Online Caching with Convex Costs

[Extended Abstract]

Ishai Menache
Microsoft Research
Redmond, WA
ishai@microsoft.com

Mohit Singh
Microsoft Research
Redmond, WA
mohits@microsoft.com

ABSTRACT

Modern software applications and services operate nowadays on top of large clusters and datacenters. To reduce the underlying infrastructure cost and increase utilization, different services share the same physical resources (e.g., CPU, bandwidth, I/O, memory). Consequently, the cluster provider often has to decide in real-time how to allocate resources in overbooked systems, taking into account the different characteristics and requirements of users. In this paper, we consider an important problem within this space – how to share *memory* between users, whose memory access patterns are unknown in advance. We assume that the overall performance (or cost) of each user is a non-linear function of the total number of misses over a given period of time. We develop an online caching algorithm for arbitrary cost functions. We further provide theoretical guarantees for *convex* functions (which capture plausible practical scenarios). In particular, our algorithm is $\alpha^\alpha k^\alpha$-competitive, where k is the memory (cache) size, and α is a constant which depends on the curvature of the cost functions. We also obtain a bi-criteria result which trades-off the performance and the memory size. Finally, we give a lower bound on the performance of any online deterministic algorithm which nearly matches the upper bound of our algorithm.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*sequencing and scheduling*; D.4.2 [**Operating Systems**]: Storage Management—*Main memory*

General Terms

Algorithms

Keywords

online caching; resource management; cloud computing; competitive analysis

SPAA'15, June 13–15, 2015, Portland, OR, USA.

DOI: http://dx.doi.org/10.1145/2755573.2755585.

1. INTRODUCTION

1.1 Background and Motivation

The cloud computing paradigm builds upon the economy of scale, where customers utilize compute resources contained in large datacenters. The potential economic gains rely on *multi-tenancy* – assigning different users and services into the same physical hardware. Multi-tenancy allows for cost reduction and increased utilization of the infrastructure, as the cloud provider can often exploit statistical multiplexing for overbooking of physical resources.

Memory is inarguably a crucial resource for many applications and services operating on the cloud, such as dynamic web applications and Databases as a Service (DaaS). In some of these systems, the physical memory is shared between different users, e.g., in Microsoft Azure SQL Database [13]. In addition, distributed memory systems have been designed and deployed in order to accelerate the performance of cloud applications. Examples include Memcached [10], an open-source in-memory key-value store, and commercial offerings such as Microsoft Azure Cache [12] and Amazon ElastiCache [2].

An important issue in such shared memory systems is how to allocate the physical memory between different users and applications. Static memory allocation are inherently both wasteful (i.e., users holding to memory which that do not utilize) and might fail to meet user requirements (e.g., a performance-sensitive user needs more memory than expected). Instead, when oversubscription is in place, the memory allocation problem can be treated as a dynamic, *online caching* problem: the "cache" here is the total available physical memory, and the provider needs to determine in real-time which pages of which users should be stored in memory, and which ones need to be evicted. Variants of the LRU algorithm, such as LRU-K[16] have been employed for many shared-memory systems, however they treat all users equally. Weighted caching [20, 3] generalizes LRU to settings where users may have different weights (priorities). Under the weighted caching model, each miss of user i has a fixed cost w_i. Unfortunately, this model is not general enough, since the costs associated with misses could be *non-linear*, i.e., each additional miss need not cost the same as previous misses of the user.

In this paper, we develop a cost-aware, online caching algorithm for the case where the cost of each user is non-linear in the number of misses. The objective of the algorithm is to minimize the sum of tenants' costs. We further provide performance guarantees when the costs are *convex*. The convexity assumption is natural in practical settings, since each additional miss has often times an increasingly greater impact on tenant performance. For example, a user can tolerate up to around M misses in a time window of T, and any number of misses greater than that will result in substan-

tial degradation in performance. Such scenarios can be captured through, e.g., piecewise-linear, convex cost functions.

Based on the algorithm developed in this paper, we have recently designed and prototyped a memory replacement mechanism for SQLVM [15], a multi-tenant DaaS system. In the SQLVM context, the shared memory is the so-called *buffer pool* memory, which serves as a cache of database pages and is crucial for the performance of user workloads. The Service Level Agreements (SLAs) between the provider and users is captured via non-linear cost functions, which could correspond, for example, to the refund paid by a service provider as a function of the total number of misses or related measures thereof; see [14] for more examples. Experiments with real workloads demonstrate the merits of our cost-aware approach [14].

1.2 Our Model and Results

We consider the problem of having multiple tenants sharing a single cache[1] and propose the following model. We have a single cache of size k which is shared among set of users U. We let $n = |U|$ denote the number of users. Each user $i \in U$, owns a set of pages P_i and we receive a sequence of T page requests $(p_1, p_2, \ldots, p_T)$ where each page p_t belongs to a unique user $i \in U$. We assume that the number of requests in unknown to the algorithm designer. At each time $1 \leq t \leq T$, the page p_t is requested and the algorithm must ensure that page p_t must either be in the cache or be fetched into the cache. If the cache is full, i.e., exactly k pages are already in the cache, we must evict one of the pages from the cache to make space for the requested page p_t. The algorithm designer does not know the whole sequence of page requests in advance, but it is rather revealed over time. When deciding the action for page p_t, only the request sequence till page p_t is known to the algorithm and not the page requests which come after page p_t. Each user $i \in U$ is also associated with a cost function $f_i : \mathbb{R} \to \mathbb{R}$ where $f_i(x)$ denotes the cost paid if user i has x misses. For our analysis, we assume each f_i is differentiable, convex, increasing and non-negative function with $f_i(0) = 0$. The objective is to minimize the sum total cost paid for all users.

Since the problem is online in nature, we appeal to *competitive analysis* to study the performance of our algorithm, see [6] for an introduction on online algorithms and competitive analysis. We compare the performance of our online algorithm to the performance of the optimal *offline* algorithm that knows the sequence of page requests σ in advance. We prove the following theorem.

THEOREM 1.1. *There exists an online deterministic algorithm for the multiple tenant caching problem which given any request sequence σ has $a_i(\sigma)$ misses for each user $i \in U$ such that*

$$\sum_{i \in U} f_i(a_i(\sigma)) \leq \sum_{i \in U} f_i(\alpha k b_i(\sigma)) \tag{1}$$

where $b_i(\sigma)$ are the number of misses by the optimal offline algorithm, and $\alpha = \sup_{x,i} \frac{x f_i'(x)}{f(x)}$.

Our algorithm is a primal-dual algorithm for a convex programming relaxation for the multiple tenant caching problem. Observe that when each f_i is a linear function, i.e., each miss costs the same, then $\alpha = 1$ and we obtain the standard k-competitive algorithm which is the best possible competitive ratio for any algorithm [19]. The following corollary is useful to illustrate our results when the convex functions are monomial functions.

[1]Throughout the paper, we use the terms "cache" and "memory" interchangeably.

COROLLARY 1.2. *Suppose that the cost function f_i for each user i is given by $f_i(x) = x^\beta$ for some $\beta \geq 1$. There exists an online algorithm which is $(\beta^\beta k^\beta)$-competitive. Thus given any request sequence σ, the algorithm has $a_i(\sigma)$ misses for each user $i \in U$ such that*

$$\sum_{i \in U} f_i(a_i(\sigma)) \leq \beta^\beta k^\beta \sum_{i \in U} f_i(b_i(\sigma)), \tag{2}$$

where $b_i(\sigma)$ are the number of misses by the optimal offline algorithm.

We also give a bi-criteria result which trades off the performance of the algorithm with the cache size. In particular, we compare the performance of our algorithm with the optimal offline algorithm where the offline algorithm must work with a cache size of $h \leq k$. In such a setting, we give an improved result in Theorem 1.3. Observe that the algorithm proving the guarantee of Theorem 1.3 is the same as the algorithm of Theorem 1.1 and is also independent of h.

THEOREM 1.3. *There exists an online deterministic algorithm for the multiple tenant caching problem which given any request sequence σ has $a_i(\sigma)$ misses for each user $i \in U$ such that*

$$\sum_{i \in U} f_i(a_i(\sigma)) \leq \sum_{i \in U} f_i(\alpha \frac{k}{k-h+1} b_i(\sigma)), \tag{3}$$

where $b_i(\sigma)$ are the number of misses by the optimal offline algorithm which is given a cache of size $h \leq k$ and $\alpha = \sup_{x,i} \frac{x f_i'(x)}{f(x)}$.

We also obtain nearly matching lower bounds for our algorithm in the following theorem.

THEOREM 1.4. *For any n and integer β, there exists an instance of the multi-tenancy caching problem with n users and cost functions $f_i(x) = x^\beta$ for each user $i \in U$, such that any online deterministic algorithm must pay a cost of at least $(\Omega(k))^\beta$ times the cost of the optimal offline solution.*

To compare the result in Theorem 1.4 with the upper bound in Corollary 1.2, we observe that lower bound and upper bounds match up to a factor of β^β.

1.3 Related Work

Competitive analysis of the caching problem was introduced by Sleator and Tarjan [19] who gave a k-competitive algorithm and also showed that the classical algorithm LRU is k-competitive. In our setting, the basic caching problem corresponds to the special case of our problem with a single user. Sleator and Tarjan [19] also showed that this is the best possible competitive ratio for any deterministic algorithm. Young [20] generalized this result to the weighted caching problem, which in our setting corresponds to the case where each of the functions f_i is linear. The algorithm in [20] uses a primal-dual framework over the corresponding linear program. While we also provide a primal-dual algorithm, our convex program builds on a different linear program which was given by Bansal, Buchbinder and Naor [3] for the weighted caching problem; [3] obtains improved competitive algorithms using randomization. We also mention that their results also generalize to the bi-criteria setting; we refer the reader to the survey by Buchbinder and Naor [8] for more details.

Caching algorithms have been widely applied for over half a century to divide memory in various computer systems, including operating systems (e.g., [9]), multi-core processors (e.g., [11]) and databases (e.g., [21] and references therein). Accordingly, much

$$\begin{array}{lll} \text{(ICP)} & \min \sum_{i=1}^{m} f_i \left(\sum_{p \in P_i} \sum_{j=1}^{r(p,T)} x(p,j) \right) & \\ \text{s.t.} & & \\ & \forall 1 \leq t \leq T \quad \sum_{p \in B(t) \setminus \{p_t\}} x(p, j(p,t)) \geq |B(t)| - k & \\ & \forall p \in P, \forall j \quad x(p,j) \in \{0,1\} & \end{array}$$

$$\begin{array}{lll} \text{(CP)} & \min \sum_{i=1}^{m} f_i \left(\sum_{p \in P_i} \sum_{j=1}^{r(p,T)} x(p,j) \right) & \\ \text{s.t.} & & \\ & \forall 1 \leq t \leq T \quad \sum_{p \in B(t) \setminus \{p_t\}} x(p, j(p,t)) \geq |B(t)| - k & \\ & \forall p \in P, \forall j \quad x(p,j) \leq 1 & \\ & \forall p \in P, \forall j \quad x(p,j) \geq 0 & \end{array}$$

Figure 1: Integer Convex Program and its Convex Programming Relaxation

research has been devoted to understand and analyze caching algorithms that work well in practice [4, 1, 17]. Recently, there has been some interest in developing caching algorithms for cloud computing scenarios [18, 5]. The latter references consider scenarios where the memory available to an algorithm can vary over time. Our work is different in the sense that we consider systems with fixed memory size with more complicated user cost models.

The rest of the paper is organized as follows. In Section 2 we give our online algorithm and prove Theorem 1.1. In Section 3, we prove the bi-criteria result (Theorem 1.3). The lower bound in Theorem 1.4 is proven in Section 4.

2. ALGORITHM

Before we describe the algorithm, we first give a convex programming formulation for the multiple tenant caching problem. We remark that the algorithm does not solve this convex program but uses it as a tool to guide the algorithm, in the spirit of primal-dual algorithms. As mentioned above, the algorithm can actually be used for general non-linear cost functions (see Section 2.5 for details), yet our performance guarantees require convexity.

We begin with some notation. We let $P = \cup_i P_i$ denote the set of all pages. For any page p, we let $i(p) \in U$ denote the user owning that page. Given any sequence σ of requests of pages which comes online, we index the sequence with time. Hence at time t, we obtain the t^{th} request in sequence σ and let p_t be the page requested at time t. We let $r(p,t)$ denote the number of requests of page p till time t. We also denote T to be length of σ. Let $B(t)$ denote the number of distinct pages requested up to time t. For any page p, the time between its two consecutive requests is an interval. We let $j(p,t)$ denotes the interval index corresponding to page p at time t (notice that $j(p,t)$ does not depend on the algorithm but only on the sequence σ).

2.1 Convex Programming Formulation

We now formulate the following integer convex programming formulation for our problem assuming that we have knowledge of the sequence of requests σ. Relaxing the integer constraints, we obtain a convex programming relaxation for our problem. We have a variable $x(p,j)$ for each page p and each $1 \leq j \leq r(p,T)$ where $x(p,j)$ will be set to 1 if the page p is evicted between its j^{th} and $(j+1)^{th}$ request. We assume that the cost paid by the algorithm is for evicting a page and not when bringing it in. We remedy this by assuming that the algorithm needs to return an empty cache, therefore each page is evicted exactly equal number of times it is brought in the cache. This is implemented by a dummy user who owns k pages and all these k pages are appended at the end of sequence σ. The cost of evicting any page for this dummy user is assumed to be infinite. Then the overall cost for user i is given by $f_i(\sum_{p \in P_i} \sum_{j=1}^{r(p,T)} x(p,j))$.

The integer convex program (ICP) is formalized in Figure 1. The constraints for the convex program are indexed by time t. For each time t, we must have all but k pages outside the cache. Moreover, page p_t which is requested at time t must not be counted in the excluded pages. We first observe that every algorithm must imply a feasible solution to (ICP) by simply setting $x(p,j) = 1$ if page p is evicted between its j^{th} and $(j+1)^{th}$-request. Moreover, the cost of the objective function is also the same as the cost of the solution given by the algorithm. Our aim will be to construct a feasible solution to (ICP) in an online fashion as the sequence σ is revealed. We emphasize that we do not solve the convex program in the algorithm. Nonetheless, as the sequence σ is revealed, we extend the convex program to include the relevant set of variables and constraints. We will use convex duality to guide the algorithm as well as for the analysis which we describe in the following sections.

2.2 Optimality Conditions

Our algorithm will aim to maintain an approximately optimal primal solution along with a Lagrangian dual solution certifying the approximate optimality. First, we describe the Lagrangian dual of the convex program (CP) and the optimality conditions. We then also list the approximate optimality conditions that will be maintained by our algorithm.

Let y_t, $z(p,j)$ and $\mu(p,j)$ denote the Lagrange multipliers corresponding to first, second and third set of constraints in the convex program (CP), respectively. Then the Lagrangian $L(x,y,z,\mu)$ is given by

$$\min \sum_{i=1}^{m} f_i \left(\sum_{p \in P_i} \sum_{j=1}^{r(p,T)} x(p,j) \right) + \sum_{p,j} z(p,j) \left(x(p,j) - 1 \right) +$$
$$+ \sum_{t} y_t \left(|B(t)| - k - \sum_{p \in B(t) \setminus \{p_t\}} x(p, j(p,t)) \right) - \sum_{p,j} \mu(p,j) x(p,j)$$

Now, the KKT optimality conditions [7] imply that (x,y,z,μ) will be optimal if we satisfy the following conditions.

1. Primal and Dual Feasibility
   (a) $\sum_{p \in B(t) \setminus \{p_t\}} x(p, j(p,t)) \geq |B(t)| - k \ \ \forall t$
   (b) $0 \leq x(p,j) \leq 1$ for all p, j
   (c) $0 \leq y, z, \mu$
2. Complementary Slackness
   (a) $y_t \left(\sum_{p \in B(t) \setminus \{p_t\}} x(p, j(p,t)) - |B(t)| + k \right) = 0 \quad \forall t$
   (b) $z(p,j) \left(x(p,j) - 1 \right) = 0 \quad \forall p, j$
   (c) $\mu(p,j) x(p,j) = 0 \quad \forall p, j$
3. Gradient conditions.
   (a) $f'_{i(p)} \left(\sum_{p' \in P_{i(p)}} \sum_{r=1}^{r(p,T)} x(p',r) \right) - \sum_{t=t(p,j+1)-1}^{t(p,j+1)-1} y_t + z(p,j) - \mu(p,j) = 0$ for all p, j

where $t(p,j)$ is the time when the j-th request to page p. The primal and dual feasibility conditions are natural. The complementary conditions ensure the dual variable is non-zero only if the primal constraints are tight. Moreover, the partial derivative of the Lagrangian objective must be non-negative for all primal variables, and if the primal variable is strictly positive then the gradient must be zero. Of course all these conditions will not be satisfied exactly by our algorithm as that would imply solving the problem exactly but we would aim to satisfy most of them. The exact invariants of the algorithm are described later.

Eliminating the dual variables μ, we obtain the following optimality conditions.

1. Primal and Dual Feasibility
   (a) $\sum_{p\in B(t)\setminus\{p_t\}} x(p,j(p,t)) \geq |B(t)|-k \ \ \forall t$
   (b) $0 \leq x(p,j) \leq 1$ for all p,j
   (c) $0 \leq y, z$
2. Complementary Slackness
   (a) $z(p,j)\,(x(p,j)-1) = 0 \quad \forall p,j,$
   (b) $y_t \left(\sum_{p\in B(t)\setminus\{p_t\}} x(p,j(p,t)) - |B(t)| + k\right) = 0 \quad \forall t,$
   (c) $f'_{i(p)}\left(\sum_{p'\in P_{i(p)}} \sum_{r=1}^{r(p,T)} x(p',r)\right) - \sum_{t=t(p,j)+1}^{t=t(p,j+1)-1} y_t +$ $z(p,j) = 0$ if $x(p,j) > 0$
3. Gradient conditions.
   (a) $f'_{i(p)}\left(\sum_{p'\in P_{i(p)}} \sum_{r=1}^{r(p,T)} x(p',r)\right) - \sum_{t=t(p,j)+1}^{t=t(p,j+1)-1} y_t +$ $z(p,j) \geq 0$ for all p,j

2.3 Primal-Dual Online Algorithm

We describe the continuous version of the algorithm in Figure 2. A discrete version of the algorithm can be simply implemented by observing that all continuous changes boil down to discrete amounts. We shall present the discrete version and discuss some implementation details in Section 2.5.

For every time $1 \leq t \leq T$, let p_t be the requested page. Do

- Initialize $x^\circ(p_t,j_t) \leftarrow 0$, $z^\circ(p_t,j_t) \leftarrow 0$, $y^\circ_t \leftarrow 0$
- If (CP) remains feasible, do nothing.
  We do need to remove any page from the cache. Either the requested page p_t is already in the cache or we have space for the new page in the cache.
- Else
  - Increase y°_t continuously.
  - For each page p outside cache (i.e., $x^\circ(p,j(p,t)) = 1$), increase $z^\circ(p,j(t))$ at same rate of y°_t
  - Let p' be the first page in the cache for which

$$f'_{i(p')}\left(m(i(p'),t-1)+1\right) - \sum_{t=t(p',j)+1}^{t(p',j+1)-1} y^\circ_t + z^\circ(p',j) = 0$$

  is satisfied; set $x^\circ(p',j(p',t)) \leftarrow 1$
  We remove page p' from the cache and bring in page p_t.

Figure 2: Algorithm ALG-CONT

Invariant of Algorithm ALG-CONT. Our algorithm will maintain primal solution x° and dual solution (y°, z°) such that we satisfy primal and dual feasibility, i.e. conditions(1a)-(1c). While the algorithm can be described in terms of the primal solution alone, we need the dual solution to guide the algorithm and prove the guarantee as claimed in Theorem 1.1. For ease of notation, for any user $i \in U$ and $1 \leq t \leq T$, we let $m(i,t)$ be the total number of evictions of pages owned by user i till time t by our algorithm, i.e., $m(i,t) = \sum_{p\in P_i} \sum_{j=1}^{r(p,t)} x^\circ(p,j)$. We let $m(i,T)$ to be the total misses for pages owned by user i. We maintain the following invariants:

1. Primal and Dual Feasibility
   (a) $\sum_{p\in B(t)\setminus\{p_t\}} x^\circ(p,j(p,t)) \geq |B(t)|-k \ \ \forall t$
   (b) $0 \leq x^\circ(p,j) \leq 1$ for all p,j
   (c) $0 \leq y^\circ, z^\circ$
2. Complementary Slackness
   (a) $z^\circ(p,j)\,(x^\circ(p,j)-1) = 0 \quad \forall p,j,$
   (b) $f'_{i(p)}\left(\sum_{p'\in P_{i(p)}} \sum_{r=1}^{r(p,\hat{t})} x^\circ(p',r)\right) - \sum_{t=t(p,j+1)-1}^{t=t(p,j)+1} y^\circ_t +$ $z^\circ(p,j) = 0$ if $x^\circ(p,j)$ is set to 1 at time $\hat{t}$.
3. Gradient conditions.
   (a) $f'_{i(p)}\left(\sum_{p'\in P_{i(p)}} \sum_{r=1}^{r(p,T)} x^\circ(p',r)\right) - \sum_{t=t(p,j+1)-1}^{t=t(p,j)+1} y^\circ_t +$ $z^\circ(p,j) \geq 0$ for all p,j

The main difference from the optimality conditions (after eliminating the dual variables μ) is the complemenatry slackness condition (2b) in the invariant as compared to condition (2c) in the optimality conditions. The algorithm tries to maintain condition (2c) as if the derivative of the lagrangian is evaluated at the current primal and dual solution and not at the final solution. At a later time t, $\sum_{r=1}^{r(p,t)} x^\circ(p',r) \geq \sum_{r=1}^{r(p,\hat{t})} x^\circ(p',r)$ and therefore, the gradient of f at the former sum might be larger than the gradient at the latter term due to convexity of f. While this does not affect feasibility of primal and dual solutions, we will violate the complementary slackness conditions. The technical heart of the analysis will be to show that we can still bound the violation in terms of the degree of f.

2.4 Proof of Theorem 1.1

The proof of Theorem 1.1 follows from the following two lemmas which we prove in this section. Lemma 2.1 states that the algorithm ALG-CONT satisfies the invariants as stated above. Then in Lemma 2.2, we show that any algorithm which satisfies the invariant conditions must satisfy the claimed guarantee in Theorem 1.1.

LEMMA 2.1. *At all times t,* ALG-CONT *satisfies the invariant conditions as claimed.*

PROOF. We check each of the invariants.

Primal and Dual Feasibility. Clearly conditions (1b) and dual feasibility (1c) are satisfied by construction. Variable $x^\circ(p,j)$ takes values from $\{0,1\}$ and $y^\circ(t)$ and $z^\circ(p,j)$ are initialized to 0 and only increase as the algorithm progresses. To see that we satisfy the condition (1a), a simple induction suffices. At $t=0$, we have $|B(t)| = 0$ and therefore, $x^\circ(p,j)$ and $(y^\circ(t), z^\circ(t))$ are feasible. In any other step, as we go from time $t-1$ to time t, one of the following happens. If after setting $x^\circ(p_t,j_t) = 0$, $z^\circ(p_t,j_t) = 0$ and $y^\circ(t) = 0$, we obtain a feasible solution then induction holds. Else,

this initialization of $x^\circ(p_t, j_t) = 0$, $z^\circ(p_t, j_t) = 0$ and $y^\circ(t) = 0$ must violate the primal feasibility constraint for time t. Observe that this can only happen in the following two scenarios. In the first case, we see the page p_t for the first time and we increase $B(t)$ by one and therefore the RHS of constraint (1a) increases by one. Otherwise, it must be the case that $x^\circ(p_t, j_t - 1) = 1$ and the term $\sum_{p \in B(t) \setminus \{p_t\}} x^\circ(p, j(p,t))$ in the feasibility constraint reduces by one since we set $x^\circ(p_t, j_t) = 0$. In either of the cases, the page p_t was not in the cache and the cache already had k pages. Thus bringing in page p_t violates the cache size constraint. But observe that the algorithm finds a page p' in the cache and sets its $x^\circ(p', j(p', t))$ to one, hence the LHS increases by one again giving us feasibility.

Complementary Slackness. To observe that we maintain condition (2a), observe that we increase $z^\circ(p, j)$ only if $x^\circ(p, j) = 1$. Since we never decrease the x° for any variable, we maintain the complementary slackness till the end.

Now consider the condition (2b). Again consider any page p and request j. If $x^\circ(p, j)$ remains zero, there is nothing to prove. Suppose $x^\circ(p, j)$ is set to one say at time $\hat{t}$. Then at time $\hat{t}$, we must have

$$f'_{i(p)}\left(m(i(p), \hat{t} - 1) + 1\right) - \sum_{t=t(p,j)+1}^{\hat{t}} y_t^\circ + z^\circ(p, j) = 0$$

which implies that

$$f'_{i(p)}\left(m(i(p), \hat{t})\right) - \sum_{t=t(p,j)+1}^{\hat{t}} y_t^\circ + z^\circ(p, j) = 0$$

since $m(i(p), \hat{t} - 1) + 1 = m(i(p), \hat{t})$.

At any later time $t \in (\hat{t}, t(p, j+1))$, if we increase y_t we increase z° at the same rate maintaining equality. After time $t > t(p, j+1)$ none of the variables in the term change proving the lemma. Thus we obtain that

$$f'_{i(p)}\left(m(i(p), \hat{t})\right) - \sum_{t=t(p,j)+1}^{t(p,j+1)-1} y_t^\circ + z^\circ(p, j) = 0,$$

as required.

Gradient Conditions. We show condition (3a). Consider any page p and its j^{th} request. Suppose the variable $x^\circ(p, j)$ is set to one at $t \in (t(p, j), t(p, j+1))$. Then the primal and dual solution at that time must satisfy

$$f'_{i(p)}\left(m(i(p), t - 1) + 1\right) - \sum_{t=t(p,j)+1}^{t(p,j+1)-1} y_t^\circ + z^\circ(p, j) = 0$$

Now consider how the LHS changes as the algorithm proceeds, if we increase y_t° for any $\hat{t} < t < t(p, j+1)$, then we also increase $z^\circ(p, j)$ at the same rate and we maintain equality. Now, we must have

$$m(i(p), T) \geq m(i(p), t - 1) + 1$$

since the LHS at least counts the eviction of page p at time t and is not counted in RHS. Using the fact that f' is an increasing function, we obtain that

$$f'_{i(p)}\left(m(i(p), T) - \sum_{t=t(p,j)+1}^{t(p,j+1)-1} y_t^\circ + z^\circ(p, j) \geq 0.\right.$$

Now, consider the case when page p is not evicted between its j^{th} and $(j+1)^{th}$ request. Thus $x^\circ(p, j)$ remains zero till the end. Firstly, this implies that page j is not the last request since we evict every page in the last request. Then, consider the expression

$$f'_{i(p)}\left(m(i(p), t') + 1\right) - \sum_{t=t(p,j)+1}^{t(p,j+1)-1} y_t^\circ + z^\circ(p, j)$$

evaluated with values of primal and dual variables at time $t' = t(p, j)$. Since only the first term is non-zero, the expression is non-negative. As we proceed with the algorithm and increase t' from $t(p, j)$ to $t(p, j+1) - 1$, the expression will never go below zero, otherwise, we would have set $x^\circ(p, j)$ to one. Since there is another request of page p where it is evicted (in particular the last one), we have $m(i(p), T) \geq m(i(p), t(p, j+1) - 1) + 1$ and therefore

$$f'_{i(p)}\left(m(i(p), T)\right) - \sum_{t=t(p,j)+1}^{t(p,j+1)-1} y_t^\circ + z^\circ(p, j) \geq 0$$

and we obtain feasibility of condition (3a).

□

We now prove the following lemma which will complete the proof of Theorem 1.1.

LEMMA 2.2. *Let $x^\star$ be an optimal solution to the convex program and x° denote any solution satisfying the invariant conditions. Then we must have*

$$\sum_{i \in U} f_i\left(m(i, T)\right) \leq \sum_{i \in U} f_i\left(\alpha k \bar{m}(i, T)\right))$$

where $\alpha = \sup_{x,i} \frac{x f'_i(x)}{f(x)}$ and $\bar{m}(i, T)$ is the number misses for user i by the optimal solution $x^\star$.

PROOF. The following auxiliary claim will be required in the sequel. This claim will help us bridge the gap introduced by violation of the complementary slackness conditions by the algorithm.

CLAIM 2.3. *Let f be a convex increasing function with $f(0) = 0$. Then*

$$f'(\sum_{j=1}^{n} x_j) \sum_{j=1}^{n} x_j \leq \alpha \sum_{j=1}^{n} x_j f'(\sum_{i=1}^{j} x_i), \tag{4}$$

where

$$\alpha = \max_x \frac{f'(x)x}{f(x)}. \tag{5}$$

As special case, note that when f is a polynomial with positive coefficients and degree β, we have $\alpha = \beta$.

PROOF. Note that it suffices to show that

$$\sum_{j=1}^{n} x_j f'(\sum_{i=1}^{j} x_i) \geq f(\sum_{j=1}^{n} x_j). \tag{6}$$

Indeed if (6) holds, then we have that the RHS of (4) satisfies

$$RHS \geq \alpha f(\sum_{j=1}^{n} x_j) \geq \frac{\sum_{j=1}^{n} x_j f'(\sum_{j=1}^{n} x_j)}{f(\sum_{j=1}^{n} x_j)} f(\sum_{j=1}^{n} x_j) = LHS.$$

To prove (6), we use the first order condition for convex functions, which implies the following set of inequalities: $f(0) - f(x_1) \geq -x_1 f'(x_1)$, $f(x_1) - f(x_1 + x_2) \geq -x_2 f'(x_1 + x_2)$, $\ldots$,$f(x_1 + \ldots x_{n-1}) - f(\sum_{j=1}^{n} x_j) \geq -x_n f(\sum_{j=1}^{n} x_j)$. Summing this set and recalling that $f(0) = 0$ immediately yields (6). $\square$

By definition, we have $m(i,T) = \sum_{p \in P_i} \sum_{j=1}^{r(p,T)} x^\circ(p,j)$ and $\bar{m}(i,T) = \sum_{p \in P_i} \sum_{j=1}^{r(p,T)} x^\star(p,j)$. For any convex function f, we have $f(y) - f(x) \geq f'(x)(y - x)$ and therefore we obtain that

$$\begin{aligned}
&\sum_{i \in U} f_i\left(\alpha k \bar{m}(i,T)\right) - \sum_{i \in U} f_i\left(m(i,T)\right) \\
=&\sum_{i \in U} f_i\left(\alpha k \sum_{p \in P_i} \sum_{j=1}^{r(p,T)} x^\star(p,j)\right) - \sum_{i \in U} f_i\left(\sum_{p \in P_i} \sum_{j=1}^{r(p,T)} x^\circ(p,j)\right) \\
\geq&\sum_{i \in U} f_i'\left(m(i,T)\right) \cdot \left(\alpha k \bar{m}(i,T) - m(i,T)\right) \\
=&\sum_{i \in U} \sum_{p \in P_i} \sum_{j=1}^{r(p,T)} f_i'\left(m(i,T)\right) \cdot \left(\alpha k \cdot x^\star(p,j) - x^\circ(p,j)\right) \\
=&\sum_{p \in P} \sum_{j=1}^{r(p,T)} \alpha k \cdot x^\star(p,j) \cdot f_{i(p)}'\left(m(i(p),T)\right) - \\
&\qquad \sum_{i \in U} \sum_{p \in P_i} \sum_{j=1}^{r(p,T)} x^\circ(p,j) \cdot f_i'\left(\sum_{p' \in P_i} \sum_{j=1}^{r(p,T)} x^\circ(p',j)\right)
\end{aligned}$$

We now have the following claim.

CLAIM 2.4. *For any i,*

$$\begin{aligned}
&\sum_{p \in P_i} \sum_{j=1}^{r(p.T)} x^\circ(p,j) \cdot f_i'\left(\sum_{p' \in P_i} \sum_{j=1}^{r(p',T)} x^\circ(p',j)\right) \\
&\leq \alpha \sum_{p \in P_i} \sum_{j=1}^{r(p,T)} x^\circ(p,j) \cdot f_i'\left(m(i,s(p,j))\right)
\end{aligned}$$

where $s(p,j)$ is the time at which $x^\circ(p,j)$ is set to 1 and T if $x^\circ(p,j)$ remains 0.

PROOF. Order the variables $x^\circ(p,j)$ for all $p \in P_i$ and all j in increasing order of $s(p,j)$ and apply Claim 2.3 on this order. Observe that

$$\sum_{(p',j'):s(p,j) \geq s(p',j')} x(p',j') = m(i,s(p,j)).$$

Thus we obtain the inequality. $\square$

Thus we obtain that

$$\begin{aligned}
&\sum_{i \in U} f_i\left(\alpha k \bar{m}(i,T)\right) - \sum_{i \in U} f_i\left(m(i,T)\right) \\
&\geq \sum_{p \in P} \sum_{j=1}^{r(p,T)} \alpha k \cdot x^\star(p,j) \cdot f_{i(p)}'\left(m(i(p),T)\right) \\
&- \sum_{i \in U} \alpha \sum_{p \in P_i} \sum_{j=1}^{r(p,T)} x^\circ(p,j) \cdot f_i'\left(m(i,s(p,j))\right) \\
&\geq \sum_{p \in P} \sum_{j=1}^{r(p,T)} \alpha k \cdot x^\star(p,j) \cdot \left(\sum_{t=t(p,j+1)}^{t=t(p,j+1)-1} y_t^\circ - z^\circ(p,j)\right) \\
&- \sum_{i \in U} \alpha \sum_{p \in P_i} \sum_{j=1}^{r(p,T)} x^\circ(p,j) \cdot \left(\sum_{t=t(p,j)+1}^{t(p,j+1)-1} y_t^\circ - z^\circ(p,j)\right)
\end{aligned}$$

where we have used condition (3a) in the first term and condition (2b) in the second term.

Thus we have

$$\begin{aligned}
&\sum_{i \in U} f_i\left(\alpha k \bar{m}(i,T)\right) - \sum_{i \in U} f_i\left(m(i,T)\right) \\
\geq&\alpha \left(k \sum_{t=1} y_t^\circ \sum_{p \in B(t) \setminus p_t} x^\star(p, j(p,t)) - k \sum_{p,j} x^\star(p,j) z^\circ(p,j)\right) \\
&- \alpha \left(\sum_{t=1} y_t^\circ \sum_{p \in B(t) \setminus p_t} x^\circ(p, j(p,t)) - \sum_{p,j} x^\circ(p,j) z^\circ(p,j)\right)
\end{aligned}$$

We now claim that the RHS is non-negative. Fix $x^\star$ and x° to their final values. We now see how the RHS changes as the algorithm changes the dual solution (y°, z°). We show that every change only increases the RHS. Initially $y^\circ = 0$ and $z^\circ = 0$ and therefore the RHS is 0. Now consider any step of the algorithm where we increase y_t° by ϵ for some t. Simultaneously, we would have increased $z^\circ(p,j)$ by ϵ for all pages outside the cache except for page p_t, page requested at time t. Let Q be the of pages not in the cache at this time. Thus $z^\circ(p, j(p,t))$ increases for exactly $|B(t)| - k - 1$ pages, all pages in Q except p_t.

$$\begin{aligned}
&\Delta\left(\alpha\left(k \sum_{t=1} y_t^\circ \sum_{p \in B(t) \setminus p_t} x^\star(p, j(p,t)) - k \sum_{p,j} x^\star(p,j) z^\circ(p,j)\right)\right) \\
=&\alpha k \epsilon \left(\sum_{p \in B(t) \setminus p_t} x^\star(p, j(p,t)) - \sum_{Q \setminus p_t} x^\star(p, j(p,t))\right) \\
\geq&\alpha k \epsilon \left(|B(t)| - k - 1 \cdot (|Q| - 1)\right) \\
=&\alpha k \epsilon
\end{aligned}$$

Thus the first term increases by at least $\alpha k \epsilon$. Now consider the change in the second term.

$$\begin{aligned}
&\Delta\left(\alpha\left(\sum_{t=1} y_t^\circ \sum_{p \in B(t) \setminus p_t} x^\circ(p, j(p,t)) - \sum_{p,j} x^\circ(p,j) z^\circ(p,j)\right)\right) \\
=&\alpha \epsilon \left(\sum_{p \in B(t) \setminus p_t} x^\circ(p, j(p,t)) - \sum_{Q \setminus p_t} x^\circ(p, j(p,t))\right) \\
\leq&\alpha \epsilon \left(\sum_{p \in B(t) \setminus \{Q \cup \{p_t\}\}} 1\right) \\
=&\alpha k \epsilon
\end{aligned}$$

For every time t, let p_t be the required page at time t. Do

- If the cache is not full or page p_t is already in cache then bring in page p_t in cache and update
$$B(p_t) \leftarrow f'_{i(p_t)}\left(m(i(p_t), t-1)+1\right).$$
- Else
  - Let p be the page in the cache with smallest $B(p)$. Remove page p from the cache and bring in p_t.
  - Set $B(p_t) \leftarrow f'_{i(p_t)}\left(m(i(p_t), t-1)+1\right)$.
  - For each $p' \notin \{p, p_t\}$ in the cache, let $B(p') \leftarrow B(p') - B(p)$.
  - For each page p' in the cache such that $i(p') = i(p)$, set
$$B(p') \leftarrow B(p') + f'_{i(p')}\left(m(i(p'), t-1)+2\right) - f'_{i(p')}\left(m(i(p'), t-1)+1\right)$$

Figure 3: Algorithm ALG-DISCRETE

where we use the fact that $|B(t) \setminus \{Q \cup \{p_t\}\}| \leq k$ since $|Q| \geq |B(t)| - k - 1$. Thus we must have that

$$\sum_{i \in U} f_i\left(\alpha k \bar{m}(i,T)\right) - \sum_{i \in U} f_i\left(m(i,T)\right) \geq 0$$

proving the lemma and Theorem 1.1.

2.5 Implementation of Algorithm ALG-CONT

We now give an implementation of ALG-CONT that does discrete updates. The algorithm, termed ALG-DISCRETE is summarized in Figure 3. The algorithm maintains a budget $B(p)$ for each page p in the cache and updates them in each iteration. A simple check shows that the ALG-CONT will be the same algorithm by observing that y_t increases in iteration t by the current value of $B(p)$ when page p is evicted.

We note that while the guarantee in Theorem 1.1 relies on the assumption that each of the cost functions f_i are convex, the algorithm ALG-CONT or its discrete implementation ALG-DISCRETE do not require the convexity assumption, and can in fact be applied for arbitrary cost functions. In fact, the cost functions f_i need not even be continuous; the derivatives in the algorithms can be replaced by their discrete versions. We indeed demonstrate in [14] that variants of our algorithms perform well in settings where the assumptions of Theorem 1.1 do not necessarily hold.

3. BI-CRITERIA APPROXIMATION

In this section, we prove Theorem 1.3. As mentioned above, we again analyze ALG-CONT. To compare the performance of our algorithm with the offline algorithm with a smaller cache size of h, we consider the convex program where the cache size is h.

We prove the following lemma which generalizes Lemma 2.2 and will prove Theorem 1.3. Observe that $x^\star$ is an optimal solution to (CP-h) (cf. Figure 4) while x° is feasible for (CP) and might not feasible for (CP-h) due to stronger constraints for smaller cache size.

LEMMA 3.1. *Let $x^\star$ be an optimal solution to the convex program (CP-h) and x° denote any solution satisfying the invariant conditions. Then we must have*

$$\sum_{i \in U} f_i\left(m(i,T)\right) \leq \sum_{i \in U} f_i\left(\alpha \frac{k}{k-h+1} \bar{m}(i,T)\right)$$

where $\alpha = \sup_{x,i} \frac{x f'_i(x)}{f(x)}$ and $\bar{m}(i,T)$ is the number misses for user i by the optimal solution $x^\star$.

PROOF. The proof follows along the same lines as proof of Lemma 2.2 and we highlight the differences. First, following the same argument as in proof of Lemma 2.2, we obtain that

$$\begin{aligned}
&\sum_{i \in U} f_i\left(\alpha \frac{k}{k-h+1} \bar{m}(i,T)\right) - \sum_{i \in U} f_i\left(m(i,T)\right) \\
\geq &\sum_{i \in U} \sum_{p \in P_i} \sum_{j=1}^{r(p,T)} f'_i\left(m(i,T)\right) \cdot \left(\alpha \frac{k}{k-h+1} \cdot x^\star(p,j) - x^\circ(p,j)\right) \\
= &\sum_{p \in P} \sum_{j=1}^{r(p,T)} \alpha \frac{k}{k-h+1} \cdot x^\star(p,j) \cdot f'_{i(p)}\left(m(i(p),T)\right) - \\
&\sum_{i \in U} \sum_{p \in P_i} \sum_{j=1}^{r(p,T)} x^\circ(p,j) \cdot f'_i\left(\sum_{p' \in P_i} \sum_{j=1}^{r(p,T)} x^\circ(p',j)\right)
\end{aligned}$$

where we have again used that a convex function f satisfies $f(y) - f(x) \geq f'(x)(y-x)$ for all x, y. Now applying Claim 2.3, we obtain that

$$\begin{aligned}
&\sum_{i \in U} f_i\left(\alpha \frac{k}{k-h+1} \bar{m}(i,T)\right) - \sum_{i \in U} f_i\left(m(i,T)\right) \\
\geq &\sum_{p \in P} \sum_{j=1}^{r(p,T)} \alpha \frac{k}{k-h+1} \cdot x^\star(p,j) \cdot f'_{i(p)}\left(m(i(p),T)\right) - \\
&\sum_{i \in U} \alpha \sum_{p \in P_i} \sum_{j=1}^{r(p,T)} x^\circ(p,j) \cdot f'_i\left(m(i, s(p,j))\right) \\
\geq &\sum_{p \in P} \sum_{j=1}^{r(p,T)} \alpha \frac{k}{k-h+1} \cdot x^\star(p,j) \cdot \left(\sum_{t=t(p,j)+1}^{t=t(p,j+1)-1} y^\circ_t - z^\circ(p,j)\right) \\
&- \sum_{i \in U} \alpha \sum_{p \in P_i} \sum_{j=1}^{r(p,T)} x^\circ(p,j) \cdot \left(\sum_{t=t(p,j)+1}^{t(p,j+1)-1} y^\circ_t - z^\circ(p,j)\right)
\end{aligned}$$

where we have used condition (3a) in the first term and condition (2b) in the second term. Rearranging, we obtain that

$$\begin{aligned}
&\sum_{i \in U} f_i\left(\alpha \frac{k}{k-h+1} \bar{m}(i,T)\right) - \sum_{i \in U} f_i\left(m(i,T)\right) \\
\geq &\alpha \left(\frac{k}{k-h+1} \sum_{t=1} y^\circ_t \sum_{p \in B(t) \setminus p_t} x^\star(p, j(p,t))\right) \\
&- \left(\alpha \frac{k}{k-h+1} \sum_{p,j} x^\star(p,j) z^\circ(p,j)\right) \\
&- \alpha \left(\sum_{t=1} y^\circ_t \sum_{p \in B(t) \setminus p_t} x^\circ(p, j(p,t)) - \sum_{p,j} x^\circ(p,j) z^\circ(p,j)\right)
\end{aligned}$$

We now claim that the RHS is non-negative. Fix $x^\star$ and x° to their final values. We now see how the RHS changes as the algorithm changes the dual solution (y°, z°). We show that every

(ICP-h)	$\min \sum_{i=1}^{m} f_i \left(\sum_{p \in P_i} \sum_{j=1}^{r(p,T)} x(p,j) \right)$	(CP-h)	$\min \sum_{i=1}^{m} f_i \left(\sum_{p \in P_i} \sum_{j=1}^{r(p,T)} x(p,j) \right)$
s.t.		s.t.	
	$\forall 1 \le t \le T \quad \sum_{p \in B(t) \setminus \{p_t\}} x(p, j(p,t)) \ge \lvert B(t) \rvert - h$		$\forall 1 \le t \le T \quad \sum_{p \in B(t) \setminus \{p_t\}} x(p, j(p,t)) \ge \lvert B(t) \rvert - h$
	$\forall p \in P, \forall j \quad x(p,j) \in \{0,1\}$		$\forall p \in P, \forall j \quad x(p,j) \le 1$
			$\forall p \in P, \forall j \quad x(p,j) \ge 0$

Figure 4: Integer Convex Program and its Convex Programming Relaxation for Cache Size h

change only increases the RHS. Initially $y^\circ = 0$ and $z^\circ = 0$ and therefore the RHS is 0. Now consider any step of the algorithm where we increase y_t° by ϵ for some t. Simultaneously, we would have increased $z^\circ(p,j)$ by ϵ for all pages outside the cache except for page p_t, page requested at time t. Let Q be the of pages not in the cache at this time. Thus $z^\circ(p, j(p,t))$ increases for exactly $|B(t)| - k - 1$ pages, all pages in Q except p_t.

$$
\begin{aligned}
&\Delta\left(\alpha\left(\frac{k}{k-h+1}\sum_{t=1} y_t^\circ \sum_{p \in B(t)\setminus p_t} x^\star(p, j(p,t))\right)\right) \\
&\quad - \Delta\left(\left(\frac{k}{k-h+1}\sum_{p,j} x^\star(p,j) z^\circ(p,j)\right)\right) \\
=&\alpha \frac{k}{k-h+1}\epsilon\left(\sum_{p \in B(t)\setminus p_t} x^\star(p, j(p,t)) - \sum_{Q \setminus p_t} x^\star(p, j(p,t))\right) \\
\ge&\alpha \frac{k}{k-h+1}\epsilon\left(|B(t)| - h - 1 \cdot (|Q| - 1)\right) \\
=&\alpha \frac{k}{k-h+1}\epsilon \cdot (k-h+1) \\
=&\alpha k \epsilon
\end{aligned}
$$

where we have the used the fact $x^\star$ satisfies the stronger constraints $\sum_{p \in B(t)\setminus p_t} x^\star(p, j(p,t)) \ge |B(t)| - h$.

Thus the first term increases by at least $\alpha k \epsilon$. Now consider the change in the second term.

$$
\begin{aligned}
&\Delta\left(\alpha\left(\sum_{t=1} y_t^\circ \sum_{p \in B(t)\setminus p_t} x^\circ(p, j(p,t)) - \sum_{p,j} x^\circ(p,j) z^\circ(p,j)\right)\right) \\
=&\alpha\epsilon\left(\sum_{p \in B(t)\setminus p_t} x^\circ(p, j(p,t)) - \sum_{Q \setminus p_t} x^\circ(p, j(p,t))\right) \\
\le&\alpha\epsilon\left(\sum_{p \in B(t)\setminus\{Q \cup \{p_t\}\}} 1\right) \\
=&\alpha k \epsilon
\end{aligned}
$$

where we use the fact that $|B(t) \setminus \{Q \cup \{p_t\}\}| \le k$ since $|Q| \ge |B(t)| - k - 1$. Thus we must have that

$$\sum_{i \in U} f_i\left(\alpha \frac{k}{k-h+1}\bar{m}(i,T)\right) - \sum_{i \in U} f_i\left(m(i,T)\right) \ge 0$$

proving Lemma 3.1 and Theorem 1.3. □

4. LOWER BOUND

In this section, we prove Theorem 1.4 by giving a worst case instance. For any n and β, we construct an instance with n users, each with cost function given by $f_i(x) = x^\beta$. Each user will own a single page and the cache size will be $n-1$. Let $\mathbb{A}$ be any algorithm. We now describe the input sequence σ which will depend on the algorithm. At any time $t \ge n-1$, the cache of the algorithm will contain exactly $n-1$ pages and therefore one of the page out of n must be missing. In sequence σ, we request exactly this missing page. Observe that this implies that the algorithm must evict a page on each request except for the first $n-1$ requests. We run this sequence for large time T and ignore the error due to the first $n-1$ page requests. Let there be r_i requests for page owned by user i. Then we have $\sum_{i=1}^{n} r_i \ge T$ and the cost of the algorithm $\mathbb{A}$ is at least $\sum_{i=1}^{n} r_i^\beta$. Now we show that the optimal solution is much smaller by giving an offline algorithm that costs much less. Of course, the cost of the optimal solution must be smaller than the cost achieved by this offline algorithm. First we divide the sequence of requests in batches of length $\frac{n-1}{2}$. At the start of the each batch, there are $n-1$ pages in the cache. We choose one page to evict making sure that it is not one of the pages in the next $\frac{n-1}{2}$ page requests. This ensures that we do not have any other eviction in the next $\frac{n-1}{2}$ page requests. Observe that there are $\frac{n+1}{2}$ different choices for which page to evict. We choose the one which has had fewest number of evictions so far. We now make two observations. First that the number of total evictions is no more than $\frac{T}{\frac{n-1}{2}}$ since we make at most one eviction per batch. Second observation is that the maximum eviction for any page is bounded by $\frac{1}{\frac{n+1}{2}}\frac{T}{\frac{n-1}{2}} + 1$. The last bound follows since there must be at least $\frac{n+1}{2}$ other pages which have nearly the same number, up to an additive factor of one, as the page with maximum number of evictions due to the rule for choosing the evictions. Thus the cost of the algorithm is bounded by $(\frac{4T}{n^2})^\beta n$. While the cost of the algorithm $\mathbb{A}$ is at least $\sum_{i=1}^{n} r_i^\beta \ge (\frac{T}{n})^\beta n$ where the last inequality follows since $\beta \ge 1$ and the sum $\sum_{i=1}^{n} r_i^\beta$ is minimized when each of r_i is equal to $\frac{T}{n}$. Thus we obtain that cost of the algorithm $\mathbb{A}$ is at least $(\frac{n}{4})^\beta$ times the cost of the optimal solution on request σ proving Theorem 1.4 since we have $k = n-1$.

5. CONCLUSION

Multi-tenancy poses substantial challenges for modern compute systems, and sharing physical memory stands out as an important problem in this space. Modeling the user performance as a non-linear (cost) function of the total number of memory misses allows to capture practical considerations of the provider, and automate memory allocation via cost-aware algorithms. In this paper, we design algorithms to solve the respective online optimization problem. We provide performance guarantees under convexity assumptions – we prove that our algorithm is $\alpha^\alpha k^\alpha$-competitive, where k is the memory (cache) size, and α is a constant which depends on the curvature of the cost functions. We also obtain a bi-criteria result which trades the performance to the memory size, and give a nearly matching lower bound on performance. Based on the algorithm developed in this paper, we have recently designed and pro-

totyped a memory replacement mechanism for SQLVM, a multi-tenant DaaS system, see [14].

In this paper, we assume that a single pool of memory has to be shared between tenants. An interesting direction for future work is to consider the case of multiple memory pools (e.g., each pool corresponds to a single physical server), where each user has to be assigned to a single pool, with potentially switching cost incurred for migrating users between servers.

Acknowledgements

We thank the reviewers for their thoughtful comments. We also thank Vivek Narasayya for helpful discussions.

6. REFERENCES

[1] S. Albers. *Competitive online algorithms*. Lecture Notes, Aarhus University, 1996.

[2] Amazon ElastiCache. `http://aws.amazon.com/elasticache/`.

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

[4] L. A. Belady. A study of replacement algorithms for a virtual-storage computer. *IBM Systems journal*, 5(2):78–101, 1966.

[5] M. A. Bender, R. Ebrahimi, J. T. Fineman, G. Ghasemiesfeh, R. Johnson, and S. McCauley. Cache-adaptive algorithms. In *SODA*, pages 958–971. SIAM, 2014.

[6] A. Borodin and R. El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.

[7] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[8] N. Buchbinder and J. Naor. The design of competitive online algorithms via a primal: dual approach. *Foundations and Trends® in Theoretical Computer Science*, 3(2–3):93–263, 2009.

[9] F. J. Corbato. A paging experiment with the Multics system. Technical report, DTIC Document, 1968.

[10] B. Fitzpatrick. Distributed caching with memcached. *Linux journal*, 2004(124):5, 2004.

[11] A. Hassidim. Cache replacement policies for multicore processors. In *ICS*, pages 501–509, 2010.

[12] Microsoft Azure Cache. `http://azure.microsoft.com/en-us/services/cache/`.

[13] Microsoft Azure SQL Database (formerly SQL Azure). `http://www.windowsazure.com/en-us/services/sql-database/`.

[14] V. Narasayya, I. Menache, M. Singh, F. Li, M. Syamala, and S. Chaudhuri. Sharing Buffer Pool Memory in Multi-Tenant Relational Database-as-a-Service. *Proceedings of the VLDB Endowment*, 8(7), 2015. Available from `http://www.vldb.org/pvldb/vol8/p726-narasayya.pdf`.

[15] V. R. Narasayya, S. Das, M. Syamala, B. Chandramouli, and S. Chaudhuri. SQLVM: Performance Isolation in Multi-Tenant Relational Database-as-a-Service. In *CIDR*, 2013.

[16] E. J. O'Neil, P. E. O'Neil, and G. Weikum. The LRU-K page replacement algorithm for database disk buffering. In *ACM SIGMOD Record*, volume 22, pages 297–306. ACM, 1993.

[17] E. J. O'Neil, P. E. O'Neil, and G. Weikum. An optimality proof of the LRU-K page replacement algorithm. *Journal of the ACM (JACM)*, 46(1):92–112, 1999.

[18] E. Peserico. Elastic paging. In *Proceedings of the ACM SIGMETRICS/international conference on Measurement and modeling of computer systems*, pages 349–350. ACM, 2013.

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

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

[21] W. Zhang and P.-A. Larson. Dynamic memory adjustment for external mergesort. In *VLDB*, volume 97, pages 25–29, 1997.