{vol}_c^T(I) = \int_{\mathrm{act}_c(I)} |E_c(\vec{s},t)|\mathrm{d}\alpha \ .$$

We also define the *gap* $\Delta_c(I)$ between total volume and extra volume for a time-interval I:

$$\Delta_c(I) := \mathrm{vol}_c^T(I) - \mathrm{vol}_c^E(I) \leq \gamma\mathrm{vol}_c^E(I) \ . \qquad (1)$$

The inequality follows because the definition of the algorithm ensures that an active color has $|E_c(\vec{s},t)| \leq (1+\gamma)|E_c(\vec{s},\vec{z})|$ (Claim 4). We extend the above definitions to individual time steps, i.e., we define $\mathrm{vol}_c^T(\tau) := \mathrm{vol}_c^T(\{\tau\})$. This allows us to rewrite the cost control budget for a time interval as

$$\begin{aligned}&\sum_{t\in I}\sum_c \mathrm{vol}_c^E(t) - \sum_{t\in I}\sum_c p_c(t)\\ &- k\sum_{t\in I} Z(t) + k(\alpha^+(I)-\alpha^-(I))f_{\max}\ .\end{aligned} \tag{2}$$

We use $X_c(t) := \sum_{\vec{v}} |E_c(\vec{v},t)|_t \alpha(\vec{v},t)$, which is the term that appears in the $\delta_c(t)$-constraint of the dual. We have

$$\begin{aligned}X_c(t) &= \int_{\alpha^-(t)}^{\alpha^+(t)} |E_c(\vec{s},t)|_t \mathrm{d}\alpha\\ &\leq \int_{\alpha^-(t)}^{\alpha^+(t)} |E_c(\vec{s},t)|\mathrm{d}\alpha\\ &= \int_{\mathrm{act}_c(t)} |E_c(\vec{s},t)|\mathrm{d}\alpha + \int_{\overline{\mathrm{act}}_c(t)} |E_c(\vec{s},t)|\mathrm{d}\alpha\\ &\leq \mathrm{vol}_c^T(t) + (\alpha^+(t)-\alpha^-(t))f_{\max}\ .\end{aligned} \tag{3}$$

Now we are ready to present the lower bound approach. Suppose we are given a partial solution to LP Dual that only specifies values for α-variables but that ensures that the $y_c(\tau)$-constraints are violated by at most a factor of W. We describe how to extend this LP-solution in order to get a dual solution with a large profit, and, hence, a large lower bound for the problem.

Our lower bound is based on partitioning the sequence of time-steps into *phases* with the following properties. For $i \in \mathbb{N}_{>0}$ the i-th phase consists of time-steps $I_i := \{\tau_i, \ldots, \tau_{i+1}-1\}$ with $\tau_1 = 1$. Each phase contains exactly k *marked colors*. We use M_i to denote the set of marked colors in a phase and U_i to denote the set of the remaining (unmarked) colors. Each marked color c collects volume at least V during the phase, i.e.,

$$\mathrm{vol}_c^T(I_i) = \int_{\mathrm{act}_c(I_i)} |E_c(\vec{s},t)|\mathrm{d}\alpha \geq V\ , \qquad \text{(Cond. I)}$$

where V is a parameter to be defined later. Typically an unmarked color will collect rather small volume but for our analysis we only require an upper bound for the volume collected by an unmarked color c, i.e.,

$$\mathrm{vol}_c^T(I_i) = \int_{\mathrm{act}_c(I_i)} |E_c(\vec{s},t)|\mathrm{d}\alpha \leq 6V\ , \qquad \text{(Cond. II)}$$

holds for an unmarked color c.

We call a marked color *new* if it was unmarked in the previous phase. Let n_i denote the number of new colors in the i-th phase. The following constraint means that the gap between total volume and extra volume is small for marked colors:

$$\sum_{c\in M_i} \Delta_c(I_i) = \sum_{c\in M_i} \Big(\mathrm{vol}_c^T(I_i) - \mathrm{vol}_c^E(I_i)\Big) \leq 10\gamma n_i V\ . \qquad \text{(Cond. III)}$$

Lemma 5. *Suppose we are given an assignment to α-variables that violates $y_c(\tau)$-constraints by at most a factor of $W \geq V$, together with a partitioning of time-steps into phases according to the above constraints. Then there exists a constant $\xi > 0$ such that the cost of an optimum solution is at least*

$$\begin{aligned}&\frac{\xi}{W}\left(\sum_i n_i V + \sum_i \sum_{c\in U_i} \mathrm{vol}_c^T(I_i)\right)\\ &+ \frac{1}{2W}\int_0^{\alpha_{\max}} \Big(|E(\vec{z},t)| - b'\Big)\mathrm{d}\alpha\ .\end{aligned}$$

Proof: Fix a pair of consecutive phases i and $i+1$, and let $I := \{\tau_i, \ldots, \tau_{i+2}-1\}$ denote the time-steps within these phases. Observe that $|M_i \cup M_{i+1}| = k + n_{i+1}$. In the following we define values for the LP variables apart from $\alpha(\vec{s},t)$ in order to obtain a dual solution.

In a first step we use the $p_c(t)$-variables to *normalize* the volume. We define $\mathrm{vol}_c^N(t) := \mathrm{vol}_c^T(t) - p_c(t)$, and call $\mathrm{vol}_c^N(t)$ the *normalized* volume of color c at step t. Rewriting Equation 2 we see that the cost control budget for interval I is

$$\begin{aligned}&\sum_{t\in I}\sum_c \Big(\mathrm{vol}_c^E(t) - p_c(t)\Big)\\ &- k\sum_{t\in I} Z(t) + k(\alpha^+(I)-\alpha^-(I))f_{\max}\\ =&\sum_{t\in I}\sum_c \Big(\mathrm{vol}_c^N(t) - \Delta_c(t)\Big)\\ &- k\sum_{t\in I} Z(t) + k(\alpha^+(I)-\alpha^-(I))f_{\max}\ .\end{aligned} \tag{4}$$

We set $p_c(t)$'s to ensure that $\mathrm{vol}_c^N(I) = V$ for marked colors (colors in $M_i \cup M_{i+1}$), and that $\mathrm{vol}_c^N(I) = \min\{\mathrm{vol}_c^T(I), V\}$ for the remaining colors. Since this means we only have to "reduce" volume we can achieve this with non-negative $p_c(t)$-values.

Rewriting the $\delta_c(\tau)$-constraint in the dual gives that we have to fulfill $X_c(\tau) - p_c(\tau) \leq Z(\tau) + B_c(\tau) - B_c(\tau-1)$, for all $\tau \in I$. Equation 3 implies that it is sufficient to fulfill

$$(\alpha^+(\tau)-\alpha^-(\tau))f_{\max} + \mathrm{vol}_c^N(\tau) \overset{!}{\leq} Z(\tau) + B_c(\tau) - B_c(\tau-1)$$

instead. We will do this while ensuring $B_c(\tau) \leq V$ for all τ, i.e., $x_c(\tau)$-constraints are violated by at most a factor of V. We set $Z(\tau_i) = V + (\alpha^+(\tau_i) - \alpha^-(\tau_i))f_{\max}$ and $B_c(\tau_i) = \mathrm{vol}_c^N(\tau_i)$. Then the above constraint is fulfilled for $\tau = \tau_i$ as $B(\tau_i - 1) \leq V$. For the other time-steps in I we fulfill the constraint by setting $Z(\tau) = (\alpha^+(\tau) - \alpha^-(\tau))f_{\max}$ and by increasing the B_c-value by $\mathrm{vol}_c^N(\tau)$, i.e., setting $B_c(\tau) = B_c(\tau-1) + \mathrm{vol}_c^N(\tau)$. Since, $\mathrm{vol}_c^N(I) \leq V$ we can do this throughout the interval without the B_c-value increasing beyond V. We get that the cost control budget for interval I is

$$\text{control-budget}(I) = \sum_{\tau\in I}\sum_c \big(\mathrm{vol}_c^N(\tau) - \Delta_c(\tau)\big) - kV\ .$$

A color that is unmarked in both phases (i.e., a color not in $M_i \cup M_{i+1}$) has $\mathrm{vol}_c^T(I) \leq 12V$, and, hence, $\mathrm{vol}_c^N(I) \geq \frac{1}{12}\mathrm{vol}_c^T(I)$. Its gap is at most $\gamma\mathrm{vol}_c^T(I)$, as the gap of any color can at most be a γ-fraction of the total volume (Equation 1).

A color in $M_i \cup M_{i+1}$ has $\mathrm{vol}_c^N(I) = V$. Exactly $2n_{i+1}$ of these colors are unmarked in one of the phases, and in this case may have a gap of at most $6\gamma V$ because the volume of an unmarked color is at most $6V$. The gap generated by all marked colors throughout both phases is at most $10\gamma(n_i + n_{i+1})V$ because of the precondition that the gap on marked colors is small.

Plugging in these observations gives a cost control budget of at least

$$\begin{aligned}&\text{control-budget}(I) \geq \\ &n_{i+1} \cdot V + \sum_{c \in R}(\tfrac{1}{12} - \gamma)\mathrm{vol}_c^T(I) - 10\gamma n_i V - 22\gamma n_{i+1} V\end{aligned} \tag{5}$$

where R denotes the colors not in $M_i \cup M_{i+1}$. Let U_i denote the set of colors that are unmarked in the i-th phase. Then

$$\sum_{c \in U_i - R}(\tfrac{1}{12} - \gamma)\mathrm{vol}_c^T(I_i) \leq \tfrac{1}{2}n_{i+1}V \ ,$$

because $|U_i - R| \leq n_{i+1}$ as each of these colors is new in phase $i+1$. Plugging this into Equation 5 gives

$$\begin{aligned}&\text{control-budget}(I) \geq \\ &\tfrac{1}{2}n_{i+1}V + \sum_{c \in U_i}(\tfrac{1}{12} - \gamma)\mathrm{vol}_c^T(I) - 10\gamma n_i V - 22\gamma n_{i+1} V \ ,\end{aligned}$$

where we used

$$\begin{aligned}&\sum_{c \in R}\mathrm{vol}_c^T(I) \\ &= \sum_{c \in U_i}\mathrm{vol}_c^T(I_i) - \sum_{c \in U_i - R}\mathrm{vol}_c^T(I_i) \\ &\quad + \sum_{c \in U_i}\mathrm{vol}_c^T(I_{i+1}) - \sum_{c \in U_i - R}\mathrm{vol}_c^T(I_{i+1}) \\ &\geq \sum_{c \in U_i}\mathrm{vol}_c^T(I_i) - \sum_{c \in U_i - R}\mathrm{vol}_c^T(I_i) \ .\end{aligned}$$

If we now generate a lower bound by grouping phases as $(1,2),(3,4),(5,6),\ldots$ and another lower bound by grouping as $(2,3),(4,5),(6,7),\ldots$ and then take the average we obtain a lower bound for the overall cost control budget:

$$\begin{aligned}&\text{control-budget} \geq \\ &\sum_i \tfrac{n_i}{8}V + \sum_i \sum_{c \in U_i}\tfrac{1}{48}\mathrm{vol}_c^T(I_i) - \tfrac{n_1}{4}V - \sum_{c \in U_\ell}\tfrac{1}{48}\mathrm{vol}_c^T(I_\ell) \ .\end{aligned}$$

Note that Phase 1 does not appear as a second phase in a group and Phase ℓ (the last phase) does not appear as first phase in a group. Therefore, the corresponding contributions of these phases is subtracted in the above sum.

In order to obtain a lower bound, we divide the dual profit (i.e., cost control budget plus base budget) by the violation W. The terms depending on n_1 and U_ℓ are removed by exploiting the additional lower bound $\mathrm{OPT} \geq \frac{1}{2}(n_1 + |U_\ell|)$, which holds as every color needs to be accessed at least once. Averaging the two lower bounds and using $V < W$ gives the first term of the sum in the lemma for a sufficiently small constant $\xi > 0$. The second term is simply the base budget. ■

We will apply the above lower bound with $W = \Theta(V)$. The second part of this lower bound pays for the cost of the base procedure. The first sum gives a lower bound of $\Theta(\sum_i n_i)$, which will pay for the cost that the scheduling algorithm for the servers experiences on new and stale colors. The double sum pays for requests within activity periods of unmarked colors. The scheduling algorithm is presented in Section IV-C and Section V.

B. Deactivating Colors

The analysis of the base procedure in Section B2 in the appendix (Lemma 28) shows that the $y_c(\tau)$-constraints in the dual solution are only violated by a factor $W_{\mathrm{base}} = \mathcal{O}(\log\log b)$ when using $\alpha_1(\vec{z},\tau)$-variables. In the following we analyze the increase in violation that is caused by using $\alpha_2(\vec{s},\tau)$-variables. The difference stems from the fact that colors may be active ($s_c < z_c$), which means that increasing $\alpha_1(\vec{z},t)$ and $\alpha_2(\vec{s},t)$ by $\mathrm{d}\alpha$ increases the left-hand side of the $y_c(t)$-constraint by different amounts. The following constraint that guides the deactivation process guarantees that the increased violation is not too large.

deactivation constraint:
The volume collected by a color during a single activity period lies between V and $3V$. Formally, suppose that a color is active during interval $[\alpha_{\mathrm{start}}, \alpha_{\mathrm{end}}]$. Then

$$V \leq \int_{\alpha_{\mathrm{start}}}^{\alpha_{\mathrm{end}}} |E_c(\vec{s},t)|\mathrm{d}\alpha \leq 3V \ .$$

The reason that we do not simply deactivate a color once the collected volume reached the threshold V is as follows. Deactivation usually incurs a cost because the corresponding color becomes marked and a server is moved to this color. If however a server is already located at the color the deactivation is for free. Therefore, we sometimes delay deactivating a color c in the hope that a server is moved to c. Then deactivation is for free. The above constraint says that we do not delay the deactivation for too long, i.e., if the color collected volume $3V$ we deactivate it even if no server is located there.

The following lemma shows that the upper bound in the above constraint guarantees that we do not increase the dual violation by too much.

Lemma 6. *The $y_c(\tau)$-constraints in our dual solution (using $\alpha_2(\vec{s},t)$-variables) are violated by at most a factor of $W = W_{\text{base}} + 6V = \mathcal{O}(W_{\text{base}} + V)$.*

Proof: Fix a color c and a time-step τ. The color c can become active at different time-steps. However, only for two activity periods can the additional violation that is generated affect the constraint for τ. To see this assume for contradiction that the color is activated at $t_1 < t_2 < t_3$ and all three activity periods influence the constraint for τ. Let s_1, s_2, and s_3 denote the s_c-values that correspond to the activity intervals started at t_1, t_2 and t_3, respectively. For τ to be affected by the first activity period we must have $s_1 < \tau \leq t$ for a time step t within the first activity period. Clearly, $t \leq t_2$. Similarly, we get $s_3 < \tau \leq t'$ (for some t'), and, hence, $s_3 < \tau \leq t_2$. However, between two consecutive activation points of a color at least $f_{\max}$ elements of color c arrive, i.e., $f_{\max}$ elements between t_2 and t_3. However, when we activate the color at time t_3 we set s_c $(= s_3)$ such that $|E_c(\vec{s}, t_3)| = \lceil f_{\max} \rceil$. But this means $t_2 \leq s_3$, which gives a contradiction.

Now, we analyze the contribution to the LHS of a $y_c(\tau)$-constraint during one activity period that ranges from α_{start} to α_{end}. The contribution is

$$\int_{\alpha^-(\tau)}^{\alpha_{\text{end}}} |E_c(\vec{s},\tau)|_t \mathrm{d}\alpha \leq \int_{\alpha_{\text{start}}}^{\alpha_{\text{end}}} |E_c(\vec{s},t)| \mathrm{d}\alpha \leq 3V \ .$$

In order to get the total contribution to the LHS of a $y_c(\tau)$-constraint we also have to take into account the time when the color is inactive. The contribution for inactive periods is at most

$$\begin{aligned}\int_{\overline{\text{act}_c}} |E_c(\vec{s},\tau)|_t \mathrm{d}\alpha &= \int_{\overline{\text{act}_c}} |E_c(\vec{s},\tau)| \mathrm{d}\alpha \\ &= \int_{\overline{\text{act}_c}} |E_c(\vec{z},\tau)| \mathrm{d}\alpha \\ &= \int_{\overline{\text{act}_c}} |E_c(\vec{z},\tau)|_t \mathrm{d}\alpha \leq W_{\text{base}} \ .\end{aligned}$$

The equalities follow because when the color is inactive we have $|E_c(\vec{s},\tau)| = |E_c(\vec{z},\tau)| \leq f_{\max} \leq |E_c(\vec{z},t)| - b' \leq |E_c(\vec{s},t)| - b'$, i.e., capping does not have any effect. Combining the results for active and inactive periods the total violation is at most $W_{\text{base}} + 6V$ as desired. ∎

C. Constructing Phases

In the following we describe how to generate phases that fulfill the preconditions of Lemma 5.

Upon a request we may decide to make a color *active*. For an active color c, $|E_c(\vec{s},\vec{z})| > 0$ holds which means that the extra volume increases. This extra volume increases the dual profit and can therefore help to pay for server moves.

A phase starts with all colors being unmarked and all colors that were marked in the previous phase (i.e., *stale colors* for the new phase) being *inactive*. If there is a request to an unmarked color in an inactive state we activate the color. Non-stale colors are deactivated once they collected volume V during their current activity period. Stale colors are deactivated by a more complicated scheme described later.

When a color is deactivated we mark the color if the corresponding activity period lies completely within the phase. After we marked k colors a new phase starts. Note that while this marking scheme is motivated by the analysis for marking algorithms [23] in paging, there are slight differences, e.g., there could be a phase with no new colors.

Claim 7. *Marked colors collect at least total volume V during a phase. This gives Cond. I.*

Proof: A marked color has an activity period completely inside the phase. During this period it collects a volume of V due to the deactivation constraint. ∎

Claim 8. *Any color collects at most volume $6V$ during the phase. This gives Cond. II.*

Proof: A color can be activated at most once in a phase, because after deactivation within the same phase it will become marked. Hence, there can exist at most two activity periods that have a (partial) overlap with the phase. In each period the color can collect a volume of at most $3V$ due to the deactivation constraint. This gives that the total collected volume is at most $6V$. ∎

The proof that Cond. III holds, is split into two parts: The volume gap on new colors and the volume gap on stale colors. We first show the former.

Claim 9. *The volume gap on new colors is at most $\sum_{c \in N_i} \Delta_c(I) \leq 6\gamma n_i V$, where I denotes the time interval of the phase, N_i denotes the set of new colors in the phase, and $n_i = |N_i|$.*

Proof: This follows directly from Claim 8 and the fact that the gap is at most a γ-fraction of the total volume (Equation 1). ∎

The missing precondition for applying Lemma 5 is a bound on the volume-gap for stale colors. We will prove such a bound in Section V. Now, we first analyze the cost generated by the scheduling algorithm for new and unmarked colors.

D. Cost Analysis

The cost of an online algorithm for serving requests consists of two parts. On the one hand there is the *hit cost*, which is the cost we incur if none of the k servers is located at the requested color. In addition the algorithm may decide to change the placement of the servers. This part of the cost is called the *movement cost*. Note that these movements may occur even without an explicit server request. Naturally, we can split this cost among the different colors. For example, the movement cost for color c in the phase is the total number of times that we changed the server assignment for color c.

1) Paying for non-stale colors: Non-stale colors are either *new* colors, i.e., colors marked in this phase but not in the previous phase, or colors that are unmarked in both phases. We derive a bound on their cost by the following simple rule for moving servers.

scheduling constraint:
- a marked color is always assigned a server.
- the remaining servers are only distributed among stale colors.

From this rule it follows that non-stale colors are served by block operations while they are unmarked, and after this they do not incur any cost because they have a server.

Claim 10. *The total movement cost on non-stale colors during a phase is n_i, where n_i is the number of new colors in the phase.*

Proof: We only incur a movement cost if we mark a color. Non-stale colors that become marked are new colors. ∎

2) Paying for stale colors: In Section V, we present a scheduling algorithm for the stale colors with the following properties.

Lemma 11. *For $V \geq D$ there is a randomized scheduling algorithm for stale colors that obeys the scheduling and deactivation constraint and guarantees that the volume gap on stale colors in Phase i is at most $3\gamma(1+\gamma)n_iV$. The expected cost on stale colors generated by this algorithm is only $\mathcal{O}(V \log k \mathbb{E}[\sum_i n_i])$.*

3) Combining the results: The bound on the volume gap from Lemma 11 (at most $3\gamma(1+\gamma)n_iV$ on stale colors) and Claim 9 (at most $6\gamma n_iV$ on new colors) directly imply (Cond. III). (Cond. I) and (Cond. II) follow from Claim 7 and Claim 8, respectively. Hence, we can use the lower bound on the optimum cost provided by Lemma 5:

$$\frac{\xi}{W}\left(\sum_i n_iV + \sum_i \sum_{c\in U_i} \mathrm{vol}_c^T(I_i)\right) + \frac{1}{2W}\int_0^{\alpha_{\max}} \Big(|E(\vec{z},t)| - b'\Big)\mathrm{d}\alpha \ . \tag{6}$$

Now we choose $V = D$, then, due to Lemma 6, $W = W_{\text{base}} + 6V = \Theta(\log\log b + D) = \Theta(D)$. In the following we argue that different cost components of our algorithm are bounded by (6) up to a factor of $\mathcal{O}(D \log k)$.

- The fractional cost of the base procedure (see Lemma 33) is at most $2\int_0^{\alpha_{\max}} \big(|E(\vec{z},t)| - b'\big)\mathrm{d}\alpha$, and can be amortized against the lower bound at a loss of a factor of $\mathcal{O}(W) = \mathcal{O}(D)$.
- The expected cost of the cost control procedure for stale colors is $\mathcal{O}(\log k \mathbb{E}[\sum_i n_i]V)$ due to Lemma 11. As the lower bound Equation 6 holds for every partitioning of the request sequence into phases generated by our algorithm, it implies a lower bound of $\mathbb{E}[\sum_i n_i]$. The cost can therefore be amortized against the lower bound at a loss of a factor of $\mathcal{O}(V \log k) = \mathcal{O}(D \log k)$.
- The movement cost of cost control procedure for non-stale colors is $\sum_i n_i$ due to Claim 10, and can be amortized against the lower bound at a loss of a factor of $\mathcal{O}(1)$.
- The hit cost of the cost control procedure for non-stale colors are at most $D\sum_c a_c$, where a_c denotes the number of non-stale activity periods for color c, i.e., activity periods for which c is a non-stale color at the point of activation. This is because a single activity period contains at most D requests. Now we show that this is larger than Equation 6 by at most a factor of $\mathcal{O}(D)$.
  Indeed, we claim that Equation (6) is in $\Omega(\sum_c a_c)$. Fix a non-stale activity period P for color c and assume that Q is the activity period following P. If P or Q finishes with a marking operation on color c then the first of these marking operation makes c into a new color for the respective phase. This new color contributes $\Omega(V/W) = \Omega(1)$ to the lower bound. Otherwise, c will be unmarked in all phases that contain parts of P. Hence, the total volume (which is at least V) collected during period P contributes to the lower bound and we again get a contribution of $\Omega(V/W) = \Omega(1)$. Since, we get a contribution of $\Omega(1)$ from any two consecutive activity periods the claim follows.

The randomized rounding procedure used to round the fractional $y_c(t)$-values has expected cost $\mathcal{O}(\sum_{c,t} y_c(t))$ (see Section B1).

Theorem 12. *There is an algorithm with competitive ratio $\mathcal{O}(D \log k) = \mathcal{O}(\log k(\log k + \log\log b))$ for the generalized reordering buffer management problem on uniform metrics.*

V. Serving Stale Colors

In this section we prove Lemma 11. For this, we describe how the algorithm schedules the servers among stale colors.

Note that for $k = 1$ the phase ends after the first color has been marked, hence the (single) stale color always has a server assigned to it. The expected service cost is therefore n_i (which is either 0 or 1), and the volume gap generated on stale colors is 0. This reproves the result for the Reordering Buffer Management problem due to Avigdor-Elgrabli and Rabani [8].

A major difficulty for the case $k > 1$ is that the collection of volume (and thereby also the increase in the volume gap) is to a large extend guided by the increase of α through the base procedure. Unfortunately, the actions of the base

procedure depend on $|E(\vec{z},t)|$, which, in turn, depends on the distribution of servers. This means that the actions of the scheduling algorithm, responsible for the server distribution, may influence the increase in α and also the request sequence itself. This makes the analysis of the randomized scheduling algorithm much more challenging and it requires a careful modeling of the adversary.

For the remainder of the section, consider a fixed adversary whose power will be defined precisely in Section V-A. A *configuration* C of the algorithm consists of a time step t along with the algorithm's decisions up to time t. At any point in time, our algorithm chooses a new configuration based on its current configuration, the adversary's actions and some random bits. Given that the adversary is fixed, the algorithm can therefore be viewed as a tree of possible choices with root C_0, the initial configuration. The vertices of this tree are all possible configurations. There is a directed edge between configurations C and C' if the algorithm, when in configuration C, decides with non-zero probability to have C' as its next configuration. Let $p(C \to C')$ denote this *transition probability*.

Given a fixed adversary, the tree is a Markov chain. This gives that we can extend the notation $p(C \to C')$ to arbitrary configurations: $p(C \to C')$ is 0 if no path from C to C' in the tree exists; otherwise, it is the product of transition probabilities on that path.

We say that C' is *reachable* from C, writing $C \to C'$, if and only if $p(C \to C') > 0$. As a simple corollary, we have

$$pt(C,X) \cdot pt(X,C') = p(C \to C') \ , \tag{7}$$

for configurations C, C', and X with $C \to X$ and $X \to C'$ (this means X lies on the path from C to C' in the tree).

If $C \to C'$, let $\mathrm{cost}(C \to C')$ denote the cost incurred by the algorithm when transitioning from configuration C to configuration C'; otherwise $\mathrm{cost}(C \to C') = 0$. Let $n(C)$ denote the number of new colors marked in the phase of C up to the time step of C.

Our high-level strategy is to define a special set $\mathcal{S}$ of *event configurations* so that the algorithm encounters only few such configurations, yet the expected cost for transitioning to the next event configuration is small. For event configurations $C, C' \in S$, we write $C \mapsto C'$ (say C' is *successor* of C), if $C \to C'$ and there is no $X \in \mathcal{S}$ with $C \to X$ and $X \to C'$. The precise definition of event configurations is deferred to Section V-B. For now, we only require them to fulfill the following properties.

(P1) The algorithm experiences at most $\mathcal{O}(D \log k)$ event configurations per phase.

(P2) For any $C \in \mathcal{S}$, $\sum_{C' \in \mathcal{S}} p(C \mapsto C') \,\mathrm{cost}(C \mapsto C') = \mathcal{O}(n(C))$.

Using these properties, we show an upper bound on the algorithm's expected cost on stale colors.

Lemma 13. *The expected hit cost and movement cost on stale colors is $\mathcal{O}(D \log k)\mathbb{E}[\sum_i n_i]$, where n_i is the number of new colors in phase i.*

Proof: We assume wlog. that each phase ends in an event configuration and let $\mathcal{P}$ denote those *phase configurations* that end a phase. Similarly to the event configurations, we say $C' \in \mathcal{P}$ is *phase successor* of phase configuration C if $C \to C'$ and there is no $X \in \mathcal{P}$ with $C \to X$, $X \to C'$; we write $C \overset{p}{\mapsto} C'$. Let $p(C \overset{p}{\mapsto} C') = p(C \to C')$ if $C \overset{p}{\mapsto} C'$ and 0 otherwise. Furthermore, let $\mathcal{B}$ denote the configurations ending the algorithm.

The expected cost of the algorithm is, by definition, $\sum_{B \in \mathcal{B}} p(C_0 \to B) \,\mathrm{cost}(C_0 \to B)$. We first show this is at most $\mathcal{O}(D \log k) \sum_{P \in \mathcal{P}} p(C_0 \to P) n(P)$. We then prove that $\sum_{P \in \mathcal{P}} p(C_0 \to P) n(P) = \mathbb{E}[\sum_i n_i]$.

Claim 14. *$\sum_{B \in \mathcal{B}} p(C_0 \to B) \,\mathrm{cost}(C_0 \to B)$ is at most $\mathcal{O}(D \log k) \sum_{P \in \mathcal{P}} p(C_0 \to P) n(P)$.*

Proof: We rewrite the cost of transitioning from C_0 to $B \in \mathcal{B}$ using all intermediate event configurations, which gives

$$\mathrm{cost}(C_0 \to B) = \sum_{\substack{C \in \mathcal{S} \\ C \to B}} \sum_{C' \in S} \mathrm{cost}(C' \mapsto C) \ .$$

Plugging this into the expected cost and changing the order of summation gives

$$\begin{aligned}
&\sum_{B \in \mathcal{B}} p(C_0 \to B) \sum_{\substack{C \in \mathcal{S} \\ C \to B}} \sum_{C' \in S} \mathrm{cost}(C' \mapsto C) \\
&= \sum_{C' \in \mathcal{S}} \sum_{B \in \mathcal{B}} \sum_{\substack{C \in \mathcal{S} \\ C \to B}} p(C_0 \to B) \,\mathrm{cost}(C' \mapsto C) \\
&= \sum_{C' \in \mathcal{S}} p(C_0 \to C') \sum_{B \in \mathcal{B}} \sum_{\substack{C \in \mathcal{S} \\ C \to B}} p(C' \to B) \,\mathrm{cost}(C' \mapsto C) \\
&= \sum_{C' \in \mathcal{S}} p(C_0 \to C') \sum_{C \in \mathcal{S}} \mathrm{cost}(C' \mapsto C) \sum_{\substack{B \in \mathcal{B} \\ C \to B}} p(C' \to B) \ .
\end{aligned}$$

The second equation used (7), the third equation changed the order of summation. As C is reachable from C', applying (7) shows that $\sum_{B \in \mathcal{B}, C \to B} p(C' \to B) = p(C' \mapsto C)$. The expected hit cost is therefore

$$\begin{aligned}
&\sum_{C' \in \mathcal{S}} p(C_0 \to C') \sum_{C \in \mathcal{S}} p(C' \mapsto C) \,\mathrm{cost}(C' \mapsto C) \\
&\leq d \cdot \sum_{C' \in \mathcal{S}} p(C_0 \to C') n(C') \ ,
\end{aligned}$$

by Property 2, where d is some constant from the $\mathcal{O}$-notation.

In order to relate this to the phase configurations, we multiply each term with $1 = \sum_{P \in \mathcal{P}} p(C \overset{p}{\mapsto} P)$, which

gives

$$\begin{aligned} & d \cdot \sum_{C \in \mathcal{S}} p(C_0 \to C) n(C) \sum_{P \in \mathcal{P}} p(C \overset{p}{\mapsto} P) \\ = \; & d \cdot \sum_{P \in \mathcal{P}} \sum_{C \in \mathcal{S}} p(C_0 \to C) n(C) p(C \overset{p}{\mapsto} P) \\ \leq \; & d \cdot \sum_{P \in \mathcal{P}} n(P) \sum_{C \in \mathcal{S}} p(C_0 \to C) p(C \overset{p}{\mapsto} P) \\ \leq \; & d \cdot D \log k \sum_{P \in \mathcal{P}} n(P) p(C_0 \to P) \ . \end{aligned}$$

Here, the first inequality uses that the number of new colors only increases during a phase. The second inequality exploits Property 1, as at most $D \log k$ event configurations are encountered in any phase. ■

We now claim that $\sum_{c \in \mathcal{P}} p(C_0 \to P) n(P) = \mathbb{E}[\sum_i n_i]$. Together with the above upper bound on the expected cost, this gives the lemma.

Fact 15. $\sum_{c \in \mathcal{P}} p(C_0 \to P) n(P) = \mathbb{E}[\sum_i n_i]$.

Proof: The expected number of new colors seen throughout the algorithm is

$$\begin{aligned} \mathbb{E}[\textstyle\sum_i n_i] &= \sum_{B \in \mathcal{B}} p(C_0 \to B) \sum_{\substack{P \in \mathcal{P} \\ P \to B}} n(P) \\ &= \sum_{B \in \mathcal{B}} \sum_{P \in \mathcal{P}} p(C_0 \to P) p(P \to B) n(P) \ . \end{aligned}$$

After changing the order of summation, this is

$$\begin{aligned} & \sum_{P \in \mathcal{P}} n(P) p(C_0 \to P) \sum_{B \in \mathcal{B}} p(P \to B) \\ = & \sum_{P \in \mathcal{P}} n(P) p(C_0 \to P) \end{aligned}$$

The equality holds because any configuration reachable from C_0 must eventually reach the end of the algorithm. ■

■

A. Model of the Adversary

Before we define the event configurations and the algorithm's strategy, this section gives a cleaner model for the adversary. We consider a single phase starting in a single state of the algorithm, i.e., the server positions and fractional α-variables are fixed for the start of the phase. We re-number the time steps with $t = 1$ being the first time step of the phase. The request sequence and the increase in α both depend on the color sequence fed to the base procedure; it is convenient to model it as a specific type of adversary that we define next.

We assume that the adversary selects an order among the explicit server requests on the stale colors. Below, we show that any request outside of this sequence may be ignored. After choosing the sequence, the adversary can do one of the following operations in each step t:

- **request**: Issue the next request from the predefined sequence.
- **volume increase**: Increase α by $d\alpha$; this increases the volume gap on active colors that are not assigned a server.

In addition there are the following events that the adversary cannot influence directly but only indirectly by altering the volume collected by a color

- **marking of a stale color**: In this case one server has to be moved away from the stale colors to be assigned to the newly marked stale color.
- **marking of a new color**: Mark a non-stale color c; in this case one server has to be moved away from the stale colors to be assigned to color c for the remainder of the phase; this increases the number $n(t)$ of new colors that appeared in the phase up to time t.

For each of the above actions the adversary *knows the random choices of the algorithm up to Step t*. The following claim shows that we can indeed assume that the adversary has to specify the order of requests at the start of the phase.

Claim 16. *After the start of the phase the order of requests to unmarked stale colors during the phase is not influenced by actions of the scheduling algorithm.*

Proof: As a reminder the rules for generating requests are as follows:

1) an inactive color gets a request if it has seen at least $f_{\max}$ requests since the most recent explicit server request
2) an active color gets a request if $|E_c(\vec{s}, t)| \geq (1 + \gamma)|E_c(\vec{s}, t')|$, where t' is the time step of the most recent explicit request to the color AND at most $D-1$ requests have been issued for c during the current activity period.

Recall that stale colors are inactive at the beginning of the phase. If at the start of the phase we know the number of items that arrived for every color since the most recent request, we can calculate how many items need to arrive in order for a request of Type 1 to be generated (which activates the color). As long as the color stays active further requests of Type 2 only depend on the number of items received for the respective color. When the color gets deactivated it becomes marked and, hence, further requests to this color are not of interest. This shows that requests to unmarked stale colors only depend on the number of items and, thus, their order cannot be influenced once the input sequence of items is fixed, which has to be done at the start of the algorithm. ■

At first glance the above claim might appear too weak to guarantee that the adversary is strong enough as the order of requests to marked colors *can* actually be influenced by actions of the block-device algorithm. The reason is that marking/deactivating a color changes how requests to the color are generated, and thus this can influence order of requests. However, our algorithm will fulfill the following

property which shows that giving the adversary the additional power of freely issuing requests to marked colors does not change the model.

irrelevant request property:
Adding/removing requests to marked colors does not influence the scheduling decisions or the cost of the algorithm.

The above property is an immediate consequence of the fact that a marked color is always assigned a server. This means we do not have to move any servers if a marked color is requested and our algorithm will not do so.

In order to get a cleaner model we restrict the choice of the online algorithm in each step. We add (many) dummy time steps, where in each time step the adversary has the choice between a limited number of actions. More concretely we have the following types of steps:

- *request time steps* $t \in \mathcal{T}_r$
  The adversary issues the next request in the predetermined sequence. The adversary *cannot* skip this step.
- *marking time steps* $t \in \mathcal{T}_n$ for new colors
  The adversary increases n or skips the action.
- *volume time steps* $t \in \mathcal{T}_v$
  The adversary either increases α by $\mathrm{d}\alpha$ or skips the action.

The above model can be obtained by fixing the time steps that should be request time steps and by adding sufficiently many time steps of each type between any two consecutive request time steps.

B. Bounding the Cost per Phase

At any time step, let ℓ_r denote the number of unmarked stale colors with at most r requests, as seen by the algorithm. Let n be the number of new colors that have been marked so far. The current *round* r^* is $\min\{r \mid \ell_r \geq n\}$. Observe that r^* never decreases, as ℓ_r never increases and n never decreases. Within round r^*, a color c is *free* if it fulfills three conditions: c must have seen exactly r^* requests, it must be unmarked and there must be a server located at it.

The algorithm's movement strategy is the following. The algorithm keeps the current server distribution unless

- a color c without server obtains its (r^*+1)th request, or
- a new color c is marked, or
- a stale color c without server is marked.

In these cases, the algorithm picks a free color c' uniformly at random and moves the server from c' to c.

Observation 17. *Any stale color with more than r^* requests is assigned a server. Any unmarked stale color with less than r^* requests is not assigned a server.*

Proof: First, observe that whenever a color is marked, it obtains a server. As marked colors never become free, a marked color keeps that server until the end of the phase.

For any time step t, let $r^*(t)$ denote the value of r^* and $r(c,t)$ the number of requests to color c up to time t. We show that $r(c,t) > r^*(t)$ implies that color c has a server. Let $t' < t$ be the last time step at which $r(c,t') = r^*(t')$. At time step $t'+1$, color c must have obtained a server. Color c was not a free color during the interval $[t'+1, t]$ as it had more than r^* requests. The server has therefore not left color c during $[t'+1, t]$.

Finally, observe that right before r^* increases, $\ell_{r^*} = n$. We know that every color with at least r^*+1 requests has a server, and n stale colors have no server. This means that right before r^* increases, no unmarked stale color with at most r^* requests has a server. As a color with less than r^* requests only obtains a server by obtaining more requests or becoming marked, unmarked stale colors with fewer than r^* requests cannot have a server. ∎

In order to get a bound on the volume gap, we proceed as follows. Observe that a color that is assigned a server does not increase its volume gap as the increase in total volume matches the increase in extra volume. In the following, we keep track of the volume that is collected by a color while it is not assigned a server. We call this the *volume cost* of the color. The volume gap is then at most a γ-fraction of the volume cost due to Equation (1).

Claim 18. *The volume cost on stale colors is at most $3(1+\gamma)V \cdot n$, where n is the number of new colors of the phase.*

Proof: Let n denote the number of new colors in the phase given a specific set of random choices. Observe that for $n = 0$ a stale color is always assigned a server. Therefore, in this case the volume cost on stale colors will be 0.

Hence, assume $n > 0$. In this case there do exist n stale colors that are not marked in the phase. Let U denote this set of colors and observe that each of these colors collects at most volume $3V$ during the phase as otw. it would be deactivated and would directly become marked (unless the phase ends at this point). Hence $\sum_{c \in U} \mathrm{vol}_c(I) \leq 3nV$. We now show that in each infinitesimal step the volume cost increases roughly by the volume collected by colors in U during the step.

Fix a time step and let M denote the set of active stale colors that are currently missing a server. The volume cost increases by

$$\sum_{c \in M} |E_c(\vec{s}, t)| \mathrm{d}\alpha \ .$$

On the other hand the volume collected by colors in U increases by $\sum_{c \in U} |E_c(\vec{s}, t)| \mathrm{d}\alpha$.

Fact 19. *The assignment of servers guarantees that $\sum_{c \in M} |E_c(\vec{s}, t)| \leq (1+\gamma) \sum_{c \in U} |E_c(\vec{s}, t)|$.*

Proof: First observe that $|M| = n$ and $|U| = n \geq n$. We sort the colors in M and U according to the number of requests they have received so far.

We now map the i-th color in M to the i-th color in U (if it exists). In the following we show that for a pair (c_m, c_u) with a color from $c_m \in M$ that is mapped to $c_u \in U$ we have $|E_{c_m}(\vec{s}, t)| \leq (1+\gamma)|E_{c_u}(\vec{s}, t)|$. This implies the fact.

Clearly, c_m has seen as most as many requests as c_u because the holes are distributed among colors with fewest requests by Observation 17. Let for $i \leq D$, s_i denote the number of extra items *right after* the i-th request. Due to the way requests are generated there are exactly $\lceil f_{\max} \rceil$ items in $E_c(\vec{s}, t)$ at the time of the first request (i.e., $s_1 = \lceil f_{\max} \rceil$). As long as $i \leq D$ we have $s_i = \lceil (1+\gamma)s_{i-1} \rceil$. Hence,

$$\begin{aligned} |E_{c_m}(\vec{s}, t)| &\leq s_{r(c_m)+1} - 1 \leq \lceil (1+\gamma)s_{r(c_m)} \rceil - 1 \\ &\leq (1+\gamma)s_{r(c_u)} \leq (1+\gamma)|E_{c_u}(\vec{s}, t)| \ , \end{aligned}$$

where $r(c)$ denotes the number of requests to color c. ■

Since the volume of elements in U is at most $3nV$ we get that the volume cost on stale colors is at most $3(1+\gamma)nV$. ■

We now define the event configurations. The beginning of the algorithm is the first event configuration. After that, event configurations are defined inductively. Any configuration is an event configuration if either n has doubled since the last event, or the number of free colors has decreased by a factor of $1/4$, or r^* increases. Furthermore, the end of a phase is an event configuration. We now prove that this definition fulfills properties 1 and 2.

Fact 20. *There can be at most $5D \log k$ event configurations in a phase.*

Proof: The number of new colors is at most k, hence it can double at most $\log k$ times. As any color receives at most D requests, $r^* \leq D$, hence at most D event configurations mark the end of a round. Within a round, the number of free colors decreases by a factor of at most $3/4$ between event configurations, hence there are at most $\log_{4/3} k < 4 \log k - 1$ such event configurations per round. Finally, there is one event configuration for the end of the phase. ■

The movement cost for marking stale colors is bounded using the following technique. According to the deactivation constraint, a color that collected volume V is *available* for marking and *must* be marked before it collects volume more than $3V$. If a color is available for marking and has a server assigned to it, we simply mark the color. This does not induce any movement cost, as the server assignment does not change. If a color reaches volume $3V$ without a server assigned to it, we mark the color and reassign a server from a free color to it. This *forced marking* induces cost 1; however, the next claim bounds the number of these forced marking operations.

Claim 21. *For any event configuration C, there can be at most $4n(C)$ forced marking operations before the next event configuration.*

Proof: Observe that a color that experiences a forced marking operation has collected volume $2V$ since it became available for marking. During this time it has never been assigned a server as it would have been marked otherwise. This means that a color that experiences a forced marking operation has volume cost at least $2n(C)V$. The total volume cost of the phase is only $3(1+\gamma)n(P)V$ by Claim 18, where P is the event configuration ending the phase. The number of new colors can only double between events, so there can be at most $2\frac{3}{2}(1+\gamma)n(C) \leq 4n(C)$ forced marking operations. ■

The following claim proves Property 2.

Claim 22. *For any $C \in \mathcal{S}$, $\sum_{C' \in \mathcal{S}} p(C \mapsto C') \operatorname{cost}(C \mapsto C') = \mathcal{O}(n(C))$.*

Proof: By definition of the event configurations, the number of new colors doubles at most between C and C' if $C \mapsto C'$. Instead of arguing about the movement cost of servers, it is therefore convenient to analyze the cost associated with each of the *holes*, distributed among the stale colors. We say that color c has a hole, if no server is located at c. The proof strategy is to show that the average expected cost per hole is only constant. As there are at most $2n(C)$ holes before the next event configuration, this implies the claim.

We say that a hole is charged cost 1 if it is located on a color that is requested or marked. Let X_i denote the cost of hole i and M_i the number of times (the color of) hole i is marked. We first show that $\mathbb{E}[X_i \mid M_i = m_i] = \mathcal{O}(1 + m_i)$, if the color of hole i has seen r^* requests before arriving in configuration C.

As the request sequence cannot be modified by the adversary after the start of the phase, we can number the free colors in the order in which they obtain their (r^*+1)th request. If there are z free colors in event configuration C, this means that a new event configuration is triggered when color $z/4$ is requested.

Whenever a hole is requested, it chooses a free color uniformly at random to move to. If the new position is among the last $3z/4$ free colors, we say the hole is *settled*. A settled hole will not be requested before the next event configuration unless the adversary decides to mark it. As at most $z/4$ free colors become non-free before the next event configuration, this means that the hole will be settled after each request or marking with probability at least $2/3$. If hole i is marked m_i times, the expected cost before the last event configuration is therefore at most the cost before i is settled for the (m_i+1)-th time. As the probability for settling a hole is $2/3$ on a single try, the expected number of movements before it is settled is less than 3. It follows

that $\mathbb{E}[X_i \mid M_i = m_i] = \mathcal{O}(1 + m_i)$, if the color of hole i has seen r^* requests before arriving in configuration C.

If the color of hole i has seen $r < r^*$ requests before arriving in configuration C, the hole might incur hit cost $r^* - r$ before the hole moves away. In order to pay for this cost, we charge it to previous rounds as follows. Suppose that at the end of a round, there are s holes on colors with less than r^* requests. Each of these colors is charged 1; as s is at most the number of new colors, this only increases the cost by at most a factor of 2.

Up to constants, the total expected cost is therefore at most $\sum_{i=1}^{2n(C)}(1+m_i) = 2n(C)+\sum_{i=1}^{2n(C)} m_i$. By Claim 21, the total number of forced marking operations is at most $4n(C)$, hence the total expected cost is $\mathcal{O}(n(C))$. ∎

Lemma 11. *For $V \geq D$ there is a randomized scheduling algorithm for stale colors that obeys the scheduling and deactivation constraint and guarantees that the volume gap on stale colors in Phase i is at most $3\gamma(1+\gamma)n_iV$. The expected cost on stale colors generated by this algorithm is only $\mathcal{O}(V \log k\mathbb{E}[\sum_i n_i])$.*

Proof: Properties 1 and 2 follow from Fact 20 and Claim 22, respectively. We can therefore apply Lemma 13, which proves the upper bound on the hit cost and movement cost.

The bound on the volume gap follows from Claim 18 and the fact that the volume gap is at most γ times the volume cost. ∎

REFERENCES

[1] Dimitris Achlioptas, Marek Chrobak, and John Noga. Competitive analysis of randomized paging algorithms. *Theoretical Computer Science*, 234(1-2):203–218, 2000.

[2] Anna Adamaszek, Artur Czumaj, Matthias Englert, and Harald Räcke. Almost tight bounds for reordering buffer management. In *Proceedings of the 43rd ACM Symposium on Theory of Computing (STOC)*, pages 607–616, 2011.

[3] Anna Adamaszek, Artur Czumaj, Matthias Englert, and Harald Räcke. Optimal online buffer scheduling for block devices. In *Proceedings of the 44th ACM Symposium on Theory of Computing (STOC)*, pages 589–598, 2012.

[4] Anna Adamaszek, Marc P. Renault, Adi Rosén, and Rob van Stee. Reordering buffer management with advice. In *Proceedings of the 11th Workshop on Approximation and Online Algorithms (WAOA)*, pages 132–143, 2013.

[5] Yuichi Asahiro, Kenichi Kawahara, and Eiji Miyano. NP-hardness of the sorting buffer problem on the uniform metric. *Discrete Applied Mathematics*, 160(10-11):1453–1464, 2012.

[6] Noa Avigdor-Elgrabli, Sungjin Im, Benjamin Moseley, and Yuval Rabani. On the randomized competitive ratio of reordering buffer management with non-uniform costs. In *Proceedings of the 42nd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 78–90, 2015.

[7] Noa Avigdor-Elgrabli and Yuval Rabani. A constant factor approximation algorithm for reordering buffer management. In *Proceedings of the 24th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 973–984, 2013.

[8] Noa Avigdor-Elgrabli and Yuval Rabani. An optimal randomized online algorithm for reordering buffer management. In *Proceedings of the 54th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 1–10, 2013.

[9] Noa Avigdor-Elgrabli and Yuval Rabani. An improved competitive algorithm for reordering buffer management. *ACM Transactions on Algorithms*, 11(4):35:1–35:15, 2015.

[10] Yossi Azar, Matthias Englert, Iftah Gamzu, and Eytan Kidron. Generalized reordering buffer management. In *Proceedings of the 31st Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 87–98, 2014.

[11] Yossi Azar, Arun Ganesh, Rong Ge, and Debmalya Panigrahi. Online service with delay. In *Proceedings of the 49th ACM Symposium on Theory of Computing (STOC)*, pages 551–563, 2017.

[12] Nikhil Bansal, Niv Buchbinder, Aleksander Madry, and Joseph Naor. A polylogarithmic-competitive algorithm for the *k*-server problem. *Journal of the ACM*, 62(5):40:1–40:49, 2015.

[13] Siddharth Barman, Shuchi Chawla, and Seeun Umboh. A bicriteria approximation for the reordering buffer problem. In *Proceedings of the 20th European Symposium on Algorithms (ESA)*, pages 157–168, 2012.

[14] Daniel K. Blandford and Guy E. Blelloch. Index compression through document reordering. In *Proceedings DCC 2002. Data Compression Conference*, pages 342–351, 2002.

[15] Avrim Blum, Carl Burch, and Adam Kalai. Finely-competitive paging. In *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 450–458, 1999.

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

[17] Niv Buchbinder and Joseph Naor. Online primal-dual algorithms for covering and packing problems. In *Proceedings of the 13th European Symposium on Algorithms (ESA)*, pages 689–701, 2005.

[18] Ho-Leung Chan, Nicole Megow, René Sitters, and Rob van Stee. A note on sorting buffers offline. *Theoretical Computer Science*, 423:11–18, 2012.

[19] Matthias Englert and Harald Räcke. Reordering buffers with logarithmic diameter dependency for trees. In *Proceedings of the 28th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1224–1234, 2017.

[20] Matthias Englert, Harald Räcke, and Matthias Westermann. Reordering buffers for general metric spaces. *Theory of Computing*, 6(1):27–46, 2010.

[21] Matthias Englert and Matthias Westermann. Reordering buffer management for non-uniform cost models. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 627–638, 2005.

[22] Hossein Esfandiari, MohammadTaghi Hajiaghayi, Mohammad Reza Khani, Vahid Liaghat, Hamid Mahini, and Harald Räcke. Online stochastic reordering buffer scheduling. In *Proceedings of the 41st International Colloquium on Automata, Languages and Programming (ICALP)*, pages 465–476, 2014.

[23] 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.

[24] Iftah Gamzu and Danny Segev. Improved online algorithms for the sorting buffer problem on line metrics. *ACM Trans. Algorithms*, 6(1), 2009.

[25] Sungjin Im and Benjamin Moseley. New approximations for reordering buffer management. In *Proceedings of the 25th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1093–1111, 2014.

[26] Sungjin Im and Benjamin Moseley. Weighted reordering buffer improved via variants of knapsack covering inequalities. In *Proceedings of the 42nd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 737–748, 2015.

[27] Rohit Khandekar and Vinayaka Pandit. Online and offline algorithms for the sorting buffers problem on the line metric. *Journal of Discrete Algorithms*, 2008.

[28] Matthias Kohler and Harald Räcke. Reordering buffer management with a logarithmic guarantee in general metric spaces. In *Proceedings of the 44nd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 33:1–33:12, 2017.

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

[30] Jens Krokowski, Harald Räcke, Christian Sohler, and Matthias Westermann. Reducing state changes with a pipeline buffer. In *Proceedings of the 9th International Fall Workshop Vision, Modeling, and Visualization (VMV)*, pages 217–224, 2004.

[31] James R. Lee. Fusible HSTs and the randomized k-server conjecture. In *Proceedings of the 59th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 438–449, 2018.

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

[33] Harald Räcke, Christian Sohler, and Matthias Westermann. Online scheduling for sorting buffers. In *Proceedings of the 10th European Symposium on Algorithms (ESA)*, pages 820–832, 2002.

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

APPENDIX

A. Modifying the Buffer Size

Theorem 23. *For any input sequence, the cost* $\mathrm{OPT}_{b'}$ *of an optimal offline solution utilizing a buffer of size* $b' = (1-\varepsilon)\cdot b$ *is at most a factor of* $(2+\varepsilon \ln b')/(1-\varepsilon)$ *larger than the cost* OPT_b *of an optimal offline solution utilizing a buffer of size* b*.*

Proof: The proof is an adapted version of Theorem 2.1 in [3].

Fix any input sequence and consider an optimal offline strategy using k servers and a buffer of size b. Without loss of generality, we may assume that each color is visited by a server exactly once. Indeed, if a color c is visited twice, we could re-color all elements that appear after a server first left c to a completely new color. This does not change OPT_b and can only increase $\mathrm{OPT}_{b'}$.

For a color c let $t_{\mathrm{start}}(c)$ and $t_{\mathrm{end}}(c)$ denote the time step when a server visits and leaves c, respectively. We call a color *finished* at time t if $t \geq t_{\mathrm{end}}(c)$. We call a color *active* if it is currently assigned a server.

In the following we construct a strategy for processing the input sequence with a buffer of size b'. We visit colors in the order given by the optimum strategy OPT_b for buffer-size b. Whenever our buffer is full and does not contain any active color, we want to perform the next movement. However, sometimes this strategy may tell us to move a server away from a color c to a color c' but because of our smaller buffer size we may not have seen all elements of c yet, i.e., we still have $t < t_{\mathrm{end}}(c)$. In this case we will add block-operations in order to remove additional elements and thereby increase t to $t_{\mathrm{end}}(c)$ or beyond. We repeat the following steps until $t \geq t_{\mathrm{end}}(c)$:

1) Choose an inactive color q, and perform a block-operation on q.
2) If still $t < t_{\mathrm{end}}(c)$ increase $p(c')$ by $n(q)$ for every unfinished color c' that finishes before $t_{\mathrm{start}}(q)$.

Here $p(c')$ is a counter that exists for every color and is initialized to 0. $n(q)$ denotes the number of elements of color q that are currently in the buffer. We call a block-operation *successful* if $t \geq t_{\mathrm{end}}(c)$ holds afterwards.

Claim 24. *A counter* $p(c')$ *cannot be increased beyond* εb*.*

Proof: Let c denote the color that finishes next according to OPT_b. Among unfinished colors this color has the largest $p(c)$-value. Assume we perform a block-operation on q with $p(c) + n(q) > \varepsilon b$, i.e. the claim would be violated if the block operation is unsuccessful.

There are at most b elements in the input sequence that occur before $t_{\mathrm{end}}(c)$ and that belong to colors q' with $t_{\mathrm{end}}(c) < t_{\mathrm{start}}(q')$, as otherwise OPT_b could not hold these items within its buffer. Observe that all elements that contributed to the increase of $p(c)$ and also the $n(q)$ elements

of q belong to this class of elements. Since we have removed them, there are at most $b-p(c)-n(q) < b'$ of these elements left. This is sufficient for t to advance to $t_{\text{end}}(c)$ and, hence, the block-operation on q is successful. ∎

Claim 25. *There exists a color q such that $\sum_{c'} \Delta p(c') \geq b'/(1+\ln b')$, where $\Delta p(c')$ is the increase of the $p(c')$-counter in case the block-operation is unsuccessful.*

Proof: Define $s := b'/(1+\ln b')$ to shorten notation, and assume for contradiction that the total counter increase for every color is at most s.

Every server move finishes one color and starts a new one. If we sort the inactive unfinished colors according to their start times, the i-th color c_i among these will have i colors that are currently unfinished but that end before $t_{\text{start}}(c_i)$. Hence, the i-th color will trigger an increase of at least $i \cdot n(c_i) \leq s$ in counters. Note that in particular this means that $n(c_i) = 0$ for $i > s$. Also note that colors c that are active or already finished have $n(c) = 0$. Since the buffer is full we have

$$b' = \sum_{i=1}^{\lfloor s \rfloor} n(c_i) \leq s \sum_{i=1}^{\lfloor s \rfloor} \frac{1}{i} < s(\ln(s)+1) < b' \ ,$$

which is a contradiction. ∎

How many block-operations can be performed if we always perform a block-operation on the color q given by Claim 25? Clearly, there are at most OPT_b successful block-operations. Every unsuccessful block-operation increases $\sum_c p(c)$ by at least $b'/(1+\ln b')$ according to Claim 25. On the other hand $\sum_c p(c) \leq \varepsilon b \cdot \text{OPT}_b$ due to Claim 24. Hence, at most $\varepsilon b \cdot (1+\ln b')/b' \cdot \text{OPT}_b = (1+\ln b') \cdot \varepsilon/(1-\varepsilon) \cdot \text{OPT}_b$ unsuccessful operations can be performed.

In total we require at most $(1+(1+\ln b') \cdot \varepsilon/(1-\varepsilon)) \cdot \text{OPT}_b \leq (2+\varepsilon \ln b')/(1-\varepsilon) \cdot \text{OPT}_b$ operations. ∎

Proof of Corollary 2: Let OPT_b denote the cost of an optimal strategy using buffer size b. Theorem 23 with a choice of $\varepsilon = 1/\log b$ implies that $\text{OPT}_{b'} \leq \mathcal{O}(1)\,\text{OPT}_b$. The LP for buffer size b' has value at most $\text{OPT}_{b'}$ and its value can only decrease by removing constraints for $(\vec{s},t)$ with $|E(\vec{s},t)| \leq b$. Therefore the modified LP has value at most $\mathcal{O}(1)\,\text{OPT}_b$. ∎

B. Analysis of the Base Procedure

1) Randomized Rounding: The y-variables, some of which are set by the base procedure and some of which are set equal to 1 by the cost control procedure can be rounded online in such a way that the buffer constraint is satisfied. This is true independent of which (if any) y-variables are set to 1 by the cost control procedure and also independent of the values of the δ-variables.

A deterministic strategy for scheduling block operations is given by a function $D : T \times C \to \mathbb{N}_0$ with the meaning that $D(\tau,c)$ describes the number of block operations for color c performed in step τ. We allow $D(\tau,c) > 1$, i.e., we will allow that the strategy performs several consecutive block operations for the same color.

We specify a number of conditions on a deterministic strategy. The conditions are chosen in such a way that any deterministic strategy that satisfies them, produces an overall solution that ensures that the buffer never stores more than b items.

In order to formulate the conditions for time t, we introduce the following notation. We use $w_c(\tau) := \max\{\beta y_c(\tau), 1\}$ to denote a scaled version of the current assignment to y-variables in the primal LP. We partition the pairs (τ,c) with $\tau \in \{z_c+1,\ldots,t\}$ into classes according to the value of $|E_c(\vec{z},\tau)|$. Here and in the following $\vec{z}$ is to be understood as the vector that our procedure maintains. We say a pair (τ,c) is in class S_i if $|E_c(\vec{z},\tau)| \in [2^i, 2^{i+1})$ (we do not care about (τ,c)-pairs that have $|E_c(\vec{z},\tau)| = 0$). We further use S^c to denote the set $\{(\tau,c) \mid \tau \in \{z_c+1,\ldots,t\}\}$ and $S = \bigcup_c S^c$.

In addition to sets S_i we also define a set L that contains (τ,c)-pairs with a "large" $|E_c(\vec{z},\tau)|$-value. Formally, we first select pairs in decreasing order of $|E_c(\vec{z},\tau)|$-value until L contains pairs whose $w_c(\tau)$-values sum up to at least $\lambda+1$ (for a parameter $\lambda \ll \beta$ to be chosen later) or $L = S$; then if the $w_c(\tau)$-values sum up to more than $\lambda+1$ we remove the last element added. Hence, if $L \neq S$ we have $\lambda \leq \sum_{(\tau,c)\in L} w_c(\tau) \leq \lambda+1$, as the $w_c(\tau)$'s are at most 1.

Our conditions for a deterministic strategy D are as follows.

1) For every color c, $\sum_{z_c \leq \tau \leq t}(D(\tau,c)+\delta_c(\tau)) \geq 1$, i.e., between time z_c and t, all items of color c are removed from the buffer at least once; either because D performed at least one block device operation or because at some point one of the k servers was located at c.
2) D mirrors the fractional solution of the LP on the sets S_i:
   $\forall S_i : \quad \lfloor \sum_{(\tau,c)\in S_i} w_c(\tau) \rfloor \leq \sum_{(\tau,c)\in S_i} D(\tau,c)$.
3) D mirrors the fractional solution of the LP on the set L of *large* (τ,c)-pairs:
   $\lfloor \sum_{(\tau,c)\in L} w_c(\tau) \rfloor \leq \sum_{(\tau,c)\in L} D(\tau,c)$.
4) For every color c and for every class S_i, $\sum_{(\tau,c)\in S_i \cap S^c} D(\tau,c) \leq 3$, this means D did not remove the same color very often in the same class.

Suppose these conditions hold at the end of time t, where $\vec{z}$ is the specific vector maintained by our base procedure, i.e., $\alpha(\vec{z},t)$ is a proper constraint. Further suppose that the proper constraint $\alpha(\vec{z},t)$ is satisfied. In this case we will now show that the buffer contains no more than b items.

Condition 1 guarantees that items of color c that appeared at time z_c or before do not influence the buffer-constraint for D at time t, since all these items have already been removed from the buffer. Hence, the following formula specifies exactly the number of items in D's buffer at time

t:

$$\text{buffer}(t) = |E(\vec{z},t)| - \sum_c \max_{\tau: D(\tau,c)+\delta_c(\tau)\geq 1} |E_c(\vec{z},\tau)| \ . \tag{8}$$

This holds because $|E_c(\vec{z},\tau)|$ (for the maximum τ with $D(\tau,c)+\delta_c(\tau)\geq 1$) specifies the items that appeared after time z_c (excluding z_c) and are evicted at time τ or before. Furthermore, remember that if $\alpha(\vec{z},t)$ is a proper constraint, we must have $\delta_c(\tau)=0$ for all c and $\tau > z_c$, i.e., there is no server at color c after time z_c. We therefore also have

$$\text{buffer}(t) = |E(\vec{z},t)| - \sum_c \max_{\tau: D(\tau,c)\geq 1} |E_c(\vec{z},\tau)| \ . \tag{9}$$

Let j denote the index of the largest class that contains a pair (τ,c) with $D(\tau,c)+\delta_c(\tau)\geq 1$. Then

$$\begin{aligned}\max_{\tau: D(\tau,c)\geq 1} |E_c(\vec{z},\tau)| \geq 2^j &\geq \frac{1}{12}\sum_{i=0}^{j} 3\cdot 2^{i+1} \\ &\geq \frac{1}{12}\sum_{(\tau,c)\in S^c} D(\tau,c)\cdot |E_c(\vec{z},\tau)| \ ,\end{aligned}$$

where the last step uses Condition 4 (the fact that D does not evict a color too often in the same class). Plugging the above into Equation 9 gives $\text{buffer}(t) \leq |E(\vec{z},t)| - \frac{1}{12}\sum_{(\tau,c)\in S} D(\tau,c)\cdot|E_c(\vec{z},\tau)|$. We need to show that this is at most b. This means we want to show that $\sum_{(\tau,c)\in S} D(\tau,c)\cdot|E_c(\vec{z},\tau)| \geq 12(|E(\vec{z},t)|-b)$.

This is encapsulated in the following lemma.

Lemma 26. *Let $\alpha(\vec{z},t)$ be a proper, non-violated constraint. A scheduling strategy that fulfills conditions 1, 2, 3, and 4 fulfills $\sum_{(\tau,c)\in S} D(\tau,c)\cdot|E_c(\vec{z},\tau)| \geq 12(|E(\vec{z},t)|-b)$.*

Proof: For this we need the following claim.

Claim 27. *Either $\lambda \leq \sum_{(\tau,c)\in L} w_c(\tau) \leq (\lambda+1)$ or $|E(\vec{z},t)|\leq b$.*

Proof: Suppose $|E(\vec{z},t)| > b$. The primal constraint $\alpha(\vec{z},t)$ is fulfilled. Therefore,

$$\begin{aligned}&\sum_c \Big(\sum_{z_c<\tau\leq t} |E_c(\vec{z},\tau)|_t \cdot y_c(\tau) + |E_c(\vec{z},t)|_t\cdot\delta_c(t)\Big)_c \\ &\geq |E(\vec{z},t)| - b' \ .\end{aligned}$$

The left term is equal to $\sum_c\sum_{z_c<\tau\leq t}|E_c(\vec{z},\tau)|_t\cdot y_c(\tau)$ because whenever $\delta_c(t)=1$ we have $z_c=t$ and hence $|E_c(\vec{z},t)|_t=0$. Since $|E(\vec{z},t)|>b$, we also have $|E_c(\vec{z},\tau)|_t \leq |E(\vec{z},t)|-b'$ and therefore, the $y_c(\tau)$ must sum to at least one. Scaling by β gives that the $w_c(\tau)$'s sum up to at least β. Hence, L will contain a $w_c(\tau)$-weight of at least λ and at most $\lambda+1$, as $\lambda \ll \beta$. ∎

In order to show the lemma we can assume that $|E(\vec{z},t)|-b>0$ as otherwise the equation trivially holds as the left hand side is always positive. The above claim then gives $\sum_{(\tau,c)\in L} w_c(\tau)\geq\lambda$.

Now assume that i_ℓ, the class-index of the pair $(\tau,c)\in L$ with smallest $|E_c(\vec{z},\tau)|$-value, fulfills $2^{i_\ell}\geq|E(\vec{z},t)|-b$. Then

$$\begin{aligned}\sum_{(\tau,c)\in S}|E_c(\vec{z},\tau)|\cdot D(\tau,c) &\geq \sum_{(\tau,c)\in L}|E_c(\vec{z},\tau)|\cdot D(\tau,c) \\ &\geq 2^{i_\ell}\sum_{(\tau,c)\in L} D(\tau,c) \\ &\geq \lambda(|E(\vec{z},t)|-b) \\ &\geq 12(|E(\vec{z},t)|-b) \ ,\end{aligned}$$

by choosing $\lambda\geq 12$. In the following we can assume $2^{i_\ell}\leq(|E(\vec{z},t)|-b)$. We have

$$\begin{aligned}\sum_{(\tau,c)\in S}|E_c(\vec{z},\tau)|D(\tau,c) &\geq \sum_{i\leq i_\ell}\sum_{(\tau,c)\in S_i}|E_c(\vec{z},\tau)|D(\tau,c) \\ &\geq \sum_{i\leq i_\ell}2^i\sum_{(\tau,c)\in S_i} D(\tau,c) \\ &\geq \sum_{i\leq i_\ell}2^i\Big(\sum_{(\tau,c)\in S_i} w_c(\tau)-1\Big) \\ &= \frac{1}{2}\sum_{i\leq i_\ell}\sum_{(\tau,c)\in S_i}2^{i+1}w_c(\tau) - \sum_{i\leq i_\ell}2^i \ .\end{aligned}$$

It follows that

$$\begin{aligned}&\sum_{(\tau,c)\in S}|E_c(\vec{z},\tau)|D(\tau,c) \\ &\geq \frac{1}{2}\sum_{i\leq i_\ell}\sum_{(\tau,c)\in S_i}|E_c(\vec{z},\tau)|w_c(\tau) - 2^{i_\ell+1} \\ &\geq \frac{\beta}{2}\sum_{i\leq i_\ell}\sum_{(\tau,c)\in S_i}|E_c(\vec{z},\tau)|_t y_c(\tau) - 2(|E(\vec{z},t)|-b) \ .\end{aligned} \tag{10}$$

Then

$$\begin{aligned}\frac{\beta}{2}\sum_{L\setminus S_{i_\ell}}|E_c(\vec{z},\tau)|_t y_c(\tau) &\leq \frac{1}{2}\sum_{(\tau,c)\in L}|E_c(\vec{z},\tau)|_t w_c(\tau) \\ &\leq (|E(\vec{z},t)|-b)\sum_{(\tau,c)\in L} z_c(\tau) \\ &\leq (\lambda+1)(|E(\vec{z},t)|-b)\end{aligned}$$

where the last step uses $\sum_{(\tau,c)\in L} w_c(\tau)\leq\lambda+1$. Adding the inequality $0\geq\frac{\beta}{2}\sum_{L\setminus S_{i_\ell}}|E_c(\vec{z},\tau)|_t y_c(\tau) - (\lambda+1)(|E(\vec{z},t)|-b)$ to Equation 10 we obtain

$$\begin{aligned}&\sum_{(\tau,c)\in S}|E_c(\vec{z},\tau)|D(\tau,c) \\ &\geq \frac{\beta}{2}\sum_{(\tau,c)\in S}|E_c(\vec{z},\tau)|_t y_c(\tau) - (\lambda+3)(|E(\vec{z},t)|-b) \\ &\geq 12(|E(\vec{z},t)|-b) \ ,\end{aligned}$$

where the last inequality holds for large enough β, and uses the fact that the primal constraint for $\alpha(\vec{z},t)$ is fulfilled and either $\delta_c(t)=0$ or $|E_c(\vec{z},t)|_t=0$. ∎

It remains to show how to update the distribution in an online manner so that the block operation strategies fulfill conditions 1, 2, 3, and 4.

Instead of constructing μ directly we will first construct distributions μ_1, μ_2, μ_3. A random buffer management strategy according to μ is then chosen by sampling strategies $\mu_1 \sim D_1$, $\mu_2 \sim D_2$, $\mu_3 \sim D_3$ and computing the strategy D by setting $D(\tau, c) := \max\{D_1(\tau, c), D_2(\tau, c), D_3(\tau, c)\}$. Any strategy, whether in the support of μ_1, μ_2, or μ_3 fulfills $\forall c, \forall S_i \sum_{(\tau,c)\in S_i \cap S^c} D(\tau, c) \leq 3$. Then it is clear that the strategy $D = \max\{D_1, D_2, D_3\}$ fulfills $\forall c, \forall S_i \sum_{(\tau,c)\in S_i \cap S^c} D(\tau, c) \leq 3$ (Condition 4).

In addition strategies in the support of μ_1 fulfill Condition 1, strategies from μ_2 fulfill Condition 2, and strategies from μ_3 fulfill Condition 3. Hence, D will fulfill all these conditions and consequently D will obey buffer-constraints.

Maintaining $\boldsymbol{\mu_1}$, $\boldsymbol{\mu_2}$, and $\boldsymbol{\mu_3}$.: In the following we describe how to update μ_1, μ_2, and μ_3, and make sure that the strategies in their support fulfill their respective conditions. The distributions μ_1 and μ_2 will always fulfill $\sum_D \mu_i(D) \cdot D(\tau, c) = w_c(\tau)$, i.e., the expected number of times the color c is removed at time τ by a random strategy D is equal to the scaled LP-variable $w_c(\tau)$ (recall that we allow a strategy to remove a color several times in the same step). It would be difficult to maintain the above property without allowing changes in the past. Therefore, we will allow a strategy at time t to alter its past behavior and to increase or decrease $D(\tau, c)$ for $\tau < t$ (such a change may incur a cost, of course). In reality, decreasing a $D(\tau, c)$-value only makes fulfilling buffer-constraints more difficult, so allowing this does not give additional power to the algorithm. Similarly, increasing a $D(\tau, c)$-value with $\tau < t$ is never better than increasing $D(t, c)$ instead, since we only care about the buffer-constraint at t as the others have already been met.

There are two types of changes that require updating the distributions. Firstly, the $w_c(\tau)$-values may increase. We will assume that these changes are infinitesimal, i.e., in each step we have to react to an increase of a $w_c(\tau)$-value from $w_c(\tau)$ to $w_c(\tau) + \varepsilon$.

The other type of change is a change to the vector z. When z is recomputed because otherwise the primal $\alpha(\vec{z}, t)$ constraint is not proper anymore, we set, for each color c for which one of the conditions is violated, z_c to the largest possible value (less or equal to t) such that $\sum_{z_c \leq \tau \leq t}(y_c(\tau) + \delta_c(\tau)) \geq 1/\beta$. The above procedure guarantees that the following properties *always* hold

- $\sum_{(\tau,c):z_c<\tau\leq t} w_c(\tau) < 3$. A primal $\alpha(\vec{z}, t)$ constraint is only proper if $\sum_{(\tau,c):z_c<\tau\leq t} w_\tau(c) \leq 2$. Procedure 1 from Section III-A, increases w-values by at most $\beta/\log^3 b$ within a single step. For large enough b this means that the above sum never exceeds 3.
- $\sum_{z_c\leq\tau\leq t} w_c(\tau) \geq 1$ In the end we want that every strategy D from μ has every color removed at least once within the range $\{z_c, \ldots, t\}$. Therefore this property is important.

Whenever z changes $\sum_{(\tau,c):z_c<\tau\leq t} w_c(\tau)$ decreases by at least 1. Therefore a fixed pair (τ, c) is only involved in at most 3 different $\alpha(\vec{z}, t)$ constraints.

For increasing $w_c(\tau)$-values we show that an increase in ε results in an expected cost of $\mathcal{O}(\varepsilon)$ for the maintenance operation. We also implement a maintenance operation with $\mathcal{O}(\varepsilon)$ expected cost when a $w_c(\tau)$-value decreases. With this decrement operation we implement a change of the vector z as follows.

Suppose we want to change the c-th coordinate of z from z_c to z'_c. This may make all conditions that depend on the values $|E_c(\vec{z}, \tau)|$ invalid. We decrement all $w_c(\tau)$-values with $\tau \in \{z'_c + 1, \ldots, t\}$ to 0. Then we change z_c to z'_c, and after that we increase the $w_c(\tau)$-values again. The cost for this is $\mathcal{O}(w_c(\tau))$ for every (τ, c)-pair involved. Since each (τ, c)-pair is only involved in a constant number of these decrement operations we obtain that the total cost for the change is only $\mathcal{O}(\sum_{\tau,c} w_c(\tau)) = \mathcal{O}(\sum_{t,c} y_c(t))$, provided that we can implement the increase and decrease operations as claimed.

Maintaining $\boldsymbol{\mu_1}$.: For maintaining μ_1 we do not require a decrement operation as Condition 1 does not depend on $|E_c(\vec{z}, \tau)|$-values. Suppose that $w_c(\tau)$ increases by ε. Then we first identify an ε-measure of strategies that have the smallest value of $\arg\max_{\tau'}\{D(\tau', c) > 0\}$, i.e., strategies that did not remove color c for a long time. For these strategies we set $D(\tau, c) = 1$. This means that the strategies evict color c in a round-robin fashion. Since $\sum_{(\tau,c)\in S^c} \sum_D \mu(D) D(\tau, c) = \sum_{(\tau,c)\in S^c} w_\tau(c) \geq 1$, Condition 1 follows.

Maintaining $\boldsymbol{\mu_2}$.: We maintain a *strengthening* of Condition 2, namely that for all S_i

$$\begin{aligned}\Big\lfloor \sum\nolimits_{(\tau,c)\in S_i} w_c(\tau) \Big\rfloor &\leq \sum\nolimits_{(\tau,c)\in S_i} D(\tau, c) \\ &\leq \Big\lceil \sum\nolimits_{(\tau,c)\in S_i} w_c(\tau) \Big\rceil \ .\end{aligned} \tag{11}$$

Suppose that the $w_{\bar{c}}(\bar{\tau})$ value for some pair $(\bar{\tau}, \bar{c})$ is increased by ε and assume that $(\bar{\tau}, \bar{c}) \in S_i$. As we want to satisfy $\sum_D \mu_1(D) D(\bar{\tau}, \bar{c}) = w_{\bar{c}}(\bar{\tau})$ we have to increase $D(\bar{\tau}, \bar{c})$ for some strategies. For this we choose an ε-fraction of strategies that have $\sum_{(\tau,c)\in S^c\cap S_i} D(\tau, c) \leq 2$ (these must exist for small enough ε as $\sum_D \sum_{(\tau,c)\in S^c\cap S_i} D(\tau, c)\mu_1(D) = \sum_{(\tau,c)\in S^c\cap S_i} w_c(\tau) \leq \sum_{(\tau,c)\in S^c} w_c(\tau) < 3$). We increase the value of $D(\tau, c)$ for the chosen strategies.

Now the constraint in Equation 11 may be violated for class S_i. Let $a = \lfloor \sum_{(\tau,c)\in S_i} w_c(\tau) \rfloor$ (before changing $w_c(\tau)$) and first assume that $\lfloor \sum_{(\tau,c)\in S_i} w_c(\tau) \rfloor$ remains equal to a. Then the strategies that just have been changed may now have $\sum_{(\tau,c)\in S_i} D(\tau, c) = a + 2$, which is not allowed.

We match these strategies to strategies that have $\sum_{(\tau,c)\in S_i} D(\tau, c) = a$. For each strategy D there must exist

a (τ, c) such that $D(\tau, c) > D'(\tau, c)$, where D' denotes the strategy that D is matched to. We decrease $D(\tau, c)$ by 1 and increase $D'(\tau, c)$ by 1. This only induces an expected cost of $\mathcal{O}(\varepsilon)$. The case in which $\lfloor \sum_{(\tau,c)\in S_i} w_c(\tau) \rfloor$ changes is analogous.

Also the decrement operation can be implemented this way. When $w_c(\tau)$ decreases we select an ε-measure of strategies that fulfill $D(\tau, c) > 0$ and decrease $D(\tau, c)$ for them. Then we do a re-balancing step.

Maintaining $\boldsymbol{\mu_3}$.: Here we maintain a strengthened version of Condition 3. Let $(\tau_1, c_1), (\tau_2, c_2), \dots$ denote the sequence of (τ, c)-pairs from S, in decreasing order of $|E_c(\vec{z}, \tau)|$. Let (τ_r, c_r) denote the first pair in this sequence that is *not* in L (note that this may not exist; then we define r as $|S| + 1$ since $L = S$). Define a function ℓ on the (τ, c)-pairs that is zero for all (τ_i, c_i), $i > r$; one for all (τ_i, c_i), $i < r$ and $w_{c_r} - (\sum_{i=1}^{r} w_{c_i}(\tau_i) - (\lambda + 1))/w_{c_r}(\tau_r)$ for (τ_r, c_r). For all (τ_i, c_i), $i \neq r$ ℓ simply is the characteristic function of the set L; only for r it measures the fraction by which (τ_r, c_r) needs to be included into L in order that the $w_c(\tau)$-values in L sum up to *exactly* $(1 + \lambda)$ (recall that during the construction of L we were aiming for it to contain a total w-weight of $\lambda + 1$. When we overshoot we remove the last element).

We maintain the constraint that

$$\Big\lfloor \sum_{(\tau,c)\in S} w_c(\tau)\ell(\tau, c) \Big\rfloor \leq \sum_{(\tau,c)\in L} D(\tau, c) \leq \Big\lceil \sum_{(\tau,c)\in S} w_c(\tau)\ell(\tau, c) \Big\rceil \ ,$$

which is a strengthening of Condition 3. In addition we maintain $\sum_D D(\tau, c)\mu_3(D) = w_c(\tau)\ell(\tau, c)$.

Suppose that a $w_c(\tau)$-value increases and that $(\tau, c) \in L$ (otherwise we don't need to do anything; observe that if w_{c_r} is increased or decreased by ε the value of $w_{c_r}(\tau_r)\ell(\tau_r, c_r)$ stays constant). We increase $D(\tau, c)$ for an ε-measure of strategies that have $D(\tau, c) < 3$. In addition we decrease $D(\tau_r, c_r)$ for an ε-measure of strategies (only if r is defined, i.e., if $L \neq S$).

Now, there may be an ε-fraction of strategies that has a value of $\sum_{(\tau,c)\in L} D(\tau, c)$ that is too large by 1, and there may exist an ε-measure of strategies for which this value is by one too low. As before we can perform re-balancing steps in order to fix this at an expected cost of $\mathcal{O}(\varepsilon)$.

2) Analyzing the Cost of the Base Procedure: We start by showing that the dual constraints $y_c(\tau)$ are not violated too much when using variables $\alpha_1(\vec{v}, t)$.

Lemma 28. *For every c and τ, $\sum_{\vec{v},t'\geq\tau} |E_c(\vec{v}, \tau)|_{t'} \cdot \alpha_1(\vec{v}, t') = \mathcal{O}(\log\log b)$.*

Proof: Let $\chi_c(\tau) = \sum_{\vec{v},t'\geq\tau} |E_c(\vec{v}, \tau)|_{t'} \cdot \alpha_1(\vec{v}, t') - 1$ denote the amount by which the dual constraint $y_c(\tau)$ is violated. $\chi_c(\tau)$ can increase if some dual variable $\alpha_1(\vec{z}, t)$ is increased. However, note that this only causes an increase of $\chi_c(\tau)$ if $z_c < \tau \leq t$. Further note that our procedure only increases $\alpha_1(\vec{z}, t)$ if the corresponding primal constraint is proper, which implies $\sum_{z_c<\tau\leq t} y_c(\tau) < 2/\beta$. Therefore, once $\sum_{\tau<i\leq t} y_c(i) \geq 2/\beta$, $\chi_c(\tau)$ does not increase any further. In the following we analyze how large $\chi_c(\tau)$ can become before $\sum_{\tau<i\leq t} y_c(i) \geq 2/\beta$. We have the following claim.

Claim 29. *A violated dual constraint $y_c(\tau)$ fulfills $\sum_{\tau<i\leq t} y_c(i) \geq e^{\chi_c(\tau)}/\log^3 b$.*

Proof: When the dual constraint $y_c(\tau)$ becomes tight (i.e., $\chi_c(\tau) = 0$) in some step t, our procedure sets $y_c(t)$ to at least $1/\log^3 b$. Therefore, at this point

$$\sum_{\tau<i\leq t} y_c(i) \geq e^{\chi_c(\tau)}/\log^3 b \tag{12}$$

as required.

In later time steps, an increase of dual variable $\alpha_1(\vec{z}, t')$ by $\mathrm{d}\alpha$ increases the violation $\chi_c(\tau)$ by $|E_c(\vec{z}, \tau)|_{t'} \cdot \mathrm{d}\alpha$. This means that the right hand side of Equation 12 increases by

$$\frac{e^{\chi_c(\tau)}}{\log^3 b} \cdot |E_c(\vec{z}, \tau)|_{t'} \cdot \mathrm{d}\alpha \ .$$

However, at the same time $\sum_{\tau<i\leq t'} y_c(i)$ increases by at least

$$\sum_{\tau<i\leq t} y_c(i) \cdot |E_c(\vec{z}, \tau)|_t \cdot \mathrm{d}\alpha \geq \frac{e^{\chi_c(\tau)}}{\log^3 b} \cdot |E_c(\vec{z}, \tau)|_{t'} \cdot \mathrm{d}\alpha$$

■

The previous claim implies that the violation of a dual constraint can grow to at most $\mathcal{O}(\log\log b)$ before $\sum_{\tau<i\leq t'} y_c(i)$ becomes $2/\beta$ after which the violation does not increase any further. ■

Our next goal is to derive a bound on the total amount by which y-variables are increased in the base procedure. Recall that $\hat{y}$ is defined in Procedure 1 and is either 0 or $1/\log^3 b$. The following two technical lemmas will be useful.

Lemma 30. *If $\alpha_1(\vec{v}, t) > 0$, then $\sum_c \sum_{\tau\leq t} |E_c(\vec{v}, \tau)|_t \cdot \hat{y}_c(\tau) \leq |E(\vec{v}, t)| - b'$.*

Proof: Fix a color c. We partition the time steps $\tau \in \{v_c + 1, \dots, t\}$ into classes according to the value of $|E_c(\vec{v}, \tau)|$. For $j \geq 0$, the class $T_{c,j}$ contains the time steps for which $|E_c(\vec{v}, \tau)| \in [2^j, 2^{j+1})$.

Claim 31. *A class $T_{c,j}$ includes at most $\mathcal{O}(\log\log b)$ time steps τ for which $\hat{y}_c(\tau) > 0$.*

Proof: Since $\alpha_1(\vec{v}, t) > 0$ the primal constraint $\alpha_1(\vec{v}, t)$ was proper and violated at time t.

Fix a class $T_{c,j}$ and let $\tau_1, \tau_2, \dots$ denote its time steps that have $\hat{y}_c(\tau_i) > 0$ in increasing order. Define $L(\tau) := \sum_{\vec{v},t'\geq\tau} |E_c(\vec{v}, \tau)|_{t'} \cdot \alpha(\vec{v}, t')$ to be the load of τ (the left hand side of the constraint $y_c(\tau)$ in the dual). Note that for $\tau \in T_{c,j}$ with $\tau \geq \tau_1$, an increase of $L(\tau)$ by $|E_c(\vec{v}, \tau)|_t \cdot \alpha(\vec{v}, t)$ also results in an increase of $L(\tau_1)$ by $|E_c(\vec{v}, \tau_1)|_t \cdot \alpha(\vec{v}, t)$ which

is the same up to a factor of 2 as τ_1 and τ_i are in the same class. In order for $\hat{y}_c(\tau_i)$ to be set to a positive value, some $L(\tau)$ for $\tau_{i-1} < \tau \leq \tau_i$ has to have increased to 1 (as a constraint needs to be tight). Hence, for every τ_i with $\hat{y}_c(\tau_i) > 0$, the load $L(\tau_1)$ will increase by at least 1/2. But due to Lemma 28, the load $L(\tau_1)$ is at most $\mathcal{O}(\log\log b)$. ■

Let $n_{c,j}$ denote the number of time steps τ in class $T_{c,j}$ that have $\hat{y}_c(\tau) > 0$. Then we have

$$\begin{aligned}\sum_c \sum_{\tau \leq t} |E_c(\vec{v},\tau)|_t \cdot \hat{y}_c(\tau) &\leq \sum_c \sum_{j>0} 2^{j+1} \frac{n_{c,j}}{\log^3 b} \\ &\leq \sum_c \mathcal{O}(\log\log b) \cdot \frac{|E_c(\vec{v},t)|}{\log^3 b} \\ &\leq \frac{|E(\vec{v},t)|}{\log b} \leq |E(\vec{v},t)| - b' \ .\end{aligned}$$

■

Lemma 32. *If* $\hat{y}_c(\tau) > 0$, *then* $\sum_{\vec{v},t\geq\tau} |E_c(\vec{v},\tau)|_t \cdot \alpha_1(\vec{v},t) \geq 1$.

Proof: $\hat{y}_c(\tau)$ is only set to a non-zero value if the dual constraint $y_c(\tau)$ is tight. Because α_1-variables are never decreased, once the constraint is tight it can only remain tight or become violated. ■

We are now ready to derive our desired bound on $\sum_{c,t} \mathrm{d}y_c(t)$.

Lemma 33. $\sum_{c,t} \mathrm{d}y_c(t) \leq 2 \sum_{\vec{v},t} \alpha_1(\vec{v},t)\big(|E(\vec{v},t)| - b'\big)$

Proof: $\sum_{c,t} \mathrm{d}y_c(t)$ consists of two components: $\sum_{c,t} \hat{y}_c(t)$ and $\sum_{c,t} \max_{\tau: z_c < \tau \leq t} \Delta(\tau, c)$. We bound both components separately.

Using Lemmas 30 and 32, we get

$$\begin{aligned}\sum_{\tau,c} \hat{y}_c(\tau) &\leq \sum_{\tau,c} \hat{y}_c(\tau) \cdot \sum_{\vec{v},t\geq\tau} |E_c(\vec{v},\tau)|_t \cdot \alpha_1(\vec{v},t) \\ &\leq \sum_{\vec{v},t} \alpha_1(\vec{v},t) \sum_c \sum_{\tau\leq t} |E_c(\vec{v},\tau)|_t \hat{y}_c(\tau) \\ &\leq \sum_{\vec{v},t} \alpha_1(\vec{v},t)\big(|E(\vec{v},t)| - b'\big) \ .\end{aligned}$$

For the second component, we have, for a time step t,

$$\begin{aligned}&\sum_c \max_{\tau: z_c < \tau \leq t} \sum_{\tau \leq i \leq t} y_c(i) \cdot |E_c(\vec{z},\tau)|_t \mathrm{d}\alpha \\ \leq\ &\sum_c \sum_{\tau \leq t} y_c(\tau) \cdot |E_c(\vec{z},\tau)|_t \mathrm{d}\alpha \\ \leq\ &\sum_c \sum_{\tau \leq t} y_c(\tau) \cdot |E_c(\vec{z},\tau)|_t \alpha_1(\vec{z},t) \ ,\end{aligned}$$

where the last step follows because $\alpha(\vec{z},t)$ is increased by $\mathrm{d}\alpha$. Because the primal $\alpha(\vec{z},t)$ constraint is violated, this is at most $(|E(\vec{z},t)| - b')\mathrm{d}\alpha_1(\vec{z},t)$. Summing over all time steps t gives the desired bound. ■