Load Imbalance and Caching Performance of Sharded Systems

Load Imbalance and Caching Performance of Sharded Systems

Lorenzo Saino, *Member, IEEE,* Ioannis Psaras, *Member, IEEE,* Emilio Leonardi, *Senior Member, IEEE* and George Pavlou, *Fellow, IEEE*

***Abstract*—Sharding is a method for allocating data items to nodes of a distributed caching or storage system based on the result of a hash function computed on the item's identifier. It is ubiquitously used in key-value stores, CDNs and many other applications. Despite considerable work that has focused on the design and implementation of such systems, there is limited understanding of their performance in realistic operational conditions from a theoretical standpoint. In this paper we fill this gap by providing a thorough modeling of sharded caching systems, focusing particularly on load balancing and caching performance aspects. Our analysis provides important insights that can be applied to optimize the design and configuration of sharded caching systems.**

***Index Terms*—Sharding, load balancing, caching.**

I. Introduction

SHARDING[1] is a widely used technique to horizontally scale storage and caching systems and to address both processing and storage capacity bottlenecks. According to this technique, a large set of data items is partitioned into a set of segments, named *shards*, based on the result of a hash function computed on the identifier of the item. Each shard is then mapped onto a physical storage or caching node. This technique practically enables partitioning data across members of a cluster and can directly identify the member of the cluster responsible for a given item by simply computing a hash function. Normally, sharding is used in conjunction with consistent hashing [3] to minimize the remapping of items as a result of cluster members joining or leaving the system.

Sharding is widely used in a variety of applications. It is for example ubiquitously implemented in database systems,[2] Web caches in enterprise networks [2], [4], CDNs [3], [5] and key-value stores [6]. It is also used to partition large forwarding tables across network cards of a router [7] or across different routers of a network [8]. More recently, sharding has been applied to single-node key-value stores to partition items across memory regions and CPU cores [9], [10].

However, despite considerable work on the design and implementation of sharded systems, there is little theoretical understanding of sharding properties under realistic conditions. In this paper, we shed light on the performance of such systems by widely extending previous work of ours on this subject [11]. Our contribution focuses on two main aspects of sharded systems: *load balancing* and *caching performance*.

We quantify the effect of common design parameters and operational conditions on load imbalance. We show that skewness in the distribution of item request frequency and size can considerably increase load imbalance. On the other hand, common techniques such as partitioning data items into chunks, replicating items across multiple shards and deploying frontend caches are all very effective solutions to mitigate it.

With respect to caching performance, we analyze the behavior of a system of shards, when each of them performs caching decisions independently. We also examine how a layer of frontend caches impacts the overall caching performance. Our main finding is that, under realistic operational conditions, a system of sharded caches yields approximately the same cache hit ratio of a single cache which is as large as the cumulative size of all shards of the system.

This result has important practical implications. In fact, it shows that a very large cache can be partitioned in smaller sharded caches and scaled horizontally without performance degradation. In addition, it makes possible to model a sharded caching system as a single cache, hence making its analysis considerably more tractable. We use the latter to analyze the interactions between a sharded caching system and a layer of frontend caches.

The remainder of this paper is organized as follows. In Sec. II, we introduce the system model adopted throughout this paper. In Sec. III, we investigate factors causing load imbalance in sharded systems and techniques to mitigate it. In Sec. IV, we analyze how sharding impacts cache hit performance. Based on these results, in Sec. V we investigate the load balancing and caching performance of a sharded system when preceded by a layer of frontend caches. Finally, we summarize our findings and draw our conclusions in Sec. VI.

II. System model

We start by introducing the system model adopted throughout this paper and explain assumptions and notation, which we also report in Tab. I.

Consistently with previous work, we assume that the system is subject to a demand conforming to the Independent Reference Model (IRM) [12], which implies that i) requests are issued for items $1, \ldots, N$ belonging to a fixed catalog of size N, ii) the probability that a given request is for item i, is

Lorenzo Saino is with Fastly, Inc., San Franscisco, CA, USA. Ioannis Psaras and George Pavlou are with the Department of Electronic and Electrical Engineering, University College London, UK. Emilio Leonardi is with the Department of Electronic and Telecommunications, Politecnico di Torino, Italy.

[1]The term *sharding* is sometimes referred to, in literature, as *hash partitioning* [1] or *hash routing* [2]. Throughout this paper, we use these terms interchangeably.

[2]To the best of our knowledge all major database implementations, both relational and NoSQL, currently support sharding.

TABLE I: Summary of notation

Symbol	Notation
N	Number of items
K	Number of shards
K'	Number of frontend caches
C	Cache size
L	Fraction of requests served by a shard
X_i	Variable valued 1 if item i assigned to shard, 0 otherwise
p_i	Probability of item i being requested
h_i	Cache hit ratio of item i
α	Zipf exponent of item request frequency distribution
$H_N^{(\alpha)}$	Generalized N^{th} order harmonic number
T, T_F, T_B	Characteristic time of a generic cache, frontend, backend

stationary and independent of previous requests. We denote with $\{p_1, p_2, \ldots, p_N\}$ the vector of such probabilities.

Equivalently, one can assume that i) the request process of item i is a stationary Poisson process with intensity $\lambda_i = p_i \Lambda$ and ii) request processes of different items are independent.

The system comprises K shards, with $K < N$, each equipped with the same amount of caching space of $C < \lfloor N/K \rfloor$ items. We assume that items are mapped uniformly to shards according to a hash function $f_H : [1 \ldots N] \rightarrow [1 \ldots K]$. Therefore, we can model the assignment of an item i to a shard using a Bernoulli random variable X_i such that

$$f_{X_i}(x) = \begin{cases} \frac{1}{K}, & x = 1 \\ 1 - \frac{1}{K}, & x = 0 \end{cases}.$$

Note that our assumptions correspond to the adoption of the so called "random hash function" model, which has been widely adopted in literature [3], [13], [14] to analyze the effects of hash functions on load balancing in distributed systems, due to its analytical tractability.

Most of our analysis does not make any assumption about the distribution of item popularity. However, when specifically studying the impact of item popularity distribution, we assume it follows a Zipf distribution, which is known to model very well item popularity distributions in the applications of our interest, such as Web content distribution [15] and key-value stores [6], [16]. According to this distribution, the probability of an item being requested is

$$p_i = \frac{i^{-\alpha}}{\sum_{i=1}^{N} i^{-\alpha}} = \frac{i^{-\alpha}}{H_N^{(\alpha)}}, \tag{1}$$

where $H_N^{(\alpha)} = \sum_{i=1}^{N} i^{-\alpha}$ corresponds to the generalized N^{th} order harmonic number and $\alpha > 0$ is a parameter affecting the skewness of the distribution. The greater the value of α, the greater the skewness of item popularity and in the limit case of $\alpha \rightarrow 0$ the distribution is uniform. The value of α which best models operational workloads varies, but a number of measurement studies [15], [17], [18], [19] covering a variety of operational systems report values between 0.6 and 1.2.

III. Load balancing

After describing the notation and assumptions used in our analysis, we now investigate how well the randomized assignment of items spreads load across shards. The results of this analysis apply to both caching and storage systems.

This problem is an instance of a *heavily loaded single-choice weighted balls-in-bins game*. A single-choice balls-in-bins game consists in placing each of N balls into one of K bins chosen independently and uniformly at random (i.u.r.). In the weighted case, each ball has a different weight, which in our case is the probability of an item being requested. In the heavily loaded case, which is the case of our interest, the number of balls is considerably larger than the number of bins, *i.e.,* $N \gg K$.

The study of balls-in-bins games has received considerable attention in the past. Unfortunately, there is very little work investigating weighted balls-in-bins games, and, to the best of our knowledge, *there is no previous work satisfactorily addressing the heavily loaded case*. There is instead a greater amount of work addressing the unweighted case. Raab and Steger [20] showed that if $N \geq K \log(K)$, the most loaded bin receives $N/K + \Theta(\sqrt{N \log(K)/K})$ balls with a probability not smaller than $1 - N^{-\theta}$ for an arbitrarily chosen constant $\theta \geq 1$. This is the tightest bound currently known.

Limiting our analysis to the assumption of uniform popularity distribution would not allow us to understand the load balancing dynamics of sharded systems under realistic conditions. In fact, it is well known that distributed caching and storage systems are normally subject to highly skewed item popularity distributions [6], [15], [16]. As we show below, skewness in item popularity distribution strongly impacts load imbalance.

In addition to considering realistic item popularity distributions in our model, we also make novel contributions by shedding light on the impact of heterogeneous item size, item chunking and multiple item replication.

In line with previous work [4], we quantify load imbalance by using the coefficient of variation of load processed by a shard $c_v(L)$,[3] where L is a random variable corresponding to the fraction of requests processed by a single shard. Other work (*e.g.,* [3], [6]), instead, quantifies load imbalance as the ratio between the load of the most loaded shard and the average load of a shard $\mathbb{E}[L]$. The latter metric is normally adopted in experimental work [6], where it can be easily measured, or in theoretical work assuming uniform item popularity distribution [3], where it can be tightly bounded using standard Chernoff arguments. Unfortunately, in theoretical work assuming non-uniform item popularity distribution like ours, Chernoff bounds cannot be easily applied. As a consequence, it is not possible to derive tight bounds on the load of the most loaded shard. For this reason, in line with previous work of this kind [4], we adopt the coefficient of variation as the metric to quantify imbalance.

A. Base analysis

We start by analyzing first how well the randomized assignment of items to shards spreads load, without making any assumption on item popularity distribution.

Adopting the notation introduced above, we can express the fraction of requests that each shard receives as

$$L = \sum_{i=1}^{N} X_i \, p_i \, . \tag{2}$$

[3]The coefficient of variation of a random variable is the ratio between its standard deviation and its mean value: $c_v(L) = \sqrt{\mathrm{Var}(L)}/\mathbb{E}[L]$.

Since $X_1, \dots, X_N$ are i.i.d. random variables, the expected value and variance of L can be calculated as

$$\mathbb{E}[L] = \mathbb{E}\left[\sum_{i=1}^{N} X_i\, p_i\right] = \sum_{i=1}^{N} \mathbb{E}[X_i]\, p_i = \frac{1}{K}\,,$$

$$\begin{aligned}\mathrm{Var}(L) &= \mathrm{Var}\left(\sum_{i=1}^{N} X_i\, p_i\right) = \sum_{i=1}^{N} \mathrm{Var}(X_i)\, p_i^2 \\ &= \frac{K-1}{K^2} \sum_{i=1}^{N} p_i^2\,.\end{aligned}$$

We can now derive the coefficient of variation $c_v(L)$ as

$$c_v(L) = \frac{\sqrt{\mathrm{Var}(L)}}{\mathbb{E}[L]} = \sqrt{K-1}\sqrt{\sum_{i=1}^{N} p_i^2}\,. \tag{3}$$

We can immediately observe from Eq. 3 that the load imbalance increases proportionally to the square root of the number of shards K and to the skewness of item popularity (quantified by $\sum_{i=1}^{N} p_i^2$). Regarding the impact of item popularity skewness on load imbalance, the minimum value of $c_v(L)$ occurs when all items are equally probable, *i.e.,* $p_i = 1/N \quad \forall i \in [1 \dots N]$, while the maximum is when one item is requested with probability 1 and all other items with probability 0. We formalize these considerations and derive the following bounds on $c_v(L)$, which apply uniformly to every distribution of the demand.

Theorem 1. *Let L be the fraction of requests a shard receives in a system of K shards subject to an arbitrary IRM demand over a catalog of N items. Then*

$$\sqrt{\frac{K-1}{N}} \le c_v(L) \le \sqrt{K-1}\,. \tag{4}$$

Proof. The theorem can be proved immediately by jointly applying Hölder's inequality and Minkowski's inequality to bound $\sum_{i=1}^{N} p_i^2$:

$$\frac{1}{N}\left(\sum_{i=1}^{N} p_i\right)^2 \le \sum_{i=1}^{N} p_i^2 \le \left(\sum_{i=1}^{N} p_i\right)^2\,.$$

Substituting $\sum_{i=1}^{N} p_i = 1$ (which holds since p_i is the probability of an item $i \in [1 \dots N]$ being requested), we can rewrite the inequality as

$$\frac{1}{N} \le \sum_{i=1}^{N} p_i^2 \le 1\,. \tag{5}$$

Substituting Eq. 3 into Eq. 5 and rearranging, we obtain Eq. 4. □

The above bounds, derived without making any assumption on item popularity, are however of little practical interest since they can span few orders of magnitude in realistic conditions (*i.e.,* $N \ge 10^6$). In the following section, we derive more useful results by making assumptions on the distribution of item popularity.

B. Impact of item popularity distribution

In this section, we provide a closed-form approximation for $c_v(L)$ under a Zipf-distributed demand that will help us understand the impact of each system variable on load imbalance.

Theorem 2. *Let L be the fraction of requests a shard receives in a system of K shards subject to an IRM demand over a catalog of N items and item request probability distributed according to a Zipf distribution with parameter α. Then*

$$c_v(L) \approx \begin{cases} \sqrt{K-1}\,\dfrac{\sqrt{(N+1)^{1-2\alpha}-1}\cdot(1-\alpha)}{\sqrt{1-2\alpha}\cdot\left[(N+1)^{1-\alpha}-1\right]} & \alpha \neq \left\{\frac{1}{2}, 1\right\} \\ \dfrac{\sqrt{(K-1)\log(N+1)}}{2(\sqrt{N+1}-1)} & \alpha = \frac{1}{2} \\ \sqrt{\dfrac{N(K-1)}{(N+1)\log^2(N+1)}} & \alpha = 1 \end{cases}. \tag{6}$$

Proof. We start by deriving an approximated closed-form expression of $H_N^{(\alpha)}$. We do so by approximating $H_N^{(\alpha)}$ with its integral expression evaluated over the interval $[1, N+1]$:

$$\sum_{i=1}^{N} \frac{1}{i^\alpha} = H_N^{(\alpha)} \approx \int_1^{N+1} \frac{dx}{x^\alpha} = \begin{cases} \frac{(N+1)^{1-\alpha}-1}{1-\alpha}, & \alpha \neq 1 \\ \log(N+1), & \alpha = 1 \end{cases}. \tag{7}$$

We then use Eq. 7 to approximate $\sum_{i=1}^{N} p_i^2$ for the three cases $\alpha \notin \left\{\frac{1}{2}, 1\right\}$, $\alpha = \frac{1}{2}$ and $\alpha = 1$:

$$\begin{aligned}\sum_{i=1}^{N} p_i^2\Big|_{\alpha \notin \left\{\frac{1}{2},1\right\}} &= \sum_{i=1}^{N}\left(\frac{i^{-\alpha}}{\sum_{j=1}^{N} j^{-\alpha}}\right)^2 = \frac{H_N^{(2\alpha)}}{\left(H_N^{(\alpha)}\right)^2} \\ &\approx \frac{\left[(N+1)^{1-2\alpha}-1\right](1-\alpha)^2}{\left[(N+1)^{1-\alpha}-1\right]^2(1-2\alpha)}\,,\end{aligned} \tag{8}$$

$$\begin{aligned}\sum_{i=1}^{N} p_i^2\Big|_{\alpha=\frac{1}{2}} &= \sum_{i=1}^{N}\left(\frac{i^{-\frac{1}{2}}}{\sum_{j=1}^{N} j^{-\frac{1}{2}}}\right)^2 = \frac{H_N^{(1)}}{\left(H_N^{\left(\frac{1}{2}\right)}\right)^2} \\ &\approx \frac{\log(N+1)}{4\left(\sqrt{N+1}-1\right)^2}\,,\end{aligned} \tag{9}$$

$$\begin{aligned}\sum_{i=1}^{N} p_i^2\Big|_{\alpha=1} &= \sum_{i=1}^{N}\left(\frac{i^{-1}}{\sum_{j=1}^{N} j^{-1}}\right)^2 = \frac{H_N^{(2)}}{\left(H_N^{(1)}\right)^2} \\ &\approx \frac{N}{(N+1)\log^2(N+1)}\,.\end{aligned} \tag{10}$$

It should be noted that Eq. 9 and Eq. 10 could also be obtained by deriving the limits of Eq. 8 for $\alpha \to \frac{1}{2}$ and $\alpha \to 1$ respectively by applying L'Hôpital's rule.

Finally, applying the approximations of Eq. 8, Eq. 9 and Eq. 10 to Eq. 3, we obtain Eq. 6. □

This result has important practical implications. The closed-form approximation makes it easier to reason about the impact

of each variable on load imbalance. Therefore it can be applied to optimize the design and configuration of sharded systems. Additionally, we use this approximation in Sec. V-A to quantify the impact of frontend caches on load imbalance and extensively evaluate its accuracy.

To understand the impact of number of shards and item popularity distribution on load imbalance better we show in Fig. 1 the value of $c_v(L)$ calculated using Eq. 6 for various values of α and K.

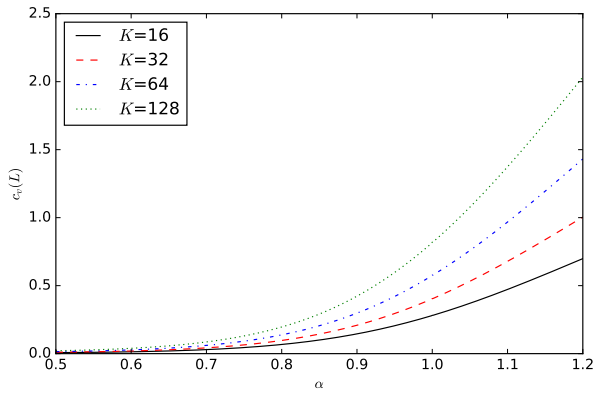


Fig. 1: $c_v(L)$ *vs.* α and K for $N = 10^6$

Analyzing Eq. 6 and Fig. 1, we can observe that the load imbalance is pretty close to 0 for low values of α and increases monotonically as α and K increase. In particular, it should be noted that *item popularity skewness considerably affects load imbalance*. This effect had been already widely experienced in operational systems [6], [21], [22]. However, the sensitivity of load imbalance with respected item distribution skewness was never investigated analytically. Our analysis also demonstrates that unweighted balls-in-bins analyses investigating purely the number of items assigned to each shard cannot capture accurately load imbalance dynamics under realistic conditions.

C. Impact of heterogeneous item size

If all items have the same size, the fraction of requests processed by a shard corresponds to the fraction of overall traffic it serves. However, if items have heterogeneous sizes, load imbalance in terms of number of requests and amount of traffic served do not coincide anymore. In this section we evaluate the load imbalance in terms of fraction of traffic served assuming that items have heterogeneous size. Since previous work did not identify any significant correlation between item size and request frequency in the applications of our interest [15], [16], we assume in our analysis that item size and access frequency are independent.

Theorem 3. *Let $L_{(\mu,\sigma)}$ be the throughput served by a shard of a system of K shards subject to an IRM demand $p_1, p_2, \ldots, p_N$, where the size of each item is a variable with mean μ and variance σ^2. Then*

$$c_v\left(L_{(\mu,\sigma)}\right) = \sqrt{K}\sqrt{\left(1-\frac{1}{K}\right)+\frac{\sigma^2}{\mu^2}}\sqrt{\sum_{i=1}^{N} p_i^2}\,. \tag{11}$$

Proof. Since we assume that item size and request frequency are independent, we can define $L_{(\mu,\sigma)}$ as

$$L_{(\mu,\sigma)} = \lambda \sum_{i=1}^{N} X_i\, S_i\, p_i\,,$$

where S_i is an arbitrary random variable with mean μ and variance σ^2, representing the size of item i and λ is the overall rate of requests served by the system.

Since we assumed that X_i and S_i are independent we have

$$\mathbb{E}[X_i \cdot S_i] = \mathbb{E}[X_i] \cdot \mathbb{E}[S_i] = \frac{\mu}{K}\,,$$

$$\begin{aligned}\mathrm{Var}(X_i \cdot S_i) &= \mathbb{E}[X_i^2 \cdot S_i^2] - \mathbb{E}[X_i \cdot S_i]^2 \\ &= \frac{1}{K}\left[\left(1-\frac{1}{K}\right)\mu^2 + \sigma^2\right].\end{aligned}$$

Since $X_1S_1, \ldots, X_NS_N$ are i.i.d., expected value and variance of $L_{(\mu,\sigma)}$ can be calculated as

$$\mathbb{E}[L_{(\mu,\sigma)}] = \mathbb{E}\left[\lambda \sum_{i=1}^{N} X_i S_i p_i\right] = \lambda \sum_{i=1}^{N} \mathbb{E}[X_i S_i]\, p_i = \frac{\lambda\mu}{K}\,,$$

$$\begin{aligned}\mathrm{Var}(L_{(\mu,\sigma)}) &= \mathrm{Var}\left(\lambda \sum_{i=1}^{N} X_i S_i p_i\right) = \lambda^2 \sum_{i=1}^{N} \mathrm{Var}(X_i S_i)\, p_i^2 \\ &= \frac{\lambda^2}{K}\left[\left(1-\frac{1}{K}\right)\mu^2 + \sigma^2\right] \sum_{i=1}^{N} p_i^2\,.\end{aligned}$$

Deriving $c_v(L_{(\mu,\sigma)}) = \sqrt{\mathrm{Var}(L_{(\mu,\sigma)})}/\mathbb{E}[L_{(\mu,\sigma)}]$ we obtain Eq. 11. □

We can observe from Eq. 11 that load imbalance still increases with $\sqrt{K}$ as in the homogeneous item size case. In addition and more importantly, *load imbalance also increases with the coefficient of variation of item size* $c_v(S) = \sigma/\mu$. We can highlight this relation by rewriting Eq. 11 as

$$c_v(L) = \Theta\left(\sqrt{K}\left(1+\frac{\sigma}{\mu}\right)\right) = \Theta\left(\sqrt{K}\left(1+c_v(S)\right)\right)\,. \tag{12}$$

From Eq.12 we can see the potentially dramatic impact of item size variance on the system performance: observe that $c_v(L) \to +\infty$ when $\sigma \to +\infty$. Such performance degradation can be greatly reduced by adopting chunking strategies as discussed in the next section.

D. Impact of chunking

The analysis carried out above shows the emergence of load imbalance as α and K increase. One way to mitigate this is to split items into chunks and map each chunk to a shard independently. Splitting large items into independent chunks and handling each chunk independently is a common practice in many systems. In this section we quantify the benefits of chunking on load balancing.

Theorem 4. *Let L_M be the load of a shard in a system where each item is split into M chunks and each chunk is mapped to*

a shard independently. Chunking reduces the load imbalance by $\sqrt{M}$, i.e.,

$$c_v\left(L_M\right) = \frac{c_v(L)}{\sqrt{M}}\,. \tag{13}$$

Proof. We can reasonably assume that chunks of the same item are requested with the same probability and let $X_{i,j}$ be a Bernoulli random variable taking value 1 if chunk j of item i is mapped to the shard under analysis and 0 otherwise. The load at each shard can then be modeled as

$$L_M = \sum_{i=1}^{N}\sum_{j=i}^{M} X_{i,j}\frac{p_i}{M} = \frac{1}{M}\sum_{i=1}^{N} Y_i\, p_i\,,$$

where $Y_i = \sum_{j=1}^{M} X_{i,j} \sim B(M, \frac{1}{K})$ is a binomial random variable.

Since $Y_1, \ldots, Y_N$ are i.i.d., $c_v(L_M)$ can be calculated as

$$\begin{aligned} c_v\left(L_M\right) &= \frac{\sqrt{\mathrm{Var}(L_M)}}{\mathbb{E}[L_M]} = \frac{\sqrt{\frac{1}{M^2}\sum_{i=1}^{N}\mathrm{Var}(Y_i)\,p_i^2}}{\frac{1}{M}\sum_{i=1}^{N}\mathbb{E}[Y_i]\,p_i} \\ &= \frac{\sqrt{(K-1)\sum_{i=1}^{N} p_i^2}}{\sqrt{M}}\,. \end{aligned} \tag{14}$$

Substituting Eq. 3 into Eq. 14, we obtain Eq 13. □

Theorem 4 shows that *chunking is a very practical and effective solution to reduce load imbalance*.

E. Impact of multiple item replication

So far we assumed that each item is stored or cached in a single location, or more precisely, that each shard is mapped to only one physical device. This is the common case in caching systems, where, since there is no need to guarantee the availability of all items in case of failures, it is preferable to cache each item at most once to maximize the number of distinct items that can be cached by the system. However, in storage systems, the availability of each item must be guaranteed in spite of failures. The common approach in this case [23] is to replicate each item R times, with $R \leq K$ and store it in the node resolved by the hash function and also in the $R-1$ neighboring nodes. When a request for an item is received, it is randomly redirected to one of these R nodes.

This approach makes it possible to trade storage efficiency with fault tolerance by setting an appropriate replication factor R. In the following, we investigate how the choice of R impacts load imbalance. Intuitively, one could argue that replicating items multiple time reduces load imbalance since it spreads the load of highly loaded shards over multiple hosts. Our analysis shows that this is in fact the case. We present our findings in the following theorem and then discuss their implications.

Theorem 5. *The load imbalance $c_v(L_R)$ of a system of K shards where each item is replicated $R \leq K$ times is equal to the load imbalance of a system of K/R shards where each item is replicated once:*

$$c_v(L_R) = \sqrt{\frac{K}{R} - 1}\sqrt{\sum_{i=1}^{N} p_i^2}\,. \tag{15}$$

Proof. We start by observing that, as a result of an item being randomly replicated R times across a system of K shards, then (i) each target is assigned an item with probability R/K and (ii) if an item has global request probability p_i, a request for that item will hit one of the R replicas with probability p_i/R.

We can now apply these observations to Eq. 2 and derive

$$L_R = \sum_{i=1}^{N}\frac{p_i}{R} Z_i\,, \tag{16}$$

where $Z_i \sim \text{Bernoulli}(R/K)$.

We can now derive $c_v(L_R)$ as follows:

$$\begin{aligned} c_v\left(L_R\right) &= \frac{\sqrt{\mathrm{Var}(L_R)}}{\mathbb{E}[L_R]} = \frac{\sqrt{\sum_{i=1}^{N}\frac{p_i}{R}\mathrm{Var}(Z_i)}}{\sum_{i=1}^{N}\frac{p_i}{R}\mathbb{E}[Z_i]} \\ &= \sqrt{\frac{K}{R} - 1}\sqrt{\sum_{i=1}^{N} p_i^2}\,. \end{aligned} \tag{17}$$

Finally, comparing Eq. 17 with Eq. 3 we can observe that the load imbalance in a system of K shards with each item replicated R times corresponds to the load imbalance of a system of K/R shards where each item is replicated once. □

Theorem 5 shows that replicating items multiple times across different nodes substantially reduces load imbalance, roughly by a factor $\sqrt{R}$. This result has important implications of practical interest. For example, in sharded systems characterized by high request frequency workloads, but moderately sized datasets, replication can be used to trade greater storage requirements with more homogenous load distribution, *i.e.,* more stable and predictable CPU utilization.

From Eq. 15, it is interesting, albeit expected, to observe the following: for $R = 1$, $c_v(L_R) = c_v(L)$, since items are replicated only once, as in the standard sharding case. At the other extreme, *i.e.,* when $R = K$, then $c_v(L_R) = 0$, which is also expected. In fact, in the latter case, each shard is assigned all the items and the sharded system becomes an array of identical storage/caching devices where requests for any item are forwarded to a randomly chosen destination.

IV. Cache hit ratio

We now investigate the performance of a sharded system in terms of cache hit ratio, assuming that each shard is not a permanent store but evicts items according to a replacement policy independently from other shards. In particular, our objective is to compare the performance of a system of caching shards with that of a single cache with capacity equal to the cumulative capacity of all the shards of the system. We assume that each shard has equal size C, with $C < \lfloor N/K \rfloor$ items and all shards operate according to the same replacement policy.

Our analysis applies the *characteristic time approximation*. The *characteristic time* T of a cache subject to an IRM demand corresponds to the time spent by an item in an LRU cache from its last request (which moves the item to the top of the cache) to its eviction. This value is a random variable specific to each item, which is hard to characterize exactly. However, approximating it with a single constant value for all items

still yields remarkably accurate results and widely simplifies performance analysis because it decouples the dynamics of a specific item from all other items.

This approximation was first proposed by Fagin [24], who additionally proved that it is asymptotically exact as cache size C and population size N tend to infinite while C/N remains constant. Che *et al.* [25] later independently rediscovered it, although with a slightly different formulation. However, they only provided intuitive arguments for its accuracy. Subsequently, Fricker *et al.* [26] provided a further *a posteriori* theoretical justification for its accuracy, which also covered configurations where the arguments of Che *et al.* did not apply. They showed that, for an LRU cache subject to a Zipf popularity distribution, the coefficient of variation of the random variable representing the characteristic time tends to vanish as the cache size grows. At the same time, the dependence of the characteristic time on the specific item becomes negligible as the catalog size grows. In addition, they also extended this approximation to the case of random and FIFO caches. Finally, Martina *et al.* [27] more recently generalized it to support a wider variety of replacement policies and network topologies.

Because of the IRM demand assumption, the probability that an item is in the cache p_{in} is equal by definition to its hit probability p_{hit}, as a consequence of the PASTA (Poisson Arrivals see Time averages) property of a Poisson process [28]. Applying the characteristic time approximation, it is possible to determine the hit ratio of each item i simply by knowing the request probability of that item p_i and the characteristic time of the cache T. In the specific case of the LRU replacement policy this corresponds to

$$p_{hit}(p_i, T) = p_{in}(p_i, T) = 1 - e^{-p_i T} \,. \tag{18}$$

The generalization of Martina *et al.* [27] shows that this method can be applied to a larger variety of cache replacement policies, including FIFO, RANDOM and q-LRU (*i.e.,* LRU with insertion probability q). We refer the reader to their work [27] for equations to derive hit ratios for these replacement policies under the characteristic time approximation.

Although all these policies have different formulations of $p_{in}(\cdot)$, it is important to highlight that, as shown by Martina *et al.* [27], they all share the property that $\forall \delta > 0$, then

$$p_{in}(p_i, T) = p_{in}\left(\delta\, p_i, T/\delta\right) \,. \tag{19}$$

As a result of this property, for the case $\delta = T$ we have $p_{in}(p_i, T) = p_{in}\left(p_i\, T, 1\right)$. This implies that p_{in} could be always expressed as a function of the product $p_i\, T$ rather than of each of the two variables independently. We will use this observation to derive the results presented in Sec. V-B, where we will use interchangeably the notation $p_{in}(p_i \cdot T)$ to indicate $p_{in}(p_i, T)$, *i.e.,* $p_{in}(p_i \cdot T) \equiv p_{in}(p_i, T)$.

We can now present the following definitions that we will use in the remainder of this section.

Definition 1. The *characteristic time* T of a cache is the average time elapsed between either the last request for an item (*e.g.,* LRU) or its insertion (*e.g.,* FIFO, RANDOM) and its eviction from the cache.

Definition 2. A cache replacement policy is *defined by characteristic time* if the steady-state per-item hit ratio of a cache operating under such policy, subject to an IRM demand, is exclusively an increasing function of the request intensity of the item and the characteristic time of the cache.

Under the characteristic time approximation, caching strategies such as LRU, FIFO, RANDOM and q-LRU belong to the class of policies *defined by characteristic time*. A special consideration needs to be made for the LFU replacement policy. While LFU cannot be defined by characteristic time, it has been shown [27] that it yields steady-state cache hit ratio performance identical to the q-LRU replacement policy (which is instead defined by characteristic time) when $q \to 0$. Therefore, all considerations regarding policies defined by characteristic time equally apply to LFU.

The characteristic time of a cache T can be computed by solving

$$\sum_{i=1}^{N} p_{in}(p_i, T) = C \,, \tag{20}$$

where, as already discussed above, $p_{in}(\cdot)$ is a function, specific for each replacement policy, that returns the cache hit ratio of an item given its probability of being requested p_i and the characteristic time of the cache T. For all cache replacement policies listed above, Eq. 20 does not have a closed-form solution, but can be easily solved numerically.

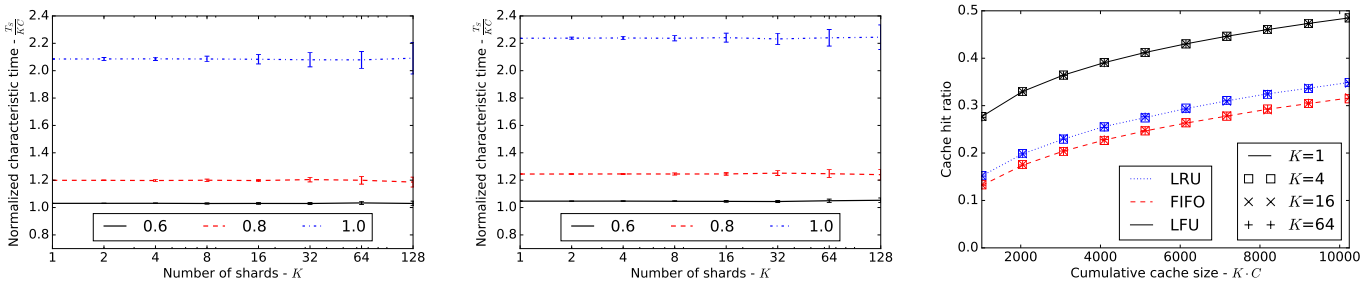
After these initial considerations, we present the following theorem and devote the remainder of this section to prove it and to discuss its applicability in realistic operational conditions.

Theorem 6. *Let $\mathcal{C}_{\mathcal{S}}$ be a caching shard of capacity C belonging to a system of K shards, all with the same capacity and operating under the same replacement policy $\mathcal{R}$ defined by characteristic time. The overall system is subject to an IRM demand $\{p_1, p_2, \ldots, p_N\} > 0$. Items $1, \ldots, N$ are independently mapped to shards $1, \ldots, K$ uniformly at random. Then, for large C, (i.e. when both N and C grow to infinity with $C/N \to \beta < 1/K$) the cache hit ratio of $\mathcal{C}_{\mathcal{S}}$ with high probability[4] tends to the cache hit ratio of a single cache of size $K \cdot C$ operating under replacement policy $\mathcal{R}$ subject to the aforementioned demand.*

Proof. The proof is presented in the appendix. □

We further validate the approximation of Theorem 6 numerically and draw results in Fig. 2a and 2b, where we show, given fixed values of cumulative cache size $KC = 16$K and number of items $N = 256$K, the value of T_S for different values of K and with item popularity distributed according to Zipf distributions with exponents $\alpha \in \{0.6, 0.8, 1.0\}$ for the cases of LRU and FIFO/RANDOM replacement policies. It should be noted that we plot the results for RANDOM and FIFO replacement policies in a single graph as it is well known that they yield identical cache hit ratios when subject to IRM demand [29] and therefore have the same characteristic time [27]. The results are consistent with our analysis. The

[4]A sequence of events $\{A_n\}_n$ occurs with high probability (w.h.p.) if $P(A_n) \to 1$ as $n \to \infty$.



(a) Characteristic time (T_S) *vs.* number of shards (K) and α (LRU case), with $KC = 16$K and $N = 256$K

(b) Characteristic time (T_S) *vs.* number of shards (K) and α (FIFO/RANDOM case), with $KC = 16$K and $N = 256$K

(c) Cache hit ratio for single cache and systems of shards, with $\alpha = 0.8$ and $N = 256$K

Fig. 2: Cache hit ratio performance of sharded systems

mean value of T_S is not affected by variations in the number of shards K and is the same in the case of a single cache ($K = 1$). The standard deviation of T_S, represented by the error bars, increases with K, as expected, but it remains low in comparison to the mean value of T_S as it is possible to see from the values of the y-axis.

The insensitivity of T_S against K is also expectedly reflected in cache hit ratio performance. We simulate systems of caching shards under the LRU, FIFO and LFU replacement policies for various caching sizes and various K and plot results in Fig. 2c. The graph shows again insensitivity of cache hit ratio with respect to K, further validating the results of our analysis.

Ji *et al.* [30] reached similar conclusions to those of Theorem 6. However their model makes more restrictive assumptions and only applies to the case of a system of LRU caches subject to a Zipf-distributed IRM demand with $\alpha > 1$ and infinite item population. Differenty, Theorem 6 applies to systems of caches operating according to any replacement policy driven by characteristic time subject to an arbitrary IRM demand.

Finally, although the asymptotic hit ratio of LRU caches has been studied extensively in the past [24], [31], [32], none of the previous work can be directly applied to the problem addressed by Theorem 6. All previous work applies only to LRU caches subject to an IRM demand. While the overall sharded system studied by Theorem 6 is subject to an IRM demand, each individual cache is instead subject to a demand where the request rate of each item is stochastic, and therefore does not conform to the IRM model. However, since Theorem 6 shows that, as cache size grows, the hit ratio of each shard tends to that of a single standalone cache, asymptotic models for single caches can be applied to evaluate the performance of sharded systems.

V. Frontend Caching

So far we analyzed the performance of sharded systems in isolation. However, in practical deployments, such systems are frequently preceded by an array of frontend nodes, as depicted in Fig. 3. These nodes can perform various functions, such as, for example, access control, terminating TCP connections or TLS sessions and redirecting requests to the relevant shards [6], [33]. These frontend nodes are frequently equipped with a smaller cache to offload the backend sharded system [23], [33]. In this section, *we investigate the interaction between this layer of frontend caches and a sharded system deployed behind it and show how the former impacts load balancing and cache hit ratio performance of the latter.*

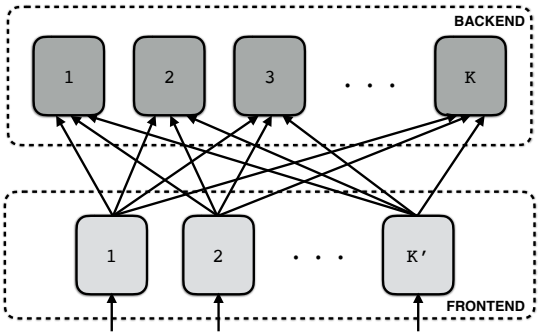


Fig. 3: Sharded system preceded by a layer of frontend caches

A. Load balancing

Intuitively, one can reasonably conjecture that placing a frontend cache does in fact reduce load imbalance at the backend. In fact, as we observed above, skewness in item popularity strongly increases load imbalance but at the same time, it can be addressed effectively by caching. Common replacement policies are more likely to cache items requested more frequently, and, as a result, they *smoothen* the skewness of the item popularity distribution of their miss streams.

Fan *et al.* [34] already investigated this aspect concluding that a small frontend cache of $O(K \log K)$ items operating according to an LFU replacement policy is in fact sufficient to reduce load imbalance. However, their analysis only addresses the load imbalance caused by an adversarial workload with knowledge of the mapping between items and shards attempting to swamp a specific shard.

In contrast, our work addresses the more general case of a frontend cache operating under a variety of replacement policies subject to an arbitrary IRM demand. We conclude that a frontend cache is in fact effective in reducing load imbalance as long as it is properly dimensioned.

For simplicity, the following analysis assumes a single frontend cache. However, it can be trivially demonstrated that the results derived here equally apply to the case of an array

of identically sized frontend caches, depicted in Fig. 3, as long as the distribution of item request frequency is identical in all frontend nodes.

Let h_i be the hit probability of item i at the frontend cache. The fraction of requests handled by a shard can be written as

$$L = \frac{\sum_{i=1}^{N} X_i \, p_i \, (1 - h_i)}{\sum_{i=1}^{N} p_i \, (1 - h_i)} \, . \tag{21}$$

It immediately follows that it its coefficient of variation is

$$c_v(L) = \frac{\sqrt{\mathrm{Var}(L)}}{\mathbb{E}[L]} = \sqrt{K-1} \frac{\sqrt{\sum_{i=1}^{N} p_i^2 (1-h_i)^2}}{\sum_{i=1}^{N} p_i (1-h_i)} \, . \tag{22}$$

After this preliminary discussion, we proceed to analyze the case of a frontend cache operating according to an arbitrary replacement policy subject to an arbitrarily-distributed IRM demand. We derive a set of sufficient conditions under which a frontend cache reduces load imbalance and apply them to the case of LRU, FIFO and RANDOM replacement policies. We then proceed to analyze the specific case of a perfect frontend cache subject to a Zipf-distributed IRM demand. For this particular case, we derive simple formulas for frontend cache dimensioning in order to minimize load imbalance.

1) Arbitrary caches: A frontend cache reduces the load imbalance among shards of a system as long as the conditions stated in the following theorem apply.

Theorem 7. *Let L_C be the load at a shard of a system preceded by a frontend cache subject to an IRM demand. Let's assume that items $1, \ldots, N$ are sorted in decreasing order of popularity,* i.e., *$p_1 \geq p_2 \geq \ldots \geq p_N$. If $h_1 \geq h_2 \geq \ldots \geq h_N$ and $p_1(1-h_1) \geq p_2(1-h_2) \geq \ldots \geq p_N(1-h_N)$, then the frontend cache reduces load imbalance,* i.e., *$c_v(L_C) \leq c_v(L)$.*

Proof. The proof is presented in the appendix. □

It is important to emphasize that the conditions provided by Theorem 7 are only sufficient and not necessary. As a matter of fact, as we show later in this section, LFU caches never meet the condition $p_1(1-h_1) \geq p_2(1-h_2) \geq \ldots \geq p_N(1-h_N)$ and yet still reduce load imbalance.

We can now apply these conditions to the cases of FIFO, RANDOM and LRU replacement policies to draw conclusions of practical interest. These analyses apply concepts from the characteristic time approximation discussed in Sec. IV.

Corollary 7.1. *The load imbalance of a sharded system preceded by a frontend cache operating according to the FIFO or RANDOM replacement policy subject to an IRM demand decreases monotonically with the size of the cache.*

Proof. It is well known that FIFO and RANDOM replacement policies yield identical cache hit ratios when subject to an IRM demand [29]. According to the generalized characteristic time approximation of [27], the cache hit ratio h_i of an item i yielded by a FIFO or RANDOM cache is

$$h_i = \frac{p_i T}{1 + p_i T} \, , \tag{23}$$

where T is the characteristic time of the cache and is a positive constant.

We now prove the theorem by showing that a frontend FIFO or RANDOM cache always meets the sufficient conditions of Theorem 7. It is immediately evident from Eq. 23 that if $p_1 \geq p_2 \geq \ldots \geq p_N$ then $h_1 \geq h_2 \geq \ldots \geq h_N$. Now we only need to demonstrate that $p_1(1-h_1) \geq p_2(1-h_2) \geq \ldots \geq p_N(1-h_N)$. This is equivalent to showing that $p_i(1-h_i) \geq p_{i+1}(1-h_{i+1})$ holds $\forall i \in 1, \ldots, N-1$. Applying 23, we can rewrite this condition as

$$\frac{p_i}{1 + p_i T} \geq \frac{p_{i+1}}{1 + p_{i+1} T} \, . \tag{24}$$

Remembering that by definition $T > 0$ and $p_i \geq 0 \, \forall i \in 1, \ldots, N$, we can rearrange Eq. 24 and observe that it holds as long as $p_i \geq p_{i+1}$ which is true by definition. □

Corollary 7.2. *A frontend cache of size C operating according to the LRU replacement policy and subject to a Zipf-distributed IRM demand with skewness α and a catalog of N items reduces load imbalance if*

$$C \leq N - \sum_{i=1}^{N} 2^{-\frac{\alpha \cdot i - \alpha}{1 - 2^{-\alpha}}} \, . \tag{25}$$

Proof. According to the characteristic time approximation [25] the cache hit ratio h_i of an item i yielded by an LRU cache is

$$h_i = 1 - e^{-p_i T} \, , \tag{26}$$

where T is the characteristic time of the cache and is the unique root of the equation

$$\sum_{i=1}^{N} h_i = \sum_{i=1}^{N} 1 - e^{-p_i T} = N - \sum_{i=1}^{N} e^{-p_i T} = C \, . \tag{27}$$

We show that a frontend LRU cache meets the sufficient conditions of Theorem 7 as long as Eq. 25 holds.

It can be immediately observed from Eq. 26 that if $p_1 \geq p_2 \geq \ldots \geq p_N$ then $h_1 \geq h_2 \geq \ldots \geq h_N$. Now we only need to verify under what conditions $p_1(1-h_1) \geq p_2(1-h_2) \geq \ldots \geq p_N(1-h_N)$. This is equivalent to verify under what conditions $p_i(1-h_i) \geq p_{i+1}(1-h_{i+1})$ holds $\forall i \in 1, \ldots, N-1$. Applying Eq. 26, we can rewrite this condition as

$$\frac{p_i}{p_{i+1}} \geq e^{(p_i - p_{i+1})T} \, .$$

Because of the Zipf-distributed demand assumption, we can substitute $p_i = i^{-\alpha} / H_N^{(\alpha)}$ and, after rearranging, we obtain

$$T \leq \frac{\alpha \log(\frac{i+1}{i}) H_N^{(\alpha)}}{i^{-\alpha} - (i+1)^{-\alpha}} \, .$$

Since the numerator monotonically decreases and the denominator monotonically increases with respect to i, the right hand side part of the inequality is minimized for $i = 1$. Therefore

$$T \leq \frac{\alpha \log(2) H_N^{(\alpha)}}{1 - 2^{-\alpha}} \leq \frac{\alpha \log(\frac{i+1}{i}) H_N^{(\alpha)}}{i^{-\alpha} - (i+1)^{-\alpha}} \, . \tag{28}$$

From Eq. 27 we can observe that C increases with T. Therefore, we can substitute Eq. 28 into Eq. 27 and rearrange to obtain Eq. 25:

$$C \leq N - \sum_{i=1}^{N} \exp\left[- \frac{i^{-\alpha} \alpha \log(2) H_N^{(\alpha)}}{H_N^{(\alpha)} (1 - 2^{-\alpha})} \right] = N - \sum_{i=1}^{N} 2^{-\frac{\alpha \cdot i - \alpha}{1 - 2^{-\alpha}}} \, .$$

□

2) Perfect cache: We now analyze the case of a perfect frontend cache of size $C < N$ that always caches the C most popular items. This corresponds to an LFU replacement policy, since we assume an IRM workload. Assuming that $p_1 \geq p_2 \geq \ldots \geq p_N$, then a perfect frontend cache permanently stores items $1, \ldots, C$.

Denoting L_C as the load after filtering from a frontend cache of C items, we can write $c_v(L_C)$ as

$$c_v(L_C) = \sqrt{K-1}\frac{\sqrt{\sum_{i=C+1}^{N} p_i^2}}{\sum_{i=C+1}^{N} p_i}\,. \tag{29}$$

We further assume that item popularity follows a Zipf distribution with parameter $\alpha > 0$ and apply the approximation of Eq. 7 to derive the following closed-form expression for $c_v(L_C)$:

$$c_v(L_C) \approx \begin{cases} \sqrt{K-1}\dfrac{\sqrt{\frac{(N+1)^{1-2\alpha}-(C+1)^{1-2\alpha}}{1-2\alpha}}}{\frac{(N+1)^{1-\alpha}-(C+1)^{1-\alpha}}{1-\alpha}} & \text{if } \alpha \notin \{\frac{1}{2}, 1\} \\ \sqrt{K-1}\dfrac{\sqrt{\log(N+1)-\log(C+1)}}{2(\sqrt{N+1}-\sqrt{C+1})} & \text{if } \alpha = \frac{1}{2} \\ \sqrt{K-1}\dfrac{\sqrt{\frac{N-C}{(N+1)(C+1)}}}{\log(N+1)-\log(C+1)} & \text{if } \alpha = 1 \end{cases} \tag{30}$$

We can now present our main theorem and four corollaries and discuss their results.

Theorem 8. *Let L_C be the load at a shard subject to a Zipf-distributed IRM demand with skewness parameter α pre-filtered by a frontend perfect cache of size C, with $C < N$. Then $c_v(L_C)$ is a convex function with respect to C and has a global minimum for*

$$C^* = \operatorname*{argmin}_C c_v(L_C) = \gamma(N+1) - 1\,, \tag{31}$$

where, if $\alpha \notin \{\frac{1}{2}, 1\}$, γ is the unique solution of

$$2(1-\alpha)\gamma^{2\alpha-1} - (1-2\alpha)\gamma^{\alpha-1} - 1 = 0 \tag{32}$$

in the open interval $\gamma \in (0,1)$, otherwise

$$\gamma = \begin{cases} -\frac{1}{4}\, W_{-1}\left(-\frac{1}{2}e^{-\frac{1}{2}}\right)^{-2} \approx 0.08 & \text{if } \alpha = \frac{1}{2} \\ -\frac{1}{2}\cdot W_0\left(-2e^{-2}\right) \approx 0.2032 & \text{if } \alpha = 1 \end{cases}\,. \tag{33}$$

where W_0 and W_{-1} denote respectively the main and lower branches of the Lambert W function.

Proof. The proof is presented in the appendix. □

We can now make further considerations for the asymptotic case of $N \to +\infty$ and approximations for large N values.

Corollary 8.1. *For $\alpha > 1$ and $N \to +\infty$, then*

$$c_v(L_C) = \sqrt{\frac{K-1}{C+1}}\cdot\frac{\alpha-1}{\sqrt{2\alpha-1}}\,. \tag{34}$$

Proof. If $\alpha > 1$ and $N \to +\infty$, then $(N+1)^{1-\alpha} \to 0$ and $(N+1)^{1-2\alpha} \to 0$. Applying these substitutions to Eq. 30 we obtain Eq. 34. □

Corollary 8.2. *For $N \to +\infty$, a perfect frontend cache always reduces load imbalance and any increase in cache size further reduces load imbalance.*

Proof. The proof is straightforward from Theorem 8, which shows that $c_v(L_C)$ is monotonically decreasing with respect to C in the interval $0 \leq C \leq C^* = \gamma(N+1) - 1$. Since

$$\lim_{N\to+\infty} C^* = \lim_{N\to+\infty} \gamma(N+1) - 1 = +\infty\,,$$

then any finite cache size reduces load imbalance. □

Corollary 8.3. *If $\alpha = \frac{1}{2}$ and N is large, then $C^* \approx 0.08 \cdot N$.*

Proof. Solving numerically the term $W_{-1}\left(-\frac{1}{2}e^{-\frac{1}{2}}\right)$ of Eq. 33 we obtain

$$C^* = 0.08 \cdot N - 0.92\,. \tag{35}$$

Since for large N the constant part of Eq. 35 is negligible, we can approximate it as $C^* \approx 0.0.08 \cdot N$. □

Corollary 8.4. *If $\alpha = 1$ and N is large, then $C^* \approx 0.2 \cdot N$.*

Proof. The proof is analogous to the case above. Solving numerically the term $W\left(-2e^{-2}\right)$ of Eq. 33 we obtain

$$C^* = 0.2032 \cdot N - 0.797\,. \tag{36}$$

Since for large N the constant part of Eq. 36 is negligible, we can approximate it as $C^* \approx 0.2 \cdot N$. □

This analysis showed that *a perfect frontend cache is effective in reducing load imbalance in all scenarios of practical interest as long as properly dimensioned.*

From Theorem 8 and its corollaries we can observe that the global minimum of $c_v(L_C)$ does not depend on the absolute values of C or N but on the quantity $\gamma = \frac{C+1}{N+1}$, as well as α. In Fig. 4 we plot all values of γ for which we observe a global minimum of $c_v(L_C)$ for various values of α, which represent the skewness of item popularity distribution. As already discussed above, typical workloads can be modeled with $\alpha \in [0.6, 1.2]$. In that interval, the values of γ that minimize load imbalance are between 0.11 and 0.24. This implies that even in the worst case scenario, the frontend cache should be smaller than 11% of the size of item population to ensure that any addition of caching space reduces load imbalance. In practice such frontend caches are rarely larger than 1% of the item population. Hence, we conclude that *any typical frontend cache reduces load imbalance.*

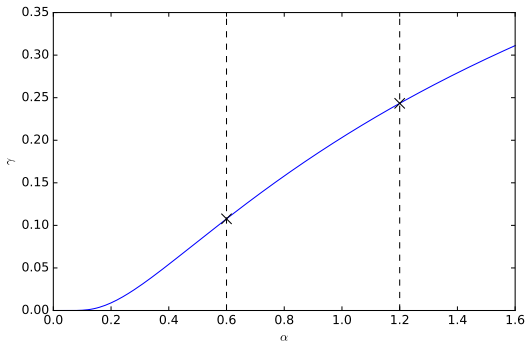


Fig. 4: Relationship between γ and α

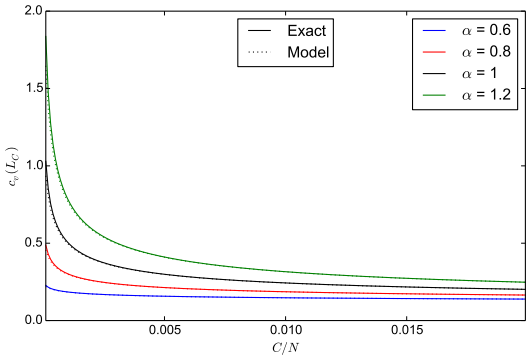


Fig. 5: $c_v(L_C)$ with perfect frontend cache ($N = 10^6$)

As a further step in our evaluation, we quantify the actual reduction of load imbalance achieved by a perfect frontend cache and plot results in Fig. 5 for various values of α and cache/catalog ratio C/N in the case of $N = 10^6$. In the plot, the lines labeled as *Exact* have been drawn using Eq. 29 while the lines labeled as *Model* have been drawn using Eq. 30.

This analysis allows us to draw three main conclusions. First of all, the load imbalance model, which relies on the approximation of Theorem 2, is very accurate. This is demonstrated by the almost perfect agreement between exact and model results. Second, the imbalance reduction is greater for more skewed item distributions. This is expected because, as discussed above, more uniform distributions induce lower load imbalance and therefore there is less improvement that a frontend cache can provide. Finally, and most importantly from a practical standpoint, *even a very small cache of around 0.5% of content catalog is sufficient to carry out a substantial reduction in load imbalance*. Further increasing cache size still reduces load imbalance but less substantially.

B. Cache hit ratio

In addition to reducing load imbalance, a layer of frontend caches impacts the caching performance of the backend sharded system. We dedicate this section to investigate the interplay between these two layers of caches and investigate how variables like size and number of frontend caches impact caching performance.

We assume that the system comprises K equally sized shards and an array of K' equally sized frontend caches. We assume that the overall system is subject to an IRM demand $\{p_1, p_2, \ldots, p_N\}$ and that each request is first handled by a randomly selected frontend cache. As a result, the demand received by each single frontend is $\{p_1/K', p_2/K', \ldots, p_N/K'\}$. Finally, we assume that both frontend and backend caches operate according to the same replacement policy and that such a policy is defined by its characteristic time.

We denote the size of a single frontend cache as C_F and the cumulative size of the backend (*i.e.,* the sum of the sizes of each shard) as C_B. Similarly, we denote the characteristic time of a single frontend cache and the backend sharded system as T_F and T_B respectively.

This analysis relies on Theorem 6 which shows that a cluster of sharded caches asymptotically yields performance identical to a single cache with size equal to the cumulative size of all shards. This result allows us to model the caching system as a two-level tree with the system of shards as root and the frontends as leaves.

Recalling the definition of characteristic time, we can observe that the probability that an item is found at a frontend cache corresponds to the probability that the item was requested in the interval $[t-T_F, t]$ independently of the state of the backend cache. Alternatively, a cache hit at the backend on an item received from a specific frontend F_j occurs only if either a request from F_j arrived in the interval $[t-T_B, t-T_F]$ or if at least one request arrived from any other frontend cache $F_{k,k\neq j}$ in the interval $[t-T_B, t]$.

We can immediately start deriving the cache hit ratio at the frontend since it does not depend on the state of the backend. We can start deriving the characteristic time T_F and the hit ratio $h_i^{(F)}$ at each frontend cache applying Eq. 20 and Eq. 18:

$$\sum_{i=1}^{N} p_{in}(p_i/K', T_F) = C_F\,, \tag{37}$$

$$h_i^{(F)} = p_{in}(p_i/K', T_F)\,. \tag{38}$$

Applying Eq. 19, we can rewrite the two equations above as

$$\sum_{i=1}^{N} p_{in}(p_i, T_F/K') = C_F\,, \tag{39}$$

$$h_i^{(F)} = p_{in}(p_i, T_F/K')\,. \tag{40}$$

Analyzing Eq. 39 and observing that p_i and C_F are constant and do not depend on the number of frontend caches K', it necessarily follows that T_F/K' is constant. As a result, we can immediately observe from Eq. 40 that *the cache hit ratio of each frontend cache $h_i^{(F)}$ does not depend on the number of frontend caches K'.*

We can proceed deriving the average rate of requests for item i missing the layer of frontend caches, denoted as $\overline{p}_i^{(B)}$ and then the characteristic time of the backend T_B:

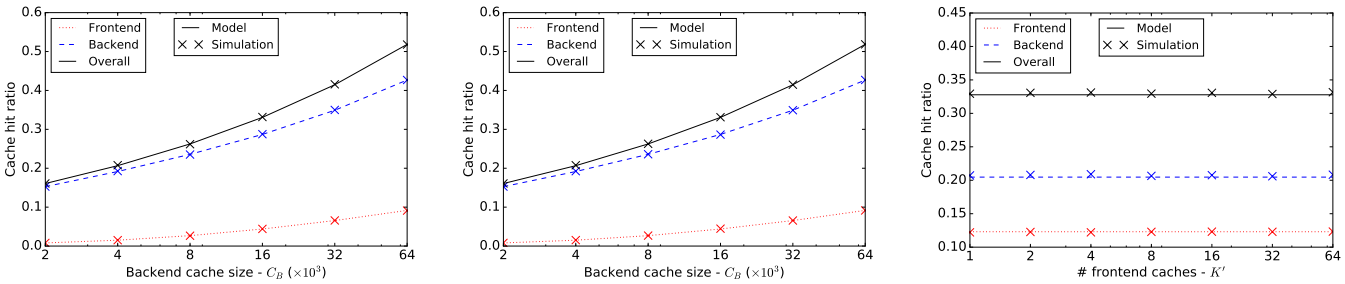
$$\overline{p}_i^{(B)} = \sum_{j=1}^{K'} \frac{p_i}{K'} \left(1 - h_i^{(F)}\right) = p_i \left(1 - h_i^{(F)}\right)\,, \tag{41}$$

$$\sum_{i=1}^{N} p_{in}(\overline{p}_i^{(B)}, T_B) = C_B\,. \tag{42}$$

It should be noted however, that in this case requests are no longer IRM, since they are not exogenous requests but are misses from another cache, which, depending on the replacement policy used, may be correlated. This is the case, for example, with LRU caches [35]. For this reason we use the overline notation.

To derive the hit ratio at the backend $h_i^{(B)}$, we apply results from [27] which allow us to compute $h^{(B)}(i|F_j)$, *i.e.,* the hit ratio of item i at the backend, given that it has been forwarded by F_j. Specifically,

$$h^{(B)}\left(i|F_j\right) \approx p_{in}(A_{i,j}) \tag{43}$$



(a) Cache hit ratio vs. C_B ($K' = 1$) (b) Cache hit ratio vs. C_B ($K' = 16$) (c) Array of frontends - Hits vs. K'

Fig. 6: Accuracy of two-layer caching model ($N = 512$K, $\alpha = 0.8$)

where

$$A_{i,j} = \frac{1}{K'} \cdot \overline{p}_i^{(B)}(j) \cdot \max(0, T_B - T_F) + \sum_{k \neq j} \frac{1}{K'} \cdot \overline{p}_i^{(B)}(k) \cdot T_B \,. \tag{44}$$

Since all frontend caches are identical and are subject to identical demand, their miss streams are statistically identical. As a result, the rate of requests reaching the backend from each frontend are identical, *i.e.,* $\overline{p}_i^{(B)} = \overline{p}_i^{(B)}(1) = \overline{p}_i^{(B)}(2) = \cdots = \overline{p}_i^{(B)}(K')$. In addition, the hit probability of an item at the backend does not depend on the specific frontend from which the request is received, *i.e.,* $h^B(i|F_j) = h_i^{(B)}$. We can therefore apply these findings and rewrite Eq. 43 as

$$h_i^{(B)} \approx \begin{cases} p_{in}\left(\overline{p}_i^{(B)}\left(T_B - \frac{T_F}{K'}\right)\right) & \text{if } T_F < T_B \\ p_{in}\left(\overline{p}_i^{(B)}\left(1 - \frac{1}{K'}\right) T_B\right) & \text{if } T_F \geq T_B \end{cases} . \tag{45}$$

The only sizing configuration of practical interest occurs when the cumulative size of frontend caches is smaller than the cumulative size of backend caches, *i.e.,* $K'C_F < C_B$. That is because in practical applications, frontends are typically built out of faster (and more expensive) memory technologies than the backend [33], [34] and have smaller capacity. We can easily observe that since the characteristic time of a cache is inversely proportional to the intensities of request processes and the request processes at the backend have lower intensities than at the frontend, even when $K'C_F = C_B$ necessarily implies that $T_F < T_B$. Therefore $K'C_F < C_B \implies T_F < T_B$.

Analyzing the $T_F < T_B$ case of Eq. 45 and recalling that T_F/K' is constant, we can observe that *the cache hit ratio at the backend does not depend on the number of frontend caches*. As a result, *the cache hit ratio of a system of sharded caches with an array of frontends is identical to that of a single frontend case*. We note that in the case where $T_F \geq T_B$, the cache hit ratio at the backend depends on the number of frontend caches K'. However, as explained above, this case is of no practical interest. Moreover, this dependency vanishes as K' grows large.

One difference that is important to highlight though is that, in both cases, since the miss stream of the frontend caches are independent [35], the greater the value of K' the less is the dependence among requests in the aggregate miss stream. As a result, the Poisson approximation used in Eq. 45 becomes more accurate. However, as we show below, even for low values of K' our model is still very accurate.

We validate the accuracy of our model by comparing the predicted cache hit ratio with values from simulations. We performed all experiments using LRU caches and a Zipf-distributed content popularity with N = 512K items and α = 0.8 and plot results in Fig. 6.

In Fig. 6a and Fig. 6b we show the accuracy of our models against various cache sizes for $K' = 1$ and $K' = 16$. In both cases we vary C_B between 2K and 64K items and maintain a constant ratio between backend and frontend cache sizes C_B/C_F equal to 8. We also maintain constant values of K and K' equal to 16. We deliberately selected small cache sizes because this is the worst case condition for our model (see Theorem 6). Despite this, our model predicts performance very accurately under all cache sizes considered.

Finally, in Fig. 6c we further evaluate the accuracy of our model maintaining constant values of C_B and C_F of 16K and 2K and varying the number of frontend caches K' from 1 to 64. Our model again predicts performance very well and, as we showed above, both frontend and backend cache hit ratio do not depend on the number of frontend caches.

VI. Summary and Conclusion

This paper presented analytical results shedding light on operational properties of sharded caching systems that, while empirically observed in the past, were poorly understood from a theoretical standpoint. Our analysis focused on two main aspects: load balancing and caching performance.

With respect to load balancing, we showed that heterogeneous request frequency and item size considerably increase load imbalance and provided closed-form expressions to quantify their impact. These results show that previous work assuming uniform request frequency is inaccurate when describing the behavior of sharded systems in typical operational conditions. On the other hand, we showed that techniques such as item chunking and replicating items multiple times across shards can effectively mitigate load imbalance.

With respect to caching performance, our key finding shows that partitioning a single cache into a cluster of smaller uncoordinated sharded caches does not cause any appreciable degradation in cache hit ratio, as the cache size grows large. This finding has remarkable practical implications because it shows that caching systems can be scaled horizontally without inter-cache coordination mechanisms and without loss in caching performance.

Finally, we evaluated how a sharded system operates when deployed behind a layer of frontend caches, which is a typical scenario in many operational deployments. This analysis provided two key insights. First, even a small frontend cache is very effective at reducing the load imbalance at the backend; we quantified its impact for various common cache replacement policies. Second, in the case of practical interest where the cumulative size of frontend caches is smaller than the cumulative size of backend caches, the cache hit ratio of the backend is not affected by the number of frontend caches but only by their sizes; we provided closed-form equations to derive the hit ratio at each layer.

We believe that all these findings have important practical applications in the design and configuration of operational sharded systems.

APPENDIX

PROOF OF THEOREM 6

Proof. We denote T_S as the characteristic time of a shard of size C and T as the characteristic time of a single cache of size $K \cdot C$ subject to the same demand of the overall sharded system. Assuming uniformly distributed random assignment of items to shards and applying Eq. 20 we obtain that T_S and T satisfy

$$\psi\left(T_S\right) = \frac{1}{C}\sum_{i=1}^{N} p_{in}(X_i\, p_i, T_S) = 1\,, \tag{46}$$

$$\phi(T) = \frac{1}{C}\sum_{i=1}^{N} p_{in}(p_i, T) = K\,, \tag{47}$$

where $\psi\left(T_S\right)$ is a random function of T_S, which depends on the r.v. $\{X_i\}_i$, while $\phi\left(T\right)$ is deterministic function.

We remark that $p_{in}(\cdot)$ in Eq. 46 and 47 are the same function because both caches operate according to the same replacement policy and p_{in} depends only on p_i and T because we assumed that such policy is determined by characteristic time.

By definition, if an item is never requested it cannot be in the cache, hence $p_{in}(0,T) = 0$. Because of this observation and since by definition $X_i \in \{0,1\}$, then we can express $p_{in}(X_i\, p_i, T_S)$in the form $X_i\, p_{in}(p_i, T_S)$. As a result, we can rewrite Eq. 46 as

$$\psi\left(T_S\right) = \frac{1}{C}\sum_{i=1}^{N} X_i\, p_{in}(p_i, T_S) = 1\,. \tag{48}$$

Now observe that

$$\begin{aligned}\mathbb{E}\left[\psi\left(T\right)\right] &= \mathbb{E}\left[\frac{1}{C}\sum_{i=1}^{N} X_i\, p_{in}(p_i, T)\right] = \frac{1}{KC}\sum_{i=1}^{N} p_{in}(p_i, T)\\ &= \frac{1}{K}\phi(T) = 1\,.\end{aligned} \tag{49}$$

We now proceed showing that $T_S \to T$ w.h.p. as C grows large. First, we note that since by definition $p_{in}(p_i, T) \in [0,1]$, $\forall i \in [1 \dots N]$, the following inequality holds:

$$\sum_{i=1}^{N} \left[p_{in}(p_i, T)\right]^2 \le \sum_{i=1}^{N} p_{in}(p_i, T)\,. \tag{50}$$

Jointly applying this inequality and Eq. 49, we derive the following upper bound of $\mathrm{Var}(\psi(T))$:

$$\begin{aligned}\mathrm{Var}\left(\psi\left(T\right)\right) &= \mathrm{Var}\left(\sum_{i=1}^{N} X_i\, p_{in}(p_i, T)\right)\\ &= \frac{K-1}{(KC)^2}\sum_{i=1}^{N}\left(p_{in}(p_i, T)\right)^2\\ &\le \frac{K-1}{(KC)^2}\sum_{i=1}^{N} p_{in}(p_i, T)\\ &= \left(1 - \frac{1}{K}\right)\frac{1}{C}\,.\end{aligned} \tag{51}$$

Then, jointly applying Chebyshev's inequality and Eq. 51 to $\psi(T)$ we obtain

$$\begin{aligned}P(|\psi(T) - \mathbb{E}\left[\psi(T)\right]| \ge \epsilon) &\le \frac{\mathrm{Var}(\psi(T))}{\epsilon^2 C^2}\\ &\le \frac{1 - 1/K}{\epsilon^2 C^2}\mathbb{E}\left[\psi(T)\right]\\ &= \left(1 - \frac{1}{K}\right)\frac{1}{\epsilon^2 C}\,.\end{aligned} \tag{52}$$

This implies that $\psi(T) \to \mathbb{E}\left[\psi(T)\right] = \frac{1}{K}\phi(T) = 1 = \psi(T_S)$ w.h.p. as C grows large.

Now, assuming $T < +\infty$, since $\psi(\cdot)$ given $\{X_i\}$, is by construction, a strictly increasing continuous function of its argument (and thus also $\psi^{-1}(\cdot)$ given $\{X_i\}$ is continuous and strictly increasing), then necessarily $T_s \to T$ w.h.p. as C and N grow large. Similarly if $T \to +\infty$ then necessarily also $T_S \to +\infty$. Therefore, since $p_{hit}(p_i, T_S) = p_{in}(p_i, T_S) \to p_{in}(p_i, T) = p_{hit}(p_i, T)$, the assertion immediately follows. □

PROOF OF THEOREM 7

To prove this theorem, we first introduce two preliminary lemmas:

Lemma 1. *Given two finite sequences $\{b_i\}_{i=1}^N$ and $\{c_i\}_{i=1}^N$, both non null and decreasing,*[5] *with $\sum_1^N c_i = 0$ then*

$$\sum_{i=1}^{N} b_i c_i \ge 0\,.$$

Proof. Since $\{c_i\}_{i=1}^N$ is decreasing with $\sum_1^N c_i = 0$, there exists an n with $1 < n < N$ such that: $c_i < 0$ for any $i \ge n$ and $c_i > 0$ for any $i < n$ (*i.e.,* $n = \min_i\{c_i < 0\}$).

Now, $\forall b \in \mathbb{R}$

$$\sum_{i=1}^{N} b_i c_i = \sum_{i=1}^{N} b_i c_i - \sum_{i=1}^{N} b c_i = \sum_{i=1}^{N} (b_i - b) c_i\,.$$

Then, the assertion follows by selecting $b = b_n$ and observing that, by construction, for any i with $1 \le i \le N$, we have $(b_i - b_n)c_i \ge 0$ (this is because $\{b_i\}_{i=1}^N$ is decreasing). □

[5]We use the terms increasing/decreasing in the weak sense, *i.e.,* not decreasing/not increasing

$$\frac{\partial c_v(L_C)}{\partial C} \approx \begin{cases} \dfrac{\sqrt{K-1}(1-\alpha)\left[2\frac{1-\alpha}{1-2\alpha}((N+1)^{1-2\alpha}-(C+1)^{1-2\alpha})-(C+1)^{-\alpha}\left((N+1)^{1-\alpha}-(C+1)^{1-\alpha}\right)\right]}{2\sqrt{\frac{(N+1)^{1-2\alpha}-(C+1)^{1-2\alpha}}{1-2\alpha}}(C+1)^{\alpha}\left[(N+1)^{1-\alpha}-(C+1)^{1-\alpha}\right]^2} & \text{if } \alpha \notin \{\frac{1}{2},1\} \\ \dfrac{\sqrt{K-1}\left[\log\left(\frac{N+1}{C+1}\right)-\sqrt{\frac{N+1}{C+1}}+1\right]}{4\sqrt{C+1}\left(\sqrt{N+1}-\sqrt{C+1}\right)^2\sqrt{\log\left(\frac{N+1}{C+1}\right)}} & \text{if } \alpha = \frac{1}{2} \\ \dfrac{\sqrt{K-1}\left[(N-C)-\frac{1}{2}(N+1)\log\left(\frac{N+1}{C+1}\right)\right]}{(C+1)^2(N+1)\log^2\left(\frac{N+1}{C+1}\right)\sqrt{\frac{N-C}{(N+1)(C+1)}}} & \text{if } \alpha = 1 \end{cases} \tag{53}$$

Lemma 2. *Given $\{a_i\}_{i=1}^N$, $\{b_i\}_{i=1}^N$, $\{c_i\}_{i=1}^N$ all non null and decreasing, with $a_i = b_i + c_i$ and $\sum_1^N c_i = 0$, then*

$$\sum_{i=1}^N a_i^2 > \sum_{i=1}^N b_i^2 .$$

Proof.

$$\sum_{i=1}^N a_i^2 = \sum_{i=1}^N (b_i+c_i)^2 = \sum_{i=1}^N b_i^2 + \sum_{i=1}^N c_i^2 + 2\sum_{i=1}^N b_i c_i \geq$$
$$\sum_{i=1}^N b_i^2 + \sum_{i=1}^N c_i^2 > \sum_{i=1}^N b_i^2 ,$$

where the second last inequality descends from Lemma 1. □

Now we are ready to prove the theorem.

Proof. Let us recall that $\{p_i\}_{i=1}^N$, $\{h_i\}_{i=1}^N$ and $\{p_i(1-h_i)\}_{i=1}^N$ are all decreasing. Observe that the coefficient of variation of a sequence does not change if we multiply all the terms of the sequence for a common strictly positive arbitrary factor. Thus we can define

$$\{q_i\}_1^N = \{Kp_i(1-h_i)\}_{i=1}^N$$

with $K := \frac{\sum_{j=1}^N p_j}{\sum_{j=1}^N p_j(1-h_j)}$ and compute $c_v^2(L_C)$, as

$$c_v^2(L_C) = \frac{\sum_{i=1}^N q_i^2}{\left(\sum_{i=1}^N q_i\right)^2} .$$

By construction we have: $\sum_{i=1}^N q_i = \sum_{i=1}^N p_i$. Furthermore we can write $q_i = p_i z_i$ with $\{z_i := K(1-h_i)\}_{i=1}^N$ increasing. Thus, we have

$$c_i := p_i - q_i = p_i(1-z_i) = p_i(1-K(1-h_i)) = p_i + Kp_i(h_i-1).$$

Now, since both $\{p_i\}_{i=1}^N$ and $\{h_i\}_{i=1}^N$ are decreasing, $\{c_i\}_{i=1}^N$ turns out to be decreasing too. Moreover $\sum_{i=1}^N c_i = \sum_{i=1}^N p_i - \sum_{i=1}^N q_i = 0$; therefore we can apply Lemma 2 to show that

$$\sum_{i=1}^N p_i^2 > \sum_{i=1}^N q_i^2 .$$

The assertion follows immediately, in fact

$$c_v^2(L) = \frac{\sum_{i=1}^N p_i^2}{(\sum_{i=1}^N p_i)^2} > \frac{\sum_{i=1}^N q_i^2}{(\sum_{i=1}^N p_i)^2} = \frac{\sum_{i=1}^N q_i^2}{(\sum_{i=1}^N q_i)^2} = c_v^2(L_C).$$

□

Proof of Theorem 8

Proof. We start deriving the partial derivative of $c_v(L_C)$ with respect to C for the three cases $\alpha \notin \{\frac{1}{2},1\}$, $\alpha = \frac{1}{2}$ and $\alpha = 1$, which we report in Eq. 53.

First, we analyze the case $\alpha \notin \{\frac{1}{2},1\}$. Solving the inequality $\partial c_v(L_C)/\partial C \geq 0$ and substituting $\gamma = \frac{C+1}{N+1}$ yields

$$\frac{1-\alpha}{1-2\alpha}\left[2(1-\alpha)\gamma^{2\alpha-1} - (1-2\alpha)\gamma^{\alpha-1} - 1\right] \geq 0 .$$

Now, we analyze the behavior of $\partial c_v(L_C)/\partial C$ to demonstrate that $c_v(L)$ has a unique global minimum for $C = C^*$.

Since by definition $0 \leq C < N$, we can immediately observe that $\gamma \in (0,1)$. More specifically, $\gamma \to 0^+$ when $C \to 0$ and $N \to +\infty$, while $\gamma \to 1^-$ when $C \to N$. The values of $\partial c_v(L_C)/\partial C$ at the boundaries of the γ interval are

$$\lim_{\gamma\to 0^+} \frac{\partial c_v(L_C)}{\partial C} = \begin{cases} -\frac{1-\alpha}{1-2\alpha} & \text{if } \alpha > 1 \\ -\infty & \text{if } \alpha \in \left(0,\frac{1}{2}\right) \cup \left(\frac{1}{2},1\right) \end{cases},$$

$$\lim_{\gamma\to 1^-} \frac{\partial c_v(L_C)}{\partial C} = 0 \quad \forall \alpha \in \mathbb{R}^+ \backslash \left\{\frac{1}{2},1\right\} .$$

Since by definition $\alpha > 0$, then

$$\lim_{\gamma\to 0^+} \frac{\partial c_v(L_C)}{\partial C} < 0 \quad \forall \alpha \in \mathbb{R}^+ \backslash \left\{\frac{1}{2},1\right\} .$$

We now investigate under what conditions $\partial \frac{\partial c_v(L_C)}{\partial C}/\partial \gamma \geq 0$ and observe that

$$\frac{\partial \frac{\partial c_v(L_C)}{\partial C}}{\partial \gamma} \geq 0 \Leftrightarrow (1-\alpha)^2\gamma^{\alpha-2}(-2\gamma^{\alpha}+1) \geq 0$$
$$\Leftrightarrow \gamma \leq 2^{-\frac{1}{\alpha}} .$$

This shows that $\partial c_v(L_C)/\partial C$ is negative for $\gamma \to 0^+$, strictly increases for $0 < \gamma < 2^{-1/\alpha}$, reaches a global maximum for $\gamma = 2^{-1/\alpha}$ and then strictly decreases for $2^{-1/\alpha} < \gamma < 1$ tending to 0 for $\gamma \to 1^-$.

From this analysis we can conclude that $\partial c_v(L_C)/\partial C$ is strictly positive at its global maximum ($\gamma = 2^{-1/\alpha}$). Since $\partial c_v(L_C)/\partial C$ is continuous over $\gamma \in (0,1)$, applying the intermediate value theorem we can conclude that there exists at least a value of $\gamma \in (0, 2^{-1/\alpha})$ for which $\partial c_v(L_C)/\partial C = 0$. Since over that interval $\partial c_v(L_C)/\partial C$ is strictly increasing, that root is unique and it is a local minimum of $c_v(L_C)$. Also, this analysis shows that $\partial c_v(L_C)/\partial C$ cannot have roots for $2^{-1/\alpha} < \gamma < 1$. Therefore, the minimum of $c_v(L)$ is global.

Finally, since we know that the only root of $\partial c_v(L_C)/\partial C$ is the global minimum of $c_v(L_C)$, we obtain Eq. 32 by rearranging $\partial c_v(L_C)/\partial C = 0$.

We now focus on the two remaining cases: $\alpha = \frac{1}{2}, 1$. Solving the inequality $\partial c_v(L_C)/\partial C \geq 0$ and substituting, as above, $\gamma = \frac{C+1}{N+1}$ yields

$$\frac{\partial c_v(L_C)}{\partial C} \geq 0 \Leftrightarrow \gamma \geq \begin{cases} -\frac{1}{4}\, W_{-1}\left(-\frac{1}{2}e^{-\frac{1}{2}}\right)^{-2} & \text{if } \alpha = \frac{1}{2} \\ -\frac{1}{2}\cdot W_0\left(-2e^{-2}\right) & \text{if } \alpha = 1 \end{cases}.$$

From this inequality it is immediately evident that $c_v(L_C)$ has a global minimum when γ is equal to the right hand side part, which corresponds to Eq. 33. □

REFERENCES

[1] D. Zhou, B. Fan, H. Lim, D. G. Andersen, M. Kaminsky, M. Mitzenmacher, R. Wang, and A. Singh, "Scaling up clustered network appliances with ScaleBricks," in *Proc. of the 2015 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '15)*, New York, NY, USA, pp. 241–254.

[2] K. Ross, "Hash routing for collections of shared web caches," *IEEE Network*, vol. 11, no. 6, pp. 37–44, 1997.

[3] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web," in *Proc. of the 29th Annual ACM Symposium on Theory of Computing (STOC'97)*, New York, NY, USA, pp. 654–663.

[4] D. G. Thaler and C. V. Ravishankar, "Using name-based mappings to increase hit rates," *IEEE/ACM Transactions on Networking*, vol. 6, no. 1, pp. 1–14, 1998.

[5] B. M. Maggs and R. K. Sitaraman, "Algorithmic nuggets in content delivery," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 3, pp. 52–66, 2015.

[6] Q. Huang, H. Gudmundsdottir, Y. Vigfusson, D. A. Freedman, K. Birman, and R. van Renesse, "Characterizing load imbalance in real-world networked caches," in *Proc. of the 13th ACM Workshop on Hot Topics in Networks (HotNets-XIII)*, New York, NY, USA, pp. 8:1–8:7.

[7] D. Perino, M. Varvello, L. Linguaglossa, R. Laufer, and R. Boislaigue, "Caesar: A content router for high-speed forwarding on content names," in *Proc. of the 10th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS'14)*, New York, NY, USA, pp. 137–148.

[8] H.-g. Choi, J. Yoo, T. Chung, N. Choi, T. Kwon, and Y. Choi, "CoRC: Coordinated Routing and Caching for Named Data Networking," in *Proceedings of the 10th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS'14)*, New York, NY, USA, pp. 161–172.

[9] H. Lim, D. Han, D. G. Andersen, and M. Kaminsky, "MICA: A holistic approach to fast in-memory key-value storage," in *Proc. of the 11th USENIX Conference on Networked Systems Design and Implementation (NSDI'14)*, Berkeley, CA, USA, pp. 429–444.

[10] X. Li, R. Sethi, M. Kaminsky, D. G. Andersen, and M. J. Freedman, "Be fast, cheap and in control with SwitchKV," in *Proc. of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI'16)*, Berkeley, CA, USA, pp. 31–44.

[11] L. Saino, I. Psaras, and G. Pavlou, "Understanding sharded caching systems," in *Proc. of the 35th IEEE International Conference on Computer Communications (INFOCOM'16)*, San Francisco, CA, USA, pp. 1–9.

[12] E. G. Coffman, Jr. and P. J. Denning, *Operating Systems Theory*. Prentice Hall Professional Technical Reference, 1973.

[13] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup protocol for internet applications," *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, pp. 17–32, 2003.

[14] U. Wieder, *Hashing, load balancing and multiple choice*. Now Publishers, 2017.

[15] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and Zipf-like distributions: evidence and implications," in *Proc. of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'99)*, New York, NY, USA, pp. 126–134.

[16] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny, "Workload analysis of a large-scale key-value store," in *Proc. of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '12)*, New York, NY, USA, 2012, pp. 53–64.

[17] S. K. Fayazbakhsh, Y. Lin, A. Tootoonchian, A. Ghodsi, T. Koponen, B. Maggs, K. Ng, V. Sekar, and S. Shenker, "Less pain, most of the gain: Incrementally deployable ICN," in *Proc. of the 2013 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM'13)*, New York, NY, USA, pp. 147–158.

[18] Y. Sun, S. K. Fayaz, Y. Guo, V. Sekar, Y. Jin, M. A. Kaafar, and S. Uhlig, "Trace-driven analysis of ICN caching algorithms on video-on-demand workloads," in *Proc. of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies (CoNEXT '14)*, New York, NY, USA, pp. 363–376.

[19] C. Imbrenda, L. Muscariello, and D. Rossi, "Analyzing cacheable traffic in ISP access networks for micro CDN applications via content-centric networking," in *Proc. of the 1st ACM Intl. Conference on Information-centric Networking (ICN'14)*, New York, NY, USA, pp. 57–66.

[20] M. Raab and A. Steger, "Balls into bins - a simple and tight analysis," in *Randomization and Approximation Techniques in Computer Science*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1998, vol. 1518, pp. 159–170.

[21] R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab, D. Stafford, T. Tung, and V. Venkataramani, "Scaling Memcache at Facebook," in *Proc. of the 10th USENIX Conference on Networked Systems Design and Implementation (NSDI'13)*, Berkeley, CA, USA, pp. 385–398.

[22] Y.-J. Hong and M. Thottethodi, "Understanding and mitigating the impact of load imbalance in the memory caching tier," in *Proc. of the 4th ACM Annual Symposium on Cloud Computing (SOCC'13)*, New York, NY, USA, pp. 13:1–13:17.

[23] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," in *Proc. of 21st ACM SIGOPS Symposium on Operating Systems Principles (SOSP'07)*, New York, NY, USA, pp. 205–220.

[24] R. Fagin, "Asymptotic miss ratios over independent references," *Journal of Computer and System Sciences*, vol. 14, no. 2, pp. 222–250, 1977.

[25] H. Che, Y. Tung, and Z. Wang, "Hierarchical Web caching systems: Modeling, design and experimental results," *IEEE Journal on Selected Areas of Communications*, vol. 20, no. 7, pp. 1305–1314, 2006.

[26] C. Fricker, P. Robert, and J. Roberts, "A versatile and accurate approximation for LRU cache performance," in *Proc. of the 24th International Teletraffic Congress (ITC'12)*, Krakow, Poland, pp. 8:1–8:8.

[27] V. Martina, M. Garetto, and E. Leonardi, "A unified approach to the performance analysis of caching systems," in *Proc. of the 33th IEEE Conference on Computer Communications (INFOCOM'14)*, Toronto, Canada, pp. 2040–2048.

[28] R. W. Wolff, "Poisson arrivals see time averages," *Operations Research*, vol. 30, no. 2, pp. 223–231, 1982.

[29] E. Gelenbe, "A unified approach to the evaluation of a class of replacement algorithms," *IEEE Transactions on Computers*, vol. C-22, no. 6, pp. 611–618, 1973.

[30] K. Ji, G. Quan, and J. Tan, "Asymptotic miss ratio of LRU caching with consistent hashing," in *Proc. of the 37th IEEE International Conference on Computer Communications (INFOCOM'18)*, Honolulu, HI, USA, pp. 450–458.

[31] J. A. Fill, "Limits and rates of convergence for the distribution of search cost under the move-to-front rule," *Theoretical Computer Science*, vol. 164, no. 1, pp. 185 – 206, 1996.

[32] P. R. Jelenković, "Asymptotic approximation of the Move-to-Front search cost distribution and Least-Recently Used caching fault probabilities," *The Annals of Applied Probability*, vol. 9, no. 2, pp. 430–464, 1999.

[33] D. G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan, "FAWN: A Fast Array of Wimpy Nodes," in *Proc. of the ACM SIGOPS 22nd Symposium on Operating Systems Principles (SOSP'09)*, New York, NY, USA, pp. 1–14.

[34] B. Fan, H. Lim, D. G. Andersen, and M. Kaminsky, "Small cache, big effect: Provable load balancing for randomly partitioned cluster services," in *Proc. of the 2nd ACM Symposium on Cloud Computing (SOCC'11)*, New York, NY, USA, pp. 23:1–23:12.

[35] P. R. Jelenković and X. Kang, "Characterizing the miss sequence of the LRU cache," *SIGMETRICS Perform. Eval. Rev.*, vol. 36, no. 2, pp. 119–121, 2008.