A caching system with object sharing

George Kesidis, Bhurvan Urgaonkar, Mahmut Kandemir
School of EECS
Pennsylvania State University
University Park, PA, 16802, USA
Email: {gik2,buu1,mtk2}@psu.edu

Takis Konstantopoulos
Dept. of Mathematical Sciences
University of Liverpool
Liverpool, L69 7ZL, UK
t.konstantopoulos@liverpool.ac.uk

Abstract

We consider a public content caching system that is shared by a number of proxies. The cache could be located in an edge-cloud datacenter and the the proxies could each serve a large population of mobile end-users. The proxies operate their own LRU-list of a certain capacity in the shared cache. The length of objects appearing in plural LRU-lists is divided among them. We provide a "working set" approximation to quickly approximate the cache-hit probabilities under object sharing. We also discuss an approach to sharing cache I/O based on token bucket mechanisms. Also, why and how a proxy may issue mock requests to exploit the shared cache is discussed.

I. Introduction

We herein consider J proxies that each service a large pool of users/processes making requests for content from a database with N data-objects via a cache of size $B \ll N$, *e.g.*, caching as part of a Content Distribution Network (CDN). Each proxy typically operates under a Least Recently Used (LRU) caching policy wherein the most recently queried for data-objects are cached. The J proxies *share* both cache memory and possibly also upload network-bandwidth to the users. For caching of encrypted data (*e.g.*, owing to copyright protections), a layered encryption strategy (as in block chains, legers) could be used to first encrypt to the network edge and then encrypt to the individual (authorized) users.

There is substantial prior work on cache sharing, including at the network edge in support of mobile end-users, *e.g.*, [14], [25], [21]. At one extreme, the queries of the proxies are aggregated and one LRU cache is maintained for all of them using the entire cache memory. At another extreme, the cache memory is statically partitioned among the proxies (without object sharing), cf. Section IV-C. For example, [9]

This research supported in part by a Cisco Systems URP gift and NSF CNS grant 1526133 and NSF CSR grant 1717571.

describes how cache memory can be partitioned according to a game wherein different proxy utilities increase with cache-hit probability. For a system with a single LRU in the cache, a lower priority (paying less) proxy could have a different tail (least recent object) pointer corresponding to lower amount of allocated memory, but different proxies would then compete for "hot" (higher ranked) objects stored in the cache. To reduce such competition, an interesting system of [11] also has a single LRU maintained in the cache but with highest priority (paying most) proxies having access to the entire cache while lower priority proxies having a *head* (most recently used object) pointer corresponding to a lower amount of allocated memory.

Now consider a scenario where different proxies may query for the same object. In [22], proxies are assigned a share of cached content based on their demand. Individual data objects are *shared* among different proxy caches that store them, each according to the LRU policy, *i.e.*, a share of their length is attributed to each proxy's cache (LRU list). In [22], the cache blocks some requests selected at random to deter a proxy from "cheating" by issuing mock requests for specific content primarily of interest only to its users in order to keep it cached (hot), while leveraging cached content apportioned to other proxies, *i.e.*, more generally popular content.

In this paper, we consider a caching system where each proxy i somehow pays for or is allocated cache memory (and possibly network I/O as well), thus preventing starvation of any proxy. Objects may be shared among different LRU-lists (each corresponding to a proxy) and a LRU-list miss but physical cache hit is accompanied by a delay corresponding to a physical cache miss. This said, a proxy may make inferences regarding the LRU-lists of others by comparing the cache hits they experience to what they would be without object sharing. Mock queries may change some near-future LRU-list misses to hits (particularly for content not in the physical cache), but will come at the cost of both memory and network I/O resources (possibly *causing* some near-future cache misses that would have been hits).

This paper is organized as follows. In Section II, we give some background on (not-shared) caches for variable-length objects. In Section III, an approach to cache memory management wherein a cached object's length is *shared* among multiple LRU-lists. In Section IV, we propose an approach to approximating hitting times for such a system of shared cache memory under the Independent Reference Model (IRM) model. In Section VI, we describe an approach to sharing cache I/O (specifically network bandwidth from the cache to its users) based on token-bucket mechanisms. In Section VII, we briefly discuss pricing issues and deterring proxies from issuing mock requests. A numerical study is given in Section V.

II. Background on a (not shared) cache for variable-length objects under the IRM

Assume that the aggregate demand process for object $n \in \{1, 2, \ldots, N\} =: [N]$ is Poisson with intensity λ_n. The Poisson demands are assumed independent. Let the total demand intensity be $\Lambda = \sum_{n=1}^{N} \lambda_n$. So, this is the classical IRM with query probabilities $p_n = \lambda_n/\Lambda$ [1], [5]. First suppose all objects are of unit length and that the cache has capacity B. The set $\mathcal{R}$ of B-permutations of $[N]$ (think of $\mathcal{R}$ as the set of injective functions $r = (r(1), \ldots, r(B))$ from $[B]$ into $[N]$) is the state-space of an LRU Markov chain. The stationary invariant distribution of this Markov chain was found by W.F. King [19]. Let π be this invariant distribution for $B = N$. In fact, π turns out to be an invariant distribution for more general interarrival distributions so long as the object-querying decisions are independent (*i.e.*, in the language of queueing theory, this is a kind of insensitivity result).

Now suppose the cache capacity is $B \ll \sum_{n=1}^{N} \ell_n$, where object n has length ℓ_n bytes (or in terms of some other common unit of length), and $\mathcal{R}$ is the set of N-permutations of $[N]$. In state $r \in \mathcal{R}$, the number of objects in the cache is given by

$$K(r) = \max \left\{ K \in [N] : \sum_{k=1}^{K} \ell_{r(k)} \leq B \right\}. \tag{1}$$

For $K \leq N$, define the K-vector $r^{(K)} = \{r(1), r(2), \ldots, r(K)\}$, *i.e.*, the N-vector r truncated to its first K elements, so a K-permutation of $[N]$. Thus, the stationary probability that the *cache occupancy* equals $r^{(K(r))}$ is

$$\sum_{\rho \in \mathcal{R}:\, \rho^{(K(r))} = r^{(K(r))}} \pi(\rho)$$

By the PASTA (Poisson Arrivals See Time Averages) theorem [26], [2], the hitting probability of object n when the objects are of variable length is

$$h_n = \sum_{r:\ r(n) \leq K(r)} \pi(r). \tag{2}$$

See the byte-hit performance metric of [4]. Because this computation is very complex, a "working-set approximation" (given below) was developed in [10].

III. Partitioning cache memory

Suppose cache memory is "virtually" allocated so that proxy i receives $b_i \leq B$ and

$$\sum_i b_i \leq B. \tag{3}$$

Each partition is managed simply by a LRU linked-list of pointers ("LRU-list" in the following) to objects stored in (physical) cache memory collectively for all the proxies.

Let $\mathcal{P}(n)$ be the set of proxies for which object n currently appears in their LRU-list, where $\mathcal{P}(n) = \emptyset$ if and only if object n is not *physically* cached. Note that $\mathcal{P}(n)$ is not disclosed to the proxies, *i.e.*, the proxies cannot with certainty tell whether objects *not* in their LRU-list are in the cache, *c.f.* Section VII.

Upon request by proxy i for object n of length ℓ_n, object n will be placed at the head of i's LRU list and all other objects in LRU-list i are correspondingly demoted in rank.

If the request for object n was a hit on LRU-list i, then nothing further is done.

If it was a miss on LRU-list i, then

- if the object is not stored in the physical cache then it is fetched from the database, stored in the cache and forwarded to proxy i;
- otherwise, the object is produced for proxy i after an equivalent delay.

Furthermore, i to $\mathcal{P}(n)$ (as in [22]), *i.e.*,

$$\mathcal{P}(n) \quad \leftarrow \quad \mathcal{P}(n) \cup \{i\}, \tag{4}$$

then add the length $\ell_n/|\mathcal{P}(n)|$ to LRU-list i and reduce the the "share" of all other caches containing n to $\ell_n/|\mathcal{P}(n)|$ (from $\ell_n/(|\mathcal{P}(n)| - 1)$).

So, if the query for object n by proxy i is a miss, its LRU-list length will be inflated and possibly exceed its allocation b_i; thus, LRU-list eviction of its tail (least recently used) object may be required. When an object m is "LRU-list evicted" by any proxy, the apportionment of ℓ_m to other LRU-lists is *increased* (inflated), which may cause other objects to be LRU-list evicted by other proxies. A simple mechanism that the cache operator could use is to evict until no LRU-list exceeds its allocated memory is to iteratively:

1) identify the LRU-list will largest overflow (length minus allocation)
2) if this largest overflow is not positive then stop
3) evict its lowest-rank object
4) reassess the lengths of all caches
5) go to 1.

This is guaranteed to terminate after a finite number of iterations because in every iteration, one object is evicted from an LRU-list and there are obviously only ever a finite number of objects per LRU-list.

Note that if during the eviction iterations, $\mathcal{P}(j) \to \emptyset$ for some object j, then j may be removed from the physical cache (physically evicted) – cached objects j in the physical cache that are not in any LRU-lists are flagged as such and have lowest priority (are first evicted if there is not sufficient room for any object that is/becomes a member of any LRU-list). Even under LRU-list eviction consensus, the physical cache

may store an object *if it has room* to try to avoid having to fetch it again from the database in the future. Section VII discusses artificially delaying a response to an LRU-list miss that this a physical cache hit.

In summary, a single proxy i can cause a new object n to enter the cache ($\mathcal{P}(n)$ changes from $\emptyset$ to $\{i\}$) whose entire length ℓ_n is applied to its cache memory allocation b_i, but a consensus is required for an object n to leave the cache ($\mathcal{P}(n) \to \emptyset$). So, the physical cache itself is not LRU. Also, as objects are requested, their apportionments to proxy LRU-lists may deflate and inflate.

Proposition 3.1: For a fixed set of active proxies i over an interval of time t, this object-sharing caching system will have a higher stationary object hit-rate per proxy compared to a *not-shared* system of LRU caches, where each proxy i's LRU cache is of size b_i in both cases.

The proof follows by a simple coupling argument to show that for each proxy, the objects in the not-shared system's cache are always a subset of what's in the LRU-list of the shared system. This follows simply because the size of object n apportioned to the shared system $\ell_n/|\mathcal{P}(n)| \leq \ell_n$, *i.e.*, $\leq$ its full size which is apportioned in the not-shared system.

IV. Approximating LRU-list hit probabilities under the IRM

A. Working-set approximations for shared-object caches under IRM

In this section, we propose an approach to computing the approximate hitting probabilities of the foregoing caching system following the Denning-Schwartz "working-set approximation" [10] for a not-shared cache under the IRM. Note that [12], [13] address the asymptotic accuracy of this approximation. Also see [8].

Let $\lambda_{i,k}$ be the mean request rate for object k, of length ℓ_k, by proxy i. A simple generalization of the working-set approximation for variable-length objects is: if $\min_i b_i \gg \max_k \ell_k$ then

$$\forall i \quad b_i = \sum_{k=1}^{N} h_{i,k}\ell_k \tag{5}$$

where

$$\forall i,k \quad h_{i,k} = 1 - \mathrm{e}^{-\lambda_{i,k}t_i} \tag{6}$$

and t_i are interpreted as (assumed common) mean eviction times of objects k in LRU-list i, *i.e.*, the time between when an object enters the cache and when it's evicted from the cache.

For our shared caching system, only a fraction of an object k's length ℓ_k will be attributed to a particular LRU-list i, depending on how k is shared over (eviction) time t_i. For all i,k, let this attribution be $L_{i,k} \leq \ell_k$, *i.e.*,

$$\forall i, \; b_i \;=\; \sum_{k=1}^{N} h_{i,k}L_{i,k} \;=\; \sum_{k=1}^{N}(1 - \mathrm{e}^{-\lambda_{i,k}t_i})L_{i,k}. \tag{7}$$

One may take

$$L_{i,k}^{(1)} \;=\; \ell_k \mathbb{E}\frac{1}{1+\sum_{j\neq i} Z_{j,k}}, \tag{8}$$

where $Z_{j,k}$ are *independent* Bernoulli random variables such that $h_{j,k} = \mathbb{P}(Z_{j,k}=1) = 1-\mathbb{P}(Z_{j,k}=0)$. That is, under the assumption of independent LRU-lists, $L_{i,k}^{(1)}$ is the stationary mean attribution of the length of object k to LRU-list i given that k is stored in LRU-list i. For example, for a system with just $J=2$ caches, *i.e.*, $j \in \{1,2\}$,

$$\begin{aligned}\mathbb{E}\frac{1}{1+\sum_{j\neq i} Z_{j,k}} &= 1\cdot(1-h_{3-j,k}) + \frac{1}{2}h_{3-j,k}\\ &= 1-\frac{1}{2}h_{3-j,k}.\end{aligned}$$

So, substituting (8) into (7) gives, for $i \in \{1,2\}$,

$$0 = b_i - \sum_{k=1}^{N}(1-\mathrm{e}^{-\lambda_{i,k}t_i})(1-\frac{1}{2}(1-\mathrm{e}^{-\lambda_{3-i,k}t_{3-i}}))\ell_k; \tag{9}$$

a system with two nonlinear equations in two unknowns t_1, t_2.

Empirically, we found that using (8) under-estimates the object hitting probabilities, *i.e.*, $L_{i,k}^{(1)}$ is too large, significantly when $J=2$. To explain this, we argue that object sharing creates a kind of *positive association* between the LRU-list hit events, because hits in one cause the objects to effectively reduce in size in others, so that they remain in the LRU-lists longer (larger eviction times), thus increasing the hit probabilities in others.

To see why, consider Prop. 4.1 below for Boolean random variables $Y_{j,k}$ indicating the *dependent* events that object k is stored in LRU-list j in steady-state.

Lemma 4.1: F_1, F_2 be two cumulative distribution functions (CDF) on $\mathbb{R}$ such that $F_1(x) \leq F_2(x)$ for all $x \in \mathbb{R}$. Let g be a decreasing (that is, non-increasing) function. Then

$$\int_{\mathbb{R}} g(x)\mathrm{d}F_1(x) \leq \int_{\mathbb{R}} g(x)\mathrm{d}F_2(x).$$

Proof: Let $F_i^{-1}(u) := \inf\{x \in \mathbb{R} :\ F_i(x) > u\}$, $i=1,2$. By change of variables in Lebesgue-Stieltjes integrals, we have

$$\int_{\mathbb{R}} g(x)\mathrm{d}F_i(x) = \int_0^1 g(F_i^{-1}(u))\mathrm{d}u.$$

Since $F_1 \leq F_2$ we have $F_1^{-1} \geq F_2^{-1}$ and so $g(F_1^{-1}(u)) \leq g(F_2^{-1}(u))$ for all $0<u<1$. ■

Proposition 4.1: For an arbitrary object index k, consider $J \geq 2$ nonnegative random variables $Y_{1,k}, \ldots, Y_{J,k}$ and J *independent* random variables $Z_{1,k}, \ldots, Z_{J,k}$ such that, for all i, $Z_{i,k}$ and $Y_{i,k}$ have the same distribution. If for any LRU-list $i \in \{1, 2, \ldots, J\}$ we have

$$\mathbb{P}\left(\sum_{j\neq i} Y_{j,k} \leq x\right) \leq \mathbb{P}\left(\sum_{j\neq i} Z_{j,k} \leq x\right) \tag{10}$$

then

$$\mathbb{E}\left(1+\sum_{j\neq i} Y_{j,k}\right)^{-1} \;\leq\; \mathbb{E}\left(1+\sum_{j\neq i} Z_{j,k}\right)^{-1}. \tag{11}$$

Proof: Simply take $g(x) = 1/(1+x)$ in Lemma 4.1 . ■

Note that according to (10), $\sum_{j\neq i} Y_j$ tends to be larger than $\sum_{j\neq i} Z_j$, similar to positive associations or positive correlations properties among random variables $Y_i \geq 0$ [18], [17], [24].

By Jensen's inequality,

$$L_{i,k}^{(1)} \quad \geq \ell_k \tfrac{1}{1+\sum_{j\neq i} h_{j,k}} =: L_{i,k}^{*} \tag{12}$$

$$\geq \ell_k \tfrac{h_{i,k}}{h_{i,k}+\sum_{j\neq i} h_{j,k}} =: L_{i,k}^{(2)}. \tag{13}$$

Empirically, we found that using the $L_{i,k}^{*}$ may give approximate hitting probabilities only marginally larger than (8).

But, empirically, we found that using (13) tends to *over*-estimate the cache hitting probabilities. Note that $\sum_i L_{i,k}^{(2)} = \ell_k$, *i.e.*, $L_{i,k}^{(2)}$ models how object k is shared over time by the different caches i, rather than the mean object length given that it is stored in the cache.

Substituting (8) into (7) gives, for $i \in \{1, 2, \ldots, J\}$,

$$0 = b_i - \sum_{k=1}^{N} h_{i,k} \mathbb{E} \frac{1}{1+\sum_{j\neq i} Z_{j,k}} \ell_k \;\; =: \frac{\partial u_i}{\partial t_i} =: \partial_i u_i \tag{14}$$

Under (6) and $\mathbb{E}Z_{j,k} = h_{j,k}$ for independent Boolean $Z_{j,k}$, equations (14) are a set of J equations in J unknowns $\{t_i\}_{j=1}^{J}$.

Note that for all the above definitions, $\forall i, k,\;\; L_{i,k} \leq \ell_k$, so one expects corresponding hit cache probabilities to be larger than without object-sharing; recall Prop. 3.1.

B. Existence and uniqueness of solution to the working-set approximation (14)

A basic assumption is that,

$$\forall i \;\; b_i < \frac{1}{J}\sum_{k=1}^{N} \ell_k, \tag{15}$$

i.e., no LRU-list is large enough to hold all of the objects even if the objects were fully shared. Note that if $\forall i\ b_i \geq \frac{1}{J}\sum_{k=1}^{N}\ell_k$, then the total available cache memory ($\geq \sum_{i=1}^{J} b_i$) is sufficiently large to hold all possible data objects (hence is not a "cache").

Proposition 4.2: If (15) holds then there are real numbers $s_j \geq 0, S_j < \infty$, such that $s_j < S_j$ and there exists a unique solution $\{t_i\}_{i=1}^{J} \in \prod_{i=1}^{J}[s_i, S_i]$ to (14).

Proof: Consider the quantities u_i as utilities of a noncooperative J-player game with strategies

$$\{t_j\}_{j=1}^{J} \in \prod_{j=1}^{J}[s_j, S_j] =: \mathcal{S}$$

where $0 \leq s_j < S_j < \infty$. First note that each u_i of (14) is continuously differentiable on $\mathcal{S}$.

For a J-dimensional vector $\underline{t} = (t_1, \ldots, t_J) \in \mathcal{S}$ let $\underline{t}_{-i}$ be the $(J-1)$-vector obtained by eliminating the entry t_i. Since the strategy-space $\mathcal{S}$ is compact and the utility functions $u_i(\underline{t})$ are strictly concave in t_i (since $\partial_i^2 u_i < 0$) a Nash equilibrium exists [3]. Alternatively, we can use Brouwer's theorem [6] to establish existence of the Nash equilibrium.

Generally, a Nash equilibrium may occur on the boundary of the strategy-space. However, note here that for an arbitrary $\underline{t}_{-i}$,

$$\begin{aligned}
\lim_{t_i \to 0} \partial_i u_i(t_i, \underline{t}_{-i}) &= b_i > 0 \quad \text{and} \\
\lim_{t_i \to \infty} \partial_i u_i(t_i, \underline{t}_{-i}) &= b_i - \sum_{k=1}^{N} \frac{1}{1 + \sum_{j \neq i}(1 - \mathrm{e}^{-\lambda_{j,k} t_j})} \ell_k \\
&\leq b_i - \frac{1}{J}\sum_{k=1}^{N} \ell_k < 0, \qquad \text{by (15).}
\end{aligned}$$

Because of this and the strict concavity of u_i in t_i, if all S_j are sufficiently large and $s_j \geq 0$ sufficiently small, then all $\partial_i u_i(\underline{t})$ are *unimodal* in t_i for all $\underline{t}_{-i}$ such that $\underline{t} \in \mathcal{S}$. As a result, the Nash equilibria are all interior to $\mathcal{S}$ so that the first-order necessary conditions for u_i-optimality must all hold, *i.e.*, (14) are satisfied.

By such unimodality and because strict concavity implies $\partial_i^2 u_i \neq 0$, uniqueness of the solution follows.

■

By the same argument:

Corollary 4.1: Proposition 4.2 is also true under (12) or (13) as well, the latter with $s_i > 0$ for all i.

Note that the diagonal-dominance conditions implying negative definiteness of the Jacobian of the gradient map, in turn implying uniqueness of the solution $\{t_i\}_{i=1}^{J}$ to (14) [23], [20], do not hold here.

C. Static cache partitioning with shared objects

Consider a caching system as described above with LRU-lists but *without* object sharing, *i.e.*, the full length of the an object is charged to each LRU list in which it resides. In this case, from any proxy's point-of-view, the system is just as static cache partitioning mentioned in Section 1. But from the cache's point of view, there may be room for additional objects in the memory even when $\sum_{i=1}^{J} b_i = B$ because every object is only store once in the cache. This additional memory could be used to store objects recently evicted from all LRU-lists (as described above), or the cache operator could attempt to *overbook* the cache, *i.e.*, operate such that $\sum_{i=1}^{J} b_i > B$. That is, the cache operator would benefit from object-sharing and its customers (the proxies) would not.

For example, considering the working-set approximation (5) for independent LRU caches without object sharing under the IRM, if the cache operator can estimate

$$\sum_{k=1}^{N} \ell_k (1 - \prod_{i=1}^{J} (1 - h_{i,k}))$$

for a current set of J proxies, then she can admit a new proxy requiring cache memory b_{J+1} if

$$b_{J+1} \;\leq\; B - \sum_{k=1}^{N} \ell_k (1 - \prod_{i=1}^{J} (1 - h_{i,k})).$$

V. Numerical results on cache memory sharing

We ran a number of experiments to test the foregoing approximations of hitting probabilities of the shared-object cache. To approximate, we solved (14) using the Newton-Raphson algorithm; this was simplified by the concavity properties and uniqueness of solution discussed in the proof of Prop. 4.2.

The result of a typical experiment for a cache shared by $J = 2$ LRU-lists is given in Table I (where hitting probabilities were evaluated by simulation with high confidence) and Table II (where (6) and (14) was used). The experiment involved LRU-lists $i = 0, 1$ of size $b_i \in \{8, 64\}$ and $N = 1000$ objects all of unit length. As a reference, we provide Table IV giving hitting probabilities of isolated caches without data-object sharing.

The "popularity" of a data object n refers to the mean rate λ_n. at which it is requested. A commonly used model for popularity is given by the so-called Zipf law:

$$\lambda_n \propto \rho(n)^{-\alpha}, \tag{16}$$

where $\alpha > 0$ is the Zipf parameter and $\rho(n)$ is the popularity rank of object n, *i.e.*, $\rho(n') = 1$ if $n' = \text{argmax}_n \lambda_n$ is unique and $\rho(n'') = N$ if $n'' = \text{argmin}_n \lambda_n$ is unique. For example, values $0.64 \leq \alpha \leq 0.83$ were given for different datasets in Table 1 of [7].

i	b_0	b_1	$h_{i,1}$	$h_{i,10}$	$h_{i,100}$	$h_{i,1000}$
0	8	8	0.501	0.113	0.0226	0.00399
0	8	64	0.503	0.113	0.0216	0.00396
0	64	8	0.998	0.687	0.189	0.0343
0	64	64	0.998	0.697	0.195	0.0400
1	8	8	0.203	0.0673	0.0226	0.00709
1	8	64	0.853	0.453	0.172	0.0574
1	64	8	0.208	0.0699	0.0231	0.00736
1	64	64	0.860	0.465	0.179	0.0609

TABLE I

EMPIRICAL HITTING PROBABILITIES FOR A SIMULATED CACHE UNDER THE IRM OF SIZE $B = 1000$ FOR UNIT-LENGTH OBJECTS ($\forall n,\ \ell_n = 1$) THAT IS SHARED BY TWO LRU-LISTS $i = 1, 2$ RESPECTIVELY WITH ZIPF POPULARITY PARAMETERS $\alpha_0 = .75$ AND $\alpha_1 = .5$. SIMULATION TIME WAS SUFFICIENTLY LONG SO THAT THESE HITTING PROBABILITIES ARE OBTAINED WITH HIGH CONFIDENCE.

cache i	b_0	b_1	$h_{i,1}$	$h_{i,10}$	$h_{i,100}$	$h_{i,1000}$
0	8	8	.571	.140	.0264	.00474
0	8	64	.805	.253	.0504	.00916
0	64	8	.996	.630	.162	.0309
0	64	64	.999	.791	.243	.0483
1	8	8	.223	.0767	.0249	.00795
1	8	64	.773	.374	.138	.0458
1	64	8	.393	.146	.0487	.0157
1	64	64	.900	.517	.205	.0701

TABLE II

HITTING PROBABILITIES NUMERICALLY APPROXIMATED USING NEWTON-RAPHSON TO SOLVE (7) WITH MEAN OBJECT LENGTHS (13) FOR $\{t_i\}_{i=1}^{J}$, AND THEN USING (6), FOR THE SHARED CACHE OF TABLE I.

cache i	b_0	b_1	$h_{i,1}$	$h_{i,10}$	$h_{i,100}$	$h_{i,1000}$
0	8	8	0.357	0.0756	0.0139	0.00248
0	8	64	0.392	0.08479	0.0156	0.00280
0	64	8	0.983	0.516	0.121	0.0227
0	64	64	0.990	0.557	0.135	0.0254
1	8	8	0.125	0.041	0.0133	0.00421
1	8	64	0.676	0.300	0.107	0.0350
1	64	8	0.136	0.0453	0.0146	0.0046
1	64	64	0.712	0.325	0.117	0.03857

TABLE III

HITTING PROBABILITIES NUMERICALLY APPROXIMATED INSTEAD USING MEAN OBJECT LENGTHS (8) FOR THE SHARED CACHE OF TABLE I, *i.e.*, SOLVING (14). USING MEAN OBJECT LENGTHS (12) GIVES SIMILAR RESULTS.

cache i	b_0	b_1	$h_{i,1}$	$h_{i,10}$	$h_{i,100}$	$h_{i,1000}$
0	8	n/a	.354	.0735	.0133	.00222
0	64	n/a	.981	.504	.123	.0200
1	n/a	8	.123	.0403	.0137	.00376
1	n/a	64	.665	.295	.105	.0343

TABLE IV

EMPIRICAL HITTING PROBABILITIES OF ISOLATED (NOT OBJECT-SHARING) CACHES UNDER THE IRM OF SIZE b_i, $i = 1, 2$, FOR UNIT-LENGTH OBJECTS ($\forall n,\ \ell_n = 1$) RESPECTIVELY WITH ZIPF POPULARITY PARAMETERS $\alpha_0 = .75$ AND $\alpha_1 = .5$.

In our typical two LRU-list experiments reported herein, we took the Zipf parameter $\alpha_0 = 0.75$ for LRU-list $i = 0$ and $\alpha_1 = 0.5$ for LRU-list $i = 1$.

First note that the cache-hit probabilities of Table IV are generally lower than those of Table I, *i.e.*, object sharing obviously increases cache-hit probabilities, recall Prop. 3.1. The cache-hit probabilities by working-set approximation of Table II (corresponding to mean object lengths (13)) are larger than those of Table I. The cache-hit probabilities of Table III (corresponding to mean object lengths (8)) are smaller (by $\sim$30%) than those of Table I. These working-set approximations required orders of magnitude less computation than system simulation with high confidence of Table I. The approximations of Table II are 10-20% higher in most cases, except when the cache-memory sizes are quite different – *e.g.*, "0,8,64" (LRU list $i = 0$ with $b_0 = 8$ and $b_1 = 64$) and "1,64,8" – the approximation is about 80% higher.

Typical results for a cache shared by three or more LRU-lists are shown in Tables V and VI (for $J = 3$ LRU-lists). Here we see that the approximation (8) is reasonably accurate.

VI. DISCUSSION OF CACHE I/O SHARING

Let $R_{i,n}$ be the incident mean-rate that proxy i requests object n having length ℓ_n. Let $\lambda_{i,n}$ be the g_i-*admitted* mean request rate, where over any interval of length t, the total length of requests admitted is at most $g_i(t)$. Thus,

$$\forall i,\ \sum_n \ell_n \lambda_{i,n} \ \leq\ \lim_{t\to\infty} \frac{g_i(t)}{t}. \tag{17}$$

Let the total mean-rate of requests for object n to the cache be

$$\lambda_n = \sum_{i=1}^{J} \lambda_{i,n}. \tag{18}$$

A proxy does not wish to unnecessarily pay more than it needs for larger demand envelope nor to have its users' requests blocked (or delayed in order to conform to its demand envelope g). So, assume that g_i is selected (purchased) by proxy i so that it minimally impedes the estimated incident rate of its users'

i	b_0	b_1	b_2	$h_{i,1}$	$h_{i,10}$	$h_{i,100}$	$h_{i,1000}$
0	8	8	8	0.368	0.075760	0.014187	0.002258
0	8	8	64	0.407	0.087653	0.015783	0.002726
0	8	64	8	0.389	0.082334	0.014859	0.002705
0	8	64	64	0.422	0.092404	0.016709	0.002814
0	64	8	8	0.983	0.513789	0.117013	0.023025
0	64	8	64	0.989	0.556809	0.132546	0.026602
0	64	64	8	0.986	0.538738	0.126236	0.023658
0	64	64	64	0.992	0.576265	0.144489	0.027240
1	8	8	8	0.126	0.041193	0.013011	0.004227
1	8	8	64	0.136	0.044836	0.013762	0.004377
1	8	64	8	0.676	0.299101	0.106945	0.034217
1	8	64	64	0.699	0.320489	0.113116	0.035743
1	64	8	8	0.136	0.043797	0.013603	0.004249
1	64	8	64	0.143	0.047648	0.014567	0.004582
1	64	64	8	0.699	0.315947	0.112926	0.036378
1	64	64	64	0.726	0.331754	0.120471	0.039156
2	8	8	8	0.708	0.114214	0.012135	0.001164
2	8	8	64	1.000	0.755988	0.129207	0.014110
2	8	64	8	0.745	0.128129	0.013006	0.001455
2	8	64	64	1.000	0.788157	0.141869	0.016281
2	64	8	8	0.771	0.138305	0.014620	0.001684
2	64	8	64	1.000	0.796751	0.141870	0.014351
2	64	64	8	0.793	0.150212	0.014683	0.001525
2	64	64	64	1.000	0.819580	0.159722	0.014161

TABLE V

EMPIRICAL HITTING PROBABILITIES FOR A SIMULATED CACHE UNDER THE IRM OF SIZE $B = 1000$ FOR UNIT-LENGTH OBJECTS ($\forall n,\ \ell_n = 1$) THAT IS SHARED BY THREE LRU-LISTS $i = 0, 1, 2$ RESPECTIVELY WITH ZIPF POPULARITY PARAMETERS $\alpha_0 = .75$, $\alpha_1 = .5$, AND $\alpha_2 = 1$. SIMULATION TIME WAS SUFFICIENTLY LONG SO THAT THESE HITTING PROBABILITIES ARE OBTAINED WITH HIGH CONFIDENCE.

requests. For example, if the aggregate demand from proxy i is Poisson (so that the mean number of requests in a time interval approximately equals the variance), then

$$g_i(t) = g_i(0) + tR_i + 3\sqrt{tR_i}, \tag{19}$$

may be selected, where $R_i = \sum_n R_{i,n}$ is the mean aggregate request rate. For large R_i recall that the (Poisson distributed) number of requests in an interval of time t approximately $\mathsf{Normal}(tR_i, tR_i)$. Hence, in selecting g_i in (19) we limit the probability of a blocked request to about 1%. Here, $g_i(0) > 0$ corresponds to a number of allowed simultaneous requests. In practice, such a demand envelope g_i may be approximated

i	b_0	b_1	b_2	$h_{i,1}$	$h_{i,10}$	$h_{i,100}$	$h_{i,1000}$
0	8	8	8	0.365	0.077604	0.014262	0.002551
0	8	8	64	0.401	0.087205	0.016095	0.002881
0	8	64	8	0.386	0.083160	0.015321	0.002742
0	8	64	64	0.421	0.092609	0.017133	0.003068
0	64	8	8	0.984	0.521259	0.122771	0.023024
0	64	8	64	0.990	0.562247	0.136623	0.025785
0	64	64	8	0.988	0.545451	0.130823	0.024625
0	64	64	64	0.993	0.584584	0.144627	0.027397
1	8	8	8	0.126	0.041602	0.013347	0.004240
1	8	8	64	0.134	0.044618	0.014330	0.004554
1	8	64	8	0.678	0.301116	0.107113	0.035193
1	8	64	64	0.704	0.319691	0.114687	0.037788
1	64	8	8	0.133	0.044198	0.014193	0.004510
1	64	8	64	0.142	0.047180	0.015167	0.004821
1	64	64	8	0.701	0.317105	0.113624	0.037423
1	64	64	64	0.725	0.335317	0.121168	0.040022
2	8	8	8	0.694	0.111572	0.011761	0.001182
2	8	8	64	1.000	0.755600	0.131419	0.013991
2	8	64	8	0.734	0.124170	0.013171	0.001325
2	8	64	64	1.000	0.786098	0.142920	0.015304
2	64	8	8	0.756	0.131431	0.013992	0.001408
2	64	8	64	1.000	0.799474	0.148437	0.015940
2	64	64	8	0.787	0.143363	0.015355	0.001546
2	64	64	64	1.000	0.824859	0.159885	0.017271

TABLE VI

HITTING PROBABILITIES NUMERICALLY APPROXIMATED INSTEAD USING MEAN OBJECT LENGTHS (8), SOLVING (14) AND SUBSTITUTING INTO (6), FOR THE SHARED CACHE OF TABLE V.

by piecewise linear one implemented by one or more token-bucket mechanisms, *e.g.*, [15], [16]. That is, a token corresponds to a common unit of object/content length (the dimension of ℓ_n).

For long-term stability, the cache I/O bandwidth C should satisfy

$$C \geq \lim_{t\to\infty} \sum_i \frac{g_i(t)}{t} \quad \Rightarrow \quad C \geq \sum_n \ell_n \lambda_n. \tag{20}$$

Admission control of new proxies and dynamic pricing could be based on in part on (20).

It is possible that, periodically, demand may exceed cache memory bandwidth, so the cache will have a *finite* request queue. The total length of in-profile [15], [16] *objects* corresponding to queued requests

will be upper bounded by

$$\max_{t>0} \sum_i g_i(t) - Ct.$$

Instead of blocking *out-of-profile* requests, one could set-up queues for both in-profile and out-of-profile requests, where the former would obviously have priority over the latter.

Finally, recall the edge-computing context in support of mobile users. If the same object is requested at the same time over a wireless channel, the necessary tokens may be shared among requesting proxies. Again, a problem here is intellectual property protections: content could be encrypted to the cache and then commonly encrypted to a *group* of authorized mobile subscribers. The cryptographic keys required for the latter may be periodically refreshed to deal with subscriber churn.

VII. Discussion of pricing and mock requests

Obviously, prices charged to proxy i could depend in part on resource reservations (g_i, b_i). Service may be organized in tiers, where tier k is associated with demand envelope $g = k\gamma$ and cache memory allocation $b = k\beta$, and $k = 1$ corresponds to an atomic or base service tier. Though the price π_k of tier k is obviously increasing with (integer) k, it may be such that π_k/k is decreasing as a kind of volume discount. Also, costs may be affected by usage – there is precedent for *penalizing* low utilization, presumably to encourage customers to better characterize their workloads and not be resource wasteful.

Recall that a query by proxy i that is a LRU-miss but a physical cache hit involves an added delay so that the net effect is a physical cache miss. So, from proxy i's point of view, there is no incentive to "free ride" on other proxies by issuing mock queries so that i's LRU list is populated primarily by objects that are popular only among its own users. Mock requests will deplete apportioned network bandwidth resources too. Also, if allocated resources are multiples of an "atomic" service tier, a proxy may not be able to purchase just a little more network bandwidth to mitigate this impact of mock requests.

References

[1] O.I. Aven, E.G. Coffman, and Y.A. Kogan. *Stochastic analysis of computer storage*. D. Reidel Publishing Co., 1987.

[2] F. Baccelli and P. Bremaud. *Elements of Queueing Theory*. Springer-Verlag, Application of Mathematics: Stochastic Modelling and Applied Probability, No. 26, New York, NY, 1991.

[3] T. Başar and G.J. Olsder. *Dynamic Noncooperative Game Theory*. Classics in Applied Mathematics, SIAM, Philadelphia, 1999.

[4] A. Balamash and M. Krunz. An overview of web caching replacement algorithms. *IEEE Communications Surveys & Tutorials*, 6(2), 2004.

[5] J. Van Den Berg and A. Gandolfi. LRU is better than FIFO under the independent reference model. *J. Appl. Prob.*, 29, 1992.

[6] K.C. Border. *Fixed Point Theorems with Applications to Economics and Game Theory*. Cambridge University Press, London, 1985.

[7] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web Caching and Zipf-like Distributions: Evidence and Implications. In *Proc. IEEE INFOCOM*, 1999.

[8] H. Che, Y. Tung, and Z. Wang. Hierarchical Web Caching Systems: Modeling, Design and Experimental Results. *IEEE JSAC*, 20(7), Sept. 2002.

[9] M. Dehghan, W. Chu, P. Nain, and D. Towsley. Sharing LRU Cache Resources among ContentProviders: A Utility-Based Approach . https://arxiv.org/abs/1702.01823, Feb. 2017.

[10] P.J. Denning and S.C. Schwartz. Properties of the working-set model. *Commun. ACM*, 15(3):191–198, March 1972.

[11] A. Eryilmaz et al. A New Flexible Multi-flow LRU Cache ManagementParadigm for Minimizing Misses. In *Proc. ACM SIGMETRICS*, 2019.

[12] R. Fagin. Asymptotic approximation of the move-to-front search cost distribution and least-recently-used caching fault probabilities, 1977.

[13] C. Fricker, P. Robert, and J. Roberts. A Versatile and Accurate Approximation for LRU Cache Performance. In *Proc. International Teletraffic Congress*, 2012.

[14] N. Golrezaei, K. Shanmugam, A.G. Dimakis, A.F. Molisch, and G. Caire. Femtocaching: Wireless video content delivery through distributed caching helpers. In *Proc. IEEE INFOCOM*, 2012.

[15] J. Heinanen, T. Finland, and R. Guerin. A single rate three color marker. *RFC 2697 available at www.ietf.org*, 1999.

[16] J. Heinanen, T. Finland, and R. Guerin. A two rate three color marker. *RFC 2698 available at www.ietf.org*, 1999.

[17] K. Joag-Dev and F. Proschan. Negative association of random variables with applications. *The Annals of Statistics*, pages 286–295, 1983.

[18] A. Khursheed and K.M.L. Saxena. Positive dependence in multivariate distributions. *Communications in Statistics - Theory and Methods*, 10(12):1183–1196, 1981.

[19] W.F. King. Analysis of paging algorithms. In *Proc. IFIP Congress*, Lyublyana, Yugoslavia, Aug. 1971.

[20] H. Moulin. Dominance Solvability and Cournot Stability. *Mathematical Social Sciences*, 7:83–102, 1984.

[21] K. Poularakis, G. Iosifidis, A. Argyriou, I. Koutsopoulos, and L. Tassiulas. Distributed Caching Algorithms in the Realm of Layered Video Streaming. *IEEE Trans. Mob. Comput.*, 18(4):757–770, 2019.

[22] Q. Pu, H. Li, M. Zaharia, A. Ghodsi, and I. Stoica. FairRide: Near-Optimal, Fair Cache Sharing. In *Proc. USENIX NDSI*, Santa Clara, CA, USA, March 2016.

[23] J.B. Rosen. Existence and uniqueness of equilibrium points for concave n-person games. *Econometrica*, 33(3):520–534, 1965.

[24] D. Wajc. Negative association: Definition, properties and applications. http://www.cs.cmu.edu/~dwajc/notes/Negative%20Association.pdf, Apr. 2017.

[25] Y. Wang, X. Zhou, M. Sun, L. Zhang, and X. Wu. A new QoE-driven video cache management scheme with wireless cloud computing in cellular networks. *Mobile Networks and Applications*, 2016.

[26] R.W. Wolff. *Stochastic Modeling and the Theory of Queues*. Prentice-Hall, Englewood Cliffs, NJ, 1989.