Generalized Reordering Buffer Management

Yossi Azar* Matthias Englert† Iftah Gamzu‡ Eytan Kidron§

Abstract

An instance of the generalized reordering buffer management problem consists of a service station that has k servers, each configured with a color, and a buffer of size b. The station needs to serve an online stream of colored items. Whenever an item arrives, it is stored in the buffer. At any point in time, a currently pending item can be served by switching a server to its color. The objective is to serve all items in a way that minimizes the number of servers color switches. This problem generalizes two well-studied online problems: the paging problem, which is the special case when $b = 1$, and the reordering buffer problem, which is the special case when $k = 1$.

In this paper, we develop a randomized online algorithm that obtains a competitive ratio of $O(\sqrt{b} \ln k)$. Note that this result beats the easy deterministic lower bound of k whenever $b < k^{2-\varepsilon}$. We complement our randomized approach by presenting a deterministic algorithm that attains a competitive ratio of $O(\min\{k^2 \ln b, kb\})$. We further demonstrate that if our deterministic algorithm can employ $k/(1-\delta)$ servers where $\delta \in (0,1)$, then it achieves a competitive ratio of $O(\min\{\ln b/\delta^2, b/\delta\})$ against an optimal offline adversary that employs k servers.

*Blavatnik School of Computer Science, Tel-Aviv University, Israel. Email: azar@tau.ac.il. Supported in part by the Israel Science Foundation (grant No. 1404/10) and by the Israeli Centers of Research Excellence (I-CORE) program (Center No. 4/11).

†Department of Computer Science and DIMAP, University of Warwick, UK. Email: englert@dcs.warwick.ac.uk. Supported in part by EPSRC award EP/D063191/1 and EPSRC grant EP/F043333/1.

‡Yahoo! Research. Email: iftah.gamzu@yahoo.com.

§Blavatnik School of Computer Science, Tel-Aviv University, Israel. Email: eytankid@tau.ac.il.

1 Introduction

We consider the *generalized reordering buffer management* problem. In this problem, there is a service station, which is equipped with k servers and a buffer of size b. Each of the servers is initially configured with some color. An online stream of colored items has to be served by the service station. Whenever an item arrives, it is stored in the buffer. At any point in time, a currently pending item can be served by removing it from the buffer and switching a server to its color. In particular, if one of the servers is already configured with the color of a pending item, this item can be served without switching any server. The goal is to serve all items while minimizing the overall number of color switches of the servers.

This problem is a natural generalization of two well-studied online problems: the *paging* problem, introduced by Sleator and Tarjan [19], is the special case when $b = 1$, and the *reordering buffer* problem, introduced by Räcke et al. [18], is the special case when $k = 1$. Apart from this, the problem is also an interesting abstraction of a number of problem scenarios occurring in computer science and manufacturing. We give two examples.

- Consider a network device that can maintain a maximum of k TCP/IP connections open at the same time. This device receives an online stream of unit sized packets that need to be forwarded to specific destinations. Arriving packets can be forwarded if they are addressed to one of the k currently open connections; otherwise, they have to be stored in a finite-sized random access packet buffer. If the buffer is full, the device needs to close one of the k open connections, and open a new one to the destination of one or more packets stored in the buffer. These packets can then be forwarded using the new connection and free up space in the buffer. The goal is to transmit all packets while minimizing the number of connection open/close operations.

- In the painting shop of a car manufacturing plant, car bodies traverse the final layer of painting, where each car body is painted with its own predetermined top coat. The painting shop is equipped with k painting machines and a finite-sized parking lot. Each machine, once configured with a color, can paint multiple cars with that color. However, switching a color in a machine causes non-negligible set-up and cleaning cost. The goal of the painting shop is to paint all incoming cars with a minimum number of color switches. The parking lot can be used to change the order in which the cars are painted, but it must never overflow.

Chan et al. [9] seem to have been the first to mention the generalized reordering buffer problem. They established that the offline variant of the problem, in which the entire stream of items is known in advance, is NP-hard. To the best of our knowledge, no other work has been done on generalized reordering buffer. In particular, no results are known for the online setting of the problem which we study.

Our results. We develop a randomized online algorithm that attains a competitive ratio of $O(\sqrt{b}\ln k)$. Note that for any $b < k^{2-\varepsilon}$ with $\varepsilon > 0$, our randomized upper bound beats the easy deterministic lower bound of k that applies for any b. Our algorithm has its roots in the randomized marking algorithm of Fiat et al. [12] for the paging problem, combined with several clean-up procedures. We emphasize that one natural approach for designing an algorithm for our generalized setting is to combine an algorithm for the reordering buffer problem with an algorithm for the paging problem. Specifically, the reordering buffer algorithm decides which color to serve next, and the paging algorithm decides which server should switch to that color. Unfortunately, we do not know if there are combinations of this nature that attain good performance guarantees. For example, we could combine the $O(\ln k)$-competitive randomized marking algorithm for paging with a deterministic algorithm for the reordering buffer problem, some of which are $O(\ln b)$-competitive or better. While one can easily prove that this combination is an $O(b \ln k)$-competitive algorithm, this guarantee is still weak; for example, if $b = k$. In fact, we were unable to show any bounds that are sublinear in k for such combinations.

We further present a deterministic online algorithm that attains a competitive ratio of $O(\min\{k^2 \ln b, kb\})$. This algorithm can complement our randomized algorithm in cases where b is large. Note that any lower bound for the paging problem also applies to the generalized reordering buffer management problem. To see this, consider any sequence of requests for pages (colors). This sequence can be modified by replacing each request to a page by b successive requests to that page. This does not change the cost for the paging problem, but it neutralizes the buffer of size b in the generalized reordering buffer problem, i.e., at any time, all pending requests are always for the same page. Therefore, no deterministic online algorithm can be better than k-competitive for our problem, and our randomized online algorithm outperforms the best possible deterministic online algorithm for any $b = o((k/\log k)^2)$. We also show that if our deterministic algorithm can employ $k/(1-\delta)$ servers where $\delta \in (0,1)$, then it achieves a competitive ratio of $O(\min\{\ln b/\delta^2, b/\delta\})$ against an optimal offline adversary that employs k servers. Notice that this implies that if our algorithm can employ a constant fraction of servers more than the optimal algorithm then it achieves logarithmic competitiveness.

Related work. The paging problem was introduced in the seminal paper of Sleator and Tarjan [19]. They proved that the LRU and FIFO strategies are k-competitive, and established that no deterministic algorithm can achieve a competitive ratio smaller than k. Karlin et al. [14] showed that the same bound is achieved by the flush-when-full strategy. Fiat et al. [12] presented a randomized $2H_k$-competitive marking algorithm, outperforming the lower bound for deterministic algorithms. Note that H_k is the kth harmonic number. They also established a lower bound of H_k on the competitive ratio that any randomized algorithm can achieve. Later on, Achlioptas et al. [1] and McGeoch and Sleator [17] provided algorithms matching this lower bound.

The reordering buffer problem was introduced by Räcke et al. [18], who devised an $O(\ln^2 b)$ -competitive algorithm. Englert and Westermann [11] presented an algorithm with an improved competitive ratio of $O(\ln b)$, which can be applied to a generalized non-uniform cost setting. Avigdor-Elgrabli and Rabani [4] developed an LP-based algorithm whose competitive ratio is $O(\ln b/\ln\ln b)$. Adamaszek et al. [2] presented an algorithm whose competitive ratio is $O(\sqrt{\ln b})$, and established lower bounds of $\Omega(\sqrt{\ln b/\ln\ln b})$ and $\Omega(\ln\ln b)$ for deterministic and randomized algorithms, respectively. Recently, Avigdor-Elgrabli and Rabani [6] developed a randomized online algorithm matching this lower bound.

Asahiro et al. [3] and Chan et al. [9] considered the offline variant of the reordering buffer problem, and established that it is NP-hard. Very recently, Avigdor-Elgrabli and Rabani [5] designed a constant factor approximation algorithm for this offline setting. The reordering buffer problem has also been studied on other metric spaces [8, 15, 13]. For example, Englert et al. [10] considered the more general variant in which items are associated with points in a metric space, and obtained a competitive ratio of $O(\ln^2 b \ln n)$, where n is the number of distinct points in the metric. Some research has been done on a maximization variant of the problem [16, 7].

2 A Randomized Algorithm

In this section, we develop a randomized algorithm whose competitive ratio is $O(\sqrt{b}\ln k)$. The algorithm sensibly combines the randomized marking algorithm due to Fiat et al. [12] with several buffer clean-up procedures.

2.1 The algorithm

We begin by briefly describing the random marking algorithm for the paging problem. Recall that in the underlying paging setting, there are k servers and a trivial buffer (i.e., $b = 1$). The algorithm is made up of *phases*. At the beginning of each phase, all colors are *unmarked*. When an item of color χ arrives, χ becomes marked and a server is moved to χ if there is no server there already. In order to decide which server moves,

the algorithm chooses a server uniformly at random from the servers which are located on unmarked colors. If there is no such server, then the current phase ends and a new phase begins. At the end of a phase, markings are removed from all colors. A crucial observation regarding this algorithm is that although it has random components, the order in which the colors are marked is deterministic and so is the partition into phases. We make use of this property also in the analysis of our algorithm.

Our algorithm combines the random marking algorithm with several buffer clean-up procedures. Similarly to the marking algorithm, our algorithm is also made up of phases. Each color has a *phase counter*, which is set to 0 at the beginning of each phase. When an item of color χ arrives, it is placed in the buffer, and the phase counter of χ is incremented by 1. When the phase counter of a color reaches $\sqrt{b}$, the color becomes marked and is served in a similar way to the marking algorithm. Note that the partition into phases is defined as in the marking algorithm.

We further define a status to each color: *marked*, *half-marked*, or *unmarked*. At the end of a phase, all half-marked colors become unmarked, and all marked colors become half-marked. Accordingly, arriving items are said to be either marked, half-marked or unmarked depending on the status of their color at their arrival time. For example, an item is said to be half-marked if its color was marked in the previous phase. We partition the buffer into two sub-buffers, each of size $b/2$. The *unmarked sub-buffer* stores unmarked items and the *half-marked sub-buffer* stores half-marked items. Note that marked items never stay in the buffer since a marked color always consists of a server that can immediately serve the arriving item. In fact, this may also be true for half-marked items, but it is never true for unmarked items. Namely, half-marked colors may or may not consist a server, but unmarked colors never consist of a server. If a half-marked item arrives, and a server is present on its corresponding color, then it is served immediately and does not need to be placed in the half-marked sub-buffer. Note that the phase counter of that color is still incremented. In order to avoid buffer overflows, we introduce the following three clean-up procedures:

Half-marked clean-up. A half-marked clean-up event takes place when the half-marked sub-buffer is full, and also at the end of each phase. Upon a *half-marked clean-up event*, all items in the half-marked sub-buffer are served. This is done by moving an arbitrary server through all the colors of items in the half-marked sub-buffer. The server then returns to its original position. Note that the number of half-marked clean-up moves in a half-marked clean-up event is one plus the number of different colors in the half-marked sub-buffer at the time of the event.

Targeted clean-up. A targeted single-color clean-up event takes place when a *buffer counter* of some color reaches $2\sqrt{b}$. The buffer counter maintains the number of items a color has in the unmarked sub-buffer. Note that this counter should not be confused with the phase counter used for marking. In a targeted single-color unmarked clean-up event, or simply *targeted clean-up event*, the items of a single color from the unmarked sub-buffer are served. An arbitrary server moves to that color and back to its original position. Hence, a targeted clean-up event consists of two targeted clean-up moves.

Unmarked clean-up. An unmarked clean-up event takes place if there are $\sqrt{b}/4$ different colors in the unmarked sub-buffer, and after $\sqrt{b}$ targeted clean-up events. Similarly to a half-marked clean-up event, upon an *unmarked clean-up event*, all items in the unmarked sub-buffer are served by an arbitrary server, which later returns to its original position. The number of unmarked clean-up moves in an unmarked clean-up event is one plus the number of different colors in the unmarked sub-buffer at the time of the event.

2.2 The analysis

We begin by pointing out that although the algorithm is randomized, it still has several deterministic aspects: the points in time in which each color becomes marked, half-marked and unmarked are deterministic, and so is the partition into phases. In particular, the partition into phases is deterministic since a phase ends just

before some color gets marked when each of precisely k different colors already got $\sqrt{b}$ items in that phase. The content of the unmarked sub-buffer at every point in time is also deterministic, and consequently, so is the point in time of every unmarked clean-up event and targeted clean-up event. Finally, we point out that although the content of the half-marked sub-buffer is not deterministic, and neither is the point in time of half-marked clean-up events, at the end of each phase the half-marked sub-buffer is emptied. Thus, the content of the entire buffer is deterministic at the beginning of every phase. The following lemma establishes the feasibility of the algorithm.

Lemma 2.1. *The algorithm never has a buffer overflow.*

Proof. Recall that the buffer is partitioned into two sub-buffers, each of size $b/2$. The half-marked sub-buffer never overflows since whenever it becomes full, a half-marked clean-up event is initiated. The unmarked items in the unmarked sub-buffer are served by a combination of unmarked clean-up events and targeted clean-up events. The unmarked clean-up events make sure that there are never more than $\sqrt{b}/4$ different colors in the unmarked sub-buffer, and the targeted clean-up events ensure that no such color has more than $2\sqrt{b}$ items in the sub-buffer. Together, there cannot be more than $b/2$ unmarked items in that sub-buffer. □

Let ON and OPT denote our algorithm and the optimal offline algorithm, respectively. We also denote the respective overall number of server moves in ON and OPT by ON and OPT. Note that ON is the expected number of moves since ON is randomized. In the remainder of this section, we prove that ON is $O(\sqrt{b}\ln k)$-competitive. We first partition ON according to four types of moves that the servers of ON do: $ON = ON^M + ON^H + ON^T + ON^U$, where ON^M is the expected number of *marking* moves, ON^H is the expected number of *half-marked clean-up* moves, ON^T is the number of *targeted clean-up* moves and ON^U is the number of *unmarked clean-up* moves. We also define ON_i to be the expected number of ON's server moves in phase i, and OPT_i is the number of OPT's server moves in phases $i-1$ and i. Note this latter asymmetry, and notice that $OPT \leq \sum_i OPT_i \leq 2 \cdot OPT$. Similarly, for a set S of consecutive phases, we define $ON_S = \sum_{i \in S} ON_i$. Note that we also use the same notation as before when considering a partition of the expected number of ON's server moves in a phase or a set of phases to the four types of moves. For example, ON_i^U is the expected number of unmarked clean-up moves that ON servers makes in phase i. We now turn to bound the ratios between each of ON's movement types and OPT. The competitive ratio ON/OPT is the sum of these ratios.

Marking moves and half-marked clean-up moves. We bound the expected number of marking moves, ON^M, and the expected number of half-marked clean-up moves, ON^H, using similar techniques. Let $H_k = \sum_{j=1}^{k} 1/j$ be the kth harmonic number, and let m_i be the number of colors that are marked in phase i but not marked in phase $i-1$. We start with a bound on the expected number of marking moves. The proof uses the same arguments as the proof for randomized marking algorithm by Fiat et al. [12].

Lemma 2.2. *For every phase i, $ON_i^M \leq m_i H_k$.*

Proof. At the beginning of phase i, all the servers of ON are located on half-marked colors. During phase i, k colors become marked. Out of these colors, m_i are unmarked at the beginning of the phase. Hence, ON has to move a server to each of those m_i colors. The remaining $k - m_i$ colors are half-marked at the beginning of the phase, and have a server on them. They also end that phase with a server on them, but it is unclear if it is the same server. That is, it may happen that a server left a color under consideration while it was still half-marked, and thus, when that color became marked again, another server made a marking move to it.

Let us focus on the k colors which are half-marked at the beginning of phase i, and let $\chi_0, \ldots, \chi_{k-m_i-1}$ be the subset of $k - m_i$ colors that become marked in phase i (according to the order in which they are marked). During the phase, some of the half-marked colors may contain servers. If a half-marked color has no server

we say that it is *void*. Notice that there can never be more than m_i void half-marked colors. When a color χ_j is marked it induces a marking move if and only if χ_j is void at that time. The probability that χ_j is void at the time that it is marked is at most $m_i/(k-j)$ since there are at most m_i void colors at that time and they are distributed uniformly at random among the $k-j$ half-marked colors not yet marked. The expected number of marking moves in phase i is

$$ON_i^M \leq m_i + \sum_{j=0}^{k-m_i-1} \frac{m_i}{k-j} \leq m_i H_k \ .$$

□

We now can use Lemma 2.2 to also derive our desired upper bound on half-marked clean-up moves.

Lemma 2.3. *For every phase i, $ON_i^H = O(\min\{m_i\sqrt{b}\ln k, m_i^2 \ln k/\sqrt{b} + m_i\})$.*

Proof. Recall that half-marked items may only accumulate at colors which had a server at the beginning of the phase, and that server left the color during that phase. Lemma 2.2 implies that there can be at most $m_i H_k$ such colors in expectation. Since at most $\sqrt{b}$ half-marked items can arrive from each such color, no more than $m_i H_k \sqrt{b}$ items enter the half-marked sub-buffer during phase i in expectation.

An immediate consequence of the above observation is that $ON_i^H = O(m_i\sqrt{b}\ln k)$. This follows since each half-marked clean-up move, except the last move in each such event, cleans at least one item. The last moves of all half-marked clean-up events add at most a multiplicative factor of 2 to the number of half-marked clean-up moves.

For the purpose of proving that $ON_i^H = O((m_i^2 \ln k)/\sqrt{b} + m_i)$, notice that the half-marked sub-buffer is full with $b/2$ items every time that half-marked clean-up event is initiated (except maybe the last half-marked clean-up event in every phase). So the number of events is upper bounded in expectation by $m_i H_k \sqrt{b}/(b/2) + 1 = 2m_i H_k/\sqrt{b} + 1$. Note that the additional 1 is due to the half-marked clean-up event done at the end of each phase. Now, at any given time, there are no more than m_i void half-marked colors, namely, colors without a server. This implies that there are no more than m_i colors in the half-marked sub-buffer, and hence, a server makes at most $m_i + 1$ moves in each such event. As a result, $ON_i^H = (2m_i H_k/\sqrt{b} + 1)(m_i + 1) = O(m_i^2 \ln k/\sqrt{b} + m_i)$. □

We next set a bound of the ratio between $ON^M + ON^H$ and OPT. For this purpose, we partition the phases into groups of *marking phase sequences*. Each marking phase sequence contains consecutive phases, and each phase belongs to exactly one marking phase sequence. We let S denote the set of phases in a marking phase sequence, and use last(S) to denote the last phase in S. Our partition maintains the property that in every marking phase sequence S (except maybe the last one), $m_i \leq 3\sqrt{b}$ for every $i \in S \setminus \{\text{last}(S)\}$, and $m_{\text{last}(S)} > 3\sqrt{b}$. Namely, the partition is set according to phases i such that $m_i > 3\sqrt{b}$. Let $m_S = \sum_{i \in S} m_i$.

Lemma 2.4. *For every marking phase sequence S, $ON_S^M + ON_S^H = O((m_S + m_{last(S)}\sqrt{b})\ln k)$.*

Proof. Notice that $ON_S^M = \sum_{i \in S} ON_i^M \leq \sum_{i \in S} m_i H_k = O(m_S \ln k)$, where the inequality holds by Lemma 2.2. In addition, observe that

$$\begin{aligned}
ON_S^H &= \sum_{i \in S \setminus \{\text{last}(S)\}} ON_i^H + ON_{\text{last}(S)}^H \\
&= \sum_{i \in S \setminus \{\text{last}(S)\}} O((m_i^2 \ln k/\sqrt{b}) + m_i) + O(m_{\text{last}(S)}\sqrt{b}\ln k) \\
&= \sum_{i \in S \setminus \{\text{last}(S)\}} O\left(m_i \ln k\right) + O(m_{\text{last}(S)}\sqrt{b}\ln k) = O((m_S + m_{\text{last}(S)}\sqrt{b})\ln k) \ ,
\end{aligned}$$

where the second equality follows from Lemma 2.3, and the third equality holds since $m_i \leq 3\sqrt{b}$ for every $i \in S \setminus \{\text{last}(S)\}$. □

We now turn to set a lower bound on OPT_S, where OPT_S is the overall number of server moves of OPT in S and the last phase before S. Recall that OPT_i is the number of server moves of OPT in phases i and $i-1$, and therefore, $OPT_S \leq \sum_{i \in S} OPT_i \leq 2 \cdot OPT_S$. We first set a bound applicable for all marking phase sequences S having a large m_S.

Lemma 2.5. *$OPT_S = \Omega(m_S/\sqrt{b})$, for every marking phase sequence S having $m_S \geq 3\sqrt{b}$.*

Proof. Observe that in each pair of phases $i-1$ and i exactly $k + m_i$ colors are marked, and the servers of OPT are present in no more than $k + OPT_i$ colors. Hence, there are at least $\sqrt{b} \cdot (m_i - OPT_i)$ items that entered the buffer of OPT during phases $i-1$ and i. The overall number of items entering OPT's buffer during S and the last phase before S is therefore at least $\sqrt{b}/2 \cdot \sum_{i \in S}(m_i - OPT_i)$, where the half factor is due to the fact that every item may be counted twice. Now, notice that in each of the OPT_S server moves, OPT can clear no more than b items from its buffer. Moreover, there can be at most b items that may stay in the buffer and not cleared at the end of S. Hence,

$$OPT_S \geq \frac{\sqrt{b}/2 \cdot \sum_{i \in S}(m_i - OPT_i) - b}{b} \geq \frac{m_S}{2\sqrt{b}} - \frac{OPT_S}{\sqrt{b}} - 1 \,,$$

where the last inequality holds since $\sum_{i \in S} OPT_i \leq 2 \cdot OPT_S$. This implies that $OPT_S = \Omega(m_S/\sqrt{b})$ since $m_S \geq 3\sqrt{b}$. □

The next lemma establishes a more specialized bound for all marking phase sequences S having a large $m_{\text{last}(S)}$.

Lemma 2.6. *$OPT_S = \Omega(m_S/\sqrt{b} + m_{last(S)})$, for every marking phase sequence S having $m_{last(S)} > 3\sqrt{b}$.*

Proof. The fact that $OPT_S = \Omega(m_S/\sqrt{b})$ follows from Lemma 2.5 by noticing that $m_S \geq m_{\text{last}(S)} > 3\sqrt{b}$. We now complete the proof by establishing that $OPT_S = \Omega(m_{\text{last}(S)})$. We next prove a somewhat stronger argument stating that $OPT_i = \Omega(m_i)$, for every phase i such that $m_i > 3\sqrt{b}$. As a consequence, we attain that $OPT_S \geq OPT_{\text{last}(S)} = \Omega(m_{\text{last}(S)})$ since $m_{\text{last}(S)} > 3\sqrt{b}$. For the purpose of proving the above argument, notice that the number of items which arrived during phases $i-1$ and i, and entered OPT's buffer is at least $\sqrt{b} \cdot (m_i - OPT_i)$. Since OPT's buffer cannot overflow, we attain that $b \geq \sqrt{b} \cdot (m_i - OPT_i)$, and therefore, $OPT_i \geq m_i - \sqrt{b} > 2m_i/3$, where the last inequality holds since $m_i > 3\sqrt{b}$. □

The main result of this subsection is the following lemma.

Lemma 2.7. *$ON^M + ON^H = O(\sqrt{b} \ln k) \cdot OPT + O(b \ln k)$.*

Proof. Notice that $OPT \geq \sum_S OPT_S/2$. Hence, it is sufficient that we establish the above mentioned bound for each marking sequence. Lemma 2.4 and Lemma 2.6 prove that $ON^M_S + ON^H_S = O(\sqrt{b} \ln k) \cdot OPT_S$, for every marking phase sequence S except maybe the last marking phase sequence. Consider the last marking phase sequence S'. If $m_{\text{last}(S')} > 3\sqrt{b}$ then the same bound also holds for S'. Otherwise, if $m_{S'} > 3b$ then from Lemma 2.5 we know that $OPT_{S'} = \Omega(m_{S'}/\sqrt{b})$, while from Lemma 2.4 we attain that $ON^M_{S'} + ON^H_{S'} = O(m_{S'} \ln k)$. Namely, the same bound ratio holds also in this case. Finally, when $m_{\text{last}(S')} \leq 3\sqrt{b}$ and $m_{S'} < 3b$, we get that $ON^M_{S'} + ON^H_{S'} = O(b \ln k)$, which is exactly the additive term in the above ratio. □

Targeted clean-up moves. We now turn our attention to bound the number of targeted clean-up moves ON^T. Recall that each targeted clean-up event consists of two server moves, and each such event happens when there are $2\sqrt{b}$ items of a single color in the unmarked sub-buffer.

Lemma 2.8. *There is no time interval during which no server of OPT moves but more than $2\sqrt{b}$ targeted clean-up events occur.*

Proof. Let us assume by way of contradiction that there exists a time interval I during which no server of OPT moves and there are more than $2\sqrt{b}$ targeted clean-up events. Recall that after every $\sqrt{b}$ targeted clean-up events, the unmarked sub-buffer is cleared by an unmarked clean-up event. This implies that the last $\sqrt{b}$ targeted clean-up events in I clear items that arrived during I. We next focus only on those $\sqrt{b}$ targeted clean-up events. We number them by $1, \ldots, \sqrt{b}$.

Let χ_j be the color cleared in the jth targeted clean-up event. We say that a targeted clean-up event j is an *OPT-present* clean-up event if OPT has a server on χ_j during I; otherwise, this event is *OPT-absent*. Let ℓ be the number of OPT-present events and $\sqrt{b}-\ell$ be the number of OPT-absent events. In each of the $\sqrt{b}-\ell$ OPT-absent events, OPT accumulates $2\sqrt{b}$ items arriving during I. We count only the first $\sqrt{b}$ items in each such event due to a reason that will be explained later. Thus, summing up over all OPT-absent events, it follows that OPT accumulates at least $\sqrt{b}(\sqrt{b}-\ell)$ items. The crucial observation needed to complete the proof is that every OPT-present clean-up event implies that there is an OPT server missing from a marked color at some phase. Specifically, let us concentrate on an OPT-present clean-up event j. The $2\sqrt{b}$ unmarked items cleared at that event must have been accumulated in the buffer of ON during at least two marking phases; otherwise, the underlying color would have been marked. Let i_j be the first phase during which the unmarked items cleared by the jth targeted clean-up event started accumulating. Let d_i be the number of OPT-present targeted clean-up events whose items started accumulating at phase i, that is, $d_i = |\{j : i = i_j\}|$. Notice that during the marking phase i, the servers of OPT are not present on at least d_i colors that become marked. Since $\sqrt{b}$ items arrive to each of those colors, OPT accumulates in its buffer at least $d_i\sqrt{b}$ items during that phase. As a result, OPT accumulated at least $\ell\sqrt{b}$ additional items as $\sum_i d_i = \ell$.

Notice that the number of items accumulated in OPT's buffer during I is at least $\sqrt{b}(\sqrt{b}-\ell)+\ell\sqrt{b} = b$, a contradiction to our assumption that OPT does not move during I. Recall that we assumed that OPT accumulates $\sqrt{b}$ (and not $2\sqrt{b}$) items in each OPT-absent event. This is required to ensure that the sets of accumulated items due to OPT-absent and OPT-present events are disjoint. In general, an item cleared in a targeted clean-up move may be counted as one of $\sqrt{b}$ items that induced a marking move. However, none of the first $\sqrt{b}$ items accumulated may be counted towards a marking move since otherwise, the underlying color becomes marked before the buffer counter reaches $2\sqrt{b}$, and thus, a targeted clean-up event cannot occur. □

Lemma 2.8 implies that $ON^T \leq 4\sqrt{b} \cdot (OPT + 1)$ since every targeted clean-up event consists of two targeted clean-up moves. Since we may assume that OPT moves at least once, this immediately gives us the following lemma.

Lemma 2.9. $ON^T = O(\sqrt{b}) \cdot OPT$.

Unmarked clean-up moves. We finally bound the number of unmarked clean-up moves ON^U. Recall that an unmarked clean-up event takes place when either (1) the unmarked sub-buffer has $\sqrt{b}/4$ different colors, or (2) after $\sqrt{b}$ targeted clean-up events. For the sake of the analysis, it is sufficient that we focus only on unmarked clean-up moves due to type (1). Clean-up moves due to type (2) can be charged against the targeted clean-up moves with an additional constant multiplicative factor. Specifically, one can observe that the number of unmarked clean-up moves of type (2) is no more than $ON^T/8$. During $\sqrt{b}$ targeted clean-up events there are $2\sqrt{b}$ targeted clean-up moves, while the unmarked clean-up event that results from this sequence of targeted

clean-ups has at most $\sqrt{b}/4$ unmarked clean-up moves; otherwise, an unmarked clean-up event of type (1) would take place before that. As a result, in the remainder of this subsection, when we refer to unmarked clean-up events or moves, we specifically mean unmarked clean-up events or moves of type (1).

Let u_i be the number of unmarked clean-up events in phase i. We partition the marking phases into groups of *clean-up phase sequences*. A clean-up phase sequence is a sequence of consecutive marking phases such that each phase belongs to exactly one clean-up phase sequence. A clean-up phase sequence S ends when $\sum_{i \in S} u_i > 60\sqrt{b}$. Let $u_S = \sum_{i \in S} u_i$, and observe that a straightforward upper bound on the number of unmarked clean-up moves in any sequence S is $ON_S^U = O(u_S\sqrt{b})$ since every unmarked clean-up event has $\sqrt{b}/4 + 1$ moves. Since we do not have an upper bound on u_S, it is convenient to consider two types of clean-up phase sequences: a clean-up phase sequence S that has a phase $i \in S$ such that $u_i > 6\sqrt{b}$, and a sequence S that does not have such a phase.

Lemma 2.10. *$OPT_i = \Omega(u_i)$ for a phase i such that $u_i > 6\sqrt{b}$.*

Proof. Notice that all the unmarked items that were cleared during phase i, with the exception of the items cleared in the first unmarked clean-up event of the phase, arrived during phase i. As a result, the number of unmarked items which arrive during phase i is at least $(u_i - 1) \cdot \sqrt{b}/4$. Note that no color is associated with more than $\sqrt{b}$ of these items. In phase $i - 1$, there are k marked colors. None of the previously mentioned $(u_i - 1) \cdot \sqrt{b}/4$ unmarked items can be from those colors as any item of those colors arriving in phase i would not be considered an unmarked item. Let us restrict our attention to the first $\sqrt{b}$ items of each of those marked colors. Summing up, we know that $(u_i - 1) \cdot \sqrt{b}/4 + k\sqrt{b}$ items arrived during phases $i - 1$ and i, and no single color has more than $\sqrt{b}$ of these items.

During phases $i-1$ and i, the servers of OPT could not have been located in more than $k+OPT_i$ different colors. Therefore, they could have served no more than $\sqrt{b} \cdot (k + OPT_i)$ of the previously mentioned items. The remaining items must have entered the buffer of OPT. Since OPT's buffer cannot accommodate more than b items, then

$$\frac{(u_i - 1)\sqrt{b}}{4} + k\sqrt{b} - \sqrt{b} \cdot (k + OPT_i) = \sqrt{b} \cdot \left(\frac{u_i - 1}{4} - OPT_i \right) \leq b \,.$$

This implies that $OPT_i \geq (u_i - 1)/4 - \sqrt{b} \geq u_i/24$, where the last inequality holds since $u_i > 6\sqrt{b}$. □

We now introduce the notion of an *extended phase*. An extended phase is defined only for phases i with $u_i > 0$. The extended phase includes phase i and phases $i - 1, i - 2$, and so on, until a phase i' with $u_{i'} > 0$. As a result, any extended phase contains at least two phases, and only the first and last of them has unmarked clean-up events. For any clean-up phase sequence S, let OPT_S be the number of moves of OPT servers during the extended phases of all relevant $i \in S$.

Lemma 2.11. *$OPT_S = \Omega(\sqrt{b})$ for any clean-up phase sequence S such that $u_i \leq 6\sqrt{b}$ in all phases $i \in S$.*

Proof. Let us assume by way of contradiction that $OPT_S \leq \sqrt{b}/60$. Let χ_S be the set of colors that OPT visited during the extended phases of S. Since OPT makes at most $\sqrt{b}/60$ moves, $|\chi_S| \leq k + \sqrt{b}/60$. Let x_i be the number of items arriving during the extended phase i whose colors are not in χ_S. We next argue that $x_i \geq u_i\sqrt{b}/30$ for every extended phase $i \in S$. Then, we get that the number of items arriving during the extended phases of S whose colors are not in χ_S is at least $\sum_{i \in S} x_i/2 \geq \sum_{i \in S} u_i\sqrt{b}/60 > b$, where the half factor is due to the fact that every item belongs to at most two extended phases, and the last inequality results since $u_S > 60\sqrt{b}$. This implies that OPT must have accumulated more than b items in its buffer, a contradiction.

We turn to prove the above-mentioned argument. Let us focus on the extended phase i, and recall that phase $i-1$ is within this extended phase. At phase $i-1$, k colors receive at least $\sqrt{b}$ items and become marked. Let R_i be the set of these colors that are not in χ_S, and let $r_i = |R_i|$. Note that at least $r_i\sqrt{b}$ items arrive out of χ_S. If $r_i \geq u_i/30$ then $x_i \geq u_i\sqrt{b}/30$, and we are done. Hence, in the remainder of this proof, we may assume that $r_i < u_i/30$ and $|\chi_S \cup R_i| < k + \sqrt{b}/60 + u_i/30$. Notice that no more than $\sqrt{b}/60 + u_i/30$ of the colors in $\chi_S \cup R_i$ may correspond to unmarked items in phase i. This follows since $\chi_S \cup R_i$ includes the k colors marked in phase $i-1$. Each unmarked clean-up event cleans items from $\sqrt{b}/4$ different unmarked colors, and at least $\sqrt{b}/4 - \sqrt{b}/60 - u_i/30$ items of colors outside χ_S. However, $\sqrt{b}/4 - \sqrt{b}/60 - u_i/30 = 7\sqrt{b}/30 - u_i/30 \geq \sqrt{b}/30$, where the last inequality follows since $u_i \leq 6\sqrt{b}$ in any phase $i \in S$. This implies that at least $u_i \cdot \sqrt{b}/30$ unmarked items were cleaned in phase i. Notice that an unmarked item cleaned in phase i must have arrived at the extended phase i, and thus, $x_i \geq u_i\sqrt{b}/30$. □

We can now complete the main contribution of this subsection.

Lemma 2.12. $ON^U = O(\sqrt{b}) \cdot OPT + O(b)$.

Proof. Notice that $OPT \geq \sum_S OPT_S/2$. Hence, it is sufficient that we establish the above mentioned bound for each clean-up phase sequence. We prove that $ON^U_S = O(\sqrt{b}) \cdot OPT_S$, for every clean-up phase sequence S except the last one. We then complete the proof by demonstrating that the last clean-up phase sequence contributes an additive value of $O(b)$.

Consider a clean-up phase sequence S that is not the last one. Observe that if there is a phase $i \in S$ such that $u_i > 6\sqrt{b}$ then $u_S = \Theta(\max_{i \in S} u_i)$. This observation uses the fact that such a clean-up phase sequence ends when $\sum_{i \in S} u_i > 60\sqrt{b}$. Using Lemma 2.10, one can infer that $OPT_S = \Omega(\max_{i \in S} u_i)$, and the claimed bound follows by recalling that $ON^U_S = O(u_S\sqrt{b})$. In case that $u_i \leq 6\sqrt{b}$ for all phases i of S, then $u_S = O(\sqrt{b})$, and therefore, $ON^U_S = O(b) = O(\sqrt{b}) \cdot OPT_S$, where the last equality follows from Lemma 2.11. Now, let us focus on the last clean-up phase sequence S'. Clearly, $u_{S'} < 60\sqrt{b}$, and hence, $ON^U_{S'} = O(b)$. □

Putting everything together. Combining the bounds from Lemma 2.7, Lemma 2.9, and Lemma 2.12, gives the main theorem of this section. Note that in adherence to competitive analysis and online algorithms research, we allow additive terms that are independent of the input stream and its properties.

Theorem 2.13. *There is a randomized algorithm whose competitive ratio is* $O(\sqrt{b} \ln k)$.

3 A Deterministic Algorithm

We develop two deterministic algorithms: the first has a competitive ratio of $O(\ln b/\delta^2)$ in a δ-augmentation setting and a competitive ratio of $O(k^2 \ln b)$ when there is no server augmentation, and the other has a competitive ratio of $O(b/\delta)$ in a δ-augmentation setting and a competitive ratio of $O(kb)$ with no augmentation. Then, one can execute the algorithm that achieves a better competitive ratio depending on the underlying parameters k, b, and δ. This results in a $O(\min\{k^2 \ln b, kb\})$-competitive algorithm, and a $O(\min\{\ln b/\delta^2, b/\delta\})$-competitive algorithm for the δ-augmentation settings. Note that in a δ-augmentation setting, an online algorithm may employ $k/(1-\delta)$ servers where $\delta \in (0,1)$, while an optimal offline adversary can employ at most k servers.

Algorithm 1. Our first algorithm sensibly combines the algorithm for the reordering buffer problem on arbitrary metric spaces [10], and the FIFO algorithm for the paging problem [19]. Specifically, we utilize the algorithm for reordering buffer to decide which color to serve next, and then use the FIFO algorithm to decide which of the servers moves to serve this color. Formally, the algorithm consists of alternating selection and

service steps. We maintain a cost $c_\chi \in [0,1]$ for every color χ, which is initially 0. During the *selection* step the buffer is full, and the cost of all inactive colors, namely, colors that currently do not consist of a server, is incremented. The cost of each color is incremented at a rate proportional to the number of items it has in the buffer. Once the cost of some color reaches 1, this color is selected, and the selection step ends. If more than one color reaches a cost of 1 then one of them is selected arbitrarily. Once the selection step ends, the service step begins. In the *service* step, a server is moved to the selected color. The choice of which server should be moved is done in a FIFO manner, that is, we move the server that has not moved the longest. Then, the cost of the selected color drops from 1 to 0, and all its items in the buffer are cleared and served. This makes room in the buffer for more items. Arriving items from active colors are served immediately, while items from inactive colors are accumulated in the buffer. Once the buffer is full again, a new selection step starts. Note that colors retain their cost from the previous selection step.

Theorem 3.1. *Algorithm 1 achieves a competitive ratio of $O(\ln b/\delta^2)$ in a δ-augmentation setting and a competitive ratio of $O(k^2 \ln b)$ when there is no server augmentation.*

Proof. Let ON and OPT denote our algorithm and the optimal offline algorithm, respectively. We also denote the overall number of server moves of ON and OPT by ON and OPT, respectively. In addition, we let ON_χ and OPT_χ be the overall number of ON and OPT server moves towards a color χ. Recall that an item of color χ, which is in the buffer during a selection step, generates a cost on χ. Note that during a given selection step, each of the b items in the buffer generate the same amount of cost.

Lemma 3.2. *The overall cost generated on color χ during the selection steps is ON_χ.*

Proof. Notice that c_χ only increases during the selection steps, and decreases from 1 to 0 whenever χ is selected and ON moves a server to it. Since this happens exactly ON_χ times, and since $c_\chi = 0$ at the beginning and end of the algorithm, then ON_χ is equal to the total cost generated on χ. □

Given $\delta \in [0,1)$, we define a phase as a sequence of $k/(1-\delta)$ consecutive selection and service steps. Since ON moves its servers in a FIFO manner, then each server moves exactly once in every phase except maybe the last one. This implies that if we define X_j to be the set of colors that ON moved its servers to during phase j, then each $|X_j| \leq k/(1-\delta)$. Let m be the number of phases in our algorithm. We may assume without loss of generality that $m > 1$; otherwise, the servers of ON make at most $k/(1-\delta)$ moves and this can be treated as an additive factor in the competitive ratio.

Let $N_j = \{\chi : \chi \in X_j \text{ and } \chi \text{ is OPT-vacant during phase } j\}$. We say that χ is OPT-vacant during phase j if there is some point of time during that phase where OPT does not have a server on χ. This implies that no single server of OPT is located on χ for the entire phase j. Let $VAC = \sum_{j=1}^{m} |N_j|$ be the overall number of ON server moves to colors in N_j in the corresponding phases j. We also define $VAC_\chi = |\{j : \chi \in N_j\}|$, and note that $VAC = \sum_\chi VAC_\chi$. Let $\alpha = \min\{3k, 2/\delta\}$.

Lemma 3.3. $ON/VAC \leq \alpha$.

Proof. We first prove that $ON/VAC \leq 3k$. Clearly,

$$ON \leq m\frac{k}{1-\delta} \tag{1}$$

since there are m phases, and in each phase ON moves at most $k/(1-\delta)$ servers. Let us consider any two consecutive phases j and $j+1$. One can easily validate that $|X_j \cup X_{j+1}| \geq k/(1-\delta)+1$ since X_j and X_{j+1}

differ in at least one color. Since OPT has k servers, VAC is incremented by at least $k/(1-\delta)+1-k$ in those two phases, and thus,

$$VAC \geq \left\lfloor \frac{m}{2} \right\rfloor \left(\frac{k}{1-\delta} + 1 - k \right) . \tag{2}$$

Combining equations 1 and 2, one obtains that

$$\frac{ON}{VAC} \leq \frac{m\frac{k}{1-\delta}}{\left\lfloor \frac{m}{2} \right\rfloor \left(\frac{k}{1-\delta} + 1 - k \right)} \leq \frac{3k}{1+\delta\cdot(k-1)} \leq 3k ,$$

where the second inequality uses the assumption that $m > 1$. We next prove that $ON/VAC \leq 2/\delta$. For this purpose, let us focus on the $m-1$ first phases. Recall that $|X_j| = k/(1-\delta)$ in each of the phases $j \in [m-1]$. Since OPT has k servers, VAC is incremented by at least $k/(1-\delta)-k$ in each of these phases, and hence,

$$\frac{ON}{VAC} \leq \frac{m\frac{k}{1-\delta}}{(m-1)\left(\frac{k}{1-\delta} - k \right)} \leq \frac{2}{\delta} ,$$

where the last inequality again utilizes the assumption that $m > 1$. □

We next upper bound the ratio VAC/OPT. For this purpose, we introduce the notion of a discount. We mark the discount of every color χ by d_χ. Note that the discount value is only used in the analysis, although we associate it with the run of ON. The discount value is incremented during the selection step of the algorithm similarly to the cost value. However, the discount value of each color is incremented at a rate proportional to the number of items in the buffer of OPT (and not ON). Moreover, the rate in which the discount is incremented is 2α times slower than the rate in which the cost is incremented. That is, if a single item in ON's buffer induces a ∂x increment to the cost of its color in some time interval in a selection step, then a single item in OPT's buffer induces a $\partial x/(2\alpha)$ increment to the discount of its color. Finally, d_χ is set to 0 whenever OPT or ON serve the color χ. Let DIS_χ be the total discount generated by items of color χ, and let $DIS = \sum_\chi DIS_\chi$.

Lemma 3.4. $ON \geq 2\alpha \cdot DIS$.

Proof. By Lemma 3.2, we know that ON is equal to the overall cost generated by all colors during all selection steps. Consequently, the lemma follows by recalling that the cost is generated at 2α times faster than the discount, and at every selection step, the buffer of ON is full, and thus, has at least the same number of items as the buffer of OPT. □

A corollary of Lemma 3.3 and Lemma 3.4 is the following.

Corollary 3.5. $DIS \leq VAC/2$.

We now introduce the notion of an amortized cost. We denote the amortized cost of a color χ by A_χ. A move of an ON server to χ during phase j is counted as an amortized move if and only if $\chi \in N_j$ and $d_\chi < 1$ during that time. In other words, the amortized cost counts ON moves to colors on which OPT does not have a statically stationed server for the entire phase, and those colors did not accumulate too much discount. Let $AMR = \sum_\chi A_\chi$.

Lemma 3.6. $VAC/AMR \leq 2$.

Proof. Notice that $VAC_\chi \leq A_\chi + DIS_\chi$ for every color χ. This holds since VAC_χ counts the number of moves to χ when χ is OPT-vacant, and these moves are either counted for A_χ or make d_χ decrease by at least 1. Recall that DIS_χ is the overall increments of d_χ over the run of the algorithm as well as the overall decrements of d_χ. As a result, we attain that

$$VAC = \sum_\chi VAC_\chi \leq \sum_\chi (A_\chi + DIS_\chi) \leq AMR + DIS \leq AMR + \frac{VAC}{2} ,$$

where the last inequality follows from Corollary 3.5. □

We next relate AMR to OPT. For this purpose, we consider some color χ, and give an upper bound on the number of times that A_χ is incremented between any two consecutive increments of OPT_χ. We note that this result essentially sets an upper bound on the number of times that A_χ is incremented before OPT_χ must be incremented. Clearly, between two consecutive increments of OPT_χ, there is a time interval in which an OPT server is located on χ and a time interval in which no OPT server is located on χ. During the time interval that an OPT server is located on χ, A_χ cannot increase by more than 2 since an ON server move is counted as amortized only in phases when χ is OPT-vacant. Specifically, an ON server move may be counted as amortized either in the phase shortly after an OPT server moved to χ or shortly before it left χ. We are left to provide an upper bound for the number of amortized moves to χ during a time interval in which no OPT server is located on χ.

Notice that if no OPT server is located on χ during some time interval then items of χ arriving during that interval accumulate in OPT's buffer. As a result, d_χ increases faster, and once $d_\chi \geq 1$, we do not count an ON server move to χ as amortized. Formally, fix a time interval in which no server of OPT is located on χ. Let r be the number amortized moves to χ during that interval, and let $x_0, \ldots, x_{r-1}$ be the number of items served by ON servers in those amortized moves.

Lemma 3.7. $\sum_{i=1}^{r-1} x_i \leq b$.

Proof. Since no server of OPT is located on χ and since OPT's buffer can accommodate at most b items then the overall arriving items is at most b. Note that the first x_0 items that are served by an ON server may be served by an OPT server before it left χ, and thus, they are not taken into account in the sum. □

Lemma 3.8. $\sum_{i=1}^{j-1} x_i < 2\alpha x_j$ *for every* $1 \leq j \leq r-1$.

Proof. During the selection steps between ON's jth amortized move to χ and the move that comes before it, the underlying x_j items collectively generate a cost of 1. At the same time, OPT has at least $\sum_{i=1}^{j-1} x_i$ items of color χ in its buffer. These items generate a discount of at least $1/x_j \cdot \sum_{i=1}^{j-1} x_i/(2\alpha)$. Since we focus on an amortized move then this discount is less than 1, and hence, $\sum_{i=1}^{j-1} x_i \leq 2\alpha x_j$. □

Using the last lemma, we can derive the following lower bound on the number of arriving items.

Lemma 3.9. $\sum_{i=1}^{j} x_i \geq (1 + 1/(2\alpha))^{j-1}$ *for every* $1 \leq j \leq r-1$.

Proof. We prove this lemma by induction on j. The lemma clearly holds when $j = 1$ since $x_1 \geq 1$. Now notice that

$$\sum_{i=1}^{j} x_i = x_j + \sum_{i=1}^{j-1} x_i > \frac{\sum_{i=1}^{j-1} x_i}{2\alpha} + \sum_{i=1}^{j-1} x_i = \left(1 + \frac{1}{2\alpha}\right) \sum_{i=1}^{j-1} x_i \geq \left(1 + \frac{1}{2\alpha}\right)^{j-1} ,$$

where the first inequality holds by Lemma 3.8, and the last inequality results by the induction hypothesis. □

Lemma 3.10. $AMR/OPT = O(\alpha \ln b)$.

Proof. Recall that we argued before that the ratio AMR/OPT is upper bounded by $2 + r$. By Lemma 3.7 and Lemma 3.9, we know that

$$\left(1 + \frac{1}{2\alpha}\right)^{r-2} \leq \sum_{i=1}^{r-1} x_i \leq b \,,$$

and thus, $r \leq \ln b / \ln(1 + 1/(2\alpha)) + 2$. It is well-known that $\ln(1 + x) \geq x/2$ for any $0 \leq x \leq 1$, and hence, $\ln(1 + 1/(2\alpha)) \geq 1/(4\alpha)$. As a result, $r \leq 4\alpha \ln b + 2$. □

By Lemmas 3.3, 3.6 and 3.10, we attain that

$$\frac{ON}{OPT} = \frac{ON}{VAC} \cdot \frac{VAC}{AMR} \cdot \frac{AMR}{OPT} \leq \alpha \cdot 2 \cdot O\left(\alpha \ln b\right) = O\left(\alpha^2 \ln b\right) \,.$$

Recall that $\alpha = \min\{3k, 2/\delta\}$. As a result, the competitive ratio of the algorithm is $O(\ln b/\delta^2)$. When there is no server augmentation then $\alpha = 3k$. Consequently, the competitive ratio is $O(k^2 \ln b)$. □

Algorithm 2. Our second algorithm is simply a FIFO algorithm. This algorithm completely ignores the buffer, and serves the items according to their arrival order. Specifically, when an item arrives, it is immediately served by either a server that is already located on the corresponding color, or by moving a server that has not moved the longest to that color.

Theorem 3.11. *There is a deterministic algorithm whose competitive ratio is $O(b/\delta)$ in a δ-augmentation setting and $O(kb)$ when there is no augmentation.*

Proof. We prove the theorem by establishing that OPT must move at least once in every sequence of $b(k + 1)$ (respectively, b/δ in δ-augmentation setting) consecutive ON server moves. For this purpose, let us focus on a sequence of consecutive online server moves during which time OPT does not move its servers. Observe that in every sub-sequence of $k + 1$ (respectively, $k/(1 - \delta)$) consecutive moves, the online servers move to $k + 1$ (respectively, $k/(1 - \delta)$) different colors. If OPT does not move during this sub-sequence, it must have accumulated at least one item (respectively, $k/(1 - \delta) - k$ items) in its buffer. After b (respectively, $b \cdot (1 - \delta)/(\delta k)$) such sub-sequences, OPT must move since its buffer is full. This implies that in every sequence of $(k + 1)b$ (respectively, $k/(1 - \delta) \cdot (b \cdot (1 - \delta)/(\delta k)) = b/\delta$) consecutive ON server moves, OPT must move at least once. □

4 Conclusions

We made a first step in analyzing the generalized reordering buffer management problem in an online setting, and provided non-trivial upper bounds on the competitive ratio. An obvious direction for future research is the design of new algorithms with further improved bounds. By now, both the paging problem and the reordering buffer problem are very well understood. It seems natural to utilize the known techniques and state of the art algorithms for these problems in order to obtain better results for generalized reordering buffer. Nevertheless, it turns out to be a challenging task to establish bounds on such combinations. We do not know whether natural combinations of such algorithms can attain good performance guarantees, although we have identified several combinations that fail. Any progress in this direction would be of great interest.

References

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

[2] Anna Adamaszek, Artur Czumaj, Matthias Englert, and Harald Räcke. Almost tight bounds for reordering buffer management. In *STOC*, pages 607–616, 2011.

[3] Yuichi Asahiro, Kenichi Kawahara, and Eiji Miyano. Np-hardness of the sorting buffer problem on the uniform metric. In *FCS*, pages 137–143, 2008.

[4] Noa Avigdor-Elgrabli and Yuval Rabani. An improved competitive algorithm for reordering buffer management. In *SODA*, pages 13–21, 2010.

[5] Noa Avigdor-Elgrabli and Yuval Rabani. A constant factor approximation algorithm for reordering buffer management. In *SODA*, pages 973–984, 2013.

[6] Noa Avigdor-Elgrabli and Yuval Rabani. An optimal randomized online algorithm for reordering buffer management. *CoRR*, abs/1303.3386, 2013.

[7] Reuven Bar-Yehuda and Jonathan Laserson. Exploiting locality: Approximating sorting buffers. *Journal of Discrete Algorithms*, 5(4):729–738, 2007.

[8] Siddharth Barman, Shuchi Chawla, and Seeun Umboh. A bicriteria approximation for the reordering buffer problem. In *ESA*, pages 157–168, 2012.

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

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

[11] Matthias Englert and Matthias Westermann. Reordering buffer management for non-uniform cost models. In *ICALP*, pages 627–638, 2005.

[12] Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel Dominic Sleator, and Neal E. Young. Competitive paging algorithms. *CoRR*, cs.DS/0205038, 2002.

[13] Iftah Gamzu and Danny Segev. Improved online algorithms for the sorting buffer problem on line metrics. *ACM Transactions on Algorithms*, 6(1):15:1–15:14, 2009.

[14] Anna R. Karlin, Mark S. Manasse, Larry Rudolph, and Daniel Dominic Sleator. Competitive snoopy caching. *Algorithmica*, 3:77–119, 1988.

[15] Rohit Khandekar and Vinayaka Pandit. Online sorting buffers on line. In *STACS*, pages 584–595, 2006.

[16] Jens S. Kohrt and Kirk Pruhs. A constant approximation algorithm for sorting buffers. In *LATIN*, pages 193–202, 2004.

[17] Lyle A. McGeoch and Daniel Dominic Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6(6):816–825, 1991.

[18] Harald Räcke, Christian Sohler, and Matthias Westermann. Online scheduling for sorting buffers. In *ESA*, pages 820–832, 2002.

[19] Daniel Dominic Sleator and Robert Endre Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985.