

# Sharing Cache Resources among Content Providers: A Utility-Based Approach

Mostafa Dehghan, Weibo Chu, Philippe Nain, Don Towsley, Zhi-Li Zhang

▶ **To cite this version:**

Mostafa Dehghan, Weibo Chu, Philippe Nain, Don Towsley, Zhi-Li Zhang. Sharing Cache Resources among Content Providers: A Utility-Based Approach. 2017. hal-01672961v1

**HAL Id: hal-01672961**

**https://inria.hal.science/hal-01672961v1**

Preprint submitted on 27 Dec 2017 (v1), last revised 18 Dec 2018 (v2)

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Sharing Cache Resources among Content Providers: A Utility-Based Approach

Mostafa Dehghan[1], Weibo Chu[2], Philippe Nain[3], Don Towsley[1], Zhi-Li Zhang[4]
[1]University of Massachusetts, Amherst, USA
[2]Northwestern Polytechnical University, Xi'an, China
[3]Inria, France, [4]University of Minnesota, Minneapolis, USA
{mdehghan, towsley}@cs.umass.edu, wbchu@nwpu.edu.cn
philippe.nain@inria.fr, zhzhang@cs.umn.edu

## ABSTRACT

In this paper, we consider the problem of allocating cache resources among multiple content providers. The cache can be partitioned into slices and each partition can be dedicated to a particular content provider, or shared among a number of them. It is assumed that each partition employs the LRU policy for managing content. We propose utility-driven partitioning, where we associate with each content provider a utility that is a function of the hit rate observed by the content provider. We consider two scenarios: i) content providers serve disjoint sets of files, ii) there is some overlap in the content served by multiple content providers. In the first case, we prove that cache partitioning outperforms cache sharing as cache size and numbers of contents served by providers go to infinity. In the second case, It can be beneficial to have separate partitions for overlapped content. In the case of two providers it is usually always beneficial to allocate a cache partition to serve all overlapped content and separate partitions to serve the non-overlapped contents of both providers. We establish conditions when this is true asymptotically but also present an example where it is not true asymptotically. We develop online algorithms that dynamically adjust partition sizes in order to maximize the overall utility and prove that they converge to optimal solutions, and through numerical evaluations we show they are effective.

## 1. INTRODUCTION

The Internet has become a global information depository and content distribution platform, where various types of information or content are stored in the "cloud", hosted by a wide array of *content providers*, and delivered or "streamed" on demand. The (nearly) "anytime, anywhere access" of online information or content – especially multimedia content – has precipitated rapid growth in Internet data traffic in recent years, both in wired and wireless (cellular) networks. It is estimated [1] that the global Internet traffic in 2019 will reach 64 times its entire volume in 2005. A primary contributor to this rapid growth in data traffic comes from online video streaming services such as Netflix, Hulu, YouTube and Amazon Video, just to name a few. It was reported [23] that Netflix alone consumed nearly a third of the peak downstream traffic in North America in 2012, and it is predicted [1] that nearly 90% of all data traffic will come from video content distributors in the near future.

Massive data traffic generated by large-scale online information access – especially, "over-the-top" video delivery – imposes an enormous burden on the Internet and poses many challenging issues. Storing, serving, and delivering videos to a large number of geographically dispersed users in particular require a vast and sophisticated infrastructure with huge computing, storage and network capacities. The challenges in developing and operating large-scale video streaming services in today's Internet [2–4] to handle user demands and meet user desired *quality-of-experience* also highlight some of the key limitations of today's Internet architecture. This has led to a call for alternate Internet architectures that connect people to content rather than servers (see [5] for a survey of representative architecture proposals). The basic premise of these content-oriented architectures is that storage is an integral part of the network substrate where content can be cached *on-the-fly*, or prefetched or "staged" *a priori*.

While there has been a flurry of recent research studies in the design of caching mechanisms [6,7,20,26], relatively little attention has been paid to the problem of storage or *cache resource allocation among multiple content providers*. In this paper, we address a fundamental research question that is pertinent to all architecture designs: *how to share or allocate the cache resource within a single network forwarding element and across various network forwarding elements among multiple content providers so as to maximize the cache resource utilization* or *provide best utilities to content providers?*

This question was addressed in [9] in an informal and heuristic manner. It proposed a utility maximization framework, which we adopt, to address the aforemen-

tioned fundamental problem. We consider a scenario where there are multiple content providers offering the same type of content, *e.g.*, videos; the content objects offered by the content providers can be all distinct or there may be common objects owned by different content providers. Due to disparate user bases, the access probabilities of these content objects may vary across the CPs. Our analysis and results are predicated on the use of Least Recently Used (LRU) cache replacement; however, we believe that they apply to other policies as well. [9] argued that, if all CPs offer *distinct* content objects, *partitioning the cache into slices of appropriate sizes, one slice per CP*, yields the best cache allocation strategy as it maximizes the sum of CP utilities. [9] also considered the case where CPs serve common content and argued that placing common content into a single LRU cache and non-common content in separate LRU caches usually provides the best performance. We make more precise statements to support these observations. In the case that common content are requested according to the same popularity distributions, regardless of provider, in the limit aggregate hit rate is maximized when three LRU partitions are established, one for the overlapped content and the other two for the non-overlap content. We also provide a counterexample that shows that such a strategy is not always optimal. However, the conclusion is that partitioning is usually best.

The above results are based on the work of Fagin [12], who characterized the asymptotic behavior of LRU for a rich class of content popularity distributions that include the Zipf distribution.

In the last part of the paper, we develop decentralized algorithms to implement utility-driven cache partitioning. These algorithms adapt to changes in system parameters by dynamically adjusting the partition sizes, and are theoretically proven to be stable and converge to the optimal solution.

Our results illustrate the importance of considering the cache allocation problem among multiple CPs and has implications in architectural designs: from the perspective of cache resource efficiency or utility maximization of CPs, cache partitioning (among CPs) should be a basic principle for cache resource allocation; it also suggests alternate content-oriented architectures which explicitly account for the role of CPs [22]. Cache partitioning also provides a natural means to effectively handle heterogeneous types of content with different traffic or access characteristics, and offer differentiated services for content delivery [16,17,19,26]. In the future Internet where network cache elements will likely be provided by various entities [27], our framework also facilitates the design of distributed pricing and control mechanisms, and allows for the establishment of a viable cache market economic model.

The main contributions of this paper can be summarized as follows:

- We establish the connection between Fagin's asymptotic results on the LRU cache and the characteristic time (CT) approximation introduced in [8], providing a stronger theoretical underpinning for the latter than previously known. Moreover, we extend Fagin's results and therefore theoretical justification of the CT approximation to a larger class of workloads that include those coming from independent content providers.
- Using Fagin's asymptotic framework we show that partitioning is the best strategy for sharing a cache when content providers do not have any content in common. On the other hand when content providers serve the same content, it can be beneficial for content providers to share a cache to serve their overlapped content. We establish this to be true for a class of popularity distributions. We also present an example where placing common content in a shared cache is not optimal.
- We develop online algorithms for managing cache partitions, and prove the convergence of these algorithms to the optimal solution using Lyapunov functions.
- We show that our framework can be used in revenue based models where content providers react to prices set by (cache) service providers without revealing their utility functions.
- We perform numerical studies and simulations to show the efficacy of cache partitioning and also the convergence of our online algorithms using different utility functions with different fairness implications.

The remainder of this paper is organized as follows. We present the problem setting and basic model in Section 2 where we make the connection between Fagin's asymptotic results and the CT approximation. We describe the cache allocation problem via the utility maximization framework in Section 3. In Section 4, we develop online algorithms for implementing utility-maximizing cache partitioning. Numerical and simulation results are presented in Section 5. In Section 6, we explore the implications of our results, and discuss future research directions and related work. Section 7 concludes the paper.

## 2. PROBLEM SETTING & BASIC MODEL

Consider a network as shown in Figure 1, where users access content, *e.g.*, videos, from $K$ content providers (CPs). CP $k$ ($k = 1, \ldots, K$) serves a set $S_k$ of $n_k$ unit size files where $n_k = |S_k|$; we will usually label

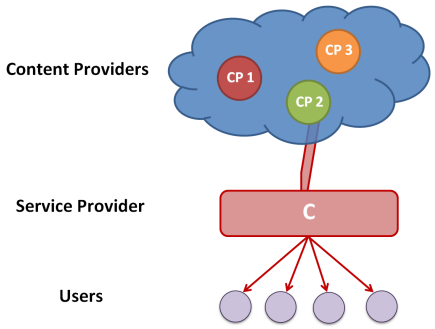



Figure 1: Network Model.

these files $i = 1, \ldots, n_k$. All CPs share a content cache, supplied by a third-party network provider, referred to as a service provider hereafter. Content providers have business relations with the service provider and pay for cache resources. There are two possible scenarios: i) the content objects offered by the CPs are all *distinct*; and ii) some *common* objects are provided by different CPs. Due to disparate user bases, the access patterns of these content objects may vary across the CPs.

We assume that requests are described by a Poisson process with request rate for file $i$ of CP $k$ being $\lambda_{k,i} = \lambda_k p_{k,i}, i \in S_k, k = 1, \ldots, K$, where $\lambda_k$ denotes the aggregate request rate for contents from CP $k$, and $p_{k,i}$ is the probability that a request to CP $k$ is for content $i$. Associated with each CP $k$ is a utility $U_k(h_k)$ that is an *increasing* and *concave* function of the hit rate $h_k$ over all its files. In its most general form, the service provider wishes to maximize the sum of utilities over all content providers, $\sum_k U_k(h_k)$, through a proper allocation of cache space to the CPs. In the simple case where $U_k(h_k) = h_k$, the objective becomes that of maximizing the overall cache hit rate, which provides a measure of the overall cache utilization efficiency.

**Cache Partitioning:** When the cache is shared among the CPs, content objects offered by all CPs compete for the storage space based on their access patterns. To restrict cache contention to smaller sets of content, the service provider can form content groups from files served by a CP or multiple CPs, and partition the cache into slices and dedicate a partition to each content group. Let $P$ denote the number of content groups/cache partitions. Also, let $V_p$ and $C_p, p = 1, \ldots, P$ denote the content groups and partition sizes, respectively. Note that $P = 1$ implies that the cache is shared as a whole, while $P > 1$ means it is partitioned.

The first question to ask is: *what is the optimal number of partitions and how should files be grouped?* To determine the number of slices and that what files should be requested from which partition, the service provider proceeds as follows. Files are first grouped into disjoint sets according to which content providers serve them. The service provider then decides how many partitions to create, and whether to dedicate a separate partition for each set of files, or have multiple sets of files share a partition. In the next section, we explain what could change if the cache manager made partitioning decisions on a per file basis rather than sets of files.

Assuming the answer to the first question, the second question is: *how should the partitions be sized?* Let $\mathbf{C} = (C_1, C_2, \ldots, C_P)$ denote the vector of partition sizes. For each content provider $k$, hit rate is a function of the partition sizes $h_k(\mathbf{C})$. For a cache of size $C$, we formulate this question as the following optimization problem:

$$\begin{aligned} \text{maximize} \quad & \sum_{k=1}^{K} U_k\Big(h_k(\mathbf{C})\Big) \\ \text{such that} \quad & \sum_{p=1}^{P} C_p \le C \\ & C_p = 0, 1, 2, \ldots; \quad p = 1, 2, \ldots, P. \end{aligned}$$

Note that the above formulation is an integer programming problem that is typically hard to solve. However, in practice caches are large and therefore we assume $C_p$ can take any real value, as the rounding error will be negligible.

**Cache Characteristic Time:** Assume a cache of size $C$ serving $n$ contents with popularity distribution $p_i$, $i = 1, \ldots, n$ . Under the independent reference model (requests are i.i.d.), Fagin [12] introduced the notion of a *window size* $T$ that satisfies

$$C = \sum_{i=1}^{n} (1 - (1 - p_i)^T).$$

The miss probability associated with a window of size $T$ is defined as

$$m(T) = \sum_{i=1}^{n} p_i (1 - p_i)^T.$$

Fagin introduced a cumulative probability distribution, $F$, that is continuously differentiable in $(0, 1)$ with $F(0) = 0$ and $F(1) = 1$. The right-derivative of $F$ at 0, denoted by $F'(0)$, may be infinite. This will allow us to account for Zipf-like distributions. Define

$$p_i^{(n)} = F(i/n) - F((i-1)/n), \quad i = 1, \ldots, n$$

the probability that page $i$ is requested. Hereafter, we will refer to $F$ as the popularity distribution. If $C/n = \beta$ then $T/n \to \tau_0$ where (see [12])

$$\beta = \int_0^1 (1 - e^{-F'(x)\tau_0}) dx \tag{1}$$

and $m(T) \to \mu$ where

$$\mu = \int_0^1 F'(x) e^{-F'(x)\tau_0} dx. \tag{2}$$

Moreover, $\mu$ is the limiting miss probability under LRU when $n \to \infty$.

Suppose that requests arrive according to a Poisson process with rate $\lambda$. Express $\beta$ as

$$\beta = \int_0^1 P(X(x) < \tau_0/\lambda)dx$$

where $X(x)$ is an exponential random variable with intensity $\lambda F'(x)$. $X(x)$ is the inter-arrival time of two requests for content of type $x$. If this time is less than $\tau_0/\lambda$, then the request is served from the cache, otherwise it is not. In practice, as $n$ is finite, this is approximated by

$$C = \beta n = \sum_{i=1}^{n}(1 - e^{-\lambda p_i^{(n)} T_c}) \tag{3}$$

where $T_c$ is the Characteristic Time (CT) for the finite content cache [8]. The aggregate miss probability is approximated by

$$m(T_c) = 1 - \sum_{i=1}^{n} p_i^{(n)}\big(1 - e^{-\lambda p_i^{(n)} T_c}\big). \tag{4}$$

Fagin's results suffice to show that as $n \to \infty$, the r.h.s. of (4) converges to the LRU miss probability.

In the context of $K$ providers, let $n_k = b_k n$, $n, b_k \in \mathbb{N}$, $k = 1, \ldots, K$. Denote $B_k := \sum_{j=1}^{k} b_j$ with $B_0 = 0$ by convention. It helps also to denote $B_K$ by $B$. Let $F_1, F_2, \ldots, F_K$ be continuous uniformly differentiable CDFs in $(0, 1)$. $F_k'(0)$ may be infinite for $k = 1, \ldots K$. If each provider has a cache that can store a fraction $\beta_k$ of its contents, then the earlier described CT approximation, (3), (4), applies with

$$p_{k,i}^{(n)} = F_k\Big(\frac{i}{b_k n}\Big) - F_k\Big(\frac{i-1}{b_k n}\Big), \quad i = 1, \ldots, b_k n. \tag{5}$$

We denote the asymptotic miss probabilities for the $K$ caches, each using LRU, by

$$\mu_k^{(p)} = \int_0^1 F_k'(x) e^{-F_k'(x)\tau_k} dx, \quad k = 1, \ldots, K \tag{6}$$

where $\tau_k$ is the solution of (1) with $\beta$ replaced by $\beta_k$.

Assume that the providers share a cache of size $C$. Let $\beta^{(s)} = (1/B)\sum_{k=1}^{K} b_k \beta_k$ be the fraction of the total number of documents that the shared cache can store. With this definition, observe that the size of the shared cache, given by $nB\beta^{(s)}$, and the size of the partitioned cache, given by $\sum_{k=1}^{K} n b_k \beta_k$, are the same, which will allow for a fair comparion between both schemes.

We introduce $\mu^{(s)}$ and $\tau_0$ through the following two equations,

$$\mu^{(s)} = \sum_{k=1}^{K} a_k \int_0^1 F_k'(x) e^{-a_k F_k'(x)\tau_0 B/b_k} dx, \tag{7}$$

Table 1: Glossary of notations.

| | |
|---|---|
| $S_k$ | set of files served by content provider $k$ |
| $p_i$ | probability that file $i$ is requested |
| $\lambda_{k,i}$ | request rate for file $i$ of CP $k$ |
| $\lambda_k$ | total request rate for CP $k$ contents |
| $F(\cdot)$ | file popularity CDF |
| $h_k$ | hit rate of CP $k$ |
| $C_p$ | capacity of partition $p$ |
| $n$ | number of files |
| $\beta$ | normalized capacity |
| $\mu$ | limiting miss probability |

$$\beta^{(s)} = 1 - \sum_{k=1}^{K} \frac{b_k}{B} \int_0^1 e^{-a_k F_k'(x)\tau_0 B/b_k} dx \tag{8}$$

where $a_k := \lambda_k/\lambda$, $k = 1, \ldots, K$.

**Theorem 1.** *Assume that we have $K$ providers with popularity distributions $F_1, \ldots, F_K$ as defined above, with numbers of contents given by $b_k n$ and request rates $\lambda_k$, $k = 1, \ldots, K$. Construct the sequence of popularity probabilities $\{p_{k,i}^{(n)}\}$, $n = 1, \ldots$ defined in (5) and cache sizes $C^{(n)}$ such that $C^{(n)}/(Bn) = \beta$. Then, the aggregate miss probability under LRU converges to $\mu^{(s)}$ given in (7), where $\tau_0$ is the unique solution of (8).*

*Proof.* See Appendix B. □

*Remark.* This extends Fagin's results to include any asymptotic popularity CDF $F$ that is continuously differentiable in $(0, 1)$ except at a countable number of points.

To help the reader with notation, a glossary of the main symbols used in this paper is given in Table 1.

## 3. CACHE RESOURCE ALLOCATION AMONG CONTENT PROVIDERS

In this section, we formulate cache management as a utility maximization problem. We introduce two formulations, one for the case where content providers serve distinct contents, and another one for the case where some contents are served by multiple providers.

### 3.1 Content Providers with Distinct Objects

Consider the case of $K$ providers with $n_k = b_k n$ contents each where $b_k, n \in \mathbb{N}$ and $k = 1, \ldots, K$. Also, let $B = \sum_{k=1}^{K} b_k$. Assume that requests to CP $k$ is characterized by a Poisson process with rate $\lambda_k$.

We ask the question whether the cache should be shared or partitioned between CPs under the LRU policy. It is easy to construct cases where sharing the cache is beneficial. However these arise when the cache size and the number of contents per CP are small. Evidence suggests that partitioning provides a larger aggregate

utility than sharing as cache size and number of contents grow. In fact, the following theorem shows that asymptotically, under the assumptions of Theorem 1, in the limit as $n \to \infty$, the sum of utilities under LRU when the cache is partitioned, is at least as large as it is under LRU when the CPs share the cache. To do this, we formulate the following optimization problem: namely to partition the cache among the providers so as to maximize the sum of utilities:

$$\max_{\beta_k} U^{(p)} := \sum_{k=1}^{K} U_k(\lambda_k(1-\mu_k(\beta_k))) \tag{9}$$
$$\text{s.t. } \beta = \sum_{k=1}^{K} \frac{b_k}{B}\beta_k,$$
$$\beta_k \geq 0, \quad k = 1, 2, \ldots, K.$$

Observe that $\lambda_k(1-\mu_k(\beta_k))$ in (9) is the hit rate of documents of CP $k$, where $\mu_k(\beta_k)$ is given by (6).

Here, $\mu_k$ is the asymptotic miss probability for content served by CP $k$, $\beta$ is the cache size constraint expressed in terms of the fraction of the aggregate content that can be cached, $b_k/B$ is the fraction of content belonging to provider $k$, and $\beta_k$ is the fraction of CP $k$ content that is permitted in CP $k$'s partition. The direct dependence of $\beta_k$ on $\mu_k$ is difficult to capture. Hence, we use (1) and (2), to transform the above problem into:

$$\max_{\tau_k} U^{(p)} := \sum_{k=1}^{K} U_k\Big(\lambda_k\big(1-\int_0^1 F_k'(x)e^{-F_k'(x)\tau_k}dx\big)\Big) \tag{10}$$
$$\text{s.t. } \beta = 1 - \sum_{k=1}^{K} \frac{b_k}{B}\int_0^1 e^{-F_k'(x)\tau_k}dx,$$
$$\tau_k \geq 0, \quad k = 1, 2, \ldots, K.$$

**Theorem 2.** *Assume $K$ providers with popularity distributions constructed from distributions $F_1, \ldots, F_K$ using (5) with number of contents $b_k n$ and request rates $\lambda_k$, $k = 1, \ldots, K$ sharing a cache of size $C^{(n)}$ such that $C^{(n)}/(Bn) = \beta$. Then, as $n \to \infty$ the sum of utilities under partitioning is at least as large as that under sharing.*

*Proof.* The sum of utilities for the shared case is

$$U^{(s)} = \sum_{k=1}^{K} U_k\Big(\lambda_k\big(1-\int_0^1 F_k'(x)e^{-a_k F_k'(x)\tau_0 B/b_k}dx\big)\Big)$$

where $\tau_0$ is the unique solution to

$$\beta = 1 - \sum_{k=1}^{K} \frac{b_k}{B}\int_0^1 e^{-a_k F_k'(x)\tau_0 B/b_k}dx.$$

When we set $\tau_k = a_k\tau_0 B/b_k$ in $U^{(p)}$ then $U^{(p)} = U^{(s)}$, proving the theorem. □

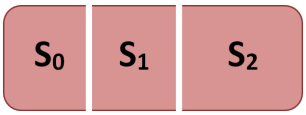



Figure 2: Partitioning cache into three slices. One partition for the set of common files, $S_0$, and two other partitions, one for the remaining files from each content provider, $S_k$.

Based on the above theorem, we focus solely on partitioned caches and use the CT approximation to formulate the utility–maximizing resource allocation problem for content providers with distinct files as follows:

$$\text{maximize} \quad \sum_{k=1}^{K} U_k\Big(h_k(C_k)\Big) \tag{11}$$
$$\text{such that} \quad \sum_{k=1}^{K} C_k \leq C,$$
$$C_k \geq 0, \quad k = 1, 2, \ldots, K.$$

In our formulation, we assume that each partition employs LRU for managing the content. Therefore, we can compute the hit rate for content provider $k$ as

$$h_k(C_k) = \lambda_k \sum_{i=1}^{n_k} p_{k,i}\big(1 - e^{-\lambda_k p_{k,i} T_k(C_k)}\big), \tag{12}$$

where $T_k(C_k)$ denotes the characteristic time of the partition with size $C_k$ dedicated to content provider $k$. $T_k(C_k)$ is the unique solution to the equation

$$C_k = \sum_{i=1}^{n_k}(1 - e^{-\lambda_k p_{k,i} T_k}). \tag{13}$$

The following theorem establishes that resource allocation problem (11) has a unique optimal solution:

**Theorem 3.** *Given strictly concave utility functions, resource allocation problem* (11) *has a unique optimal solution.*

*Proof.* In Appendix A, we show that $h_k(C_k)$ is an increasing concave function of $C_k$. Since $U_k$ is assumed to be an increasing and strictly concave function of the cache hit rate $h_k$, it follows that $U_k$ is an increasing and strictly concave function of $C_k$. The objective function in (11) is a linear combination of strictly concave functions, and hence is concave. Meanwhile, as the feasible solution set is convex, a unique maximizer called the optimal solution, exists. □

Last, it is straightforward to show that partitioning is at least as good as sharing, for finite size systems using the CT approximation.

## 3.2 Content Providers with Common Objects

Here, we first assume there are only two content providers in the network and then consider the gen-

eral case. There are three sets of content, $S_0$ of size $n_0$ served by both providers, and $S_1$ and $S_2$, sizes $n_1$ and $n_2$ served separately by each of the providers. Requests are made to $S_0$ at rate $\lambda_{0,k}$ from provider $k$ and to $S_k$ at rate $\lambda_k$. Given a request is made to $S_0$ from provider $k$, it is for content $i$ with probability $p_{0,k,i}$. Similarly, if the request is for content in $S_k$, $k = 1, 2$, it is for content $i$ with probability $p_{k,i}$.

We have two conceivable cases for the files in $S_0$: 1) each content provider needs to maintain its own copy of the content, *e.g.*, due to security reasons, or 2) one copy can be kept in cache to serve requests to either of the content providers. The first case can be treated as if there is no common content between the two content providers, and hence can be cast as problem (11). For the second case, we consider three strategies for managing the cache:

- **Strategy 1 (S1):** sharing the whole cache as one large partition,
- **Strategy 2 (S2):** partitioning into two dedicated slices, one for each CP,
- **Strategy 3 (S3):** partitioning into three slices, one shared partition for the set of common contents, and two other partitions for the remaining files of each CP, as shown in Figure 2.

The following theorem states that **S3** performs at least as well as **S1** in an asymptotic sense.

**Theorem 4.** *Assume that we have two providers with a set of shared files $S_0$, and sets of non-shared files , $S_1, S_2$ with numbers of files $n_k = b_k n$, $b_k, n \in \mathbb{N}$, and $k = 0, 1, 2$. Assume that requests to these sets occur with rates $\lambda_{0,k}$ and $\lambda_k$ and content popularities are described by asymptotic popularity distributions $F_{0,k}$, and $F_k$ . Construct the sequence of popularity probabilities $\{p_{k,i}^{(n)}\}$, $n = 1, \ldots$ similar to (5) and cache sizes $C^{(n)}$ such that $C^{(n)}/(Bn) = \beta$. Then, the asymptotic aggregate LRU miss probability is at least as small under* **S3** *as under* **S1**.

The proof is similar to the proof of Theorem 2.

Neither **S2** nor **S3** outperforms the other for all problem instances, even asymptotically. However, we present a class of workloads for which asymptotically **S3** outperforms **S2** and then follow it with an example where **S2** outperforms **S3**.

Consider the following workload where the asymptotic popularity distributions of requests to the two providers for the shared content are identical, $F_{0,1} = F_{0,2}$.

**Theorem 5.** *Assume that we have two providers with a set of shared files $S_0$ and sets of non-shared files, $S_1, S_2$ with numbers of files $n_k = b_k n$, $b_k \in \mathbb{N}$, $n = 1, \ldots,$ and $k = 0, 1, 2$. Assume that requests are described by Poisson processes with rates $\lambda_{0,k}$ and $\lambda_k$, $k = 1, 2$, and content popularities are described by asymptotic popularity distributions $F_{0,1} = F_{0,2}$, and $F_1, F_2$. Construct the sequence of popularity probabilities $\{p_{k,i}^{(n)}\}$, $n = 1, \ldots$ similar to (5) and cache sizes $C^{(n)}$ such that $C^{(n)}/(Bn) = \beta$. Then the asymptotic aggregate hit probability under LRU is at least as large under* **S3** *as under* **S2**.

The proof is found in Appendix C

Below is an example where **S2** outperforms **S3**. The asymptotic popularity distributions for the shared content are given by

$$F_{0,1}(x) = \begin{cases} 2x/11 & 0 < x \leq 1/2 \\ (20x - 9)/11 & 1/2 < x < 1 \end{cases}$$

and

$$F_{0,2}(x) = \begin{cases} 300x/151 & 0 < x \leq 1/2 \\ (2x + 149)/151 & 1/2 < x < 1 \end{cases}$$

with request rates $\lambda_{0,1} = 1.1$ and $\lambda_{0,2} = 15.1$. The asymptotic popularities of the non-shared contents are $F_1(x) = F_2(x) = x$ with request rates $\lambda_1 = 20$ and $\lambda_2 = 30$. Last, there are equal numbers of content in each of these sets, $n_0 = n_1 = n_2$. If we set $\beta = 2/3$, then the aggregate hit probability under **S3** with optimal partitioning is .804, which is slightly lower than the aggregate hit probability, .816, under **S2** with optimal partitioning.

The above examples show that the workloads of the content providers can affect which strategy is optimal. However, we argue that partitioning into three slices should provide the best performance in most practical situations, where content providers have similar popularity patterns for the contents they commonly serve. This is unlike the second example where the two content providers have disparate rates for the common contents they serve. In Section 5, we will show that even if two content providers have dissimilar request rates for their common contents, partitioning into three slices does better.

Based on the above argument for the performance of partitioning into three slices in the case of two content providers, for $K$ content providers with common files, one should create a partition for each set of files that are served by a number of content providers. A procedure for creating the optimal set of partitions $\mathcal{P}$ with the files routed to each partition is given in Algorithm 1. Algorithm 1 runs in $O(|S|^2)$ where $S$ denotes the set of all files served by all content providers. Note that the number of partitions can grow exponentially with the number of content providers.

Once the set of partitions $\mathcal{P}$ and the set of files corresponding to each partition is determined, the optimal

**Algorithm 1** Partitioning a cache serving $K$ content providers with possibility of common files among some content providers.

$S \leftarrow S_1 \cup S_2 \cup \ldots \cup S_K$.
$\mathcal{P} \leftarrow \emptyset$.
**for** $f \in S$ **do**
    $M_f \leftarrow \{k :$ Content provider $k$ serves files $f\}$.
    **if** Exists $(V, M) \in P$ such that $M = M_f$ **then**
        $V \leftarrow V \cup \{f\}$.
    **else**
        $\mathcal{P} \leftarrow \mathcal{P} \cup \{(\{f\}, M_f)\}$.

partition sizes can be computed through the following optimization problem:

$$\begin{aligned} \text{maximize} \quad & \sum_k U_k(h_k) \qquad (14) \\ \text{such that} \quad & \sum_{p=1}^{|\mathcal{P}|} C_p \leq C \\ & C_p \geq 0, \quad p = 1, 2, \ldots, |\mathcal{P}|, \end{aligned}$$

where hit rate for CP $k$ is computed as

$$h_k = \sum_{p=1}^{|\mathcal{P}|} \lambda_k \sum_{i \in V_p} p_{k,i}\big(1 - e^{-\lambda_i p_{k,i} T_p(C_p)}\big)$$

where $V_p$ denotes the set of files requested from partition $p$, and $\lambda_i \triangleq \sum_k \lambda_{k,i}$ denotes the aggregate request rate for content $i$ through all content providers, and $T_p$ denotes the characteristic time of partition $p$.

**Theorem 6.** *Given strictly concave utility functions, resource allocation problem* (14) *has a unique optimal solution.*

*Proof.* In Appendix D, we show that the optimization problem (14) has a concave objective function. Since the feasible solution set is convex, a unique maximizer exists. □

## 3.3 Implications

**Cache Partitioning: Number of Slices, Management Complexity and Static Caching.** In our utility maximization formulations (11) and (14) and their solution, the cache is only partitioned and allocated per CP for a set of *distinct* content objects owned by the CP; a cache slice is allocated and shared among several CPs only for a set of *common* content objects belonging to these CPs. This is justified by cache management complexity considerations, as further partitioning of a slice allocated to a CP to be exclusively utilized by the same CP simply incurs additional management complexity. In addition, we show in Appendix F that partitioning a cache slice into smaller slices and *probabilistically routing* requests to content objects of a CP is sub-optimal.

Table 2: $\alpha$-fair utility functions

| $\alpha$ | $U_k(h_k)$ | implication |
|---|---|---|
| 0 | $h_k$ | hit rate |
| 1 | $\log h_k$ | proportional fairness |
| 2 | $-1/h_k$ | potential delay |
| $\infty$ | $\lim_{\alpha\to\infty} \frac{h_k^{1-\alpha}-1}{1-\alpha}$ | max-min fairness |

As an alternative to CP-oriented cache allocation and partitioning approach, one could adopt a per-object cache allocation and partitioning approach (regardless of the CP or CPs which own the objects). Under such an approach, it is not hard to show that the optimal *per-object* cache allocation strategy that maximizes the overall cache hit rate is equivalent to the *static caching* policy [18]: the cache is only allocated to the $C$ most popular objects among all content providers. Alternatively, such a solution can also be obtained using the same CP-oriented, utility maximization cache allocation framework where only the most popular content from each provider is cached.

**Utility Functions and Fairness.** Different utility functions in problems (11) and (14) yield different partition sizes for content providers. In this sense, each utility function defines a notion of fairness in allocating storage resources to different content providers. The family of $\alpha$-fair utility functions expressed as

$$U(x) = \begin{cases} \frac{x^{1-\alpha}-1}{1-\alpha} & \alpha \geq 0, \alpha \neq 1; \\ \log x & \alpha = 1, \end{cases}$$

unifies different notions of fairness in resource allocation [24]. Some choices of $\alpha$ lead to especially interesting utility functions. Table 2 gives a brief summary of these functions. We will use these utilities in Section 5 to understand the effect of particular choices for utility functions, and in evaluating our proposed algorithms.

# 4. ONLINE ALGORITHMS

In the previous section, we formulated cache partitioning as a convex optimization problem. However, it is not feasible to solve the optimization problem offline and then implement the optimal strategy. Moreover, system parameters can change over time. Therefore, we need algorithms that can implement the optimal strategy and adapt to changes in the system by collecting limited information. In this section, we develop such algorithms.

## 4.1 Content Providers with Distinct Contents

The formulation in (11) assumes a hard constraint on the cache capacity. In some circumstances it may

be appropriate for the cache manager to increase the available storage at some cost to provide additional resources for the content providers. One way of doing this is to turn cache storage disks on and off based on demand [25]. In this case, the cache capacity constraint can be replaced with a penalty function $P(\cdot)$ denoting the cost for the extra cache storage. Here, $P(\cdot)$ is assumed to be convex and increasing. We can now write the utility and cost driven caching formulation as

$$\begin{aligned} \text{maximize} \quad & \sum_k U_k(h_k(C_k)) - P(\sum_k C_k - C) \\ \text{such that} \quad & C_k \geq 0, \quad k = 1, \ldots, K. \end{aligned} \tag{15}$$

Let $W(\mathbf{C})$ denote the objective function in (15) defined as

$$W(\mathbf{C}) = \sum_k U_k(h_k(C_k)) - P(\sum_k C_k - C).$$

A natural approach to obtaining the maximum value for $W(\mathbf{C})$ is to use a gradient ascent algorithm. The basic idea behind a gradient ascent algorithm is to move the variables $C_k$ in the direction of the gradient,

$$\begin{aligned} \frac{\partial W}{\partial C_k} &= \frac{\partial U_k}{\partial C_k} - P'(\sum_k C_k - C), \\ &= U_k'(h_k)\frac{\partial h_k}{\partial C_k} - P'(\sum_k C_k - C). \end{aligned}$$

Note that since $h_k$ is an increasing function of $C_k$, moving $C_k$ in the direction of the gradient also moves $h_k$ in that direction.

By gradient ascent, partition sizes should be updated according to

$$C_k \leftarrow \max\Big\{0, C_k + \gamma_k\Big[U_k'(h_k)\frac{\partial h_k}{\partial C_k} - P'(\sum_k C_k - C)\Big]\Big\},$$

where $\gamma_k$ is a step-size parameter.

**Theorem 7.** *The above gradient ascent algorithm converges to the optimal solution.*

*Proof.* Let $\mathbf{C}^*$ denote the optimal solution to (15). We show in Appendix E that $W(\mathbf{C}^*) - W(\mathbf{C})$ is a Lyapunov function, and the above algorithm converges to the optimal solution. □

#### 4.1.1 Algorithm Implementation

In implementing the gradient ascent algorithm, we restrict ourselves to the case where the total cache size is $C$. Defining $\eta \triangleq P'(0)$, we can re-write the gradient ascent algorithm as

$$C_k \leftarrow \max\Big\{0, C_k + \gamma_k\Big[U_k'(h_k)\frac{\partial h_k}{\partial C_k} - \eta\Big]\Big\}.$$

In order to update $C_k$ then, the cache manager needs to estimate $\frac{\partial U_k}{\partial C_k} = U_k'(h_k)\frac{\partial h_k}{\partial C_k}$ by gathering hit rate information for each content provider. Instead of computing $U_k'(h_k)$ and $\partial h_k/\partial C_k$ separately, however, we suggest using

$$\frac{\partial U_k}{\partial C_k} \approx \frac{\Delta U_k}{\Delta C_k} = \frac{U_k(h_k^t) - U_k(h_k^{t-1})}{C_k^t - C_k^{t-1}},$$

where the superscripts $t$ and $t-1$ denote the iteration steps. We then use $\frac{\Delta U_k}{\Delta C_k}$ as an estimate of $U_k'(h_k)\frac{\partial h_k}{\partial C_k}$ to determine the value of $C_k$ at the next iteration.

Moreover, since we impose the constraint that $\sum_k C_k = C$, we let $\eta$ take the mean of the $\frac{\Delta U_k}{\Delta C_k}$ values. The algorithm reaches a stable point once the $\frac{\Delta U_k}{\Delta C_k}$s are equal or very close to each other. Algorithm 2 shows the rules for updating the partition sizes.

**Algorithm 2** Online algorithm for updating the partition sizes.

1: Start with an initial partitioning $\mathbf{C}^0 \leftarrow (C_1 \cup C_2 \cup \ldots \cup C_K)$.
2: Estimate hit rates for each partition by counting the number of hit requests $\mathbf{h}^0 \leftarrow (h_1, \ldots, h_K)$.
3: Make arbitrary changes to the partition sizes $\mathbf{\Delta}^0$, such that $\sum_k \mathbf{\Delta}_k^0 = 0$, $\mathbf{C}^1 \leftarrow \mathbf{C}^0 + \mathbf{\Delta}^0$.
4: Estimate the hit rates $\mathbf{h}^1$ for the new partition sizes.
5: $t \leftarrow 1$.
6: $\delta_k^t \leftarrow \Big(U_k(h_k^t) - U_k(h_k^{t-1})\Big)/\Big(C_k^t - C_k^{t-1}\Big)$.
7: $\eta^t \leftarrow \Big(\sum_k \delta_k^t\Big)/K$.
8: **if** $\max_k \{\delta_k^t - \eta^t\} > \epsilon$ **then**
9: $\quad \mathbf{\Delta}_k^t = \gamma(\delta_k^t - \eta^t)$.
10: $\quad \mathbf{C}^{t+1} \leftarrow \mathbf{C}^t + \mathbf{\Delta}^t$.
11: $\quad$ Estimate hit rates $\mathbf{h}^{t+1}$.
12: $\quad t \leftarrow t + 1$.
13: $\quad$ **goto** 6.

### 4.2 Content Providers with Common Content

We now focus on the case where some contents can be served by multiple content providers. Algorithm 1 computes the optimal number of partitions for this case. Let $\mathcal{P}$ and $\mathbf{C} = (C_1, \ldots, C_{|\mathcal{P}|})$ denote the set of partitions and the vector of partition sizes, respectively. The hit rate for content provider $k$ can be written as

$$h_k(\mathbf{C}) = \sum_{p=1}^{|\mathcal{P}|} \sum_{i \in V_p} \lambda_{ik}(1 - e^{-\lambda_i T_p}),$$

where $V_p$ denotes the set of files requested from partition $p$, and $\lambda_i$ denotes the aggregate request rate at partition $p$ for file $i$.

Similar to (15), we consider a penalty function for violating the cache size constraint and rewrite the optimization problem in (14) as

$$\text{maximize} \quad \sum_k U_k(h_k(\mathbf{C})) - P(\sum_p C_p - C)$$

**Algorithm 3** Online algorithm for updating the partition sizes.

1: Compute the number of partitions $P$ using Algorithm 1.
2: Start with an initial partitioning $\mathbf{C}^0 \leftarrow (C_1 \cup C_2 \cup \ldots \cup C_P)$.
3: Estimate hit rates for each provider/partition pair $\mathbf{H}^0 \leftarrow \begin{pmatrix} h_{11} & \ldots & h_{1P} \\ \vdots & \ddots & \vdots \\ h_{K1} & \ldots & h_{KP} \end{pmatrix}$.
4: Make arbitrary changes to the partition sizes $\mathbf{\Delta}^0$, such that $\sum_p \mathbf{\Delta}_p^0 = 0$, $\mathbf{C}^1 \leftarrow \mathbf{C}^0 + \mathbf{\Delta}^0$.
5: Estimate the hit rates $\mathbf{H}^1$ for the new partition sizes.
6: $t \leftarrow 1$.
7: $\delta_p^t \leftarrow \sum_k U_k'(h_k^{t-1})(h_{kp}^t - h_{kp}^{t-1})/(C_p^t - C_p^{t-1})$.
8: $\eta^t \leftarrow \left(\sum_p \delta_p^t\right)/P$.
9: **if** $\max_p \{\delta_p^t - \eta^t\} > \epsilon$ **then**
10: $\quad \mathbf{\Delta}_p^t = \gamma(\delta_p^t - \eta^t)$.
11: $\quad \mathbf{C}^{t+1} \leftarrow \mathbf{C}^t + \mathbf{\Delta}^t$.
12: $\quad$ Estimate hit rates $\mathbf{H}^{t+1}$.
13: $\quad t \leftarrow t + 1$.
14: $\quad$ **goto** 6.

$$\text{such that} \quad C_p \geq 0, \quad p = 1, \ldots, |\mathcal{P}|. \tag{16}$$

Let $W(\mathbf{C})$ denote the objective function in the above problem. Taking the derivative of $W$ with respect to $C_p$ yields

$$\frac{\partial W}{\partial C_p} = \sum_k U_k'(h_k)\frac{\partial h_k}{\partial C_p} - P'(\sum_p C_p - C).$$

Following a similar argument as in the previous section, we can show that a gradient ascent algorithm converges to the optimal solution.

#### 4.2.1 Algorithm Implementation

The implementation of the gradient ascent algorithm in this case is similar to the one in Section 4.1.1. However, we need to keep track of hit rates for content provider $k$ from all partitions that store its files. This can be done by counting the number of hit requests for each content provider and each partition through a $K \times |\mathcal{P}|$ matrix, as shown in Algorithm 3. Also, we propose estimating $\partial W/\partial C_p$ as

$$\frac{\partial W}{\partial C_p} \approx \sum_k U_k'(h_k)\frac{\Delta h_{kp}}{\Delta C_p},$$

where $\Delta h_{kp}$ denotes the change in aggregate hit rate for content provider $k$ from partition $p$ resulted from changing the size of partition $p$ by $\Delta C_p$.

## 5. EVALUATION

In this section, we perform numerical simulations, first to understand the efficacy of cache partitioning on the hit rate and utility observed by content providers, and second to evaluate the performance of our proposed online algorithms.

### 5.1 Hit Rate Improvement

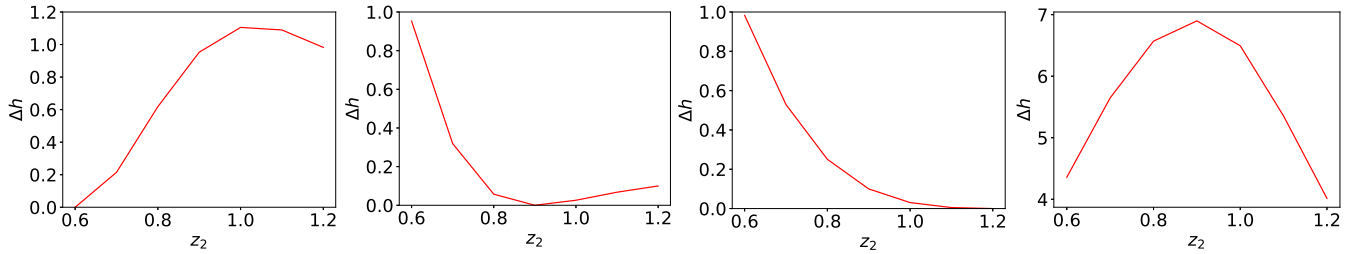
To show hit rate improvement, we consider a caching system where there are two content providers competing for a shared LRU cache with size $C = 1000$. The two content providers serve $n_1 = n_2 = 10000$ content objects (so that 5% of content can be held in cache), and requests for these files arrive according to Poisson processes with rates $\lambda_1 = \lambda_2 = 100$. We assume that content popularity of both providers follow a Zipf's law, *i.e.*, $p_i \propto 1/i^z$, with skewness parameters $z_1$ and $z_2$, respectively.

We compare the observed aggregate hit rate of the system under both cache partitioning and sharing (non-partitioning) for a variety of content popularity pattern combinations, *i.e.*, by varying $z_2$ while fixing $z_1$ and keeping all other parameters unchanged. The results are shown in Figure 3, where the $y$ axis gives the difference of the observed aggregate hit rate for the two policies, *i.e.*, $\Delta h = hit_{partitioning} - hit_{sharing}$. Obviously we can see that: 1) in all four different content popularity pattern combinations, cache partitioning outperforms cache sharing, thus verifying our theoretical result; 2) as compared with Zipf distributions, the hit rate difference is even larger when content popularity of one provider follows a uniform distribution (see Figure 3(d) vs Figure 3(a), 3(b), 3(c)), which indicates that the more difference of traffic distributions of content providers, the more hit rate gains.

### 5.2 Utility-based Cache Partitioning

For understanding the efficacy of utility-based cache partitioning, we setup a base case where an LRU cache of size $C = 10^4$ is shared by two content providers that serve $n_1 = 10^4$ and $n_2 = 2 \times 10^4$ contents. The content popularities follow Zipf distributions with parameters $z_1 = 0.6$ and $z_2 = 0.8$, respectively. Requests for the files from them arrive as Poisson processes with rates $\lambda_1 = 15$ and $\lambda_2 = 10$. The utilities of the two content providers are $U_1(h_1) = w_1 \log h_1$ and $U_2(h_2) = h_2$. Unless otherwise specified, we let $w_1 = 1$ so that the two content providers are equally important to the service provider.

We consider two scenarios here. In the first scenario, the two content providers serve completely separate files. In the second scenario, files $S_0 = \{1, 4, 7, \ldots, 10^4\}$, are served by both providers. For each scenario the appropriate optimization formulation is chosen.




(a) $z_1 = 0.6$ (b) $z_1 = 0.9$ (c) $z_1 = 1.2$ (d) uniform distribution of CP 1

Figure 3: Observed aggregate hit rate gains under different combinations of content popularity pattern

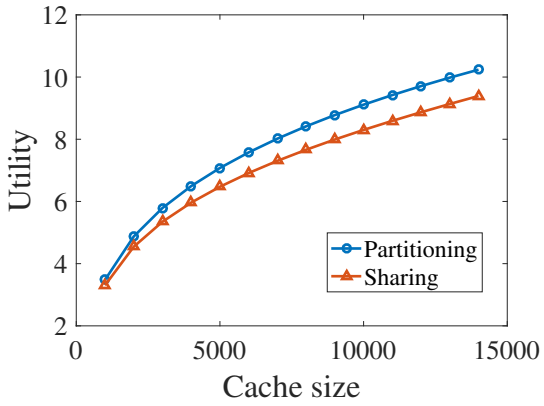
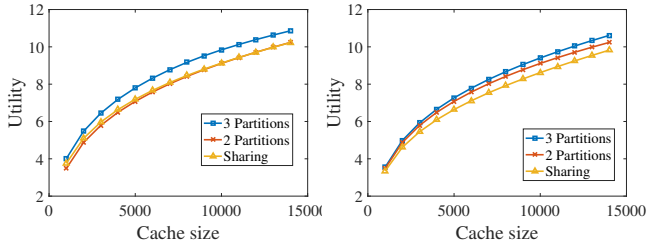



Figure 4: Efficacy of cache partitioning when content providers serve distinct files.




(a) (b)

Figure 5: Efficacy of cache partitioning when some content is served by both content providers. Request rates for the common contents from the two content providers are set to be (a) similar, and (b) dissimilar.

We first look at solutions of optimization problems (11) and (14). Here, we measure the gain in total utility through partitioning the cache by computing the utility obtained by sharing the cache between the content providers and the utility obtained by partitioning the cache. Figure 4 shows the utility gain when content providers serve distinct files. In this example, the aggregate utility increases by 10% from partitioning the cache.

Figure 5 shows the utilities for the case when files in $S_0 = \{1, 4, 7, \ldots, 10^4\}$ are served by both content providers. Two cases are considered here: a) request rates for the common content are similar for two content providers. This is done by letting $p_{k,1} > p_{k,2} > \ldots > p_{k,n_0}$ for both providers. b) Requests rates from the two content providers for the common files are set to be dissimilar. This is done by setting the file popularities for the second CP as $p_{2,1} < p_{2,2} < \ldots < p_{2,n_0}$. In both cases partitioning the cache into three slices shows the best performance.

We next look at the effect of various parameters on cache partitioning, when CPs serve distinct contents and when they serve some common content with similar popularities. We fix the parameters of the second content provider, and study the effect of changing weight parameter $w_1$ and aggregate request rate $\lambda_1$ of the first content provider. We also change the Zipfian file popularity distribution parameter $z_1$. To study the effect of the utility function, we take it to be the $\alpha$–fair utility function and vary $\alpha$ for the first content provider, $\alpha_1$.

Figure 6 shows how hit rates and partition sizes of the two content providers vary as functions of $w_1$, $\lambda_1$, $z_1$ and $\alpha_1$. As expected, by increasing the weight $w_1$, content provider one gets a larger share of the cache, and hence a higher hit rate. Increasing $\lambda_1$ has no effect on the partition sizes. This is because the first content provider uses the log utility function, and it is easy to see that the derivative $U_1'(h_1)\partial h_1/\partial C_p$ does not depend on the aggregate rate. In our example, changing the aggregate request rate for the second content provider with $U_2(h_2) = h_2$ results in different partition sizes. As the popularity distribution for contents from

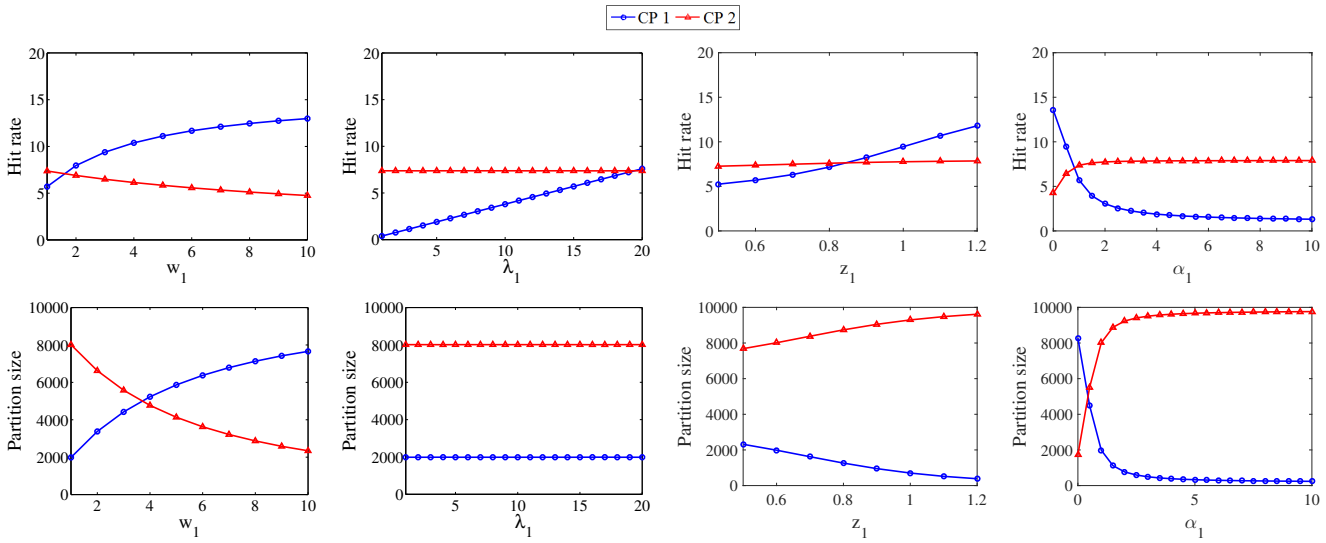



Figure 6: Effect of the parameters on hit rates and partition sizes when content providers serve distinct files.

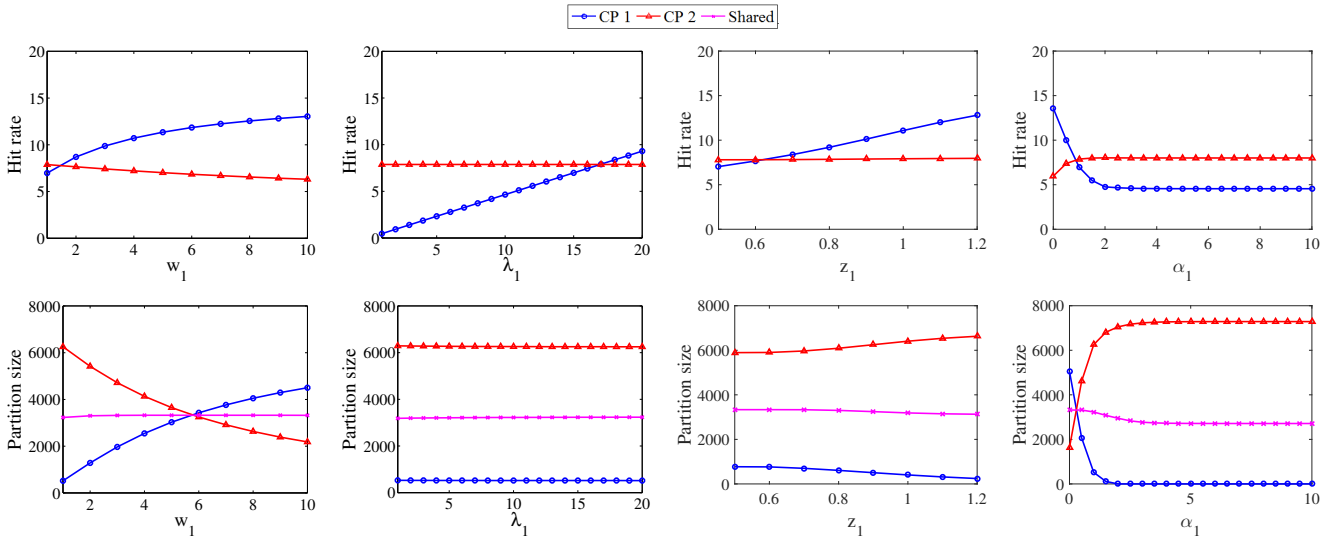



Figure 7: Effect of the parameters on hit rates and partition sizes when some content is served by both content providers.

the first content provider becomes more skewed, *i.e.*, as $z_1$ increases, the set of popular files decreases in size. Consequently, the dedicated partition size for content provider one decreases as $z_1$ increases. Increasing $\alpha_1$ changes the notion of fairness between the two content providers in favor of the second content provider, and the size of the partition allocated to the first content provider and its hit rate decreases as $\alpha_1$ increases.

Figure 7 repeats the same experiment for the case when some common content is served by both content providers. The cache is partitioned into three slices in this case, one of them storing common content. Very similar behavior as in Figure 6 is observed here.

To understand the fairness notion of the $\alpha$-fair utility functions, we next use the same utility function for both of the content providers, and vary the value of $\alpha$ to see how the hit rates and partition sizes change. Figure 8 shows the effect of $\alpha$ on hit rates and partition sizes for the case when content providers serve distinct files. As $\alpha$ increases, partition sizes change so that hit rates become closer to each other. This is expected since the $\alpha$-fair utility function realizes the max-min notion of fairness as $\alpha \to \infty$.

Figure 9 shows the changes in resource allocation based on the $\alpha$-fair notion of fairness when common content is served by the content providers.

## 5.3 Online Algorithms

Here, we evaluate the online algorithms presented in Section 4 through numerical simulations. Requests are generated according to the parameters presented in the beginning of the section, and the service provider ad-

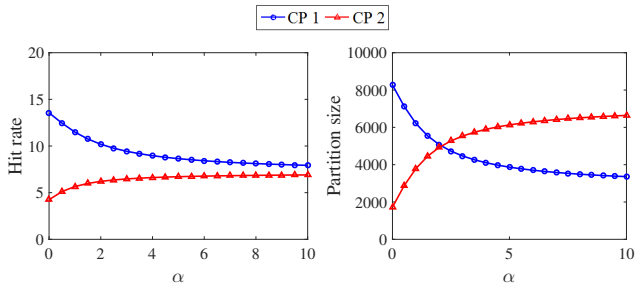



Figure 8: $\alpha$-fair resource allocation for content providers serving distinct content. $U_k(h_k) = h_k^{1-\alpha}/(1-\alpha)$.

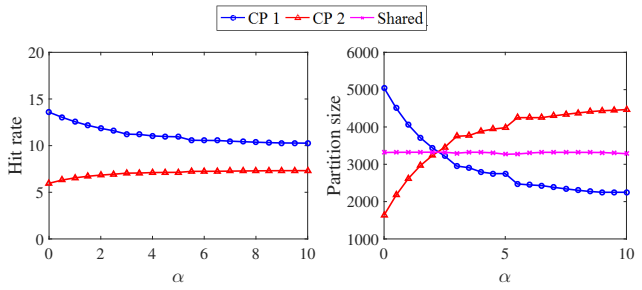



Figure 9: $\alpha$-fair resource allocation when some content is served by both content providers. $U_k(h_k) = h_k^{1-\alpha}/(1-\alpha)$.

justs partition sizes based on the number of hits between iterations. The service provider is assumed to know the utility functions of the content providers. The utility function of the first content provider is fixed to be $U_1(h_1) = \log h_1$. We consider three utility functions for the second content provider, namely $U_2(h_2) = h_2$, $U_2(h_2) = \log h_2$ and $U_2(h_2) = -1/h_2$.

We first consider the case where content providers serve distinct files. We initially partition the cache into two equal size slices $C_1 = C_2 = 5000$ and use Algorithm 2 to obtain the optimal partition sizes. Figure 10 shows how the partition sizes for the two content providers change at each iteration of the algorithm and that they converge to the optimal values computed from (11), marked with dashed lines.

Next, we consider the case where some content is served by both content providers. We first partition the cache into three slices of sizes $C_1 = C_2 = 4000$ and $C_3 = 2000$, where slice 3 serves the common content, and use Algorithm 3 to obtain the optimal partitioning. Figure 11 shows the changes in the three partitions as the algorithm converges to a stable point. For each partition the optimal size computed by (14) is shown by dashed lines.

## 5.4 Performance Over Real World Traces

We also conduct a trace-based simulation to investigate the performance of our online algorithm over real-world traffic. The trace consists of 8626163 requests accessing 142040 content objects, with a time duration of 268 minutes 20 seconds. The cache size is set as 7000 and the sampling interval of the online algorithm is 5 minutes. In the first experiment we randomly partition the content into two disjoint sets with the same sizes and allocate them to CP 1 and CP 2 respectively. Figure 12 shows how partition sizes evolve as time goes on. Clearly we can see that our algorithm adapts to workload changes as real-world traffic is often non-stationary. In the second experiment 30% of the content objects are selected as the shared content among the two providers. CP 1 accounts for 40% of the requests for these shared content while CP 2 accounts for 60%. The performance in this case is given in Figure 13. Again we observe our algorithm adapts to workload changes. The trace-based simulation suggests that our algorithm can be adopted by cache providers for managing their cache resources.

# 6. DISCUSSIONS AND RELATED WORK

In this section, we explore the implications of utility-driven cache partitioning on monetizing caching service and present some future research directions. We end with a brief discussion of the related work.

**Decomposition.** The formulation of the problem in Section 3 assumes that the utility functions $U_k(\cdot)$ are known to the system. In reality the content providers may not want to reveal their utility functions to the service provider. To handle this case, we decompose optimization problem (11) into two simpler problems.

Suppose that cache storage is offered as a service and the service provider charges content providers at a constant rate $r$ for storage space. Hence, a content provider needs to pay an amount of $w_k = rh_k$ to obtain hit rate $h_k$. The utility maximization problem for content provider $k$ can be written as

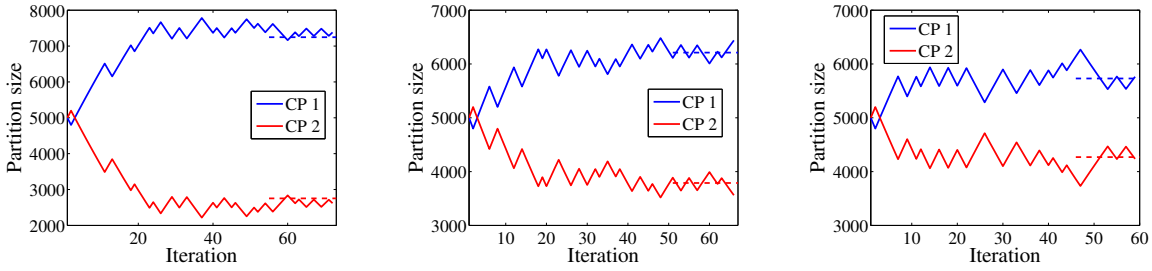
$$\begin{aligned} \text{maximize} \quad & U_k(\frac{w_k}{r}) - w_k \\ \text{such that} \quad & w_k \geq 0 \end{aligned} \tag{17}$$

Now, assuming that the service provider knows the vector $\mathbf{w}$, for a proportionally fair resource allocation, the hit rates should be set according to

$$\begin{aligned} \text{maximize} \quad & \sum_{k=1}^{K} w_k \log(h_k) \\ \text{such that} \quad & \sum_p C_p = C. \end{aligned} \tag{18}$$

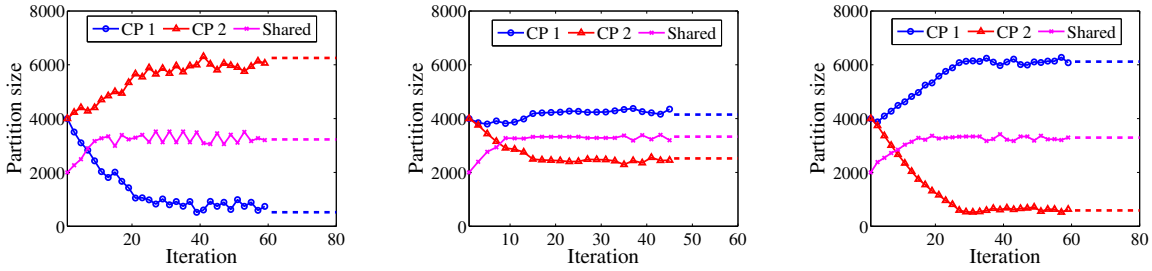
It was shown in [15] that there always exist vectors $\mathbf{w}$ and $\mathbf{h}$, such that $\mathbf{w}$ solves (17) and $\mathbf{h}$ solves (18); furthermore, the vector $\mathbf{h}$ is the unique optimal solution.

**Cost and Utility Functions.** In Section 4, we defined a penalty function denoting the cost of using additional storage space. One might also define cost functions based on the consumed network bandwidth. This



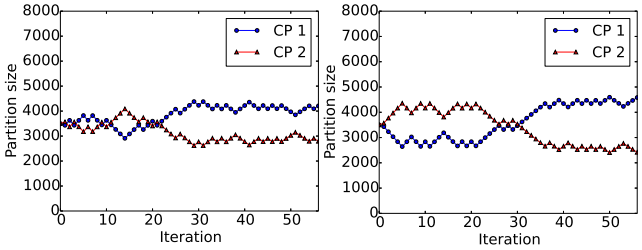

(a) $U_2(h_2) = h_2$. (b) $U_2(h_2) = \log h_2$. (c) $U_2(h_2) = -1/h_2$.

Figure 10: Convergence of the online algorithm when content providers serve distinct files. $U_1(h_1) = \log h_1$.



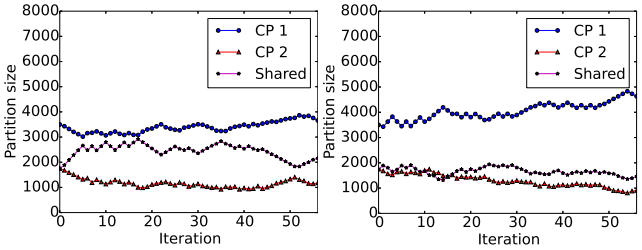

(a) $U_2(h_2) = h_2$. (b) $U_2(h_2) = \log h_2$. (c) $U_2(h_2) = -1/h_2$.

Figure 11: Convergence of the online algorithm when some content is served by both content providers. $U_1(h_1) = \log h_1$.




(a) $U_2(h_2) = h_2$. (b) $U_2(h_2) = \log h_2$.

Figure 12: Performance of the online algorithm over real-world trace when content providers serve disjoint content. $U_1(h_1) = h_1$.




(a) $U_2(h_2) = h_2$. (b) $U_2(h_2) = \log h_2$.

Figure 13: Performance of the online algorithm over real-world trace with some content served by both providers. $U_1(h_1) = h_1$.

is especially interesting in modeling in-network caches with network links that are likely to be congested.

Optimization problems (11) and (14) use utility functions defined as functions of the hit rate. It is reasonable to define utility as a function of the hit *probability*. Whether this significantly changes the problem, *e.g.*, in the notion of fairness, is a question that requires further investigation. One argument in support of utilities as functions of hit rates is that a service provider might prefer pricing based on request rate rather than the cache occupancy. Moreover, in designing hierarchical caches a service provider's objective could be to minimize the internal bandwidth cost. This can be achieved by defining the utility functions as $U_k = -P_k(m_k)$ where $P_k(m_k)$ denotes the cost associated with miss rate $m_k$ for content provider $k$.

**Related Work.** Internet cache management issues have been extensively studied in the context of web caching (e.g., see [8, 13] and references therein). In this context, biased replacement policies for different kinds of content classes [16] and differentiated caching services via cache partitioning [17,19] haven been proposed and studied. None of these studies explicitly deal with the cache allocation problem among multiple content providers. The emergence of content-oriented networking has renewed research interests in cache management issues for content delivery, especially in the design of cache replacement policies for the content-oriented architecture [6, 7, 20, 26]. The cache allocation problem among content providers has attracted relatively little

attention. Perhaps most closely related to our work is the study in [14] where a game-theoretic cache allocation approach is developed. This approach requires the content providers to report the true demands from their content access. In contrast, we develop a general utility maximization framework for studying the cache allocation problem. Since its first proposal by Kelly *et al.* [15], the network utility maximization framework has been applied to a variety of networking problems from stability analysis of queues [11] to the study of fairness in network resource allocation [21]. A utility maximization framework for caching policies was developed in [10] to provide differentiated services to content. This framework was adopted by [9] to study the cache resource allocation problem in an informal and heuristic manner. We make precise statements to support the observations in [9]. In this respect, our contribution lies in establishing the key properties of CP utilities as a function of cache sizes and in postulating cache partitioning as a basic principle for cache sharing among content providers. Furthermore, we develop decentralized algorithms to implement utility-driven cache partitioning, and prove that they converge to the optimal solution.

## 7. CONCLUSION

We proposed utility-based partitioning of a cache among content providers, and formulated it as an optimization problem with constraints on the service provider's cache storage size. Utility-driven cache partitioning provides a general framework for managing a cache with considerations of fairness among different content providers, and has implications on market economy for service providers and content distributors. We considered two scenarios where 1) content providers served disjoint sets of files, or 2) some content was served by multiple content providers. Using Fagin's asympototic framework we showed that caching performance can be imporved by partitioning the cache. We then developed decentralized algorithms for each scenario to implement utility-driven cache partitioning in an online fashion. These algorithms adapt to changes in request rates of content providers by dynamically adjusting the partition sizes. We theoretically proved that these algorithms are globally stable and converge to the optimal solution, and through numerical evaluations illustrated their efficiency.

## 8. REFERENCES


[1] Cisco visual networking index: Forecast and methodology, 2014-2019. White paper, May 2015.

[2] Adhikari, V. K., Chen, Y., Jain, S., and Zhang, Z.-L. Vivisecting YouTube: An active measurement study. In *INFOCOM* (2012).

[3] Adhikari, V. K., Guo, Y., Hao, F., Hilt, V., and Zhang, Z.-L. A tale of three CDNs: An active measurement study of Hulu and its CDNs. In *Global Internet Symposium* (2012).

[4] Adhikari, V. K., Guo, Y., Hao, F., Varvello, M., Hilt, V., Steiner, M., and Zhang, Z.-L. Unreeling Netflix: Understanding and improving multi-CDN movie delivery. In *INFOCOM* (2012).

[5] Bengt Ahlgren *et al.* A survey of information-centric networking. *Comm. Magazine* (2012).

[6] Borst, S., Gupta, V., and Walid, A. Distributed caching algorithms for content distribution networks. In *INFOCOM* (2010).

[7] Carofiglio, G., Gehlen, V., and Perino, D. Experimental evaluation of memory management in content-centric networking. In *ICC* (2011).

[8] Che, H., Wang, Z., and Tung, Y. Analysis and design of hierarchical web caching systems. In *INFOCOM* (2001), pp. 1416–1424.

[9] Chu, W., Dehghan, M., Towsley, D., and Zhang, Z.-L. On allocating cache resources to content providers. In *ACM ICN* (2016), pp. 154–159.

[10] Dehghan, M., Massoulie, L., Towsley, D., Menasche, D., and Tay, Y. A utility optimization approach to network cache design. In *INFOCOM* (2016).

[11] Eryilmaz, A., and Srikant, R. Fair resource allocation in wireless networks using queue-length-based scheduling and congestion control. *IEEE/ACM Transaction on Networking 15*, 6 (December 2007), 1333–1344.

[12] Fagin, R. Asymptotic miss ratios over independent references. *Journal of Computer and System Sciences 14*, 2 (1977), 222–250.

[13] Feldman, M., and Chuang, J. Service differentiation in web caching and content distribution. In *IASTED CCN* (2002).

[14] Hoteit, S., El Chamie, M., Saucez, D., and S., S. On fair network cache allocation to content providers. *INRIA Technical Report* (2015).

[15] Kelly, F. Charging and rate control for elastic traffic. *European Transactions on Telecommunications 8* (1997), 33–37.

[16] Kelly, T., Chan, Y. M., Jamin, S., and MacKie-Mason, J. K. Biased replacement policies for web caches: Differential quality-of-service and aggregate user value. In *Web Caching Workshop* (1999).

[17] Ko, B.-J., Lee, K.-W., Amiri, K., and Calo, S. Scalable service differentiation in a shared storage cache. In *ICDCS* (2003), pp. 184–193.

[18] Liu, Z., Nain, P Niclausse, N., and Towsley, D. Static caching of web servers. In

*Multimedia Computing and Networking Conference* (1998).

[19] Lu, Y., Abdelzaher, T., and Saxena, A. Design, implementation, and evaluation of differentiated caching services. *IEEE Transactions on Parallel and Distributed Systems 15*, 5 (May 2004), 440–452.

[20] Martina, V., Garetto, M., and Leonardi, E. A unified approach to the performance analysis of caching systems. In *INFOCOM* (2014).

[21] Neely, M. J. *Stochastic Network Optimization with Application to Communication and Queueing Systems*. Morgan and Claypool Publishers, 2010.

[22] Ramadan, E., Narayanan, A., and Zhang, Z.-L. CONIA: Content (provider)-oriented, namespace-independent architecture for multimedia information delivery. In *MuSIC Workshop* (2015), pp. 1–6.

[23] Sandvine. Global internet phenomena report, 2014.

[24] Srikant, R., and Ying, L. *Communication Networks: An Optimization, Control, and Stochastic Networks Perspective*. Cambridge University Press, 2013.

[25] Sundarrajan, A., Kasbekar, M., and Sitaraman, R. Energy-efficient disk caching for content delivery. In *ACM e-Energy* (2016).

[26] Zhang, G., Li, Y., and Lin, T. Caching in information centric networking: A survey. *Computer Networks 57*, 16 (2013), 3128–3141.

[27] Zhang, Z.-L. Feel free to cache: Towards an open CDN architecture for cloud-based content distribution. In *CTS* (2014).

# APPENDIX

## A. HIT RATE IS A CONCAVE AND INCREASING FUNCTION OF CACHE SIZE.

**Lemma 1.** *The hit rate $h_k$ is a concave and strictly increasing function of $C_k$.*

*Proof.* Differentiating (12) and (13) w.r.t. $C_k$ gives

$$\begin{aligned}\frac{dh_k(C_k)}{dC_k} &= \lambda_k^2 \sum_{i=1}^{n_k} p_{k,i}^2 e^{-\lambda_k p_{k,i} T_k(C_k)} \frac{dT_k(C_k)}{dC_k} \\ 1 &= \lambda_k \sum_{i=1}^{n_k} p_{k,i} e^{-\lambda_k p_{k,i} T_k(C_k)} \frac{dT_k(C_k)}{dC_k}.\end{aligned}$$

The latter equation implies that $dT_k(C_k)/dC_k > 0$, which in turn implies from the former that $dh_k(C_k)/dC_k > 0$. This proves that $h_k(C_k)$ is strictly increasing in $C_k$. Differentiating now the above equations w.r.t. $C_k$ yields

$$\begin{aligned}\frac{1}{\lambda_k^2}\frac{d^2h_k(C_k)}{dC_k^2} &= \sum_{i=1}^{n_k} p_{k,i}^2 e^{-\lambda_k p_{k,i} T_k(C_k)} g_{k,i} \\ 0 &= \sum_{i=1}^{n_k} p_{k,i} e^{-\lambda_k p_{k,i} T_k(C_k)} g_{k,i} \quad (19)\end{aligned}$$

with $g_{k,i} := d^2T_k(C_k)/dC_k^2 - \lambda_k p_{k,i}(dT_k(C_k)/dC_k)^2$. Assume without loss of generality that $0 \le p_{k,1} \le \cdots \le p_{k,n_k} \le 1$. (19) implies that there exists $1 \le l \le n_k$ such that $g_{k,i} \ge 0$ for $i = 1, \ldots, l$ and $g_{k,i} \le 0$ for $i = l+1, \ldots, n_k$. Hence,

$$\begin{aligned}\frac{1}{\lambda_k^2}\frac{d^2h_k(C_k)}{dC_k^2} &\le \sum_{i=1}^{l} p_{k,i} e^{-\lambda_k p_{k,i} T_k(C_k)} g_{k,i} \\ &\quad + \sum_{i=l+1}^{n_k} p_{k,i}^2 e^{-\lambda_k p_{k,i} T_k(C_k)} g_{k,i} \\ &= \sum_{i=l}^{n_k} p_{k,i} e^{-\lambda_k p_{k,i} T_k(C_k)} g_{k,i}(1 - p_{k,i}) \le 0.\end{aligned}$$

This proves that $h_k(C_k)$ is concave in $C_k$. □

## B. PROOF OF THEOREM 1

*Proof.* We first construct a CDF $F$ from the CP specific CDFs, $\{F_k\}$. When the providers share a single cache, documents are labelled $1, \ldots, Bn$, so that documents $B_{k-1}n+1, \ldots, B_k n$ are the $b_k n$ documents with service provider $k$. Denote $A_k := \sum_{j=1}^{k} a_j$ with $A_0 = 0$ in what follows. Define

$$F(x) = \sum_{k=1}^{K} \Big( A_{k-1} + a_k F_k\Big(\frac{B}{b_k}x - \frac{B_{k-1}}{b_k}\Big)\Big) \times \mathbf{1}\left\{\frac{B_{k-1}}{B} \le x \le \frac{B_k}{B}\right\}. \quad (20)$$

Let

$$p_i^{(n)} := F\left(\frac{i}{Bn}\right) - F\left(\frac{i-1}{Bn}\right), \quad i = 1, \ldots, Bn$$

It is easy to see that

$$p_{B_{k-1}n+i}^{(n)} = a_k\, p_{k,i}^{(n)}, \quad i = 1, \ldots, b_k n;\ k = 1, \ldots, K.$$

Note that $F$ may not be differentiable at $x \in \{B_1/B, B_2/B, \ldots B_{K-1}/B\}$ and, hence, we cannot apply the result of [12, Theorem 1] directly to our problem.

Let

$$\beta^{(s)}(n, \tau_0) = 1 - \frac{1}{Bn}\sum_{i=1}^{Bn}\Big(1 - p_i^{(n)}\Big)^{Bn\tau_0}$$

be the fraction of documents in the cache. Here, $Bn\tau_0$ corresponds to the window size in [12].

We have

$$\beta^{(s)}(n,\tau_0) = 1 - \frac{1}{Bn}\sum_{k=1}^{K}\sum_{i=1}^{b_k n}\big(1-p^{(n)}_{B_{k-1}n+i}\big)^{Bn\tau_0}$$
$$= 1 - \frac{1}{Bn}\sum_{k=1}^{K}\sum_{i=1}^{b_k n}\big(1-a_k p^{(n)}_{k,i}\big)^{Bn\tau_0}. \quad (21)$$

We are interested in $\beta^{(s)} = \lim_{n\to\infty}\beta^{(s)}(n,\tau_0)$.

By applying the result in [12, Theorem 1], we find that

$$\lim_{n\to\infty}\frac{1}{Bn}\sum_{i=1}^{b_k n}\big(1-a_k p^{(n)}_{k,i}\big)^{Bn\tau_0} = \frac{b_k}{B}\int_0^1 e^{-\tau_0 a_k B F'_k(x)/b_k}dx \quad (22)$$

for $k = 1,\ldots,K$, so that

$$\beta^{(s)} = 1 - \sum_{k=1}^{K}\frac{b_k}{B}\int_0^1 e^{-a_k F'_k(x)\tau_0 B/b_k}dx.$$

Equation (7) is derived in the same way.

Last, it follows from Theorems 2 and 4 in [12] that $\mu^{(s)}$ is the limiting aggregate miss probability under LRU as $n\to\infty$. □

## C. PROOF OF THEOREM 5

*Proof.* The optimization problems under strategies 2 and 3 are

$$\begin{aligned}\min_{\tau_1,\tau_2}\quad & \sum_{k=1}^{2}(a_{0,k}\mu^{(s2)}_{0,k}(\tau_k) + a_k\mu^{(s2)}_k(\tau_k))\\ \text{s.t.}\quad & \sum_{k=1}^{2}(\beta^{(s2)}_{0,k}(\tau_k)+\beta^{(s2)}_k(\tau_k)) \le \beta,\\ & \beta^{(s2)}_{0.k},\ \beta^{(s2)}_k \ge 0,\quad k=1,2.\end{aligned}$$

and

$$\begin{aligned}\min_{\tau_k}\quad & \sum_{k=0}^{2} a_k\mu^{(s3)}_k(\tau_k)\\ \text{s.t.}\quad & \sum_{k=0}^{2}\beta^{(s3)}_k(\tau_k) \le \beta, \quad (23)\\ & \beta^{(s3)}_k \ge 0,\quad k=0,1,2.\end{aligned}$$

where $\lambda_0 = \lambda_{0,1}+\lambda_{0,2}$, $a_{0,k} = \lambda_{0,k}/\sum_{k=0}^{2}\lambda_k$, and $a_k = \lambda_k/\sum_{k=0}^{2}\lambda_k$,

$$\mu^{(s2)}_{0,k}(\tau) = \int_0^1 F'_0(x)e^{-\frac{a_{0,k}(b_0+b_k)}{(a_{0,k}+a_k)b_0}F'_0(x)\tau}dx$$
$$\mu^{(s2)}_k(\tau) = \int_0^1 F'_k(x)e^{-\frac{a_{0,k}(b_0+b_k)}{(a_{0,k}+a_k)b_k}F'_k(x)\tau}dx$$
$$\beta^{(s2)}_{0,k}(\tau) = \frac{b_0}{2b_0+b_1+b_2}\big(1-\int_0^1 e^{-\frac{a_{0,k}(b_0+b_k)}{(a_{0,k}+a_k)b_0}F'_0(x)\tau}dx\big)$$
$$\beta^{(s2)}_k(\tau) = \frac{b_k}{2b_0+b_1+b_2}\big(1-\int_0^1 e^{-\frac{a_k(b_0+b_k)}{(a_{0,k}+a_k)b_k}F'_k(x)\tau}dx\big)$$

and

$$\mu^{(s3)}_k(\tau) = \int_0^1 F'_k(x)e^{-F'_k(x)\tau}dx$$
$$\beta^{(s3)}_k(\tau) = \frac{b_k}{b_0+b_1+b_2}\big(1-\int_0^1 e^{-F'_k(x)\tau}dx\big)$$

We make the following observation:

- $\mu^{(s2)}_{0,k}(\tau) = \mu^{(s3)}_0(\frac{a_{0,k}(b_0+b_k)}{(a_{0,k}+a_k)b_0}\tau)$, $k=1,2$,

Let $\mu^{(s2)*}$ denote the minimum miss probability under strategy 2, which is achieved with $\tau^*_1$ and $\tau^*_2$. Let $\mu^{(s3)}(\tau_1,\tau_2,\tau_3)$ denote the miss probability under strategy 3 where $\tau_k$, $k=0,1,2$ satisfy (23). Set $\tau_k = \frac{a_{0,k}(b_0+b_k)}{(a_{0,k}+a_k)b_0}\tau^*_k$, $k=1,2$ for strategy 3 and allocate provider $k$ a cache of size $\beta^{(s2)}_k$ under strategy 3 for its non-shared content. The aggregate miss probability for non-shared content is then the same under the two strategies and given by $\mu^{(s2)}_1(\tau^*_1)+\mu^{(s2)}_2(\tau^*_2)$.

Under strategy 2, the amount of shared content stored in the cache is $\beta_- = \beta^{(s2)*}_{0,1}+\beta^{(s2)*}_{0,2}$. We allocate a cache of that size to the shared content under strategy 3 and without loss of generality assume that $\mu^{(s2)}_{0,1}(\tau^*_1) \ge \mu^{(s2)}_{0,2}(\tau^*_2)$. The miss probability over all shared content under strategy 2 is

$$\sum_{k=1}^{2}\frac{a_{0,k}}{a_{0,1}+a_{0,2}}\mu^{(s2)}_{0,k}(\tau^*_k) \ge \mu^{(s2)}_{0,2}(\tau^*_2) = \mu^{(s3)}_0(\tau_2)$$

Note that strategy 3 requires only a cache of size $\beta^{(s2)}_{0,2} < \beta_-$ to achieve a smaller miss probability for the shared content than strategy 2 can realize. Adding additional storage to the shared partition can only decrease the hit probability further, thus proving the theorem.

□

## D. PROOF OF THEOREM 6

*Proof.* Let $P$ and $\mathbf{C} = (C_1,\ldots,C_P)$ denote the number of partitions and the vector of partition sizes, respectively. The hit rate for content provider $k$ in this case can be written as

$$h_k(\mathbf{C}) = \sum_{p=1}^{P}\sum_{i\in V_p}\lambda_{ik}(1-e^{-\lambda_i T_p}),$$

where $V_p$ denotes the set of files requested from partition $p$, and $\lambda_i = \sum_k \lambda_{ik}$ denotes the aggregate request rate for file $i$.

We can re-write the expression for $h_k$ as the sum of the hit rates from each partition, since distinct files are requested from different partitions. We have

$$h_k(\mathbf{C}) = \sum_{p=1}^{P} h_{kp}(C_p),$$

where $h_{kp}(C_p)$ denotes the hit rate for files requested from partition $p$ from content provider $k$. Since $h_{kp}$ is assumed to be a concave increasing function of $C_p$, $h_k$ is sum of concave functions and hence is also concave. □

## E. PROOF OF THEOREM 7

*Proof.* We first note that since $W(\mathbf{C})$ is a strictly concave function, it has a unique maximizer $\mathbf{C}^*$. Moreover $V(\mathbf{C}) = W(\mathbf{C}^*) - W(\mathbf{C})$ is a non-negative function and equals zero only at $\mathbf{C} = \mathbf{C}^*$. Differentiating $V(\cdot)$ with respect to time we obtain

$$\begin{aligned}\dot{V}(\mathbf{C}) &= \sum_k \frac{\partial V}{\partial h_k}\dot{h_k} \\ &= -\sum_k \left( U_k'(h_k) - P'(\sum_k C_k - C)\frac{\partial C_k}{\partial h_k} \right) \dot{h_k}.\end{aligned}$$

For $\dot{h_k}$ we have

$$\dot{h_k} = \frac{\partial h_k}{\partial C_k}\dot{C_k}.$$

From the controller for $C_k$ we have

$$\dot{C_k} = \gamma_k \left( U_k'(h_k)\frac{\partial h_k}{\partial C_k} - P'(\sum_k C_k - C) \right).$$

Since $\frac{\partial h_k}{\partial C_k} \geq 0$, we get

$$\begin{aligned}\dot{V}(\mathbf{C}) &= -\sum_k \gamma_k \frac{\partial h_k}{\partial C_k} \left( U_k'(h_k) - P'(\sum_k C_k - C)\frac{\partial C_k}{\partial h_k} \right)^2 \\ &\leq 0.\end{aligned}$$

Therefore, $V(\cdot)$ is a Lyapunov function, and the system state will converge to $\mathbf{C}^*$ starting from any initial condition. A description of Lyapunov functions and their applications can be found in [24]. □

## F. PARTITIONING AND PROBABILISTIC ROUTING IS SUB-OPTIMAL.

**Lemma 2.** *Partitioning a cache, and probabilistically routing content requests to different partitions is sub-optimal.*

*Proof.* Assume we partition the cache of size $C$ into two slices of size $C_1$ and $C_2$, and route requests to partition one with probability $p$, and with probability $1-p$ route to partition two. Let $h_P$ denote the hit rate obtained by partitioning the cache. From non-decreasing concavity of $h(C)$ we have

$$\begin{aligned}h_P &= ph(C_1) + (1-p)h(C_2) \\ &\leq ph(C) + (1-p)h(C) = h(C).\end{aligned}$$

□