Efficient Online Weighted Multi-Level Paging

Nikhil Bansal*
CWI Amsterdam and TU Eindhoven
Netherlands
N.Bansal@cwi.nl

Joseph (Seffi) Naor†
Computer Science Department
Technion, Israel
Israel
naor@cs.technion.ac.il

Ohad Talmon
Computer Science Department
Technion
Israel
ohad@cs.technion.ac.il

ABSTRACT

We study the writeback-aware caching problem, a variant of classic paging where paging requests that modify data and requests that leave data intact are treated differently. We give an $O(\log^2 k)$ competitive randomized algorithm, answering an open question of Beckmann *et al.* [8] and Even *et al.* [21] about the existence of a randomized poly-logarithmic competitive algorithm. Our algorithm also works for arbitrary page weights. We also give an $O(k)$ competitive deterministic algorithm, extending the previous result of Beckmann et al. [8] to the weighted setting.

Interestingly, we also show that any polynomial-time randomized algorithm must be $\Omega(\log^2 k)$-competitive, assuming $NP \not\subset BPP$, based on a connection to online set-cover and using ideas of Feige and Korman [24]. This gives a surprising separation from the classical paging problem, where several tight $O(\log k)$-competitive algorithms are known.

A key underlying observation is that writeback-aware caching is algorithmically equivalent to Read/Write (RW) paging, which is an interesting problem on its own and has also been studied previously. We consider a further generalization of RW-paging to *multi-level paging*, where a page can have ℓ different types (RW-paging corresponds to $\ell = 2$), and give $O(k)$-competitive deterministic and $O(\log^2 k)$-competitive polynomial-time randomized algorithms, without any dependence on ℓ.

Our randomized algorithms are based on first finding an $O(\log k)$-competitive fractional solution to an online linear problem that has additional complicating constraints beyond the usual covering or packing type. Then, we round this fractional solution online losing an additional $O(\log k)$ factor. For $\ell = 1$, which corresponds to the well-studied weighted paging, our approach gives a substantially simpler and natural rounding algorithm compared to the previous approaches, which might be of independent interest, albeit at the loss of an additional $O(\log k)$ factor.

*Supported by the ERC Consolidator Grant 617951 and the NWO VICI grant 639.023.812.
†Supported in part by US-Israel BSF grant 2018352 and by ISF grant 2233/19 (2027511).

SPAA '21, July 6–8, 2021, Virtual Event, USA.

ACM ISBN 978-1-4503-8070-6/21/07…$15.00
https://doi.org/10.1145/3409964.3461801

CCS CONCEPTS

• **Theory of computation → Caching and paging algorithms**; *Rounding techniques*; Linear programming.

KEYWORDS

Online Algorithms, Competitive Analysis, Caching

ACM Reference Format:
Nikhil Bansal, Joseph (Seffi) Naor, and Ohad Talmon. 2021. **Efficient Online Weighted Multi-Level Paging. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '21), July 6–8, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3409964.3461801**

1 INTRODUCTION

Paging is one of the earliest and most extensively studied problems in online computation and competitive analysis [1, 3, 5, 6, 14, 15, 25, 29–31, 34, 35, 37, 39], including works on non-standard caching models, e.g., elastic caches [27], caching with time windows [28], caching with dynamic weights [21], and caching with machine learning predictions [33]. In fact, online paging has become a focal point for many of the recent developments in competitive analysis, e.g., the online primal-dual method, projections, and mirror descent [16–18]. Our paper studies *writeback-aware caching*, a non-standard caching model studied recently, and its generalizations. Before describing this model, let us first recall the standard weighted paging problem.

Weighted Paging. In this problem, there is a universe of n pages, each page has a weight (fetch cost), and there is a cache that can hold up to k pages. At each time step a page is requested, if the requested page is already in the cache then no cost is incurred, otherwise the algorithm must fetch the page into the cache, incurring a cost equal to its weight. The goal is to minimize the total cost incurred.

By now, weighted paging is well understood. In their seminal paper, Sleator and Tarjan [35] showed that any deterministic algorithm is at least k-competitive, and that LRU (Least Recently Used) is exactly k-competitive for unweighted paging. The k-competitive bound was later generalized to weighted paging as well [20, 38]. When randomization is allowed, Fiat *et al.* [25] gave the elegant Randomized Marking algorithm for unweighted paging, which is $\Theta(\log k)$-competitive against an oblivious adversary.

Building on a long line of work, Bansal et al. [5] gave a $\Theta(\log k)$-competitive randomized algorithm for weighted paging based on the online primal-dual framework [4, 18]. Their algorithm uses a two-step approach. First, a deterministic competitive algorithm is designed for a suitable fractional version of the problem. Then, a

randomized online algorithm is obtained by *rounding* the deterministic fractional solution online. In particular, the online rounding maps a fractional solution into a probability distribution over integral cache states. This mapping is maintained carefully, so that when the fractional solution changes, the probability distribution can be updated, so that the expected cost in updating it is $O(1)$ times the fractional cost. This online rounding is highly implicit and impractical, in sharp contrast, e.g., to the elegant Randomized Marking algorithm of [25] for unweighted online paging, which is easy to describe and implement in practice.

1.1 Our Model and Results

In writeback-aware caching, paging requests that modify the data (called *write requests*) and requests that leave the data intact (called *read requests*) are treated differently. Upon eviction, a modified data item needs to be written back to memory, incurring a high cost, while an unmodified data item can just be discarded, incurring a much lower cost. This distinction is further motivated by other recent trends in memory systems (see [8] for more details). Furthermore, this lack of symmetry between the cost of read and write has provided new perspectives on algorithm design, where write-efficiency is required, and has lead to a long line of recent research on algorithms that mitigate writing into memory [9–13, 19, 26].

The theoretical study of writeback-aware caching was recently initiated by Beckmann *et al.* [8]. A simpler version of the problem was already studied by Farach-Colton and Liberatore [22] who showed that the offline problem is NP-complete. Beckmann *et al.* [8] generalized classical paging algorithms and obtained the first $O(k)$-competitive deterministic algorithm for writeback-aware caching. They left the problem of obtaining randomized online paging algorithms in this setting open. Even *et al.* [21] considered a model in which page weights may change over time. Note that in writeback-aware caching, when a page residing in the cache is modified by a write request, its weight increases. However, the model of [21] is too restrictive to capture writeback-aware caching, and in fact [21] state obtaining a randomized algorithm for writeback-aware caching as an open problem.

Results for writeback-aware caching. We explore writeback-aware caching in full generality and assume that read and write costs can be page-dependent, in contrast to the *uniform* read and write costs assumption of [8]. We call this problem the *weighted writeback-aware caching*. We obtain the following results.

Theorem 1.1. *There is an $O(k)$-competitive deterministic algorithm for weighted writeback aware caching.*

This extends the results of [8] to the weighted setting. When randomization is allowed, we show the following.

Theorem 1.2. *There is an $O(\log^2 k)$-competitive randomized algorithm for weighted writeback-aware caching against an oblivious adversary.*

At first glance, writeback-aware caching may seem as a simple variant of weighted paging. However, it turns out that there are fundamental differences from weighted paging. In particular, we show the following hardness result, which gives a surprising separation from weighted paging.

Theorem 1.3. *For writeback-aware caching, unless $NP \subset BPP$, there is no polynomial-time randomized algorithm that is $o(\log^2 k)$-competitive against an oblivious adversary.*

In particular, the competitive factor we achieve for writeback-aware caching in Theorem 1.2 is tight, and cannot be improved from $O(\log^2 k)$ to $O(\log k)$, while maintaining polynomial-time running time. Our techniques also imply the following.

Theorem 1.4. *For writeback-aware caching, any randomized online algorithm (possibly exponential-time) that first solves the fractional problem and then does online rounding, must lose a factor of $\Omega(\log k)$ in the rounding.*

These hardness results are based on the observation that writeback-aware caching can encode the online set-cover problem, and is thus qualitatively different, and much harder than weighted paging.

Later, we will elaborate on the new algorithmic ideas required for writeback-aware caching beyond those for weighted paging. We also develop a new online rounding technique for writeback-aware caching which is very simple to implement. In particular, this gives the first *distribution-free* rounding for weighted paging with a poly-logarithmic competitive ratio.

Read/Write paging and Multi-level paging. Our main algorithmic tool is a more general variant of weighted paging, that we call *Read/Write* (RW) paging. In this problem each page p has two copies, $(p, 1)$ and $(p, 2)$, corresponding to *write* and *read* copies of p, respectively. Requests for pages are either write or read. A write request for page p can only be served by copy $(p, 1)$, while a read request can be served by both $(p, 1)$ and $(p, 2)$. The cache is not allowed to contain both copies of a page, and evicting $(p, 1)$ costs at least as much as evicting $(p, 2)$.

We show that RW paging is (algorithmically) equivalent to writeback-aware caching and design our algorithms for the writeback-aware caching problem through this problem. We describe the precise relation between the two problems in Section 2.

In fact, we generalize RW paging to multi-level paging, where there are ℓ copies of each page, at levels $1, \ldots, \ell$. A request (p, i) for page p at level i can be satisfied by any copy of page p at levels $1, \ldots, i$. For any page p, *at most one* copy, over all levels, can be stored in the cache. The eviction costs are arbitrary for each page and level, provided that for each page they are monotonically non-increasing over the levels. Note that RW-paging corresponds to the case $\ell = 2$.

Interestingly, RW and multi-level paging are intriguing problems by themselves, with several natural motivations. In practical systems, multi-level paging models situations where a request for a data item can be satisfied by several pages. For example, Intel's Optane SSD [36] can serve requests in different granularities, e.g., fetching a 4KB aligned chunk into the cache can serve read requests for any of the 8 sectors contained in it, and may be preferable to fetching individual sectors. A different setting in which a request can be satisfied by more than one data item (called substitutability) is caching of a training set in deep learning training [32].

We prove Theorem 1.1 and Theorem 1.2 in this general setting of multi-level paging.

Theorem 1.5. *There is an $O(k)$-competitive determinsitic algorithm, and an $O(\log^2 k)$-competitive, polynomial-time randomized online algorithm for ℓ-level multi-level paging.*

We remark that our bounds have no dependence on the number of levels ℓ.

1.2 Overview of techniques

Our randomized online algorithms follow the two-step approach of weighted paging. First, a fractional deterministic competitive algorithm is designed, and then a randomized online algorithm is obtained by rounding the deterministic fractional solution online.

Non-covering LP. The deterministic algorithms (both fractional and integral) for RW-paging and multi-level paging follow from many of the ideas developed in previous works, but there are some crucial differences. First, the constraint that for any page p there can be at most one copy (p, i) over all $i \in [\ell]$, introduces non-trivial interactions between the copies of a page. This introduces additional comparison (or precedence) constraints (of the type $x_i \leq x_j$) in the LP formulation, in addition to the standard covering constraints for paging problems. Second, we need a somewhat non-standard LP, where the variables correspond to prefixes of copies of a page.

To illustrate the variety of techniques, we provide both primal-dual algorithms, and algorithms with a potential function based analysis.

Distribution-free rounding. Our online rounding algorithm makes its decisions at each step using a very simple rule, based only on the fractional solution and the pages in *current* cache. It is thus *distribution free*, in contrast to the way fractional solutions to weighted paging are rounded, where the online rounding maintains a probability distribution over integral cache states, and updates it when the fractional solution changes.

Roughly, our rounding algorithm tries to mimic the fractional algorithm, but evicts pages $O(\log k)$ times more aggressively. However, as it can only evict pages currently residing in cache, it works with the right conditional probability versions of these fractional variables. Due to the random evictions, the cache may still have more than k pages, which is handled via a *fixing* step, that evicts some pages carefully. As the eviction rule is $O(\log k)$ times more aggressive, the cost for the fixing step can be amortized with the overall fractional cost. However, making these ideas precise requires some care, and we use a careful coupling argument together with induction over time.

An obvious benefit of our rounding algorithm is that it is easy to implement and is very efficient. Moreover, combined with a simple multiplicative update rule for computing the fractional solution, this gives a very simple overall algorithm.

Implications for weighted paging. The $O(\log^2 k)$-competitive algorithms we develop for writeback-aware caching and RW-paging are also applicable to weighted paging. Even though this factor is inferior to the known $O(\log k)$- competitive algorithm of Bansal et al. [5], the $O(\log^2 k)$-competitive algorithm is an extremely simple and clean algorithm which is very easy to implement.

The hardness results. To show Theorem 1.3, we observe that the set cover problem can be reduced to RW-paging, and that this reduction works both in the offline and online settings. Usually, in online algorithms, one does not consider polynomial-time computability issues, but in an elegant and surprising result [24], Feige and Korman showed that unless NP $\subset$ BPP, there is no $o(\log m \log n)$-competitive algorithm that runs in polynomial-time, for the online set cover problem on n elements and m sets. Fortunately, the online instances in the result of [24] have a nice structure, and work directly with our reduction, allowing us to prove Theorem 1.3.

Finally, Theorem 1.4 follows by considering an integrality gap instance for the set cover problem and the reduction above.

1.3 Organization

The paper is organized as follows. In Section 2 we formally define the problems we consider in the paper. In Section 3 we prove our lower bounds on the competitive factor for the RW-paging problem by reducing from the online set cover problem.

In Section 4 we show the algorithmic results for the weighted multi-level paging problem. Specifically, in Section 4.1 we develop a deterministic integral $O(k)$-competitive algorithm and in Section 4.2 we develop a deterministic fractional $O(\log k)$-competitive algorithm. In the full version of the paper we give an alternate deterministic integral $O(k)$-competitive algorithm via the online primal-dual method for the problem.

Finally, in Section 4.3 we describe the distribution-free online rounding. For ease of exposition and since it is of independent interest, we first describe the rounding for weighted paging in Section 4.3.1, yielding a randomized integral $O(\log^2 k)$-competitive algorithm for the problem. Later, in Section 4.3.3, we generalize it to multi-level paging.

2 PRELIMINARIES

We define the problems we study, and set up the notation.

Weighted Paging. In the online weighted paging problem there is a cache of size k and a universe of $n > k$ pages. Each page p is associated with a positive eviction cost $w(p) \geq 1$[1]. At each time step t there is a request for one of the pages, denoted by p_t. If p_t is already in the cache then no cost is incurred, otherwise the algorithm must bring p_t into the cache, possibly evicting some other page. The goal is to serve a given request sequence, while minimizing the total cost of page evictions.

Writeback-Aware Caching. The basic setting here is the same as in weighted paging. However, each request is either a *read* request or a *write* request. We say that a page p in the cache is *dirty* at time t if, since the last time p was loaded into the cache, there was a write request to p. Otherwise, we say that p is *clean*. Evicting a dirty page costs more than evicting a clean page. We thus associate with each page p two possible weights, $w_1(p) \geq w_2(p) \geq 1$, where $w_1(p)$ is the eviction cost of p when it is dirty, and $w_2(p)$ is the eviction cost when it is clean. The goal is to serve a given request sequence, with minimum total cost.

[1] Total fetch cost and eviction cost are equal up to an additive constant.

RW-Paging. The basic setting here is the same as in weighted paging, except that for each page p we now have a pair of pages $(p,1),(p,2)$, where $(p,1)$ is the *write copy* of p, and $(p,2)$ is the *read copy* of p. The eviction costs satisfy $w(p,1) \geq w(p,2) \geq 1$. A request for page $(p,1)$ is served by fetching it into the cache, while a request for $(p,2)$ can be served with either fetching $(p,1)$ or $(p,2)$ into the cache. A crucial restriction is that at most one page of each pair, $(p,1)$ and $(p,2)$ can be in the cache at the same time. The goal is to serve a given request sequence with minimum cost.

Equivalence of RW-Paging and Writeback-Aware Caching. We now show that writeback-aware caching and RW-paging are algorithmically equivalent. Consider an instance of the writeback-aware caching with cache size k and n pages with eviction costs $w_1(p) \geq w_2(p) \geq 1$. Define the following RW-Paging instance in which there is a cache of size k and n pairs of pages $(p,1),(p,2)$ with eviction costs $w(p,i) = w_i(p)$. Given a request sequence for the writeback-aware caching problem, define a request sequence for the RW-paging problem in which every write request to a page p is replaced by a request for $(p,1)$, and every read request to p is replaced by a request to $(p,2)$. This reduction goes in both directions: from writeback-aware caching to RW-paging and vice versa.

Lemma 2.1. *Consider the two instances of writeback-aware caching and RW-paging obtained by the above reduction. Then the integral optima of both these instances are equal. In particular, there is an α-competitive algorithm for RW-paging iff there is an α-competitive algorithm for writeback-aware caching.*

Proof. Consider an integral solution S to the RW-paging instance. This solution defines a solution S' to the writeback-aware caching instance in a natural way. At any point of time, S and S' maintain the same cache with respect to every page p, i.e. if S has page (p,i) in the cache, then S' has p in its cache. Solutions S and S' have the same cost except for the case when $(p,2)$ is replaced by $(p,1)$ in S. In this case S' has no cost as p just becomes dirty in S'. Thus, the cost of S' is not higher than the cost of S.

Conversely, consider an integral solution S' to the writeback-aware caching instance, and define a solution S for the RW-paging instance as follows. For every page p, consider a time interval $[t_1,t_2]$ in which p is fetched to the cache at time t_1 and remains in the cache until it evicted at time t_2. If p is dirty at time t_2, then we load $(p,1)$ to the cache at time t_1 and evict it at time t_2, otherwise we load $(p,2)$ to the cache at time t_1 and evict it at time t_2. The cost of solution S is not higher than the cost of S'. Thus, the integral optima of the two instances are of equal value. □

Weighted Multi-Level Paging. This problem is a generalization of RW-paging. The basic setting is the same as in weighted paging. In this problem, requests are given for both a page and a level: for each page p there are ℓ copies $(p,1),(p,2),\ldots,(p,\ell)$, representing the levels, where level 1 is the highest, and ℓ the lowest. A request (p,i) for page p and level i can be served by any copy of p of level higher than i, i.e. any (p,j) such that $j \leq i$. Eviction costs are associated with the copies of page p, satisfying: $w(p,1) \geq w(p,2) \geq \cdots \geq w(p,\ell) \geq 1$. For every page p, the cache is restricted to contain at most one of its copies $(p,1),(p,2),\ldots,(p,\ell)$ at any time.

Fractional Versions. Our randomized algorithms will be based on fractional versions and LP formulations of these problems. We present a linear programming (LP) formulation for the ℓ-level paging along the same lines as weighted paging (see [7] for details).

Let $y(p,i,t)$ denote the fraction of (p,i) present in the cache at time t. For each p,i, we define the *prefix variables* $u(p,i,t) = 1 - \sum_{j=1}^{i} y(p,j,t)$, so that $u(p,i,t) = 0$ if the sum of fractions of (p,j) for $j \leq i$ is 1. In other words, if (p_t,i_t) is requested at time t, then $u(p_t,i_t,t)$ fraction of page p must be fetched in total (among copies $1,\ldots,i_t$) to the cache.

Note that these variables must satisfy that $u(p,i,t) \geq u(p,i+1,t)$ for every $p,t,i = 1,\ldots\ell-1$. For page p, $1-u(p,\ell,t)$ is the total space in the cache occupied by copies of page p.

To make sure that our cache is feasible we need that the cumulative fraction of pages in the cache at any time is at most k, or equivalently, $\sum_p u(p,\ell,t) \geq n-k$.

As in weighted paging, we use the variables $z(p,i,t)$ to denote the movement cost for $u(p,i)$ at time t. As the weights are geometrically decreasing, the objective below is within a factor of 2 of the objective that measures that change in $y(p,i,t)$.

This yields the following linear program.

$$\min \quad \sum_t \sum_p \sum_{i\in\ell} z(p,i,t)w(p,i) + \sum_t \infty \cdot u(p_t,i_t,t)$$

$$\text{s.t.} \sum_{p:p\in S} u(p,\ell,t) \geq |S| - k \qquad \forall S \subset [n]$$

$$u(p,i-1,t) - u(p,i,t) \geq 0 \qquad i = \ell,\ldots,2$$

$$z(p,i,t) - u(p,i,t) + u(p,i,t-1) \geq 0 \qquad \forall p,t,i = 1,\ldots,\ell$$

$$z(p,i,t), u(p,i,t) \geq 0 \qquad \forall p,t,i = 1,\ldots,\ell$$

As in weighted paging, the first set of constraints ensures that the cache has at most k pages. The term $u(p_t,i_t,t)\cdot\infty$ in the objective function ensures that any finite cost solution must set $u(p_t,i_t,t) = 0$, and hence satisfies (p_t,i_t) at time t. Note that while the box constraints $u(p,\ell,t) \leq 1$ are implied by the first set of constraints (see e.g. [7]), it is not immediately clear why $u(p,i,t) \leq 1$ for $i < \ell$. The following claims shows why this holds, and hence that this LP is a valid formulation.

Claim 2.2. *In any feasible solution to the LP above, we can assume that $u(p,i,t) \leq 1$ for every p,i,t.*

Proof. Given any feasible solution (u,z), consider the solution $(\tilde{u},z)$, where $\tilde{u}(p,i,t) = \min\{u(p_t,i,t),1\}$. We claim that $(\tilde{u},z)$ is feasible. Indeed, the first set of constraints only involves $u(p,\ell,t)$ variables which are already ≤ 1, the second set of constraints are still satisfied, and finally the third set of constraints are satisfied as $x-y \geq \min(x,1)-\min(y,1)$ for any $x \geq y$. Moreover, replacing u by $(\tilde{u})$ and choosing the optimum $\tilde{z}$ can only reduce the objective. □

Finally, we note that for $\ell = 1$, our LP coincides with the one for weighted paging.

3 LOWER BOUNDS

We now prove Theorem 1.3. As mentioned previously, we give a reduction from the online set cover problem, defined below, and use the hardness result of Feige and Korman [24].

Definition 3.1 (Online Set Cover)**.** *There is a universe of elements $U = \{1, \ldots, n\}$ and a family $F = \{S_1, \ldots, S_m\}$ of subsets of U. Both U and F are known upfront to the online algorithm. At each step, an element $e \in U$ appears, and the algorithm must choose a set $S \in F$ containing it, unless e is already covered by some previously chosen set. The goal is to minimize the total number of sets chosen.*

Consider the following reduction from Online Set Cover to Online RW paging.

Reduction. Let (U, F) be the set system, with $n = |U|$ and $m = |F|$, and let $e_1, \ldots, e_t$ be the elements requested in an online set cover instance. We define an RW-paging instance as follows.

The cache has size $k = m$, the number of sets in F. For each set $S \in F$, there are two pages $(p_S, 1)$ and $(p_S, 2)$, corresponding to the write and read copies for S. Similarly, for each element $e \in U$, there are two pages $(e, 1)$ and $(e, 2)$, corresponding to the write and read copies for e.

Let ℓ be a parameter ($\gg mn$) that we specify later. Consider the following request sequence.

(1) **Initialization:** For each set S in F, there is a write request for S.
(2) **Sequence for element e:** For each element e, let $\overline{F}_e := \{S : S \in F, e \notin S\}$ be the collection of sets that do not contain e. Let $\rho(e)$ denote the request sequence consisting of a read request for e, followed by a read request for each set in $\overline{F}_e$. When element e is requested in the online set cover instance, do the following:
   (a) Give the request sequence $\rho(e)$ for ℓ times.
   (b) Give a read request for each $S \in F$.
(3) **Terminate:** For each set S in F, there is a write request for S.

Let $T = \{e_1, \ldots, e_t\}$ denote the set of elements requested in the online set cover instance. We now show that this request sequence has low cost if and only if there is a small set cover for T. For convenience, we only consider eviction cost. Let the cost of evicting a write page and a read page be w and 1, respectively.

Lemma 3.2. *(Completeness) If C is a set cover for T of size $|C| = c$, the RW-instance has a solution with cost at most $c(w+1) + 2t$, where $t = |T|$. Moreover, at the start and at the end of this solution, the cache consists of the write pages $(p_S, 1)$ for each set $S \in F$.*

Proof. Consider the following solution. Initially, the cache contains all the pages $(p_S, 1)$ for each $S \in F$. Then in Step 1, all requests are served for free. After Step 1, evict pages $(p_S, 1)$ for $S \in C$, and replace them by $(p_S, 2)$ for each $S \in C$. Evicting these c write pages, incurs a cost of cw.

For each element e in T, and before Step 2*a* begins, evict some page $(p_S, 2)$ where S is some set containing e. Such a set S always exists as C is a valid set cover for the elements in T. Evicting the single page $(p_S, 2)$ incurs cost 1. In Step 2a, as the sequence $\rho(e)$ consists only of read requests to e and to sets S' that do not contain e. As one of $(p'_S, 1)$ or $(p'_S, 2)$ is present for each such S', these requests can be served for free, and hence all the ℓ copies of $\rho(e)$ can be served at no cost.

After Step 2a, evict the page $(p_e, 2)$ and load the page $(p_S, 2)$ back. This incurs a cost of 1. Now, the cache contains either $(p_S, 1)$ or $(p_S, 2)$ for each set $S \in F$, and hence the Step 2b incurs 0 cost.

As Step 2 occurs t times, once for element $e \in T$, the overall cost in this step is at most $2t$.

Finally, before Step 3, evict pages $(p_S, 2)$ for $S \in C$ and replace them by $(p_S, 1)$. This incurs cost c. The requests in Step 3 can now be served for free, and moreover the cache consists of the same pages as when it started. □

Let us set $\ell := mnw$ to be a large parameter. The following shows that in any solution with reasonable cost, the sets S corresponding to the evicted pages $(p_S, 1)$ must form a valid set cover of T.

Lemma 3.3. *(Soundness) Let D be the write pages $(p_S, 1)$ evicted by the algorithm. If D is not a valid set cover for T, then the cost for serving the instance is at least ℓ.*

Proof. In Step 1 and Step 3, for each set $S \in F$, there is a write request for S, and hence the page $(p_S, 1)$ must be present in the cache at both these times.

Let D denote the sets S for which the page $(p_S, 1)$ was evicted sometime between the two write requests for S. If D is not a valid set cover for T, consider some element $e \in T$ that is not covered by sets in D, and hence in particular, $D \subset \overline{F}_e$.

Consider Step 2a for element e. As $\rho(e)$ consists of read requests to e and all the pages in $\overline{F}_e$ (and hence in D), when the read requests for $e \cup D$ arrive, there is only $|D|$ space in the cache available for these $|D| + 1$ pages, and hence there must be at least one eviction, leading to an overall cost of at least ℓ. □

Lower bound instance. Feige and Korman [24] gave the following reduction from offline set cover to online set cover.

Theorem 3.4 (Corollary 2.3.2 [24], slightly restated). *Let A be any randomized algorithm for online set cover. There exists a polynomial-time reduction from (offline) Set Cover to Online Set Cover such that given an offline Set Cover instance $I = (X, F)$, consisting of $|X| = N$ elements and $|F| = M$ sets and whose optimal cover size is c, produces an online set cover instance I' on $n = N(N-1)$ elements, $m = NM/2$ sets with q different online request sequences $\rho_1, \ldots, \rho_q$ such that:*

(1) The optimal offline cover size of each sequence ρ_j for $j \in [q]$ is c. The length of each ρ_j is $N \log N$.
(2) The expected number of sets used by A is at least $(c \log N)/2$, where the expectation is over the randomness is A, and the uniform distribution over the sequences $\rho_1, \ldots, \rho_q$.

Combined with the $\Omega(\ln N)$ approximation hardness for (offline) set-cover [23] (details in Section 2.3.2 of [24]) yields the following:

Theorem 3.5 ([24], Theorem 2.3.4). *If there is an $o(\log n \log m)$-competitive randomized polynomial-time algorithm for online set cover, then NP $\subset$ BPP.*

Moreover, in these instances n and m are polynomially related. Together with the reduction from online set cover to RW-paging we get:

Theorem 3.6. *Unless NP $\subset$ BPP, there is no $o(\log^2 k)$-competitive polynomial-time randomized algorithm for RW-paging.*

Proof. Consider the online set cover instance in Theorem 3.4. We apply the reduction from the set system I' to RW-paging. The cache size is $k = m$. The overall request sequence for RW-paging consists of $h = k$ phases, where at each phase, we randomly choose an online set cover request sequence ρ_i, and give the paging requests specified by Steps 1-3 corresponding to ρ_i.

We claim that the offline cost is at most $2hcw$. Suppose that the initial cache consists of pages $p_{S,w}$ for each of the m sets S. Otherwise, we pay an initial cost of at most kw to get to this state, which is at most hcw. Let C_i denote some optimum set cover for ρ_i, consider the solution given by Lemma 3.2. As and the solution starts and ends at the same cache state in each phase, the total cost incurred over the h phases is at most $h((w+1)c + 2n)$, as $\rho_i \ll n$ for each i. Choosing $w = n$, this is at most $2hwc$.

Now consider the online solution. First, by Lemma 3.3, if during any phase i the evicted write pages do not form a valid set cover for ρ_i, then the cost is at least $\ell = m^2nw = k^2nw \geq k(kwc)$ as trivially $c \leq |\rho_i| \ll n$. So, we can assume that in each phase i, a valid set cover for ρ_i is computed online. But, then, by Theorem 3.5, this cover has expected size $\Omega(c \log^2 n)$, resulting in expected eviction cost $\Omega(\log^2 n)cw$ for the pages $p_{S,W}$ per phase. This gives the claimed lower bound of $\Omega(\log^2 k)$ on the competitive ratio. □

Integrality Gap. We now prove Theorem 1.4.

Given a set system (U, F), and a set of requested elements T, a valid fractional set cover solution satisfies $x_S \geq 0$ for all $S \in F$ and $\sum_{S:e\in S} x_S \geq 1$ for all $e \in T$.

In the fractional version of RW-paging, to serve $(p, 1)$, one unit of $(p, 1)$ must be in the cache, and to serve $(p, 2)$, the sum of the fractions $(p, 1)$ and $(p, 2)$ must be 1. We claim that a simple modification of Lemma 3.2 gives a fractional solution to RW-paging of cost at most $|x|_1 w + 2t$, i.e. the size $|C|$ of the set cover is replaced by the size of the fractional set cover $|x|_1$.

Let x be a fractional set cover for T, consider the solution in Lemma 3.2, where instead of evicting the copies $(p_S, 1)$ for $S \in C$ integrally and replacing them by $(p_S, 2)$, we evict each $(p_S, 1)$ fractionally by x_S, and load $(p_S, 2)$ to extent x_S. When request $\rho(e)$ arrives, we evict fractions adding up to 1 from pages $(p_S, 2)$, where $e \in S$, and fetch $(p_e, 2)$ to extent 1. The above is possible as x is a valid fractional set cover for elements in T. The bounds on the cost now follow directly by the arguments in Lemma 3.2.

Now, consider any set of elements T for which the integral set cover is $\Omega(\log n)$ times larger than the fractional set cover, i.e. $|C| = \Omega(\log n)|x|_1$. Consider any randomized algorithm that takes fractional solution x, and maintains a distribution over integral cache states. By Lemma 3.3, at least half of these cache states must evict write copies $(p_S, 1)$ corresponding to sets S in a valid integral cover for T, otherwise the expected cost is already too high. But then, these cache states must evict $\Omega(\log n \cdot \|x\|_1)$ write copies, incurring cost $\Omega(\log n)$ times the fractional cost.

4 MULTI-LEVEL PAGING

We now consider the weighted multi-level paging problem. We first give an $O(k)$-deterministic algorithm in Section 4.1. In Section 4.2 we give a deterministic fractional $O(\log k)$-competitive algorithm, and in Section 4.3 we give the online rounding procedure. Together, this gives the randomized $O(\log^2 k)$-competitive algorithm.

Recall that in the multi-level paging problem, requests correspond to both a page and a level: for each page p there are ℓ copies $(p, 1), (p, 2), \ldots, (p, \ell)$, representing the levels. A request (p, i) can be served by any copy (p, j) such that $j \leq i$. We assume with loss of generality that the eviction costs satisfy $w(p, 1) \geq \ldots \geq w(p, \ell) \geq 1$, and moreover that $w(p, i)/w(p, i+1) \geq 2$, for all $i = 1, \ldots, \ell - 1$. This loses a factor of at most 2, otherwise we can simply merge two levels for p. For every page p, the cache can contain at most one of its copies $(p, 1), (p, 2), \ldots, (p, \ell)$ at any time.

4.1 Deterministic $O(k)$-Competitive Algorithm

We now give a deterministic online $O(k)$-competitive algorithm. We first give a water-filling based algorithm with a simple potential function proof. In the full version of the paper, we give a primal-dual proof of the same result.

Algorithm. For each copy (p, i), we maintain a water-level $f(p, i, t) \in [0, w(p, i)]$. Let us assume that initially at $t = 0$, the cache is empty, and the water-level $f(p, i, 0) = w(p, i)$ for all $p \in [n], i \in [\ell]$.

At time t, upon arrival of request (p_t, i_t) do the following:

(1) If the request is already satisfied, i.e., there is some (p_t, j) for $j \leq i_t$ in the cache, do nothing.
(2) Otherwise, fetch (p_t, i_t) and set $f(p_t, i_t) = 0$.
   (a) If there is another copy (p_t, j) for $j > i_t$ of page p_t, evict this copy.
   (b) Otherwise, let $S = \{q : q \in \text{ON} \setminus \{p_t\}\}$, and let (q, i_q) denote the copy of $q \in S$. If $|S| = k$, raise $f(q, i_q)$ at rate 1 for each $q \in S$, until $f(q, i_q) = w(q, i_q)$ for some q, evict this (q, i_q).

Analysis. We show the following result, which implies a $4k$ competitive ratio for arbitrary weights.

Theorem 4.1. *Assuming that $w(q, i) \geq 2w(q, i+1)$ for all q, i, the algorithm above is $2k$ competitive.*

Proof. Let OFF denote some optimal offline solution, and ON denote the online solution. Without loss of generality, we assume that OFF is integral. Let $v(p, i, t)$ denote variables $u(p, i)$ corresponding to OFF, i.e. for any page p, if OFF has (p, i) in the cache at time t, then $v(p, 1, t) = \cdots = v(p, i-1) = 1$ and $v(p, i) = \cdots = v(p, \ell) = 0$. If no copy of p is in OFF, then $v(p, i, t) = 1$ for all $i \in [\ell]$. In particular, it means that $v(p, \ell) = 0$ for at most k different pages p.

We give a potential function analysis. Define

$$\Phi = \sum_{p\in\text{ON}} k \cdot v(p, i_p, t)(w(p, i_p) - f(p, i_p, t)) + \sum_{p\in\text{ON}} f(p, i_p, t).$$

We use the convention that for online evicting (p, i) incurs cost $w(p, i)$, but fetching it gives $w(p, i)/2$ profit (i.e. incurs cost $-w(p, i)/2$). For offline, we incur cost $w(p, i)$ to evict (p, i) and fetching has no cost. Now, fetching and evicting some (p, i) incurs a total online cost of $w(p, i)/2$, so this affects online cost by factor at most 2. Under this convention, we show that for all possible events at time t,

$$\Delta(\text{ON}) + \Delta\Phi \leq k \cdot \Delta(\text{OFF}), \tag{1}$$

where $\Delta\Phi$ is change in potential at time t, and $\Delta(\text{ON})$ and $\Delta(\text{OFF})$ denote the online and offline cost at t. Clearly, this gives the claimed $2k$ competitive ratio. We defer the details to the full version. □

4.2 Fractional $O(\log k)$-Competitive Algorithm

We now give a deterministic online fractional $O(\log k)$-competitive algorithm and analyze it using a potential function. We use the same notation as in Section 4.1.

The fractional Algorithm. Upon the arrival of request (p_t, i_t) do the following. We drop t from the variables u and y for notational convenience.

(1) Set $u(p_t, j) = 0$ for $j \geq i_t$, and keep $u(p_t, j)$ unchanged for $j < i_t$.
This is equivalent to setting, $y(p_t, i_t) = 1 - \sum_{j<i} y(p_t, j)$ and $y(p_t, j) = 0$ for $j > i_t$, i.e., we evict pages of level $j > i_t$, and fetch enough (p_t, i_t) so that there is one unit of page p_t in levels $1, \ldots, i_t$.
(2) For each $q \neq p_t$, let i_q be the largest index with $y(q, i_q) > 0$. For each such $q \neq p_t$ for which such an index i_q exists do the following: Decrease, $y(q, i_q, t)$ at rate

$$\frac{1}{w(q, i_q)}(u(q, i_q) + \eta) \tag{2}$$

until $\sum_{q\in[n]} u(q, \ell) \geq n - k$, where we set $\eta = 1/k$.

Analysis. We will do a potential function analysis, and show that at each step

$$\Delta(\text{On}) + \Delta(\Phi) \leq c\Delta(\text{Off}) \tag{3}$$

for $c = 4(\log k)$. As previously, we assume that the weights satisfy $w(q, j)/w(q, j + 1) \geq 2$.

Consider the potential

$$\Phi = 2 \sum_{q\in[n]} \sum_{j\in[\ell]} w(q, j)v(q, j) \ln \frac{1 + \eta}{u(q, j) + \eta}.$$

We first consider the offline move. Without loss of generality, we assume that OFF is integral.

Lemma 4.2. *If OFF evicts some (q, j), then $\Delta\Phi \leq 4w(q, j) \ln(1 + 1/\eta)$. If OFF fetches any page, the potential can only decrease.*

Proof. When OFF evicts (q, j), the quantities $v(q, j), v(q, j + 1), \ldots, v(q, \ell)$ go up from 0 to 1. So

$$\Delta(\Phi) \leq 2 \sum_{h=j}^{\ell} w(q, h) \ln(1 + 1/\eta) \leq 4 \cdot w(q, j) \ln(1 + 1/\eta)$$

If OFF fetches some (q, j), the quantities $v(q, j), v(q, j+1), \ldots, v(j, \ell)$ go down from 1 to 0. So the potential can only decrease due to this change (as $\ln((1 + \eta)/(u(q, j) + \eta))$ is always non-negative). □

We now analyze the online move. We can assume that offline has already served the request (p_t, i_t). We first consider step 1 of the algorithm.

Lemma 4.3. *$\Phi = 0$ in step 1, and as the online fetching cost is 0, this implies that (3) holds.*

Proof. As OFF already has (p_t, h) in its cache for some $h \leq i_t$, we have $v(p_t, i_t) = \ldots = v(p_t, \ell) = 0$. The only update that online does in step 1 is to set $u(p_t, j) = 0$ for $j \geq i_t$. So, $\Delta\Phi = 0$. □

Now consider the rule when pages are evicted. We do a continuous analysis.

Lemma 4.4. *$\Delta(ON) + \Delta(\Phi) \leq 0$, as pages other than p_t are evicted.*

Proof. Let S be the set of pages q such that $q \neq p_t$ and $u(p, \ell) < 1$. Recall that the pages in S are precisely those for which we evict some fraction of (q, i_q). Also, by the definition of i_q, we have that $u(q, i_q) = u(q, \ell)$. Moreover, when $y(q, i_q)$ is decreased by ϵ, then by definition of $u(q, j)$, all $u(q, j)$ for $j \geq i_q$ rise by exactly ϵ.

Now, the movement cost is

$$\begin{aligned}\sum_{q\in S} w(q, i_q)|dy(q, i_q)| &= \sum_{q\in S} w(q, i_q) \frac{(u(q, i_q) + \eta)dy}{w(q, i_q)} \\ &= \sum_{q\in S} (u(q, i_q) + \eta)dy \\ &\leq (|S| - (k - 1) + \eta|S|)dy \\ &\leq 2(|S| - (k - 1))dy.\end{aligned}$$

The first step uses that the update rule is (2), and the last step uses that $\eta|S| = |S|/k \leq |S| - (k - 1)$ for $|S| \geq k$, which is the case as otherwise the cache was feasible. The first inequality follows as $u(q, i_q) = u(q, \ell)$, and since $\sum u(q, \ell) < n - k$,

$$\begin{aligned}\sum_{q\in S} u(q, \ell) &= \sum_{q\in[n]} u(q, \ell) - \sum_{q:u(q,\ell)=1} u(q, \ell) \\ &< n - k - (n - 1 - |S|) = |S| - (k - 1)\end{aligned}$$

To show that (3) holds, it suffices to show that the potential decreases by at least $2(|S| - (k - 1))dy$.

For every page $q \neq p_t$, as $x_{q,i}$ can only increase, $u(q, i)$ can only increase and hence the potential can only decrease. So let us simply consider the contribution to the decrease from the $|S| - (k - 1)$ pages q for which $v(q, \ell) = 1$ (recall that $v(q, \ell)$ can be 0 for at most k pages, and p_t is one of them), but $u(q, \ell) < 1$. Call this set of pages T. For each such page in T, the potential decreases by

$$\sum_{j\geq i_q} w(q, j)v(q, j) \frac{du(q, j)}{u(q, j) + \eta} = \sum_{j\geq i_q} w(q, j) \frac{1}{w(q, i_q)} dy \geq dy,$$

as for $j \geq i_q$, we have $u(q, j) = u(q, i_q)$ and hence $du(q, j) = du(q, i_q) = (u(q, i_q) + \eta)dy/w(q, i_q)$, and $v(q, j) = 1$ for $q \in T$. □

4.3 Online Rounding

We now describe online rounding for multi-level paging. Our rounding algorithm has a simple form and is *local* in the following sense. Let $x(t - 1)$ denote the fractional solution at the end of time t. The algorithm maintains a feasible cache state $C(t-1)$ based on $x(t-1)$ and its own internal randomness until time $t - 1$. When a request arrives at time t and the fractional solution changes to $x(t)$, the new cache state $C(t)$ only depends on the state $C(t - 1)$, fractional solutions $x(t - 1)$ and $x(t)$, and the random choices at time t. We emphasize that the rounding is independent of the way the fractional solution is generated. We say that the rounding algorithm is β-competitive, if its expected cost is at most β times the fractional cost of the solution x. For our rounding, $\beta = O(\log k)$.

While the rounding algorithm itself is simple to describe and implement, the analysis is a bit subtle and uses a coupling argument. To illustrate the main idea we first describe the rounding for $\ell = 1$, which corresponds to weighted paging. In fact, this is interesting by itself, as previous rounding algorithms for weighted paging [2, 5], while $O(1)$-competitive, are much more complicated, and maintain

a distribution over several integral cache states, thus making them very unnatural and hard to implement.

4.3.1 Rounding for weighted paging. We assume that for each page p, $w_p \geq 1$, and for $i = 1, 2, \ldots$, let $P_i = \{p : w_p \in (2^{i-1}, 2^i]\}$. We refer to P_i as weight *class* i. Let $P_{\geq i} = \{p : w_p \geq 2^{i-1})\}$ denote the set of pages with weight at least 2^{i-1}. For each class i, we denote the fractional space used by pages in $P_{\geq i}$ as

$$k_{\geq i}(t) = \sum_{p \in P_{\geq i}} (1 - x_p(t)).$$

Without loss of generality, we assume that at time t, for any page $p \neq p_t$, $x_p(t) - x_p(t-1) \geq 0$, and the total fraction of pages evicted upon any request is at most 1. Let $\beta = 4 \log k$, and let

$$y_p(t) = \min(\beta x_p(t), 1).$$

At time t, let us define $\Delta(y_p(t)) = y_p(t) - y_p(t-1)$.

We also need the following simple claim, the proof of which appears in the full version of the paper.

Lemma 4.5. *Any feasible fractional x can be assumed to satisfy, while losing at most a factor of two in the objective, that for each p and t, the value $x_p(t)$ is an integer multiple of $\delta = 1/(4k)$.*

The rounding algorithm. Consider Algorithm 1. Intuitively, it is

Algorithm 1: Randomized Rounding Step

```
At time t, when the request for page p_t arrives:
    if p_t ∉ C(t−1) then
        add p_t to C(t−1)

    For each page p ≠ p_t:
        if p ∈ C(t−1) then
            evict p independently with probability
              Δy_p(t)/(1 − y_p(t−1))

    end
    For i in decreasing order
        if |P_{≥i} ∩ C(t−1)| > ⌈k_{≥i}(t)⌉ then
            Type-i Reset: Evict an arbitrary page p ≠ p_t
              from P_i ∩ C(t−1)
    end
    Set C(t) = C(t−1)
end
```

trying to mimic the fractional algorithm, by evicting pages $O(\log k)$ times more aggressively, and as it can only evict pages that are currently in the cache, it uses the change in *conditional* probabilities $\Delta y_p(t)/(1 - y_p(t-1))$. Since the choices are random, there is a possibility that the cache still has too many pages. To fix this, it considers prefixes of weight class from heaviest to lightest, and ensures that the cumulative page count in the cache does not exceed that in the fractional solution.

We show that the algorithm is feasible in the sense that whenever the condition

$$|P_{\geq i} \cap C(t-1)| > \lceil k_{\geq i}(t) \rceil \tag{4}$$

occurs, there is always a page $p \neq p_t$ in class P_i that can be evicted. We call (4) the type-i reset condition.

4.3.2 Analysis of rounding for weighted paging. We first note that $p_t \in C(t)$ as the algorithm adds p_t if $p_t \notin C(t-1)$, and does not evict it in any step at time t. Moreover, the following lemma (that we prove later) will imply that $|C(t)| \leq k$ at all times.

Lemma 4.6. *For any class i, the number of pages in $P_{\geq i} \cap C(t)$ is at most $\lceil k_{\geq i}(t) \rceil$.*

Setting $i = 1$ and noting that $P_{\geq 1} = [n]$, and using that $\sum_p x_p(t) \geq n - k$ as $x(t)$ is feasible, gives that,

$$k_1(t) = \sum_p (1 - x_p(t)) = n - \sum_p x_p(t) \leq n - (n - k) = k.$$

The following lemma bounds the cost.

Lemma 4.7. *The expected cost of the solution produced is $O(\log k)$ times that of the fractional solution x.*

Note that as the adversary is oblivious, the request sequence ρ can be assumed to be fixed in advance, and as the fractional algorithm is deterministic, the solution $x(t)$ (and hence $y(t)$) can also be assumed to be fixed in advance. In particular it does not depend on the random choices made in the rounding algorithm.

The idea behind the analysis is the following. There are two types of costs: eviction costs due to the change $\Delta y(t)$ in the fractional solution, and evictions due to the reset steps. The first type of cost clearly is at most $O(\log k)$ times the fractional cost. For the second type of cost, as the algorithm evicts $O(\log k)$ times more aggressively in comparison to x, if the fractional solution has 1 unit of space for the incoming page p_t, we expect our algorithm to have $O(\log k)$ space in expectation (for simplicity we are ignoring weights in this heuristic argument, handling which requires some care). So, the probability that we need to evict some page to make room for p_t should be $\exp(-\Omega(\log k)) = 1/\text{poly}(k)$. Using Lemma 4.5, we can amortize this and charge it to the total optimum cost.

As the distribution on cache state $C(t)$ defined by the algorithm is quite non-explicit (due to the reset steps), to make these ideas above precise, we will work with a suitable coupling between the distribution on $C(t)$ and the product distribution with marginals $y_p(t)$. We now give the details.

Product distribution and Coupling. Fix a time t, and the solution $y(t)$. Consider the product distribution $D(t)$ on subsets U of pages with marginals $(1 - y_p(t))$, i.e. for $U \subseteq [n]$,

$$\Pr_{D(t)}[U] = \prod_{p \in U} (1 - y_p(t)) \prod_{p \notin U} y_p(t).$$

Note that the the support of $D(t)$ consists of all possible subsets U of $[n]$, including those of size more than k. Moreover, if $y_p(t) = 1$, then p is not contained in any set in the support of $D(t)$. Let $E(t)$ denote the (implicit) distribution produced by our algorithm on the cache states $C(t)$. Note that the support of $E(t)$ consists of subsets of $[n]$ of size at most k.

Given two probability distributions π_1 and π_2 on discrete sets S_1 and S_2, recall that a *coupling* between π_1 and π_2 is a probability distribution γ over $S_1 \times S_2$ such that for all $s_1 \in S_1$ and $s_2 \in S_2$, it holds that

$$\sum_{s_2 \in S_2} \gamma(s_1, s_2) = \pi_1(s_1) \qquad \text{and} \qquad \sum_{s_1 \in S_1} \gamma(s_1, s_2) = \pi_2(s_2).$$

Given a coupling γ and $s_1 \in S_1, s_2 \in S_2$, we say that s_1 is coupled to s_2 whenever $\gamma(s_1, s_2) > 0$.

Local rule for product distributions. Let $y(t)$ be some fractional solution at time t, and $D(t)$ be the product distribution corresponding to $y(t)$ as defined above. When $y(t)$ changes to $y(t+1)$, consider the following *local rule* for updating a set U. For each page $p \in [n]$, if $y_p(t+1) > y_p(t)$, then if $p \in U$, delete p with probability $(y_p(t+1) - y_p(t))/(1 - y_p(t))$. Otherwise, if $y_p(t+1) < y_p(t)$, and $p \notin U$, then add p to U with probability $(y_p(t) - y_p(t+1))/y_p(t)$. The following lemma is direct.

Lemma 4.8. *Let U be a random set distributed according to $D(t)$. The random set U' produced by applying the local rule to U is distributed according to $D(t+1)$, the product distribution corresponding to $y(t+1)$.*

The coupling. Note that Steps 4-8 in the Algorithm exactly correspond to the local rule above (but applied to $C(t)$), as for $p \neq p_t$, we have $y_p(t+1) \geq y_p(t)$ and for $p = p_t$, $\Delta y_p(t) = -y_p(t-1)$, and so the local rule will load p_t to C whenever p_t is not in C. We now exhibit a suitable coupling between $E(t)$ and $D(t)$ by giving an inductive construction over time, which satisfies a crucial subset condition stated below.

Lemma 4.9. *At any time t, and given any $y(t)$, there is a coupling $\gamma(t)$ between the distribution on caches $E(t)$ and the product distribution $D(t)$, such that if $C \in E(t)$ is coupled with a set $U \in D(t)$, then $C \subseteq U$.*

Proof. We construct the coupling inductively over time. In the base case, at $t = 0$, we can assume that $y_p(0) = 1$, so that both $E(0)$ and $D(0)$ are fully supported on the empty set and the coupling trivially exists.

Let $\gamma(t-1)$ be a coupling at time $t-1$ satisfying the claimed property. Consider a set C distributed according to $E(t-1)$, and let U be some set to which C is coupled according to $\gamma(t-1)$. Let C' and U' denote the random sets obtained by applying the local rule to C and U respectively, as $y(t-1)$ changes to $y(t)$, but using the same random choices for a page while applying the local rule to C and U'. We claim that $C' \subset U'$, which would give the desired coupling $\gamma(t)$. To see this, consider a page p and suppose $y_p(t) \geq y_p(t-1)$. As $C \subset U$, by the coupling of the random choices, if p is removed from U, then either it is also removed from C, unless it was already absent from C. Either way, if p is not present in U', then it is also not present in C'. Similarly, suppose $y_p(t) \leq y_p(t-1)$. If p is added to C, by the coupling of the random choices, it will also be added to U, unless it is already present in U. In either case, if $p \in C'$ then $p \in U'$. As this holds for each p, we have that $C' \subset U'$.

Finally, when the reset rule for each class i is applied in step 12, the set C' can only get smaller, and hence the property $C' \subset U'$ is still maintained. □

Feasibility. We now prove Lemma 4.6. We will apply induction over time t. The conditions of Lemma 4.6 clearly hold at time $t = 0$, when the cache is empty. It suffices to show the following.

Lemma 4.10. *If the conditions of Lemma 4.6 hold at time $t-1$, then at time t, there is at most one index i for which type-i reset occur. Moreover, in this case the condition for the type-i reset can be violated by at most 1, i.e. $|P_{\geq i} \cap C(t-1)| = \lceil k_{\geq i}(t) \rceil + 1$, and the cache has at least one page p of class i, where $p \neq p_t$.*

Proof. Fix an index i, and suppose $|P_{\geq i} \cap C(t-1)| \leq \lceil k_{\geq i}(t-1) \rceil$ holds at the end of time $t-1$. Suppose the requested page p_t lies in some class $< i$. Then, as the fractions $x_p(t)$ for pages in classes $\geq i$ can only be increased, and their cumulative increase is at most 1 (to satisfy the request for p_t). So, $k_{\geq i}(t-1) - 1 \leq k_{\geq i}(t) \leq k_{\geq i}(t-1)$, and hence $\lceil k_{\geq i}(t) \rceil \geq \lceil k_{\geq i}(t-1) \rceil - 1$, and as the condition for class i was satisfied at the end of $t-1$, the condition for type-i reset can be violated by at most 1.

Now, if there already is type-i' reset for some class $i' > i$, by the above, the condition will also hold for class i at time t and there will be no type-i reset. On the other hand, if i is the largest index for which type-i reset happens, and there was no class i page in $C(t-1)$, then

$$|P_{\geq i} \cap C(t-1)| = |P_{\geq i+1} \cap C(t-1)| \leq \lceil k_{\geq i+1}(t) \rceil \leq \lceil k_{\geq i}(t) \rceil,$$

contradicting the assumption that there was a type-i reset.

Now, suppose that p_t lies in some class $\geq i$. Then as p_t is fetched, $k_{\geq i}(t-1) \leq k_{\geq i}(t) \leq k_{\geq i}(t-1) + 1$. As the condition for class i was satisfied at the end of $t-1$, and pages other p_t can only possibly be deleted from $C(t-1)$, the type-i reset will occur iff p_t was not in the cache at time $t-1$, no pages from $P_{\geq i} \cap C(t-1)$ are evicted, no type-i' reset occurs for $i' > i$ and $k_{\geq i}(t) = k_{\geq i}(t-1)$. This also implies that the condition can be violated by at most 1. We consider two sub-cases depending on whether p_t lies in class i or $> i$.

If p_t lies in class i, and p_t is the only class i page, we claim that there cannot be a type-i reset. Indeed, as $1 - x_{p_t}(t) = 1$, $k_{\geq i+1}(t) = k_{\geq i}(t) - 1$, then if the condition was violated for class i, then it was also violated for the class $i+1$, resulting in a type-i' reset for $i' > i$.

Finally, if p_t lies in class $> i$, and there is no class i page in $C(t-1)$, and there is a type-i reset, then

$$|P_{\geq i+1} \cap C(t-1)| = |P_{\geq i} \cap C(t-1)| = \lceil k_{\geq i}(t) \rceil + 1 = \lceil k_{\geq i+1}(t) \rceil + 1$$

resulting in a type-i' reset for some $i' > i$, contradicting the assumption that there was a type-i reset. □

Cost analysis. We now analyze the online eviction cost. We first bound the cost due to applying the local rules, and then the cost due to the reset steps.

Lemma 4.11. *The eviction cost due to applying the local rules is at most β times the fractional cost of x.*

Proof. By the coupling in Lemma 4.9, and the local rules for deletion, if a page p in some C is evicted, then it is also evicted from any state U that C is coupled with. As each page p under the product distribution $D(t)$ has probability exactly $y_p(t)$ of being missing from a random state U, The total expected eviction cost for the product distribution D is exactly $\sum_t \sum_p w_p (y_p(t) - y_p(t-1))_+ \leq \sum_t \sum_p w_p \beta (x_p(t) - x_p(t-1))_+$. □

Lemma 4.12. *The expected cost of resets is at most $16k \exp(-\beta/4)$ times the cost of the fractional solution x. In particular, for $\beta = 4 \log k$, this is $O(1)$ times the fractional cost.*

Proof. Let us fix a time t, and class i. If $x_p(t) = x_p(t-1)$ for all pages $p \in P_{\geq i}$, then $k_{\geq i}(t) = k_{\geq i}(t-1)$ and no type-i reset can occur. By Lemma 4.5, as $x_p(t')$ for any t' is a multiple of $1/4k$,

$|k_{\geq i}(t) - k_{\geq i}(t-1)| \geq 1/4k$ whenever $k_{\geq i}(t) \neq k_{\geq i}(t-1)$. We will show that whenever $k_{\geq i}(t) \neq k_{\geq i}(t-1)$, the probability of a type-i reset at time t is at most $\exp(-\beta/4)$.

Then the total expected cost over all time and over all types of resets can be bounded by

$$\begin{aligned}
&\sum_t \sum_i 2^i \Pr[\text{ Type-}i \text{ reset at } t | \mathbf{1}_{k_{\geq i}(t) \neq k_{\geq i}(t-1)}] \cdot \mathbf{1}_{k_{\geq i}(t-1) \neq k_{\geq i}(t)} \\
&\leq \sum_t \sum_i 2^i \exp(-\beta/4) \cdot \left(\sum_{j \geq i} \sum_{p \in P_j} \mathbf{1}_{x_p(t-1) \neq x_p(t)} \right) \\
&\leq \exp(-\beta/4) \sum_t \sum_j \sum_{p \in P_j} \sum_{i \leq j} 2^i \cdot \mathbf{1}_{x_p(t-1) \neq x_p(t)} \\
&\leq \exp(-\beta/4) \sum_t \sum_j \sum_{p \in P_j} 2^{j+1} \cdot \left(4k \cdot |k_{\geq p}(t-1) - k_{\geq p}(t)|\right).
\end{aligned}$$

As $\sum_t w_p |x_p(t-1) - x_p(t)|$ is at most twice the total eviction cost of p (assuming $x_p(0) = 1$, and $w_p \geq 2^{j-1}$ for $p \in P_j$), the expected cost is at most $32k \exp(-\beta/4)$ times the fractional cost of x.

To bound the probability of type-i reset, by the coupling property, whenever cache C is coupled with some U, it suffices to bound the probability that the condition is violated for U under the distribution $D(t)$. Let $S = \{p : y_p(t) < 1, p \in P_{\geq i}\}$. Let U denote the random set under $D(t)$ restricted to pages in $P_{\geq i}$. Note that $U \subset S$, as page $p \in P_{\geq i}$ does not lie in U if $y_p(t) = 1$. We are interested in upper bounding $\Pr[|U| \geq \lceil k_{\geq i} \rceil + 1]$. As $k_{\geq i} = \sum_{p \in P_i} (1 - x_p) \geq \sum_{p \in S} (1 - x_p) = |S| - \sum_{p \in S} x_p$, it suffices to upper bound $\Pr\left[|U| \geq (|S| - \sum_{p \in S} x_p) + 1\right]$, or equivalently, $\Pr\left[|S \setminus U| \leq (\sum_{p \in S} x_p) - 1\right]$.

So we can assume that $\sum_{p \in S} x_p \geq 1$. For $p \in S$, let Y_p be the random variable which is 1 if $p \notin U$ and 0 otherwise. Then, $\sum_{p \in S} Y_p = |S \setminus U|$, and as $\mathbb{E}[Y_p] = y_p = \beta x_p$ (as $y_p < 1$). Denoting $\mu = \sum_{p \in S} \mathbb{E}[Y_p]$, we wish to bound $\Pr\left[\sum_p Y_p \leq \mu/\beta - 1\right]$. For $\mu \in [\beta, 2\beta)$, as the Y_p-s are integral, this is the same as

$$\Pr\Big[\sum_p Y_p = 0\Big] \leq \prod_{p \in S} (1 - y_p) \leq \exp\Big(-\sum_{p \in S} y_p\Big) \leq \exp(-\beta).$$

For $\mu \geq 2\beta$, and as $\beta = \Omega(\log k) \geq 2$,

$$\begin{aligned}
\Pr\Big[\sum_p Y_p \leq \mu/\beta - 1\Big] &\leq \Pr\Big[\sum_p Y_p \leq \mu/2\Big] \\
&\leq \exp(-\mu/8) \leq \exp(-\beta/4). \qquad \square
\end{aligned}$$

4.3.3 Rounding for Multi-level paging. Our approach is similar to that of weighted paging. However the rounding rule is a bit more involved, as the variables y correspond to differences of u variables (recall that $u(p,i,t) = 1 - \sum_{j=1}^{i} y(p,j,t)$, where $y(p,j,t)$ is the fraction of the j-th copy of the page p in the cache), so it can happen that some $y(p,i,t)$ increases (i.e. it is fetched), even when all $u(p,i,t)$ are increased. As before, we can assume that the values of $u(p,j,t)$ and $y(p,j,t)$ are integer multiples of $\delta = 1/(4k)$.

The rounding algorithm. Without loss of generality, we assume that at time t, solution u never increases the fraction of a copy of a page $p \neq p_t$. Algorithm 2 describes the local rounding step.

Analysis. The analysis follows similar steps as the analysis given for weighted paging. First, consider the current request (p_t, i_t). The algorithm loads (p_t, i_t) to the cache whenever there is no other copy $(p_t, i), i \leq i_t$ that can serve the request, and hence the current request (p_t, i_t) is served by $C(t)$.

Algorithm 2: Randomized Rounding Step

```
1  At time t, when the request for page (p_t, i_t) arrives:
2    if (p_t, j) ∈ C(t−1) for j > i_t then
3      evict (p_t, j).
4
5    if (p_t, j) ∉ C(t−1) for j ≤ i_t then
6      add (p_t, i_t) to C(t−1).
7
8    Foreach p ≠ p_t:
9      For i = 1, 2, ..., ℓ in increasing order
10       if (p, i) ∈ C(t) then
11         replace (p, i) with (p, i+1) with probability
             Δv(p,i,t)/[v(p,i−1,t) − v(p,i,t−1)]. (for
             i = ℓ we evict)
12     end
13   end
14   For i in decreasing order
15     if |P_{≥i} ∩ C(t−1)| > ⌈k_{≥i}(t)⌉ then
16       Type-i Reset: Evict an arbitrary page (p, j) with
           p ≠ p_t from P_i ∩ C(t−1)
17   end
18   Set C(t) = C(t−1)
19 end
```

Lemma 4.13. *For every page p and time t, $C(t)$ contains at most one copy of p.*

We now define an "almost" product distribution $D(t)$ that will be coupled with the distribution $C(t)$ over the random cache states generated by our rounding algorithm, similarly to the analysis of weighted paging. In the distribution $D(t)$, independently for each page p, we pick a copy of page p such that: (i) copy (p,i) is picked with probability $v(p,i-1,t) - v(p,i,t)$; (ii) none of the copies is picked with probability $v(p,\ell,t)$; (iii) at most one copy is picked. (This can be easily achieved by picking a random threshold θ_p uniformly at random from $[0,1]$.) We note that the way we define the distribution $D(t)$ guarantees that any set $U(t)$ sampled by $D(t)$ contains at most one copy of each page. The following lemma shows that this is indeed the "right" distribution for our coupling. The proof is given in the full version of the paper.

Lemma 4.14. *Applying the local rules of the loop at line 9 of Algorithm 2 on a cache $U(t)$ distributed according to $D(t)$ produces a cache $U(t+1)$ distributed according to $D(t+1)$.*

From here on, the proof follows along the same lines as the proof for weighted paging. The distribution $E(t)$ on caches $C(t)$ is coupled with $D(t)$, such that the coupling satisfies the subset property for each prefix, i.e. for every prefix $(p,1), \ldots, (p,i)$ for $1 \leq i \leq \ell$ if some copy $(p,j) : j \leq i$ is in the cache , then it is also in the coupled state under $D(t)$. As the marginals for prefixes are $1 - v(p,i,t)$ during $D(t)$, each prefix has β times more space of missing pages in expectation than u, and the expected reset cost can be bounded in an identical way to Lemma 4.12.

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. An $O(\log k)$-competitive algorithm for generalized caching. *ACM Trans. Algorithms*, 15(1):6:1–6:18, 2019.

[3] Susanne Albers, Sanjeev Arora, and Sanjeev Khanna. Page replacement for general caching problems. In *Symposium on Discrete Algorithms*, pages 31–40, 1999.

[4] N. Alon, B. Awerbuch, Y. Azar, N. Buchbinder, and J. Naor. The online set cover problem. In *Symposium on the Theory of Computation.*, pages 100–105, 2003.

[5] Nikhil Bansal, Niv Buchbinder, and Joseph Naor. A primal-dual randomized algorithm for weighted paging. *J. ACM*, 59(4):19, 2012.

[6] Nikhil Bansal, Niv Buchbinder, and Joseph Naor. Randomized competitive algorithms for generalized caching. *SIAM J. Comput.*, 41(2):391–414, 2012.

[7] Nikhil Bansal, Niv Buchbinder, and Joseph (Seffi) Naor. Towards the randomized k-server conjecture: A primal-dual approach. In *Symposium on Discrete Algorithms*, 2010.

[8] Nathan Beckmann, Phillip B. Gibbons, Bernhard Haeupler, and Charles McGuffey. Writeback-aware caching. In *Algorithmic Principles of Computer Systems, APOCS@SODA, 2020*, pages 1–15, 2020.

[9] Naama Ben-David, Guy E. Blelloch, Jeremy T. Fineman, Phillip B. Gibbons, Yan Gu, Charles McGuffey, and Julian Shun. Parallel algorithms for asymmetric read-write costs. In *Symposium on Parallelism in Algorithms and Architectures, SPAA*, pages 145–156, 2016.

[10] Naama Ben-David, Guy E. Blelloch, Jeremy T. Fineman, Phillip B. Gibbons, Yan Gu, Charles McGuffey, and Julian Shun. Implicit decomposition for write-efficient connectivity algorithms. In *International Parallel and Distributed Processing Symposium, IPDPS*, pages 711–722, 2018.

[11] Guy E Blelloch, Jeremy T Fineman, Phillip B Gibbons, Yan Gu, and Julian Shun. Sorting with asymmetric read and write costs. In *Symposium on Parallelism in Algorithms and Architectures*, pages 1–12, 2015.

[12] Guy E. Blelloch, Jeremy T. Fineman, Phillip B. Gibbons, Yan Gu, and Julian Shun. Efficient algorithms with asymmetric read and write costs. In *European Symposium on Algorithms, ESA*, volume 57 of *LIPIcs*, pages 14:1–14:18, 2016.

[13] Guy E. Blelloch, Yan Gu, Julian Shun, and Yihan Sun. Parallel write-efficient algorithms and data structures for computational geometry. In *Symposium on Parallelism in Algorithms and Architectures, SPAA*, pages 235–246, 2018.

[14] A. Blum, M. Furst, and A. Tomkins. What to do with your free time: algorithms for infrequent requests and randomized weighted caching, 1996.

[15] Avrim Blum, Carl Burch, and Adam Kalai. Finely-competitive paging. In *Symposium on Foundations of Computer Science (FOCS)*, page 450, 1999.

[16] Sébastien Bubeck, Michael B Cohen, Yin Tat Lee, James R Lee, and Aleksander Mądry. K-server via multiscale entropic regularization. In *symposium on theory of computing*, pages 3–16, 2018.

[17] Niv Buchbinder, Shahar Chen, and Joseph Naor. Competitive analysis via regularization. In *Symposium on Discrete algorithms*, pages 436–444, 2014.

[18] Niv Buchbinder, Joseph Seffi Naor, et al. The design of competitive online algorithms via a primal–dual approach. *Foundations and Trends® in Theoretical Computer Science*, 3(2–3):93–263, 2009.

[19] E. Carson, J. Demmel, L. Grigori, N. Knight, P. Koanantakool, O. Schwartz, and H. V. Simhadri. Write-avoiding algorithms. In *International Parallel and Distributed Processing Symposium (IPDPS)*, pages 648–658, 2016.

[20] Marek Chrobak and Lawrence L. Larmore. An optimal on-line algorithm for k-servers on trees. *SIAM J. Comput.*, 20(1):144–148, 1991.

[21] Guy Even, Moti Medina, and Dror Rawitz. Online generalized caching with varying weights and costs. In *Symposium on Parallelism in Algorithms and Architectures, SPAA*, pages 205–212, 2018.

[22] Martin Farach-Colton and Vincenzo Liberatore. On local register allocation. *Journal of Algorithms*, 37(1):37–65, 2000.

[23] U. Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998.

[24] U. Feige and S. Korman. On the use of randomization in the online set-cover problem. In *Thesis work, unpublished*, 2005.

[25] Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel Dominic Sleator, and Neal E. Young. Competitive paging algorithms. *Journal of Algorithms*, 12(4):685–699, 1991.

[26] Yan Gu, Yihan Sun, and Guy E. Blelloch. Algorithmic building blocks for asymmetric memories. In *European Symposium on Algorithms, ESA*, volume 112 of *LIPIcs*, pages 44:1–44:15, 2018.

[27] Anupam Gupta, Ravishankar Krishnaswamy, Amit Kumar, and Debmalya Panigrahi. Elastic caching. In *Symposium on Discrete Algorithms, SODA*, pages 143–156, 2019.

[28] Anupam Gupta, Amit Kumar, and Debmalya Panigrahi. Caching with time windows. In *Symposium on Theory of Computing*, 2020.

[29] S. Irani. Randomized weighted caching with two page weights. *Algorithmica*, 32(4):624–640, 2002.

[30] Sandy Irani. Competitive analysis of paging: A survey. In *Proc. of the Dagstuhl Seminar on Online Algorithms*, 1996.

[31] Sandy Irani. Page replacement with multi-size pages and applications to web caching. In *Symposium on Theory of computing*, pages 701–710, 1997.

[32] Abhishek Vijaya Kumar and Muthian Sivathanu. Quiver: An informed storage cache for deep learning. In *USENIX Conference on File and Storage Technologies (FAST 20)*, pages 283–296, 2020.

[33] Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. In *International Conference on Machine Learning, ICML, 2018*, volume 80, pages 3302–3311, 2018.

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

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

[36] Kan Wu, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. Towards an unwritten contract of intel optane SSD. In *USENIX Workshop on Hot Topics in Storage and File Systems, HotStorage*, 2019.

[37] Neal Young. On-line caching as cache size varies. In *Symposium on Discrete algorithms*, pages 241–250, 1991.

[38] Neal E. Young. The k-server dual and loose competitiveness for paging. *Algorithmica*, 11(6):525–541, 1994.

[39] Neal E. Young. On-line file caching. In *Symposium on Discrete algorithms*, pages 82–86, 1998.