Efficient Caching with Reserves via Marking

Sharat Ibrahimpur
Department of Mathematics, London School of Economics and Political Science, UK

Manish Purohit
Google Research, USA

Zoya Svitkina
Google Research, USA

Erik Vee
Google Research, USA

Joshua R. Wang
Google Research, USA

Abstract

Online caching is among the most fundamental and well-studied problems in the area of online algorithms. Innovative algorithmic ideas and analysis – including potential functions and primal-dual techniques – give insight into this still-growing area. Here, we introduce a new analysis technique that first uses a potential function to upper bound the cost of an online algorithm and then pairs that with a new dual-fitting strategy to lower bound the cost of an offline optimal algorithm. We apply these techniques to the Caching with Reserves problem recently introduced by Ibrahimpur et al. [10] and give an $O(\log k)$-competitive fractional online algorithm via a marking strategy, where k denotes the size of the cache. We also design a new online rounding algorithm that runs in polynomial time to obtain an $O(\log k)$-competitive randomized integral algorithm. Additionally, we provide a new, simple proof for randomized marking for the classical unweighted paging problem.

2012 ACM Subject Classification Theory of computation → Caching and paging algorithms

Keywords and phrases Approximation Algorithms, Online Algorithms, Caching

Digital Object Identifier 10.4230/LIPIcs.ICALP.2023.80

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2305.02508` [11]

Funding *Sharat Ibrahimpur*: Received funding from the following sources: NSERC grant 327620-09 and an NSERC DAS Award, European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement no. ScaleOpt–757481), and Dutch Research Council NWO Vidi Grant 016.Vidi.189.087.

1 Introduction

Caching is a critical component in many computer systems, including computer networks, distributed systems, and web applications. The idea behind caching is simple: store frequently used data items in a cache so that subsequent requests can be served directly from the cache to reduce the resources required for data retrieval. In the classical unweighted caching problem, a sequence of page requests arrives one-by-one and an algorithm is required to maintain a small set of pages to hold in the cache so that the number of requests not served from the cache is minimized.

Traditional caching algorithms, both in theory and practice, are designed to optimize the global efficiency of the system and aim to maximize the *hit rate*, i.e., fraction of requests that are served from the cache. However, such a viewpoint is not particularly suitable for

cache management in a multi-user or multi-processor environment. Many cloud computing services allow multiple users to share the same physical workstations and thereby share the caching system. In such multi-user environments, traditional caching policies can lead to undesirable outcomes as some users may not be able to reap any benefits of the cache at all. Recently, Ibrahimpur et al. [10] introduced the *Caching with Reserves* model that ensures certain user-level fairness guarantees while still attempting to maximize the global efficiency of the system. In this formulation, a cache of size k is shared among m agents and each agent i is guaranteed a reserved cache size of k_i. An algorithm then attempts to minimize the total number of requests that are not served from the cache while guaranteeing that any time step, each agent i holds at least k_i pages in the cache. Unlike the classical paging problem, *Caching with Reserves* is NP-complete even in the offline setting when the algorithm knows the entire page request sequence ahead of time and Ibrahimpur et al. [10] gave a 2-approximation algorithm. They also gave an $O(\log k)$-competitive online fractional algorithm for *Caching with Reserves* via a primal-dual technique and then design a rounding scheme to obtain an $O(\log k)$-competitive online randomized algorithm. Unfortunately, the rounding scheme presented in [10] does not run in polynomial time and the fractional primal-dual algorithm, while simple to state, also does not yield itself to easy implementation.

Caching and its many variants have been among the most well-studied problems in theoretical computer science. It has long been a testbed for novel algorithmic and analysis techniques and it has been investigated via general techniques such as potential function analysis, primal-dual algorithms, and even learning-augmented algorithms. For the classical unweighted caching problem, a particularly simple algorithm, *randomized marking* [9], is known to yield the optimal competitive ratio (up to constant factors). At any point in time, the randomized marking algorithm partitions the set of pages in cache into *marked* and *unmarked* pages and upon a cache miss, it evicts an unmarked page chosen uniformly at random. Cache hits and pages brought into the cache are marked. When a cache miss occurs, but there are no more unmarked pages, a new *phase* begins, and all pages in the cache become unmarked. In this paper, we build upon this algorithm, adapting it to caching with reserves.

Our Contributions

We study the *Caching with Reserves* model of Ibrahimpur et al. [10] in the online setting and improve upon those results. Our first main result is a simpler fractional algorithm that is a generalization of randomized marking for classical caching.

▶ **Theorem 1.** *There is an $O(\log k)$-competitive fractional* marking *algorithm for online* Caching with Reserves*. The competitive guarantee holds even when the optimal offline algorithm is allowed to hold fractional pages in the cache.*

We remark that our algorithm in Theorem 1 and its analysis are more involved than those of the classical randomized marking algorithm. One complication is that due to the reserve constraints, a marking-style algorithm for the caching with reserves setting cannot evict an arbitrary unmarked page. Another key difficulty comes from the fact that even the notion of a *phase* is non-trivial to define in our setting. In particular, unlike in classical caching, it can happen that the cache still contains unmarked pages, but none of them can be evicted to make space for a new page, because of the reserve constraints. Thus, we need a rule to isolate agents whose reserve constraints prevent the algorithm from having a clean end of a phase, while also ensuring that the already marked pages of such isolated agents are not erased prematurely. To this end, we introduce the notion of *global* and *local* phases to effectively model the state of each agent. We elaborate on this in Section 3.1.

Our analysis of the fractional marking algorithm introduces two novel components that may be of independent interest. First, we upper-bound the total cost incurred by our fractional marking algorithm using a new potential function. This potential function, introduced in Section 3.3, depends only on the decisions of the algorithm and is independent of the optimal solution. To the best of our knowledge, all previous potential function based analyses of (variants of) caching [4, 5, 6, 10] define a potential function that depends on the optimal solution. Second, we introduce a new lower bound for the cost of the optimal solution via the dual-fitting method. Our techniques also yield a new simple proof that the classical *randomized marking* [9] for unweighted paging is $O(\log k)$-competitive (see the full version [11] for more details).

We also design a new online rounding algorithm that converts a deterministic, fractional algorithm into a randomized, integral algorithm while only incurring a constant factor loss in the competitive ratio. Via a careful discretization technique (inspired by Adamaszek et al. [2]), the new rounding algorithm runs in polynomial time and only uses limited randomization. Our fractional marking algorithm (Algorithm 1) maintains that at any point in time, a particular page p is either completely in the cache or at least $1/k$ fraction of the page has been evicted. We exploit this key property to show that the fractional solution at any time t can be discretized so that the fraction of any page that is evicted is an integral multiple of $1/k^3$. This discretization allows us to maintain a distribution over feasible integer cache states with bounded support.

▶ **Theorem 2.** *There is a polynomial-time $O(\log k)$-competitive randomized integral algorithm for online caching with reserves.*

Other Related Work

The unweighted caching (also known as paging) problem has been widely studied and its optimal competitive ratio is well-understood even up to constant factors. Tight algorithms [1, 14] are known that yield a competitive ratio of exactly H_k, where H_k is the kth harmonic number. Recently, Agrawal et al. [3] consider the *parallel paging* model where a common cache is shared among p agents – each agent is presented with a request sequence of pages and the algorithm must decide how to partition the cache among agents at any time. It allows the p processors to make progress simultaneously, i.e., incur cache hits and misses concurrently. Multi-agent paging has also been extensively studied in the systems community [7, 16, 17] often in the context of caching in multi-core systems. Closely related to the *Caching with Reserves* setting, motivated by fairness constraints in multi-agent settings, a number of recent systems [12, 13, 15, 18] aim to provide *isolation guarantees* to each user, i.e., guarantee that the cache hit rate for each user is at least as much as what it would be if each user is allocated its own isolated cache. Also motivated by fairness constraints, Chiplunkar et al. [8] consider the Min-Max paging problem where the goal is to minimize the maximum number of page faults incurred by any agent.

2 Preliminaries and Notation

Formally, an instance of the *Caching with Reserves* problem consists of the following. We are given a number of agents m and a total (integer) cache capacity k. Let $[m]$ denote the set $\{1, \ldots, m\}$. Each agent $i \in [m]$ owns a set of pages $\mathcal{P}(i)$ (referred to as i-pages) and has a *reserved cache size* $k_i \geq 0$. Pages have a unique owner, i.e. $\mathcal{P}(i) \cap \mathcal{P}(j) = \varnothing$ for all $i \neq j$, and we use $\mathcal{P} \triangleq \cup_{i \in [m]} \mathcal{P}(i)$ to refer to the universe of all pages. For any page $p \in \mathcal{P}$, let $ag(p)$ be

the unique agent that owns p. We assume without loss of generality that at least one unit of cache is not reserved: $\sum_{i\in[m]} k_i < k$.[1] At each timestep t, a page $p_t \in \mathcal{P}$ is requested. We can wrap all these into an instance tuple: $\sigma = (m, k, \{\mathcal{P}(i)\}, \{k_i\}, \{p_t\})$.

An *integral* algorithm for the *Caching with Reserves* problem maintains a set of k pages in the cache such that for each agent i, the cache always contains at least k_i pages from $\mathcal{P}(i)$. At time t, the page request p_t is revealed to the algorithm. If this page is not currently in the cache, then the algorithm is said to incur a *cache miss* and it must fetch p_t into the cache by possibly evicting another page q_t. For any (integral) algorithm $\mathcal{A}$ we write its total cache misses on instance σ as $\text{cost}_{\mathcal{A}}(\sigma)$.

A *fractional* algorithm for the *Caching with Reserves* problem maintains a fraction $x_p \in [0, 1]$ for how much each page $p \in \mathcal{P}$ is in the cache such that the total size of pages in the cache is at most k, i.e, $\sum_{p\in\mathcal{P}} x_p \le k$, and the total size of i-pages is at least k_i, i.e., $\sum_{p\in\mathcal{P}(i)} x_p \ge k_i$. At time t, the page request p_t is revealed to the algorithm, which incurs a fractional cache miss of size $1 - x_{p_t}$. The algorithm must then fully fetch p_t into cache $(x_{p_t} \leftarrow 1)$ by possibly evicting other pages. For any (fractional) algorithm $\mathcal{A}$ we again write its total size of cache misses on instance σ as $\text{cost}_{\mathcal{A}}(\sigma)$.

Let $\text{cost}_{OPT}(\sigma)$ be the cost of the optimal offline algorithm on instance σ.

▶ **Definition 3** (Competitive Ratio). *An online algorithm $\mathcal{A}$ for* Caching with Reserves *is said to be c-competitive, if for any instance σ, $\mathbb{E}[\text{cost}_{\mathcal{A}}(\sigma)] \le c \cdot \text{cost}_{OPT}(\sigma) + b$, where b is a constant independent of the number of page requests in σ. The expectation is taken over all the random choices made by the algorithm (if any).*

3 Fractional $O(\log k)$-Competitive Algorithm for Caching with Reserves

For any time t and page $p \in \mathcal{P}$, the algorithm maintains a variable $y_p^t \in [0, 1]$ representing the portion of page p that is outside the cache. Then $x_p^t \triangleq 1 - y_p^t$ represents the portion of p that is in cache. Algorithm 1 ensures feasibility at all times t: the total of all y values is exactly the complementary cache size $|\mathcal{P}| - k$, i.e., $\sum_{p\in\mathcal{P}} y_p^t = |\mathcal{P}| - k$; and the total y value for pages of any agent i is within its respective complementary reserve size, $\sum_{p\in\mathcal{P}(i)} y_p^t \le |\mathcal{P}(i)| - k_i$. When a request for page p_t arrives at time t, the algorithm fully fetches p_t into the cache by paying a fetch-cost of $y_{p_t}^t$ while simultaneously evicting a total of $y_{p_t}^t$ amount of other suitably chosen pages.

3.1 Fractional Algorithm

The complete algorithm (referred to as Algorithm $\mathcal{A}$ in the proofs) is presented in Algorithm 1. We present a high-level discussion here. At any time t, we say that an agent i is *tight* if $\sum_{p\in\mathcal{P}(i)} x_p^t = k_i$, i.e., the algorithm is not allowed to further evict any pages (even fractionally) of agent i. Conversely, an agent i is *non-tight* if $\sum_{p\in\mathcal{P}(i)} x_p^t > k_i$.

The algorithm is a fractional marking algorithm and runs in phases where each phase corresponds to a maximal sequence of page requests that can be served while maintaining feasibility and ensuring that no "marked" pages are evicted. Within each phase, the currently requested page p_t is fully fetched into cache by continuously evicting an infinitesimal amount of an "available" (described below) unmarked page q with the smallest y_q value; if there are multiple choices of q, then all of them are simultaneously evicted at the same rate. Page p_t gets marked after it has been served and this mark may only be erased at the end of a phase.

[1] If all of cache is reserved, the problem decomposes over agents into the standard caching task.

At the end of a phase, an agent i is designated as *isolated* if strictly fewer than k_i i-pages are marked in the cache at this time point. This designation changes to non-isolated as soon as k_i i-pages get marked at some point in the future. An isolated agent essentially runs a separate instance of caching on its own pages and in its own reserved space. At the end of a phase, the marks of pages owned by non-isolated agents (i.e., agents with at least k_i marked i-pages) are erased.

It remains to describe when a page q is considered available for eviction. Clearly, $y_q^t < 1$ must hold, since otherwise page q is already fully outside the cache. Moreover, $ag(q)$ must be *non-tight*, i.e., evicting page q must not violate the reserve constraint of the agent that owns it. The last condition for q to be considered available for eviction depends on whether the agent $i_t := ag(p_t)$ is *isolated* or not: (i) if agent i_t is isolated, then $ag(q) = i_t$ should hold, i.e., only unmarked i_t-pages are available for eviction; and (ii) if agent i_t is not isolated, then $ag(q)$ should also be non-isolated. We recall again that among all available pages for eviction, pages with the smallest y_q^t value are evicted first.

Notation. Let $\mathcal{I}(t) \subsetneq [m]$ denote the set of isolated agents at time t. For a global phase r_0, we use $\mathcal{I}(r_0)$ to denote the set of isolated agents at the end of phase r_0. Let $\mathcal{T}(t)$ denote the set of *tight* agents at time t. At any time t, let r_i^t denote the value of the local phase counter r_i for agent i, and let $R_i = r_i^T$ be the total number of local phases for agent i. By definition, for any agent $i \in [m]$, $P(i, r_i - 1)$ and $P(i, r_i)$ denote the set of i-pages in the cache (integrally) at the beginning of the r_ith local phase and the end of the r_ith local phase for agent i, respectively. For any agent $i \in [m]$, let $M(i,t) \subseteq \mathcal{P}(i)$ denote the set of marked i-pages in the cache and $U(i,t) = P(i, r_i^t - 1) \setminus M(i,t)$ denote the set of unmarked i-pages.

We emphasize that the notion of *unmarked* pages will only be relevant while referring to pages in $P(i, r_i^t - 1)$ for some i, t; in particular, every i-page $q \in \mathcal{P}(i) \setminus (P(i, r_i^t - 1) \cup M(i,t))$ is not *marked*, but we do not refer to it as *unmarked*. Analogous to the notion of clean and stale pages used by the randomized marking algorithm [9], we define *clean, pseudo-clean* and *stale* pages as follows. Fix an agent i and let r_i be its local phase counter at time t. Any i-page $q \in P(i, r_i - 1)$ is considered *stale*. The currently requested page p_t is said to be *clean* if $p_t \notin P(i, r_i - 1)$. Next, we say that the currently requested page p_t is *pseudo-clean* if $p_t \in P(i, r_i - 1)$ and $y_{p_t}^t = 1$ holds right before Algorithm $\mathcal{A}$ starts to fetch p_t into the cache. Lemmas 7 and 8 show that a pseudo-clean page necessarily belongs to an agent who was isolated at the start of (global) phase r_0 but is non-isolated at time t. To simplify notation, we drop the superscript t from all notation whenever the time index is clear from the context.

The following lemma compiles a list of key invariants that are maintained throughout the execution of the algorithm that follow directly from an examination of Algorithm 1.

▶ **Lemma 4.** *Algorithm 1 maintains the following invariants.*

- **(i)** *When a new phase begins, all marked pages belong to isolated agents.*
- **(ii)** *At any time t, all isolated agents are tight.*
- **(iii)** *At any time t and for any agent i, all unmarked pages of agent i have the same y value.*
- **(iv)** *Any page belonging to an isolated agent is (fractionally) evicted only in those timesteps when a different page of the same agent has been requested.*

The following lemmas show that the algorithm is well-defined and that the operations in Lines 12 and 18 of Algorithm 1 are always feasible.

[2] Agent i_t is considered non-tight here because fetching p_t while evicting other $q \in \mathcal{P}(i_t) \setminus p_t$ does not violate reserve feasibility.

Algorithm 1 Fractional Marking Algorithm for *Caching with Reserves*.

```
/* Initialization */
r_0 ← 1                                                  /* global phase counter */
r_i ← 1, ∀i ∈ [m]                                        /* local phase counters */
Let P(i,0) ⊂ P(i) be set of i-pages in the initial cache (assume |P(i,0)| ≥ k_i) ∀i ∈ [m]
All agents i ∈ [m] are non-isolated and all pages p in the cache are unmarked
for each page request p_t of agent i_t do
    if y_{p_t} = 0, i.e., x_{p_t} = 1, then
        Mark page p_t and serve the request.
    else if agent i_t is isolated, then
        /* Continuously fetch page p_t while uniformly evicting all unmarked
           i_t-pages. */
        Set y_{p_t} ← 0, mark page p_t and serve the request.
        Increase y_q at the same rate for all unmarked i_t-pages in the cache until the cache
          becomes feasible, i.e. Σ_{p∈P} y_p ≥ |P| − k holds.
        if agent i_t now has k_i marked pages then
            Designate i_t as non-isolated
    else if ∃ page q owned by some non-tight agent[2] and satisfying y_q < 1, then
        /* Continuously fetch page p_t while uniformly evicting all unmarked
           pages (belonging to any non-tight agent) with the least y-value. */
        Set y_{p_t} ← 0, mark page p_t and serve the request.
        Increase y_q at the same rate for all unmarked pages of non-tight agents with the
          smallest y values until the cache becomes feasible, i.e. Σ_{p∈P} y_p ≥ |P| − k holds.
    else
        /* End of phase */
        for each agent i ∈ [m] do
            if i has strictly fewer than k_i marked pages, then
                Designate i as isolated.
            else
                /* i is non-isolated and undergoes a phase reset */
                Set P(i, r_i) ← collection of all (integral and marked) i-pages in cache.
                Set r_i ← r_i + 1
                All marked i-pages are now unmarked
        Set r_0 ← r_0 + 1
        Re-process the current page request p_t in the new phase
```

▶ **Lemma 5.** *If the requested page p_t has $x^t_{p_t} \in [0,1)$ and agent i_t is isolated, then p_t can be fetched fully by evicting unmarked pages of agent i_t.*

Proof. As agent i_t is isolated when page p_t is requested, $\sum_{p\in\mathcal{P}(i)} x^t_p = k_i$ (by invariant (ii) in Lemma 4) and i_t has fewer than k_i marked pages in cache. Hence $\sum_{p\in U(i,t)} x^t_p \geq 1$ and $\sum_{p\in U(i,t)\setminus\{p_t\}} x^t_p \geq 1 - x^t_{p_t}$. ◀

▶ **Lemma 6.** *If the requested page p_t has $0 < x^t_{p_t} < 1$ and its owner i_t is non-isolated, then there is always enough fractional mass of pages belonging to non-tight agents that can be evicted to fully fetch page p_t. In particular, line 18 of Algorithm 1 is well-defined.*

Proof. Suppose page p_t is fetched in a continuous manner. To show that page p_t can be fetched fully, it suffices to show that at any instantaneous time t when $x^t_{p_t} < 1$, there always exists an unmarked page q belonging to a non-tight agent i such that $x^t_q > 0$, i.e. page

q can be evicted. Observe that $k = \sum_{q \in \mathcal{P}} x_q^t = \sum_{i \in \mathcal{T}(t)} k_i + \sum_{i \notin \mathcal{T}(t)} \sum_{q \in \mathcal{P}(i)} x_q^t$. Due to integrality of k and $\{k_i\}_{i \in [m]}$, we must have $\sum_{i \notin \mathcal{T}(t)} \sum_{q \in \mathcal{P}(i)} x_q^t$ is an integer. Since any marked page q always has $x_q^t = 1$, $\mu := \sum_{i \notin \mathcal{T}(t)} \sum_{q \in U(i,t)} x_q^t$ is also an integer. Further, since $i_t \notin \mathcal{T}(t)$ and $x_{p_t}^t > 0$, we must have $\mu \geq 1$ and hence there exists a page q belonging to some non-tight agent i with $x_q^t > 0$ as desired. ◀

Since we always evict an available page with the least y value, at any time step t, all available pages (i.e., unmarked pages q belonging to non-tight agents and satisfying $y_q < 1$) have the same y value at all times. We denote this common y-value by h^* and refer to the corresponding set of evictable pages (with y-value h^*) as the *frontier*. The following two key structural lemmas formalize this property.

▶ **Lemma 7.** *At any time t, let i be an agent that was isolated at the beginning of the current phase, and let q be one of its unmarked pages. Then $y_q^t < 1$ if and only if i is still isolated at time t.*

Proof. For the *if* direction, suppose that i is still isolated. By invariant (ii), it is also tight. By invariant (iii), all its unmarked pages have the same x-value and in total they occupy $k_i - |M(i,t)| > 0$ units of cache space. Thus, $x_q^t > 0$ and $y_q^t < 1$.

For the *only if* direction, suppose that i is no longer isolated. Just before the k_ith i-page to be marked was requested, $(k_i - 1)$ i-pages were marked. Since i was tight, the total x value of all its unmarked pages must have been 1. Then the algorithm replaced all of them with the k_ith marked page, and the x-value of all remaining unmarked pages became 0. Thus, the property holds for a newly non-isolated agent i. This property continues to hold for the rest of the phase since y_q never decreases for an unmarked page. ◀

▶ **Lemma 8.** *At any time t, there is a value $h_t^* \in [0,1]$ such that: for any agent i that was non-isolated at the beginning of the current phase and any unmarked i-page q, $y_q \leq h_t^*$ holds and $y_q = h_t^*$ holds whenever i is non-tight.*

Proof. We prove the lemma by induction. Clearly, the lemma holds at the start of the phase: all unmarked pages belonging to non-isolated agents have y-value 0. Now consider a time t during phase r such that the lemma holds for all timepoints before t in this phase. We may also assume that $y_{p_t} > 0$, since otherwise none of the variables are modified in this timestep.

By induction hypothesis, any unmarked i-page q satisfies $y_q \leq h_{t-1}^*$, and this inequality is tight whenever i is non-tight. If agent i_t is non-tight, then its unmarked pages are already part of the frontier. Otherwise, Algorithm $\mathcal{A}$ fetches p_t fully into the cache by increasing the y-value of other unmarked i_t-pages until one of the following happens: (a) p_t is fully fetched. In this case, i_t continues to remain tight; or (b) The y-value of unmarked i_t-pages becomes equal to the frontier's y-value, h_{t-1}^*. In the latter case, unmarked i_t-pages become part of the frontier and the y-value of the frontier is uniformly increased until p_t gets fully fetched into the cache. If some agent i' becomes tight before the fetch operation is completed, then its unmarked pages get excluded from the frontier and the corresponding y-values remain unchanged for the rest of this timestep. In all cases, the lemma continues to hold since the y-value of the frontier is never decreased and only tight agents get dropped from the frontier. ◀

▶ Remark 9. Within any phase, h_t^* is non-decreasing over time and takes values 0 and 1 at the endpoints. This follows from the fact that $\mathcal{A}$ never decreases y_q for an unmarked page $q \neq p_t$.

The following lemma shows that any page that is not completely in Algorithm $\mathcal{A}$'s cache must be evicted to at least a $1/k$ portion. This property will be useful to us in Sections 3.3 and 4.

▶ **Lemma 10.** *At the end of any time step t, for any page $p \in \mathcal{P}$, we have $y_p^t = 0$ or $y_p^t \geq 1/k$.*

Proof. First, note that for all marked pages, we have $y_p^t = 1 - x_p^t = 0$. Let $i \in \mathcal{T}(t)$ be any tight agent. Then we have $k_i = \sum_{p \in \mathcal{P}(i)} x_p^t = |M(i,t)| + \sum_{p \in U(i,t)} x_p^t$. By Lemma 4 (part iii), all unmarked pages of agent i have the same y value: $y_p^t = 1 - x_p^t = h_i$ (say). Since k_i is integral, we have either $h_i = 0$ or h_i. Rearranging, we have $|U(i,t)|h_i = |M(i,t)| + |U(i,t)| - k_i$. Since all terms on the RHS are integral, we have either $h_i = 0$ or $h_i \geq 1/|U(i,t)| \geq 1/k$.

By Lemma 8, all unmarked pages p belonging to non-tight agents satisfy $y_p^t = h_t^*$. Let $U(t)$ be the set of all unmarked pages belonging to all non-tight agents that were also non-isolated at the beginning of the phase. Recall that by definition, we have $|U(t)| \leq k$ since all pages in $U(t)$ must have been fully in the cache at the beginning of the current phase. Since we have $k = \sum_{i \in \mathcal{T}(t)} k_i + \sum_{i \notin \mathcal{T}(t)} \sum_{p \in \mathcal{P}(i)} x_p^t$, once again by integrality of k and $\{k_i\}$, we must have that $\sum_{p \in U(t)} y_p^t = \sum_{p \in U(t)} h_t^*$ is an integer. Hence, either $h_t^* = 0$ or $h_t^* \geq 1/|U(t)| \geq 1/k$. ◀

3.2 Analysis Overview

At any time t, we consider the set of y values of pages in $\bigcup_{i \in [m]} P(i, r_i - 1)$ as the *state* of the system. We define a non-negative potential function Ψ that is purely a function of this state. For any page request p_t, we attempt to bound the algorithm's cost by an increase in the potential function, thereby bounding the total cost incurred by the algorithm by the final value of the potential function. There are two difficulties with this approach: (i) when a phase ends, the potential function abruptly drops since all the unmarked pages that were fully evicted no longer contribute to the state, and (ii) when the agent i_t was isolated at the beginning of the phase but is now non-isolated, the change in potential is not sufficient to cover the fetch cost. In both these situations we charge the cost incurred by the online algorithm to a new quantity that is a function of the sets $\{P(i, r_i)\}$. To complete the analysis, we show that this quantity is upper-bounded by the cost of the optimal solution.

3.3 Potential Function Analysis

Consider the function $\phi : [0,1] \to \mathbb{R}_{\geq 0}$ defined as:

$$\phi(h) \triangleq 2h \cdot \ln(1 + kh) \tag{1}$$

As h goes from 0 to 1, $\phi(h)$ increases from 0 to $2\ln(1+k)$.

The potential at any time t is defined as follows:

$$\Psi(t) \triangleq \sum_{i=1}^{m} \sum_{p \in U(i,t)} \phi(y_p^t) \tag{2}$$

Note that only unmarked pages at any time t contribute to the potential. So when page p_t is fetched at time t and marked, it stops contributing to the potential. But since ϕ is monotone, the newly evicted pages increase their contribution to the potential. We remark that the potential is purely a function of the state of the system as defined by the y values of unmarked pages in the cache and is thus always bounded by a quantity independent of the length of the page request sequence.

▶ **Lemma 11.** *For any $h \geq 1/k$, we have $\phi(h) \geq h$ and $\phi'(h) \geq 1 + 2\ln(1+kh)$.*

Proof. The first conclusion follows from the logarithmic inequality $\ln(1+x) \geq x/(1+x)$ which holds for any nonnegative x: we have $\phi(h) = 2h\ln(1+kh) \geq h \cdot 2kh/(1+kh) \geq h$ whenever $kh \geq 1$. Next, $\phi'(h) = \frac{d\phi}{dh} = 2(1 - 1/(1+kh) + \ln(1+kh))$. So, for any $h \geq 1/k$ we have $1/2 \geq 1/(1+kh)$, which gives the other conclusion. ◀

The rest of this section is devoted to proving the following theorem where we bound the total cost incurred by the algorithm in terms of the sets $\{P(i, r_i)\}$ and the number of requests to pseudo-clean pages.

▶ **Theorem 12.** *The following bound holds on the cost incurred by $\mathcal{A}$ to process the first T page requests:*

$$\text{cost}_{\mathcal{A}}(\sigma) \leq 2\ln(1+k) \cdot \left(mk + \sum_{t=1}^{T} \mathbb{1}_{p_t \text{ is pseudo-clean}} + \sum_{i \in [m]} \sum_{r_i=1}^{R_i} |P(i, r_i - 1) \setminus P(i, r_i)| \right).$$

Recall that the algorithm incurs a cost of $y^t_{p_t}$ to fetch page p_t at time t. So the total cost incurred by the algorithm is simply $\text{cost}_{\mathcal{A}}(\sigma) = \sum_t y^t_{p_t}$. We first bound this cost for time steps when the requested page p_t is at least partially in the cache, i.e., $y^t_{p_t} < 1$. Recall by Lemmas 5 and 6, the algorithm does not undergo a phase transition in this time step.

▶ **Lemma 13.** *Consider any time step t such that $y^t_{p_t} < 1$ for the currently requested (unmarked) page p_t. Let $\Delta\Psi(t)$ denote the change in the potential function during time step t. Then $y^t_{p_t} \leq \Delta\Psi(t)$.*

Proof. We assume that $y^t_{p_t} \geq \frac{1}{k}$, since otherwise by Lemma 10, we must have $y^t_{p_t} = 0$ and the lemma follows trivially. Since $y^t_{p_t} < 1$, by Lemma 8, either agent i_t is tight or we have $y^t_q = y^t_{p_t}$ for every unmarked page q owned by any non-tight agent i that was non-isolated at the start of this phase. In either case, the pages that get evicted to make space for p_t have their initial y values at least $y^t_{p_t} \geq 1/k$. The potential function Ψ changes in this step due to two factors: (i) Ψ drops as page p_t stops contributing to the potential as soon as it gets marked; and (ii) Ψ increases as the y-value of (fractionally) evicted pages increases in this step.

Let $h \triangleq y^t_{p_t}$. At the beginning to time t, page p_t contributed exactly $\phi(h) = 2h\ln(1+kh)$ to the potential; This contribution is lost as soon as p_t gets marked. To prove the lemma, it suffices to show that the rate of increase in the potential function (without including p_t's contribution) is at least $1 + 2\ln(1+kh)$ throughout the eviction of an h amount of unmarked pages belonging to non-tight agents: the 1 term in total pays for the fetch-cost of h and the $2\ln(1+kh)$ term in total pays for the $2h\ln(1+kh)$ loss in potential. This directly follows from Lemma 11 from the fact that the y-values of pages that are fractionally evicted in this timestep were already at least $h \geq 1/k$. Here, we also use the monotonicity of the function $h' \mapsto \ln(1+kh')$. ◀

We still need to bound the cost incurred by the algorithm when the incoming request is to a page that is fully outside the cache. Note that the algorithm incurs exactly unit cost for all such time steps. The following lemma shows that the total cost incurred by the algorithm can be bounded by the drop in potential function at the end of a phase and by a term that depends only on the change in the potential function while processing a request to a page fully outside the cache.

▶ **Lemma 14.** *For any global phase r_0, let $\Delta\Psi(r_0)$ denote the change in the potential function at the end of phase r_0 (line 30 in Algorithm 1). Let R_0 denote the total number of global phases and T denote the time at the end of phase R_0. Then we have the following upper bound on the cost incurred by $\mathcal{A}$ for processing the first T page requests:*

$$\text{cost}_{\mathcal{A}}(\sigma) \leq 2mk\ln(1+k) + \sum_{t\in[T]:y^t_{p_t}=1} (1-\Delta\Psi(t)) - \sum_{r_0=1}^{R_0} \Delta\Psi(r_0)$$

Proof. We have:

$$\begin{aligned}
\text{cost}_{\mathcal{A}}(\sigma) &= \sum_{t\in[T]} y^t_{p_t} = \sum_{t\in[T]:y^t_{p_t}<1} y^t_{p_t} + |\{t\in[T] : y^t_{p_t}=1\}| \\
&\leq \sum_{t:y^t_{p_t}<1} \Delta\Psi(t) + |\{t : y^t_{p_t}=1\}| && \text{(Using Lemma 13)} \\
&= \Psi(T) - \Psi(0) - \sum_{t:y^t_{p_t}=1} \Delta\Psi(t) - \sum_{r_0=1}^{R_0} \Delta\Psi(r_0) + |\{t : y^t_{p_t}=1\}|.
\end{aligned}$$

The lemma follows since $\Psi(T) \leq 2mk\ln(1+k)$ and $\Psi(0)=0$. The bound on $\Psi(T)$ is because we have m agents each with $|P(i,r_i-1)| \leq k$, and $\phi(1) = 2\ln(1+k)$. ◀

So, it is enough to bound the total cost and drop in potential for time steps when the requested page is fully outside the cache and also to bound the drop in potential when the phase changes.

Proof of Theorem 12. Consider any time step t such that the currently requested page p_t is fully outside the cache, i.e. $y^t_{p_t}=1$. We differentiate such requests into two cases depending on whether the page p_t is in the set $P(i_t, r_{i_t}-1)$ at the time or not. In other words, we do a case analysis on p_t being clean or pseudo-clean. (Recall that only unmarked pages in $P(i_t, r_{i_t}-1)$ contribute to the potential).

Case 1: $p_t \notin P(i_t, r_{i_t}-1)$, i.e, p_t is clean. Since page $p_t \notin U(i,t)$, it does not contribute to the potential before (or after) the request has been served. Consider any page q that is evicted (fractionally) by the algorithm in this step. By Lemma 4, before the eviction, we have $y_q = 0$ or $y_q \geq 1/k$. In either case, by Lemma 11, we have $\Delta\phi(y_q) \geq \Delta y_q$ where Δy_q denotes the change in y-value of page q in this step. Since we have $\sum_q \Delta y_q = y^t_{p_t} = 1$, we have $\Delta\Psi(t) = \sum_q \Delta\phi(y_q) \geq 1$.

Case 2: $p_t \in P(i_t, r_{i_t}-1)$, i.e., p_t is pseudo-clean. By the same reasoning as above, we have $\sum_{q\neq p_t} \Delta\phi(y_q) \geq 1$. However, in this case, page p_t also contributed exactly $2\ln(1+k)$ to the potential at the beginning of the time step. So we have $\Delta\Psi(t) = \sum_{q\neq p_t} \Delta\phi(y_q) - 2\ln(1+k) \geq 1 - 2\ln(1+k)$.

Combining the two cases we get:

$$\sum_{t:y^t_{p_t}=1} (1-\Delta\Psi(t)) \leq 2\ln(1+k)\cdot|\{t : y_{p_t}=1 \text{ and } p_t \in P(i_t, r_{i_t}-1)\}| \tag{3}$$

Consider the end of some phase r_0 and let i be a non-isolated agent. Let r_i denote the current local phase of agent i that must also end along with the global phase r_0. Consider any unmarked page q in $U(i,t)$. As the phase r_0 is ending, page q must be fully evicted and thus contributes $\phi(1)$ to the potential. Once phase r_0 ends and phase r_0+1 begins,

page q no longer contributes to the potential. Note that the set of such unmarked pages is exactly $P(i, r_i - 1) \setminus P(i, r_i)$. Hence, the change in potential at the end of (global) phase r_0 is given by:

$$\Delta\Psi(r_0) = -2\ln(1+k) \cdot \sum_{i \notin \mathcal{I}(r_0)} |P(i, r_i - 1) \setminus P(i, r_i)|$$

Since an agent only changes its local phase when it is non-isolated at the end of a global phase, we have:

$$\sum_{r_0=1}^{R_0} \Delta\Psi(r_0) = -2\ln(1+k) \cdot \sum_{i \in [m]} \sum_{r_i=1}^{R_i} |P(i, r_i - 1) \setminus P(i, r_i)|. \tag{4}$$

The theorem now follows from Lemma 14. ◀

3.4 A Lower Bound on OPT through Dual Fitting

In this section, we give a novel LP-based lower bound on the cost of any offline algorithm for caching with reserves via dual-fitting. This lower bound analysis is new even for the classical unweighted paging setting. Crucially, the lower bound derived here perfectly matches the two terms used to bound the cost of the fractional algorithm $\mathcal{A}$ in Theorem 12, thereby completing the proof of our main result (Theorem 1).

We now describe the linear relaxation of the caching with reserves problem and its dual program. The following notation will be useful. For any page $q \in \mathcal{P}$, let $t_{q,1} < t_{q,2} < \ldots$ denote the time steps when q is requested in the online sequence. For an integer $a \geq 0$, define $I(q, a) = \{t_{q,a} + 1, \ldots, t_{q,a+1} - 1\}$ to be the time interval between the ath and $(a+1)$th requests for q. We define $t_{q,0} \triangleq 0$ for all pages. Let $a(q, t)$ denote the number of requests to page q that have been seen until time t (inclusive). Hence, by definition, for any time t and page $q \in \mathcal{P} \setminus \{p_t\}$, we have $t \in I(q, a(q, t))$. The primal LP has variables $y(q, a) \in [0, 1]$ which denote the portion of page q that is evicted between its ath and $(a+1)$th requests, i.e., $1 - y(q, a)$ portion of q is held in the cache during the time-interval $I(q, a)$. For convenience, we define $n \triangleq |\mathcal{P}|$ and $n_i \triangleq |\mathcal{P}(i)|$ for any $i \in [m]$. The first and second set of primal constraints encode the cache size constraint and the agent-level reserve constraints for all times. The dual LP has variables $\alpha(t)$ and $\beta(t, i)$ corresponding to these primal constraints. We also have dual variables $\gamma(q, a)$ corresponding to the primal constraint encoding $y(q, a) \leq 1$. Besides nonnegativity, the dual has a single constraint for each interval $I(q, a)$. The primal and dual LPs are stated below. We emphasize that we use these linear programs purely for analysis and the algorithm itself does not need to solve any linear program.

Primal LP

$$\min \quad \sum_{q \in \mathcal{P}} \sum_{a \geq 1} y(q, a)$$

subject to:

$$\sum_{q \in \mathcal{P}, q \neq p_t} y(q, a(q, t)) \geq n - k \qquad \forall t \tag{5}$$

$$\sum_{q \in \mathcal{P}(i), q \neq p_t} y(q, a(q, t)) \leq n_i - k_i \qquad \forall t, \forall i \tag{6}$$

$$y(q, a) \leq 1 \qquad \forall q, \forall a \tag{7}$$

$$y \geq 0 \tag{8}$$

Dual LP

$$\max \sum_t (n-k)\alpha(t) - \sum_{t,i} (n_i - k_i)\beta(t, i) - \sum_{q,a} \gamma(q, a)$$

subject to:

$$\sum_{t \in I(q,a)} \big(\alpha(t) - \beta(t, ag(q))\big) - \gamma(q, a) \leq 1 \ \forall q, \forall a \tag{9}$$

$$\alpha, \beta, \gamma \geq 0 \tag{10}$$

Consider time T that marks the end of a global phase R_0 for some integer R_0. Let $\mathrm{OPT} = \mathrm{cost}_{\mathrm{OPT}}(\sigma)$ denote the total cost incurred by an optimal offline algorithm. By weak LP duality, the objective function of the Dual LP yields a lower bound on OPT for any feasible dual solution. We now construct an explicit dual solution (α, β, γ) whose objective value is roughly equal to the total number of clean and pseudo-clean pages seen by the algorithm. See Section 3.1 to recall relevant notation and terminology. The dual solution is updated at the end of each (global) phase in two stages. Updates in the first stage, denoted $\mathsf{update}(r_0, 1)$, are simple and account for stale pages belonging to non-isolated agents that got evicted in the most recent local phase for that agent. Updates in the second stage, denoted $\mathsf{update}(r_0, 2)$, are more involved and account for the pseudo-clean pages of agents who lost their isolated status in the current phase. The dual solution that we maintain will always be approximately feasible up to $O(1)$ factors, so the objective value of this dual solution serves as a lower bound on $\mathrm{OPT}(T)$ within a constant factor. We remark that the assumption that T marks the end of a phase is without loss of generality since it can lead to at most an additive $O(k)$ loss in the lower bound. Formally, we show the following.

▶ **Theorem 15.** *Let T denote the timepoint when global phase R_0 ends, and let $(R_i)_{i\in[m]}$ denote the corresponding local phase counters. Let (α, β, γ) denote the dual solution that is constructed by the end of time T, i.e., the solution that arises from a sequential application of dual updates in the order $\mathsf{update}(1,1), \mathsf{update}(1,2), \mathsf{update}(2,1), \mathsf{update}(2,2), \ldots, \mathsf{update}(R_0,1)$, and $\mathsf{update}(R_0,2)$. We have:*

(a) *The dual solution is approximately feasible: for any i-page q and an integer $a \geq 0$, $\sum_{t\in I(q,a)}(\alpha(t) - \beta(t,i)) - \gamma(q,a) \leq 5$ holds.*

(b) *The dual objective value of (α, β, γ) is:*

$$\begin{aligned}\mathrm{dual}(R_0) &\triangleq \sum_{t=1}^{T}(n-k)\alpha(t) - \sum_{t=1}^{T}\sum_{i\in[m]}(n_i - k_i)\beta(t,i) - \sum_{q\in\mathcal{P}}\sum_{a=1}^{a(q,T)}\gamma(q,a)\\ &= \sum_{i\in[m]}\sum_{r_i=1}^{R_i}|P(i,r_i-1)\setminus P(i,r_i)| + \sum_{r_0=1}^{R_0}\sum_{i\in\mathcal{I}(r_0-1)\setminus\mathcal{I}(r_0)}(|P(i,r_i-1)\cup P(i,r_i)| - k_i).\end{aligned}$$

We first show how Theorem 15 implies that our fractional algorithm $\mathcal{A}$ is $O(\log k)$-competitive.

Proof of Theorem 1. In Theorem 12 we proved the following upper bound on the cost incurred by $\mathcal{A}$ for processing the first T page requests:

$$\mathrm{cost}_{\mathcal{A}}(\sigma) \leq 2\ln(1+k)\cdot\Big(mk + \sum_{t=1}^{T}\mathbb{1}_{p_t \text{ is pseudo-clean}} + \sum_{i\in[m]}\sum_{r_i=1}^{R_i}|P(i,r_i-1)\setminus P(i,r_i)|\Big).$$

Clearly, the second nontrivial term in the above cost-expression matches the first term in the expression for $\mathrm{dual}(R_0)$. Now consider an arbitrary global phase $r_0 \in \{1,\ldots,R_0\}$ and a timestep t in this phase. By definition, a pseudo-clean page p_t is necessarily stale, i.e., $p_t \in P(i_t, r_{i_t}-1)$ holds, and it must be that agent i_t was isolated at the start of phase r_0 but is non-isolated by time t. Therefore, $i_t \in \mathcal{I}(r_0-1)\setminus\mathcal{I}(r_0)$ and the following holds:

$$|P(i,r_i-1)\cup P(i,r_i)| - k_i \geq |P(i,r_i)| - k_i \geq |\{t \in \text{phase } r_0 : p_t \text{ is pseudo-clean}\}|.$$

In the above, the final inequality is because among all pages in $P(i,r_i)$ (w.r.t. the order in which they were marked by $\mathcal{A}$), the first k_i pages are not pseudo-clean. Thus, the first nontrivial term in the cost-expression for $\mathcal{A}$ can be bounded by the second term in $\mathrm{dual}(R_0)$.

Overall, we have shown that $\mathrm{cost}_{\mathcal{A}}(\sigma) \leq 2\ln(1+k)\cdot\big(mk + \mathrm{dual}(R_0)\big)$ holds. Since the dual solution is $O(1)$-feasible, we get that $\mathcal{A}$ is $O(\log k)$-competitive. ◀

We now furnish the details of our dual updates. Initially, all our dual variables $\{\alpha(t)\}, \{\beta(i,t)\}, \{\gamma(q,a)\}$ with $t \in [T], i \in [m], q \in \mathcal{P}, a \in [a(q,T)]$ are set to zero. We assume that the dual updates are applied in the sequence given in Theorem 15. That is, the set of updates in $\{\mathsf{update}(r_0, s)\}_{r_0 \in [R_0], s \in \{1,2\}}$ are applied in increasing order of r_0 and within each phase first stage updates are applied first. With a slight abuse of notation, let $\text{dual}(r_0, s)$ denote the objective value of the dual solution right after updates until $\mathsf{update}(r_0, s)$ (inclusive) have been applied where $r_0 \in [R_0], s \in \{1, 2\}$. Note that $\text{dual}(R_0) = \text{dual}(R_0, 2)$. We also define $\text{dual}(0, 1) = \text{dual}(0, 2) := 0$. Throughout our updates, we ensure that the dual objective value never decreases, i.e., $0 \leq \text{dual}(1, 1) \leq \text{dual}(1, 2) \leq \cdots \leq \text{dual}(R_0, 1) \leq \text{dual}(R_0, 2)$ holds. We remark that β variables may decrease and this only happens in the second stage; However, the α and γ variables never decrease.

In Section 3.4, we describe the first stage of updates and show that the gain in the dual objective corresponds to the first term in Theorem 15(b). In Section 3.4, we describe the second stage of updates and show that the gain in the dual objective corresponds to the second term in Theorem 15(b). Lastly, in Section 3.4, we show that the dual solution that we maintain is always feasible up to constant factors and thus complete the proof of Theorem 15.

First Stage of Dual Updates

Fix a phase $r_0 \in [R_0]$ and consider the set $\mathcal{I}(r_0) \subsetneq [m]$ of agents that are designated as isolated at the end of phase r_0. Let $C(r_0)$ denote the set of timesteps t (in this phase) when the following two conditions hold: (a) $y^t_{p_t} = 1$ in the fractional algorithm $\mathcal{A}$ just before p_t is requested; and (b) i_t is not isolated at time t. Define $\ell^{(r_0)} \triangleq |C(r_0)|$. It is not hard to see that the following is an equivalent expression for $\ell^{(r_0)}$.

$$\ell^{(r_0)} := \sum_{i \notin \mathcal{I}(r_0-1) \cup \mathcal{I}(r_0)} |P(i, r_i) \setminus P(i, r_i - 1)| + \sum_{i \in \mathcal{I}(r_0-1) \setminus \mathcal{I}(r_0)} (|P(i, r_i)| - k_i). \tag{11}$$

Observe that for agents who are non-isolated both at the start and end of phase r_0, $\ell^{(r_0)}$ counts all their clean pages. However, for agents who were isolated at the start of this phase but are no longer isolated by the end, $\ell^{(r_0)}$ only counts clean and pseudo-clean pages that are requested *after* the agent has become non-isolated. Roughly speaking, the motivation for the definition of $\ell^{(r_0)}$ comes from the intuition that an offline algorithm should incur, on an average, a cost of $\Omega(\ell^{(r_0)})$ to serve page requests in phase r_0.

Description of $\mathsf{update}(r_0, 1)$. For each time $t \in C(r_0)$, we separately apply the following updates. First, we increase $\alpha(t)$ by $1/\ell^{(r_0)}$. Next, we increase $\beta(t, i)$ by $1/\ell^{(r_0)}$ for every agent $i \in \mathcal{I}(r_0)$. Last, for each agent $i \notin \mathcal{I}(r_0)$, we increase $\gamma(q, a(q, t))$ by $1/\ell^{(r_0)}$ for every i-page $q \in \mathcal{P}(i) \setminus (P(i, r_i - 1) \cup P(i, r_i))$.

It will be clear from the description of our updates that the α and β variables that were modified in $\mathsf{update}(r_0, 1)$ were previously at 0. However, no such guarantee holds for the affected γ variables. We also remark that the same $\gamma(q, a)$ variable can be increased more than once during $\mathsf{update}(r_0, 1)$; this happens when there are multiple times $t \in C(r_0)$ with the same $a(q, t)$ value. In fact, since the $\gamma(q, a)$ variables arise from intervals $I(q, a)$ that can possibly span across multiple phases, it is possible that the same γ variable is increased by different $1/\ell^{(r_0)}$ amounts across different $\mathsf{update}(r_0, 1)$ steps.

For convenience, let $t \in$ phase r_0 be a shorthand for all timepoints in phase r_0. The following result will be useful to us.

▶ **Lemma 16.** *Let (α, β, γ) denote the dual solution that is obtained right after* update$(r_0, 1)$ *has been applied. We have: (a) $\sum_{t \in phase\ r_0} \alpha(t) = 1$; and (b) $\sum_{t \in phase\ r_0} \beta(t,i) = 1$ for any agent $i \in \mathcal{I}(r_0)$.*

Proof. Follows directly from our choice of $\ell^{(r_0)} = |C(r_0)|$. ◀

Our key technical result in this section is that the gain in the dual objective value that comes from update$(r_0, 1)$ is equal to the number of stale pages owned by non-isolated agents that were not requested in their most recent local phases. For convenience, we use the prefix Δ to refer to changes that occured during update$(r_0, 1)$.

▶ **Lemma 17.** *We have* $\Delta\text{dual}(r_0, 1) = \sum_{i \notin \mathcal{I}(r_0)} |P(i, r_i - 1) \setminus P(i, r_i)|$, *where* $\Delta\text{dual}(r_0, 1) \triangleq \text{dual}(r_0, 1) - \text{dual}(r_0 - 1, 2)$ *is the change in the dual objective after* update$(r_0, 1)$.

Proof. Since the only affected $\alpha(t)$ and $\beta(t,i)$ variables have are those with $t \in C(r_0)$ and they are all increased by exactly $1/|C(r_0)|$, we get:

$$\begin{aligned}
\Delta\text{dual}(r_0, 1) &= \sum_t (n-k)\Delta\alpha(t) - \sum_{t,i} (n_i - k_i)\Delta\beta(t,i) - \sum_{q,a} \Delta\gamma(q,a) \\
&= (n-k) - \sum_{i \in \mathcal{I}(r_0)} (n_i - k_i) - \sum_{q,a} \Delta\gamma(q,a) \\
&= \Big(\sum_{i \notin \mathcal{I}(r_0)} n_i \Big) - k + \Big(\sum_{i \in \mathcal{I}(r_0)} k_i \Big) - \sum_{q,a} \Delta\gamma(q,a)
\end{aligned}$$

Now observe that for every $t \in C(r_0)$ and $i \notin \mathcal{I}(r_0)$, the update$(r_0, 1)$ step increases the $\gamma(q,a)$ variable corresponding to exactly $n_i - |P(i, r_i - 1) \cup P(i, r_i)|$ unique i-pages, each by an amount $1/\ell^{(r_0)}$. So we have $\sum_{q,a} \Delta\gamma(q,a) = (1/\ell^{(r_0)}) \cdot \sum_{t \in C(r_0)} \sum_{i \notin \mathcal{I}(r_0)} (n_i - |P(i, r_i - 1) \cup P(i, r_i)|) = \sum_{i \notin \mathcal{I}(r_0)} (n_i - |P(i, r_i - 1) \cup P(i, r_i)|)$. Substituting back into the equation above, we get:

$$\begin{aligned}
\Delta\text{dual}(r_0, 1) &= \Big(\sum_{i \notin \mathcal{I}(r_0)} n_i \Big) - k + \Big(\sum_{i \in \mathcal{I}(r_0)} k_i \Big) - \sum_{i \notin \mathcal{I}(r_0)} (n_i - |P(i, r_i - 1) \cup P(i, r_i)|) \\
&= \Big(-k + \sum_{i \in \mathcal{I}(r_0)} k_i + \sum_{i \notin \mathcal{I}(r_0)} |P(i, r_i)| \Big) + \sum_{i \notin \mathcal{I}(r_0)} |P(i, r_i - 1) \setminus P(i, r_i)|
\end{aligned}$$

The lemma follows from observing that the above group of terms within the parentheses is 0: this is because the cache (of size k) at the end of phase r_0 consists exactly k_i (fractional) pages for isolated agents $i \in \mathcal{I}(r_0)$ and exactly $|P(i, r_i)|$ (integral) pages for non-isolated agents $i \notin \mathcal{I}(r_0)$. ◀

Second Stage of Dual Updates

We now describe the second stage of dual updates that are carried out at the end of each phase $r_0 \in [R_0]$. Unlike the first stage, where we only increased the α and β variables of time steps in phase r_0, in the second stage we decrease the β variables of time steps in the previous phase $r_0 - 1$.

Description of update$(r_0, 2)$. These dual updates correspond to agents that were isolated at the end of phase $r_0 - 1$ but are no longer isolated at the end of phase r_0. Consider an agent $i \in \mathcal{I}(r_0 - 1) \setminus \mathcal{I}(r_0)$. For every time t in phase $r_0 - 1$ with $\beta(t,i) > 0$ (i.e., $t \in C(r_0 - 1)$), we do the following: we increase $\gamma(q, a(q,t))$ by $\beta(t,i)$ for all i-pages $q \in \mathcal{P}(i) \setminus \big(P(i, r_i - 1) \cup P(i, r_i)\big)$ followed by resetting $\beta(t,i)$ to 0.

For clarity, we note the following: (i) resetting $\beta(t,i)$ to zero is the only dual update when a variable is *decreased*; (ii) the $\beta(t,i)$ updates are applied to timepoints in phase r_0-1 (i.e., the previous phase); and (iii) the $\gamma(q,a(q,t))$ variables that we updated above were unchanged while applying $\mathsf{update}(r_0-1,1)$ at the end of phase r_0-1 because their owner i was designated as isolated at that time. The reason for decreasing the $\beta(t,i)$ variables is that it leads to an increase in the dual objective, which will be needed to pay for costs associated with pseudo-clean pages. We formalize this in the following lemma.

▶ **Lemma 18.** *We have* $\Delta\mathrm{dual}(r_0,2)=\sum_{i\in\mathcal{I}(r_0-1)\setminus\mathcal{I}(r_0)}\left(|P(i,r_i-1)\cup P(i,r_i)|-k_i\right)$ *where* $\Delta\mathrm{dual}(r_0,2)\triangleq\mathrm{dual}(r_0,2)-\mathrm{dual}(r_0,1)$ *is the change in the dual objective after* $\mathsf{update}(r_0,2)$.

Proof. Fix an agent $i\in\mathcal{I}(r_0-1)\setminus\mathcal{I}(r_0)$. In Lemma 16 we showed that after $\mathsf{update}(r_0-1,1)$, $\sum_{t\in\text{phase }r_0-1}\beta(t,i)=1$ and $\beta(t,i)\in\{0,1/\ell^{(r_0-1)}\}$. Consider any time t in phase r_0-1 with $\beta(t,i)>0$. By the definition of $\mathsf{update}(r_0,2)$, we decrease $\beta(t,i)$ by $1/\ell^{(r_0-1)}$ while increasing $\gamma(q,a(q,t))$ by the same amount for all pages $q\in\mathcal{P}(i)\setminus\big(P(i,r_i-1)\cup P(i,r_i)\big)$. Recalling the coefficients in the dual objective function, we see that the updates corresponding to agent i increases the dual objective by exactly:

$$(n_i-k_i)-|\mathcal{P}(i)\setminus(P(i,r_i)\cup P(i,r_i-1))|=|(P(i,r_i)\cup P(i,r_i-1))|-k_i.$$ ◀

Approximate Dual Feasibility

We finish this section by showing that the dual solution is always approximately feasible.

▶ **Lemma 19.** *Let* (α,β,γ) *denote the dual solution that is obtained right after* $\mathsf{update}(r_0,s)$ *has been applied for some* $r_0\in[R_0]$ *and* $s\in\{0,1\}$. *For any* i*-page* q *and an integer* $a\geq 1$ *satisfying* $a\leq a(q,T)$, *we have* $\sum_{t\in I(q,a)}(\alpha(t)-\beta(t,i))-\gamma(q,a)\leq 5$.

Proof. First of all, for the purposes of this proof, the specific values of r_0 and s are irrelevant, so we ignore them. Fix some i-page q and an integer a satisfying $a\leq a(q,T)$. Recall that $I(q,a)=\{t_{q,a}+1,\ldots,t_{q,a+1}-1\}$, where $t_{q,a'}$ denotes the time when q is requested for the a'th time; We redefine $t_{q,a+1}$ to be $T+1$ if $t_{q,a+1}>T$ holds.

The lemma holds trivially if $I(q,a)$ is empty, so we assume otherwise. Let $r_0^b,r_0^e\in[R_0]$ denote the global phases that contain timesteps $t_{q,a}+1$ and $t_{q,a+1}-1$, respectively. Clearly, $r_0^b\leq r_0^e$. Another easy case of the lemma is when $r_0^e\leq r_0^b+1$ holds. The desired conclusion follows easily because all the dual variables are nonnegative and the sum of all $\alpha(t)$ variables in any phase is at most 1 (by Lemma 16). Formally,

$$\sum_{t\in I(q,a)}\big(\alpha(t)-\beta(t,i)\big)-\gamma(q,a)\leq\sum_{t\in\text{phase }r_0^b}\alpha(t)+\sum_{t\in\text{phase }r_0^e}\alpha(t)\leq 2.$$

Now suppose that $r_0^b+2\leq r_0^e$ holds. Define $Z:=\{r_0^b+1,\ldots,r_0^e-1\}$. Repeating the above calculation, we get:

$$\sum_{t\in I(q,a)}(\alpha(t)-\beta(t,i))-\gamma(q,a)\leq 2+\left\{\sum_{r_0\in Z}\sum_{t\in\text{phase }r_0}\big(\alpha(t)-\beta(t,i)\big)\right\}-\gamma(q,a),$$

so the crux of the lemma is to show that the sum of $\delta(t)\triangleq\alpha(t)-\beta(t,i)$ over timepoints spanning phases in Z is not much larger than $\gamma(q,a)$. For a phase $r_0\in Z$, we overload the notation $\delta(r_0)$ to mean $\sum_{t\in\text{phase }r_0}\delta(t)$. Note that by nonnegativity of β variables and Lemma 17, $\delta(r_0)\leq 1$ for every $r_0\in Z$. We do a case analysis on phase $r_0\in Z$ to get a better handle on the changes that happens during our dual update procedures.

(a) Suppose that $i \in \mathcal{I}(r_0) \cap \mathcal{I}(r_0+1)$ holds. Since i is isolated by the end of phase r_0, we know that any increase in $\alpha(t)$ (as part of $\mathsf{update}(r_0, 1)$) for some $t \in C(r_0)$ is accompanied with the same increase in $\beta(t,i)$. Since $i \in \mathcal{I}(r_0+1)$ holds, $\mathsf{update}(r_0+1, 2)$ does not decrease/reset any of the $\{\beta(t,i)\}_{t \in \text{phase } r_0}$ variables to 0. Thus, $\delta(r_0) = 0$ holds. Note that there can be an arbitrary number of phases r_0 that fall under this case, but this is not a problem for us since $\delta(r_0) = 0$.

(b) Suppose that $i \in \mathcal{I}(r_0) \setminus \mathcal{I}(r_0+1)$ and $q \in P(i, r_i - 1) \cup P(i, r_i)$ hold. We rely on the trivial bound $\delta(r_0) \leq 1$ for this case. Since i is isolated by the end of phase r_0 but is non-isolated by the end of phase r_0+1, the local phase counter r_i goes up by 1 at the end of phase r_0+1, and subsequently the $P(i,\cdot)$ set gets updated with some new collection of marked i-pages. By definition of $I(q,a)$, there are no page-requests for q during any of the phases in Z. Thus, there can be at most 2 local phase increments for agent i before q gets dropped from the $P(i,\cdot)$ set; By the design of $\mathcal{A}$, page q cannot enter any of the future $P(i, r_i)$ until the next time it is requested, which does not happen during any of the phases in Z.

(c) Suppose that $i \in \mathcal{I}(r_0) \setminus \mathcal{I}(r_0+1)$ and $q \notin P(i, r_i - 1) \cup P(i, r_i)$ hold. Similar to case (a) above, we know that any increase in $\alpha(t)$ (as part of $\mathsf{update}(r_0, 1)$) for some $t \in C(r_0)$ is accompanied with the same increase in $\beta(t,i)$. Now, although $\mathsf{update}(r_0+1, 2)$ decreases/resets all the $\{\beta(t,i)\}_{t \in C(r_0)}$ variables to 0, it also increases $\gamma(q,a)$ by the same amount since $q \notin P(i, r_i - 1) \cup P(i, r_i)$. Thus, $\sum_{t \in \text{phase } r_0} \alpha(t)$ equals the increase in $\gamma(q,a)$ due to $\mathsf{update}(r_0+1, 2)$. So, the difference is essentially 0.

(d) Suppose $i \notin \mathcal{I}(r_0)$ and $q \in P(i, r_i - 1) \cup P(i, r_i)$ hold. We rely on the trivial bound $\delta(r_0) \leq 1$ for this case. Since i is non-isolated by the end of phase r_0, the local phase counter r_i goes up by 1 at the end of phase r_0. Repeating the argument from case (c), there can be at most 1 more local phase increment for agent i before q gets dropped from the $P(i,\cdot)$ set for the rest of the phases in Z.

(e) Suppose $i \notin \mathcal{I}(r_0)$ and $q \notin P(i, r_i - 1) \cup P(i, r_i)$ hold. Since i is non-isolated by the end of phase r_0 and q is not in $P(i, r_i - 1) \cup P(i, r_i)$, we know that any increase in $\alpha(t)$ (as part of $\mathsf{update}(r_0, 1)$) for some $t \in C(r_0)$ is accompanied with the same increase in $\gamma(q,a)$. Thus, $\sum_{t \in \text{phase } r_0} \alpha(t)$ equals the increase in $\gamma(q,a)$ due to $\mathsf{update}(r_0, 1)$. So, the difference is essentially 0.

From the above case analysis, it follows that $\{\sum_{r_0 \in Z} \sum_{t \in \text{phase } r_0} \big(\alpha(t) - \beta(t,i)\big)\} - \gamma(q,a)$ is bounded by the number of phases $r_0 \in Z$ for which case (b) or (d) hold. Since we argued that there can be at most 3 such occurences, $\sum_{t \in I(q,a)} (\alpha(t) - \beta(t,i)) - \gamma(q,a) \leq 2 + 3 = 5$ holds. ◀

We now prove the main theorem in this section by combining the above lemmas.

Proof of Theorem 15. The first part of the theorem follows from Lemma 19. The second part follows directly from Lemmas 17 and 18 since we have $\text{dual}(R_0) = \sum_{r_0=1}^{R_0} \big(\Delta\text{dual}(r_0, 1) + \Delta\text{dual}(r_0, 2)\big)$ ◀

4 Rounding

In this section we show how to convert the fractional Algorithm 1 into a randomized integral online algorithm for *Caching with Reserves*, each step of which runs in polynomial time.

The algorithm maintains a uniform distribution on $N = k^3$ valid cache states. In each step, these states are updated based on the actions of the fractional algorithm. The randomized algorithm selects one of these states uniformly at random in the beginning, and then follows it throughout the run.

Initially, all N cache states in the distribution are the same as the initial cache state in Algorithm 1. Given the fractional algorithm values x_p^t after each page request, the distribution is updated in two steps. First, we produce a discretized version of these fractions, $\tilde{x}_p^t$, which are also a feasible fractional solution. This is based on the technique in [2]. Second, we update the N cache states so that all of them remain valid, and for each page p, exactly $N \cdot \tilde{x}_p^t$ of the states contain p. This is based on the technique in [10]. We note that the rounding procedure can be done online, as it does not need the knowledge of any future page requests.

4.1 Discretization Procedure

In this subsection, we explain how to perform the first step: discretizing the fractional algorithm's values x_p^t into $\tilde{x}_p^t$ which are multiples of $\frac{1}{N}$. Our procedure is quite simple: we iterate over the pages in any order π that arranges all pages belonging to the same agent consecutively (i.e., order the agents arbitrarily and order each agent's pages arbitrarily, but do not interleave pages from different agents). Then for $i \in [|\mathcal{P}|]$, set:

$$\tilde{x}_{\pi(i)}^t \triangleq \left\lfloor \sum_{j=1}^{i} x_{\pi(j)}^t \right\rfloor_{1/N} - \left\lfloor \sum_{j=1}^{i-1} x_{\pi(j)}^t \right\rfloor_{1/N}$$

where $\lfloor a \rfloor_b$ denotes rounding a down to the nearest multiple of b; formally: $\lfloor a \rfloor_b \triangleq b \lfloor a/b \rfloor$.

▶ **Lemma 20.** *Discretization satisfies the following guarantees:*

1. $\tilde{x}_p^t$ *is a multiple of* $1/N$
2. $\left|\tilde{x}_p^t - x_p^t\right| < 1/N$
3. *for each agent* i, $\left|\sum_{p\in\mathcal{P}(i)} \tilde{x}_p^t - \sum_{p\in\mathcal{P}(i)} x_p^t\right| < 1/N$
4. *if* $x_p^t \in \{0,1\}$, *then* $\tilde{x}_p^t = x_p^t$

Due to space constraints, the proof of the above lemma is deferred to the full version [11].

▶ **Corollary 21.** *If* $\{x_p^t\}$ *satisfy total cache capacity* ($\sum_{p\in\mathcal{P}} x_p^t \le k$) *and reserve requirements* ($\sum_{p\in\mathcal{P}(i)} x_p^t \ge k_i$), *then so do* $\{\tilde{x}_p^t\}$.

Proof. The total cache capacity constraint continues to hold due to a telescoping argument:

$$\begin{aligned}\sum_{p\in\mathcal{P}} \tilde{x}_p^t &= \sum_{i\in[|\mathcal{P}|]} \tilde{x}_{\pi(i)}^t = \sum_{i\in[|\mathcal{P}|]} \left[\left\lfloor \sum_{j=1}^{i} x_{\pi(j)}^t \right\rfloor_{1/N} - \left\lfloor \sum_{j=1}^{i-1} x_{\pi(j)}^t \right\rfloor_{1/N} \right] \\ &= \left\lfloor \sum_{j=1}^{|\mathcal{P}|} x_{\pi(j)}^t \right\rfloor_{1/N} \le \sum_{j=1}^{|\mathcal{P}|} x_{\pi(j)}^t \le k\end{aligned}$$

Next, we will prove that reserve cache sizes are satisfied. For the sake of contradiction, suppose that for some agent i, $\sum_{p\in\mathcal{P}(i)} \tilde{x}_p^t < k_i$. Since the right-hand side of this inequality is an integer and therefore a multiple of $1/N$, the left-hand side, which is also a multiple of $1/N$ due to being a sum of multiples of $1/N$ (by Lemma 20's first guarantee), must be at least a full multiple of $1/N$ less than the right-hand side: $\sum_{p\in\mathcal{P}(i)} \tilde{x}_p^t \le k_i - 1/N$. But this contradicts Lemma 20's third guarantee, $\left|\sum_{p\in\mathcal{P}(i)} \tilde{x}_p^t - \sum_{p\in\mathcal{P}(i)} x_p^t\right| < 1/N$. Therefore for all agents i, $\sum_{p\in\mathcal{P}(i)} \tilde{x}_p^t \ge k_i$, completing the proof. ◀

▶ **Corollary 22.** $\sum_p \left|\tilde{x}_p^t - x_p^t\right| \le k^2/N$

Proof. Without loss of generality, there are at most k agents since we can combine all agents that do not have any reserve. Each agent i has at most k fractional pages by the algorithm (the i-pages that were in cache when i's local phase began). The x value of each fractional page is distorted by at most $1/N$ by Lemma 20's second guarantee, while not being distorted for non-fractional pages by the fourth guarantee. This completes the proof. ◀

▶ **Lemma 23.** *Let x_p^t and x_p^{t+1} be the amounts of each page p in cache in the fractional algorithm for two consecutive time steps, and $\tilde{x}_p^t$ and $\tilde{x}_p^{t+1}$ be the corresponding discretized values. Then the cost of cache update from $\tilde{x}^t$ to $\tilde{x}^{t+1}$ (call it $\tilde{c}$) is at most twice the cost of cache update from x^t to x^{t+1} (call it c).*

Proof. Lemma 10 implies that either $c = 0$ (i.e., the requested page was already in cache and there is no change to the cache state), or $c \ge 1/k$. In the first case, there is no change to the discretized cache state either, so $\tilde{c} = 0$. So we focus on the second case. By triangle inequality, for any page p,

$$|\tilde{x}_p^t - \tilde{x}_p^{t+1}| \le |\tilde{x}_p^t - x_p^t| + |x_p^t - x_p^{t+1}| + |x_p^{t+1} - \tilde{x}_p^{t+1}|.$$

We note that since cost is incurred for adding pages to cache, and both the original fractional solution and the discretized one add as much page mass to cache as they evict, $2c = \sum_p |x_p^t - x_p^{t+1}|$, and similarly for $\tilde{c}$. Summing the above inequality over p, we get

$$2\tilde{c} = \sum_p |\tilde{x}_p^t - \tilde{x}_p^{t+1}| \le \sum_p |x_p^t - x_p^{t+1}| + 2k^2/N = 2c + 2/k \le 4c,$$

where we used $\sum_p (|\tilde{x}_p^t - x_p^t| + |x_p^{t+1} - \tilde{x}_p^{t+1}|) \le 2k^2/N$ by Corollary 22, then $N = k^3$, then $c \ge 1/k$. ◀

4.2 Updating the Distribution of Cache States

In this subsection, we explain how to perform the second step: updating the N cache states. We would like (i) all cache states to be valid, (ii) exactly $N \cdot \tilde{x}_p^t$ (integral due to our discretization step) of the states to contain page p, and (iii) to not use too many evictions.

Formally, let $\mathcal{X}$ be a set of N cache states with k pages each, which corresponds to the discretized values $\tilde{x}_p^t$ for time step t. Given the discretized values $\tilde{x}_p^{t+1}$ for time step $t+1$, we show how to transform $\mathcal{X}$ into $\mathcal{X}'$, in which each page p appears in exactly $N \cdot \tilde{x}_p^{t+1}$ cache states and such that each cache state satisfies all the reserve requirements.

Let P be a multiset of pages whose fraction in the cache increased from time t to $t+1$, with each page p appearing $\max(0, (\tilde{x}_p^{t+1} - \tilde{x}_p^t)N)$ times. Let Q be an analagous multiset for decreases, with each page appearing $\max(0, (\tilde{x}_p^t - \tilde{x}_p^{t+1})N)$ times. Since the total amount of pages in the cache is unchanged, $|P| = |Q|$. We find a matching between pages in P and Q and use it to transform $\mathcal{X}$ into $\mathcal{X}'$ gradually, one pair at a time. The matching is constructed as follows. First, any pages from P and Q that belong to the same agent are matched up. Then, the remaining pages in P and Q are matched up arbitrarily.

▶ **Lemma 24.** *Let $(p_1, q_1), (p_2, q_2), \ldots$ be the matching between P and Q described above. Then for any j, the fractional solution that adds $1/N$ fraction of pages $p_1, \ldots, p_j$ to $\tilde{x}^t$ and removes $1/N$ fraction of pages $q_1, \ldots, q_j$ from it satisfies all the reserve requirements.*

We defer the proof of the above lemma to the full version [11].

We now show how to modify $\mathcal{X}$ with the next pair (p, q) from the matching. This follows the procedure in [10], with the difference that we work on a limited number of N sets, and the amount of increase in p and decrease in q is fixed at $1/N$.

Let $\mathcal{X}$ be the current set of cache states (possibly modified by the previous page pairs). If there is a cache state $S \in \mathcal{X}$ such that $p \notin S$ and $q \in S$, add p to S and remove q from S. Otherwise, find cache states $S \in \mathcal{X}$ and $T \in \mathcal{X}$ with $p \notin S$ and $q \in T$, add p to S and remove q from T. Next, move some page $r \in S \setminus T$ from S to T to adjust the set sizes back to k.

At this point, each page is in the correct number of cache states. However, reserve requirements could be violated by one page for $ag(q)$ or $ag(r)$ in the cache states from which the corresponding pages were removed. In such a case, suppose the requirement is violated for agent i in a cache state $V \in \mathcal{X}$. Since, by Lemma 24, each reserve requirement is satisfied on average, there must be another set $W \in \mathcal{X}$ which has strictly more than k_i pages belonging to agent i. We move one such page from W to V. Now V has $k + 1$ pages, so there must be an agent j which has more than k_j pages in V. We move one of j's pages from V to W to restore the sizes. This completes the update, resulting in new valid sets corresponding to the fractions $\tilde{x}^{t+1}$.

We conclude by bounding the cost of update to $\mathcal{X}$, and thus the expected cost of the randomized algorithm, relative to the cost of the fractional algorithm.

▶ **Lemma 25.** *The cost of update to $\mathcal{X}$ is at most 6 times the cost of fractional cache update from $\tilde{x}^t$ to $\tilde{x}^{t+1}$.*

Proof. Each pair (p, q) in the matching corresponds to a cost of $1/N$ incurred by the discretized fractional solution. In the updates to sets in $\mathcal{X}$, each time a page is removed from one of the cache states incurs a cost of $1/N$ to the randomized algorithm. At most, the following six removals are done: remove q from T; remove r from S; two pages each are swapped to fix the reserve requirements for $ag(q)$ and $ag(r)$. ◀

The proof of Theorem 2 follows by combining Lemmas 23 and 25.

References

1 Dimitris Achlioptas, Marek Chrobak, and John Noga. Competitive Analysis of Randomized Paging Algorithms. *Theoretical Computer Science*, 234(1):203–218, 2000. doi:10.1016/S0304-3975(98)00116-9.

2 Anna Adamaszek, Artur Czumaj, Matthias Englert, and Harald Räcke. An $O(\log k)$-Competitive Algorithm for Generalized Caching. *ACM Transactions on Algorithms*, 15(1):6:1–6:18, 2018. doi:10.1145/3280826.

3 Kunal Agrawal, Michael A. Bender, Rathish Das, William Kuszmaul, Enoch Peserico, and Michele Scquizzato. Tight Bounds for Parallel Paging and Green Paging. In *Proceedings of the 32nd Symposium on Discrete Algorithms*, pages 3022–3041, 2021. doi:10.1137/1.9781611976465.180.

4 Nikhil Bansal, Niv Buchbinder, and Joseph Seffi Naor. A Simple Analysis for Randomized Online Weighted Paging. *Unpublished Manuscript*, 2010.

5 Nikhil Bansal, Niv Buchbinder, and Joseph (Seffi) Naor. Towards the Randomized k-Server Conjecture: A Primal-Dual Approach: (Extended Abstract). In *Proceedings of the 21st Symposium on Discrete Algorithms*, pages 40–55, 2010. doi:10.1137/1.9781611973075.5.

6 Nikhil Bansal, Christian Coester, Ravi Kumar, Manish Purohit, and Erik Vee. Learning-Augmented Weighted Paging. In *Proceedings of the 33rd Symposium on Discrete Algorithms*, pages 67–89, 2022. doi:10.1137/1.9781611977073.4.

7 Jichuan Chang and Gurindar S Sohi. Cooperative Cache Partitioning for Chip Multiprocessors. In *Proceedings of the 21st ACM International Conference on Supercomputing*, pages 242–252, 2007. doi:10.1145/1274971.1275005.

8 Ashish Chiplunkar, Monika Henzinger, Sagar Sudhir Kale, and Maximilian Vötsch. Online Min-Max Paging. In *Proceedings of the 34th Symposium on Discrete Algorithms*, pages 1545–1565, 2023. doi:10.1137/1.9781611977554.ch57.

9 Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel D. Sleator, and Neal E. Young. Competitive Paging Algorithms. *Journal of Algorithms*, 12(4):685–699, 1991. doi:10.1016/0196-6774(91)90041-V.

10 Sharat Ibrahimpur, Manish Purohit, Zoya Svitkina, Erik Vee, and Joshua R. Wang. Caching with Reserves. In *Proceedings of the 25th International Conference on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, volume 245, pages 52:1–52:16, 2022. doi:10.4230/LIPIcs.APPROX/RANDOM.2022.52.

11 Sharat Ibrahimpur, Manish Purohit, Zoya Svitkina, Erik Vee, and Joshua R. Wang. Efficient Caching with Reserves via Marking. *CoRR*, abs/2305.02508, 2023. doi:10.48550/arXiv.2305.02508.

12 Wu Kan, Tu Kaiwei, Patel Yuvraj, Sen Rathijit, Park Kwanghyun, Arpaci-Dusseau Andrea, and Remzi Arpaci-Dusseau. NyxCache: Flexible and Efficient Multi-tenant Persistent Memory Caching. In *Proceedings of the 20th USENIX Conference on File and Storage Technologies (FAST)*, pages 1–16, 2022. URL: https://www.usenix.org/conference/fast22/presentation/wu.

13 Mayuresh Kunjir, Brandon Fain, Kamesh Munagala, and Shivnath Babu. ROBUS: Fair Cache Allocation for Data-parallel Workloads. In *Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD)*, pages 219–234, 2017. doi:10.1145/3035918.3064018.

14 Lyle A. McGeoch and Daniel D. Sleator. A Strongly Competitive Randomized Paging Algorithm. *Algorithmica*, 6:816–825, 1991. doi:10.1007/BF01759073.

15 Qifan Pu, Haoyuan Li, Matei Zaharia, Ali Ghodsi, and Ion Stoica. FairRide: Near-Optimal, Fair Cache Sharing. In *Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 393–406, 2016. URL: https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/pu.

16 Harold S. Stone, John Turek, and Joel L. Wolf. Optimal Partitioning of Cache Memory. *IEEE Transactions on Computers*, 41(9):1054–1068, 1992. doi:10.1109/12.165388.

17 G Edward Suh, Larry Rudolph, and Srinivas Devadas. Dynamic Partitioning of Shared Cache Memory. *The Journal of Supercomputing*, 28(1):7–26, 2004. doi:10.1023/B:SUPE.0000014800.27383.8f.

18 Yinghao Yu, Wei Wang, Jun Zhang, and Khaled Ben Letaief. LACS: Load-Aware Cache Sharing with Isolation Guarantee. In *Proceedings of the 39th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 207–217. IEEE, 2019. doi:10.1109/ICDCS.2019.00029.