Distributed Selfish Caching[§][*]

Nikolaos Laoutaris[†][‡], Georgios Smaragdakis[†], Azer Bestavros[†]

Ibrahim Matta[†], Ioannis Stavrakakis[‡]

Abstract

Although cooperation generally increases the amount of resources available to a community of nodes, thus improving individual and collective performance, it also allows for the appearance of potential mistreatment problems through the exposition of one node's resources to others. We study such concerns by considering a group of independent, rational, self-aware nodes that cooperate using on-line caching algorithms, where the exposed resource is the storage at each node. Motivated by *content networking* applications – including web caching, CDNs, and P2P – this paper extends our previous work on the off-line version of the problem, which was conducted under a game-theoretic framework, and limited to object replication. We identify and investigate two causes of mistreatment: (1) cache *state interactions* (due to the cooperative servicing of requests) and (2) the adoption of a *common scheme* for cache management policies. Using analytic models, numerical solutions of these models, as well as simulation experiments, we show that on-line cooperation schemes using caching are fairly robust to mistreatment caused by state interactions. To appear in a substantial manner, the interaction through the exchange of miss-streams has to be very intense, making it feasible for the mistreated nodes to detect and react to exploitation. This robustness ceases to exist when nodes fetch and store objects in response to remote requests, *i.e.*, when they operate as Level-2 caches (or proxies) for other nodes. Regarding mistreatment due to a common scheme, we show that this can easily take place when the "outlier" characteristics of some of the

[*]E-mail:{nlaout, gsmaragd, best, matta}@cs.bu.edu, istavrak@di.uoa.gr

[†] Computer Science Dept, Boston University, Boston, Massachusetts, USA.

[‡] Dept of Informatics and Telecommunications, University of Athens, Greece.

[§] A. Bestavros and I. Matta are supported in part by a number of NSF awards, including CNS Cybertrust Award #0524477, CNS NeTS Award #0520166, CNS ITR Award #0205294, and EIA RI Award 0202067. I. Stavrakakis is supported in part by EU IST projects CASCADAS and E-NEXT. N. Laoutaris is supported by a Marie Curie Outgoing International Fellowship of the EU MOIF-CT-2005-007230.

nodes get overlooked. This finding underscores the importance of allowing cooperative caching nodes the flexibility of choosing from a diverse set of schemes to fit the peculiarities of individual nodes. To that end, we outline an emulation-based framework for the development of mistreatment-resilient distributed selfish caching schemes.

1 Introduction

Background, Motivation, and Scope: Network applications often rely on distributed resources available within a cooperative grouping of nodes to ensure scalability and efficiency. Traditionally, such groupings are dictated by an overarching, common strategic goal. For example, nodes in a CDN such as Akamai or Speedera cooperate to optimize the performance of the overall network, whereas IGP routers in an Autonomous System (AS) cooperate to optimize routing within the AS.

More recently, however, new classes of network applications have emerged for which the grouping of nodes is more "ad hoc" in the sense that it is not dictated by organizational boundaries or strategic goals. Examples include overlay protocols [3, 6] and peer-to-peer (P2P) applications. Two distinctive features of such applications are (1) individual nodes are autonomous, and as such, their membership in a group is motivated solely by the selfish goal of *benefiting* from that group, and (2) group membership is warranted only as long as a node is interested in being part of the application or protocol, and as such, group membership is expected to be fluid. In light of these characteristics, an important question is this: *Are protocols and applications that rely on sharing of distributed resources appropriate for this new breed of ad-hoc node associations?*

In this paper, we answer this question for networking applications, whereby the distributed resource being shared amongst a group of nodes is *storage*. While our work and methodology is applicable for a wide range of applications that rely on distributed shared storage, we target the distribution of voluminous content as our application of choice.[1]In particular, we consider a group of nodes that store information objects and make them available to their local users as well as to remote nodes. A user's request is first received by the local node. If the requested object is stored locally, it is returned to the requesting user immediately, thereby incurring

[1]Unlike content distribution for static (typically small) web objects such as html web pages and images, voluminous content requires treating the storage as a limited resource [21].

a minimal access cost. Otherwise, the requested object is searched for, and fetched from other nodes of the group, at a potentially higher access cost. If the object cannot be located anywhere in the group, it is retrieved from an origin server, which is assumed to be outside the group, thus incurring a maximal access cost.

Under an *object replication* model, once selected for replication at a node, an object is stored permanently at that node (*i.e.*, the object cannot be replaced later). In [20] we established the vulnerability of *socially optimal* (SO) object replication schemes in the literature to *mistreatment* problems. We define a mistreated node to be a node whose access cost under SO replication is higher than the minimal access cost that the node can guarantee under greedy local (GL) replication. Unlike centrally designed/controlled groups where all constituent nodes have to abide by the ultimate goal of optimizing the social utility of the group, an autonomous, selfish node will not tolerate such a mistreatment. Indeed, the emergence of such mistreatments may cause selfish nodes to secede from the replication group, resulting in severe inefficiencies for both the individual users as well as the entire group.

In [20], we resolved this dilemma by proposing a family of *equilibrium* (EQ) object placement strategies that (a) avoid the mistreatment problems of SO, (b) outperform GL by claiming available "cooperation gain" that the GL algorithm fails to utilize, and (c) are implementable in a distributed manner, requiring the exchange of only a limited amount of information. The EQ strategies were obtained by formulating the *Distributed Selfish Replication* (DSR) game and devising a distributed algorithm that is always capable of finding pure Nash equilibrium strategies for this particular game.

Distributed Selfish Caching: Proactive replication strategies are not practical in a highly dynamic content networking setting, which is likely to be the case for most of the Internet overlays and P2P applications we envision. This is due to a variety of reasons: (1) Fluid group membership makes it impractical for nodes to decide what to replicate based on what (and where) objects are replicated in the group. (2) Access patterns as well as access costs may be highly dynamic (due to bursty network/server loads), necessitating that the selection of replicas and their placement be done continuously, which is not practical. (3) Both the identification of the appropriate re-invocation times [24] and the estimation of the non-stationary demands (or equivalently, the timescale for a stationarity assumption to hold) [14] are non-trivial problems. (4) Content objects may be dynamic and/or may expire, necessitating the use of "pull" (*i.e.*, on-demand *caching*) as opposed to "push" (*i.e.*, pro-active replication) approaches. Using on-demand caching is the most widely acceptable and natural solution to all of these issues because it requires no *a priori* knowledge of local/group demand patterns and, as a consequence, responds dynamically to changes in these patterns over time (*e.g.*, introduction of new objects, reduction in the popularity of older ones, *etc.*)

Therefore, in this paper we consider the problem of *Distributed Selfish Caching* (DSC), which can be seen as the *on-line* counterpart to the DSR problem. In DSC, we adopt an *object caching* model, whereby a node employs demand-driven temporary storage of objects, combined with replacement. At this juncture, it is important to note that we make a clear distinction between *replication* and *caching*. While these terms may be seen as similar (and indeed used interchangeably in much of the literature), we note that for our purposes they carry quite different meanings and implications. Replication amounts to maintaining permanent copies whereas caching amounts to maintaining temporary copies. This changes fundamentally the character and the methodologies used in analyzing DSR and DSC.

Causes of Mistreatments Under DSC: We begin our examination of DSC by first considering the operational characteristics of a group of nodes involved in a distributed caching solution. This examination will enable us to identify two key culprits for the emergence of mistreatment phenomena.

First, we identify the mutual *state interaction* between replacement algorithms running on different nodes as the prime culprit for the appearance of mistreatment phenomena. This interaction takes place through the so called "remote hits". Consider nodes v, u and object o. A request for object o issued by a user of v that cannot be served at v but could be served at u is said to have incurred a *local miss* at v, but a *remote hit* at u. Consider now the implications of the remote hit at u. If u does not discriminate between hits due to local requests and hits due to remote requests (which is the default behavior of the Internet Cache Protocol (ICP) / Squid web cache [8] and other systems (e.g., Akamai Content Distribution Network, IBM Olympic Server Architecture), then the remote hit for object o will affect the state of the replacement algorithm in effect at u. If u is employing Least Recently Used (LRU) replacement, then o will be brought to the top of the LRU list. If it employs Least Frequently Used (LFU) replacement, then its frequency will be increased, and so on with other replacement algorithms [29]. If the frequency of remote hits is sufficiently high, *e.g.*, because v has a much higher local request rate and thus sends an intense miss-stream to u, then there could be performance implications for the second: u's cache may get invaded by objects that follow v's demand, thereby depriving the user's of u from valuable storage space for caching their own objects. This can lead to the mistreatment of u, whose cache is effectively "hijacked" by v.

Moving on, we identify a second, less anticipated, culprit for the emergence of mistreatment in DSC. We call it the *common scheme* problem. To understand it, one has to observe that most of the work on cooperative caching has hinged on the fundamental assumption that all nodes in a cooperating group adopt a common scheme. We use the word "scheme" to refer to the combination of: (i) the employed *replacement algorithm*, (ii) the employed *request redirection algorithm*, and (iii) the employed *object admission algorithm*. Cases (i) and (ii) are more or less self-explanatory. Case (iii) refers to the decision of whether to cache locally an incoming object after a local miss. The problem here is that the adoption of a common scheme can be beneficial to some of the nodes of a group, but harmful to others, particularly to nodes that have special characteristics that make them "outliers". A simple case of an outlier, is a node that is situated further away from the center of the group, where most nodes lie. Here distance may have a topological/affine meaning (*e.g.*, number of hops, or propagation delay), or it may relate to dynamic performance characteristics (*e.g.*, variable throughput or latencies due to load conditions on network links or server nodes). Such an outlier node cannot rely on the other nodes for fetching objects at a small access cost, and thus prefers to keep local copies of all incoming objects. The rest of the nodes, however, as long as they are close enough to each other, prefer not to cache local copies of incoming objects that already exist elsewhere in the group. Since such objects can be fetched from remote nodes at a small access cost, it is better to preserve the local storage for keeping objects that do not exist in the group and, thus, must be fetched from the origin server at a high access cost. In this setting, a common scheme is bound to mistreat either the outlier node or the rest of the group.

In addition to the identification of the two causes of mistreatments in a DSC setting, this paper presents a number of concrete results regarding each one of these two causes. These results are intended to be used as basic design guidelines on dealing with selfishness in current and future caching applications. Admittedly we do not present a full cache design since this would require addressing additional issues like data consistency, cache associativity, and other application specific details. We avoid obscuring the presentation with such details, which to a large extended are orthogonal to selfishness, our main theme here.

Mistreatment Due to Cache State Interaction: Regarding the state interaction problem, our investigations answer the following basic question: *"Could and under which schemes do mistreatments arise in a DSC group?"*. More, specifically:

+ We show that state interactions occur when nodes do not discriminate between local and remote *hits* upon updating the state of their replacement algorithms.

+ To materialize, state interactions require substantial request rate imbalance, *i.e.*, one or more "overactive" nodes must generate disproportionally more requests than the other nodes. Even in this case, mistreatment of less active nodes depends on the amount of storage that they posses: Mistreatment occurs when these nodes have abundant storage, otherwise they are generally immune to, or even benefit from, the existence of overactive nodes.

+ Comparing caching and replication with regard to their relative sensitivities to request rate imbalance, we show that caching is much more robust than replication.

+ Regarding the vulnerability of different replacement algorithms, we show that "noisier" replacement algorithms are more prone to state interactions. In that regard, we show that LRU is more vulnerable than LFU.

+ Even the most vulnerable LRU replacement is quite robust to mistreatment as it requires a very intense miss-stream in order to force a mistreated node to maintain locally unpopular objects in its cache (thus depriving it of cache space for locally popular objects). In particular, the miss-stream has to be strong enough to counter the sharp decline in the popularity of objects in typically skewed workloads.

+ Robustness to mistreatment due to state interaction evaporates when a node operates as a Level-2 cache [36] (L2) for other nodes. L2 caching allows all remote requests (whether they hit or miss) to affect the local state (as opposed to only hits under non-L2 caching), leading to a vulnerability level that approaches the one under replication.

Mistreatment Due to Use of Common Scheme: We classify cooperative caching schemes into two groups: *Single Copy* (SC) schemes, *i.e.*, schemes where there can be at most one copy of each distinct object in the group – two examples of SC schemes are HASH based caching [32] and LRU-SC [10]; *Multiple Copy* (MC) schemes, *i.e.*, schemes where there can be multiple copies of the same object at different nodes.

+ We show that the relative performance ranking of SC and MC schemes changes with the "tightness" of a cooperative group. SC schemes perform best when the inter-node distances are small compared to the distance to the origin server; in such cases the maintenance of multiple copies of the same object becomes unnecessary.[2] MC schemes improve progressively as the inter-node distances increase, and eventually outperform the SC schemes.

+ We demonstrate a case of mistreatment due to common scheme by considering a tight group of nodes that operate under SC and a unique outlier node that has a larger distance to the group. We show that this node is mistreated if it is forced to follow the same SC scheme.

Towards Mistreatment-Resilient DSC Schemes: More constructively, we present a framework for the design of mistreatment-resilient DSC schemes. Our framework allows individual nodes to decide autonomously (*i.e.*, without having to trust any other node or service) whether they should stick to, or secede from a DSC caching group, based on whether or not their participation is beneficial to their performance compared to a selfish, greedy scheme. Resilience to mistreatments is achieved by allowing a node to emulate the performance gain possible by switching from one scheme to another, or by adapting some control parameters of its currently deployed DSC scheme. We use a simple control-theoretic approach to dynamically parametrize the DSC scheme in use by a local node. We evaluate the performance of our solution by considering caching in wireless mobile nodes [37] where distances and download rates depend on mobility patterns. We show that our adaptive schemes can yield substantial performance benefits, especially under skewed demand profiles.

2 Related work

Apart from our previous work on distributed selfish replication [20], we are aware of only two additional works on game-theoretic aspects of replication, one due to Chun et al. [4] (distributed selfish replication under infinite storage capacities) and the other due to Erçetin and Tassiulas [9] (market-based resource allocation in content delivery); we are not aware of any previous work on distributed selfish caching. The issue of dynamic adjustment of caching schemes has been raised recently also by Sivasubramanian et al. [33], in a completely different context from ours (consistency control of cached objects). For studies on the social utility of

[2]We do not consider load balancing concerns in this study.

distributed caching we recommend Korupolu and Dahlin [16] and Rodriguez et al. [31].

Our consideration of DSC builds upon many facets of caching in general, and cooperative web caching protocols and systems in particular – facets which have been studied extensively in the literature. Given the multi-faceted nature of the relationship between our work and this body of literature, and for the sake of a better exposition of our contributions, rather than enumerating these studies here, we discuss how we leverage and relate to such works throughout the paper, as appropriate.

3 Definitions and Notation

Let o_i, $1 \leq i \leq N$, and v_j, $1 \leq j \leq n$, denote the ith unit-sized object and the jth node, and let $O = \{o_1, \ldots, o_N\}$ and $V = \{v_1, \ldots, v_n\}$ denote the corresponding sets. Node v_j is assumed to have storage capacity for up to C_j unit-sized objects, a total request rate λ_j (total number of requests per unit time, across all objects), and a demand described by a probability distribution over O, $\vec{p}_j = \{p_{1j}, \ldots, p_{Nj}\}$, where p_{ij} denotes the probability of object o_i being request by the local users of node v_j. Successive requests are assumed to be independent and identically distributed.[3] Later in this paper, we make the specific assumption that the popularity of objects follows a power-law profile, *i.e.*, the ith most popular object is requested with probability $p_i = K/i^a$. Such popularity distributions occur in many measured workloads [2, 25] and, although used occasionally in our work (*e.g.*, in Section 4.1 to simplify an analytic argument, or in Section 5 for producing numerical results), they do not constitute a basic assumption, in the sense that mistreatment can very well occur with other demand distributions that do not follow such a profile.

Let t_l, t_r, t_s denote the access cost paid for fetching an object locally, remotely, or from the origin server, respectively, where $t_s > t_r > t_l$.[4] User requests are serviced by the closest node that stores the requested object along the following chain: local node, group, and origin server. Each node employs a replacement algorithm for managing the content of its cache and employs an object admission algorithm for accepting (or not) incoming objects.

[3]The Independent Reference Model (IRM) [5] is commonly used to characterize cache access patterns [1, 2]. The impact of temporal correlations was shown in [14, 30] to be minuscule, especially under typical, Zipf-like object popularity profiles.

[4]The assumption that the access cost is the same across all node pairs in the group is made only for the sake of simplifying the presentation. Our results can be adapted easily to accommodate arbitrary inter-node distances.

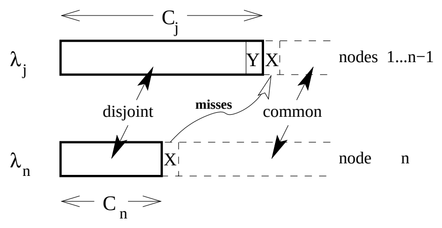

Figure 1: Reference model for the study of mistreatment due to state interaction.

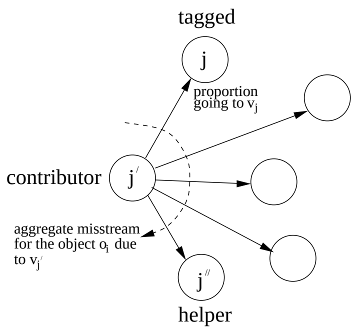
tagged
j
proportion going to v_j
contributor
j'
aggregate misstream for the object o_i due to $v_{j'}$
j''
helper

Figure 2: Graphical illustration for the explanation of Eq. (7).

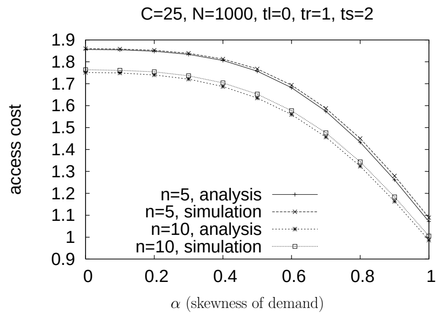

Figure 3: Validation of the approximate analytical model of Section 4.2 through comparison with simulation results on the social cost of the group.

4 Mistreatment Due to State Interaction: Analysis

Our goal in this section is to understand the conditions under which mistreatment may arise as a result of (cache) state interactions. We start in Section 4.1 with a replacement-agnostic model that focuses on the rate-imbalance (between the local request stream and the remote miss stream) necessary for mistreatment to set in. Next, in Section 4.2, we present a more detailed analytical model that allows for the derivation of the average access cost in a distributed caching group composed of n nodes that operate under LRU replacement.

4.1 General conditions

We would like to determine the level of *request rate imbalance* that is necessary for mistreatment to be feasible. We model this imbalance through the ratio λ_n/λ_j, where λ_j denotes the request rate of any normally-behaving node v_j, while λ_n denotes the request rate of an overactive node, which we use to instigate mistreatment problems. As a convention we assume this overactive node to be the last (n^{th}) node of the group.

We focus on the interaction between v_j and v_n. Fig. 1 shows a particular choice of demand patterns that fosters the occurrence of mistreatment. The initial most popular objects in $\vec{p}_j$ and $\vec{p}_n$ up to the two capacities (C_j for v_j and C_n for v_n) are completely disjoint, while the remaining ones in the middle part of the two distributions are identical; both demands are assumed to be power-law with parameter a. Let X denote the most popular object that is common to both distributions. A boundary condition for the occurrence of mistreatment can be obtained by considering the ratio λ_n/λ_j that results in a switch of ranking between X and Y at v_j, where Y denotes the least most popular object that would be kept in the cache of

v_j under a perfect ranking of objects according to the local demand, if no miss-stream was received. To derive the condition for the switch we first note that X is the (C_j+1)th most popular object for v_j and the (C_n+1)th most popular one for v_n. Y is the C_jth most popular object for v_j. Let $f(n)$ denote a function that captures the operation of different object location mechanisms in a group of n nodes (used for locating and retrieving objects from remote nodes). For example, $f(n)=1$ can be used for modeling request flooding (following a local miss, a request is sent to all other nodes in the group); $f(n)=1/(n-1)$ can be used for modeling index-based mechanisms [10] (following a local miss, a request is sent to only one of the nodes that appear to be storing the object according to some index). The boundary condition for the occurrence of the switch can be written as follows:

$$\begin{aligned}&\lambda_j p_{C_j} \le \lambda_j p_{C_j+1} + \lambda_n p_{C_n+1} f(n) \Rightarrow \lambda_j \frac{1}{(C_j)^a} \le \lambda_j \frac{1}{(C_j+1)^a} + \lambda_n \frac{1}{(C_n+1)^a} f(n) \Rightarrow \\ &\frac{\lambda_n}{\lambda_j} \ge \frac{(C_n+1)^a}{f(n)} \left(\frac{1}{C_j^a} - \frac{1}{(C_j+1)^a} \right)\end{aligned} \tag{1}$$

Writing a continuous approximation for the rate of change of $1/C^a$ with respect to C, we get:

$$\frac{d\left(\frac{1}{C^a}\right)}{dC} = \frac{\frac{1}{C^a} - \frac{1}{(C+1)^a}}{C-(C+1)} \approx -a \cdot \frac{1}{C^{1+a}} \Rightarrow \frac{1}{C^a} - \frac{1}{(C+1)^a} \approx a \cdot \frac{1}{C^{1+a}} \tag{2}$$

Using the approximation from Eq. (2) on Eq. (1) we obtain:

$$\frac{\lambda_n}{\lambda_j} \ge \frac{(C_n+1)^a}{f(n)} \cdot a \frac{1}{(C_j)^{1+a}} \sim \frac{a}{f(n) C_j} \left(\frac{C_n}{C_j} \right)^a \tag{3}$$

Eq. (3) states that the amount of imbalance in request rates ($\frac{\lambda_n}{\lambda_j}$) that is required for the occurrence of mistreatment is: (i) increasing with C_n, (ii) decreasing with C_j, and (iii) increasing when request flooding is employed for locating remote objects (in this case all the nodes get the full miss-stream from v_n, otherwise the miss-stream weakens by being split into $n-1$ parts).

Now assume that as a result of the received miss-stream, k objects of v_j are switched (objects with ids $C_j, \ldots, C_j-k+1$ evicted, objects $C_j+1, \ldots, C_j+k$ inserted); k can be computed from a condition similar to that in (Eq. 1). Define the *Loss* of v_j as the reduction in the probability mass of the objects that it caches locally.

$$Loss = \sum_{i=C_j-k+1}^{C_j} p_i - \sum_{i=C_j+1}^{C_j+k} p_i = K \cdot (H_{C_j}^{(a)} - H_{C_j-k}^{(a)} - H_{C_j+k}^{(a)} + H_{C_j}^{(a)}) = K \cdot (2H_{C_j}^{(a)} - H_{C_j-k}^{(a)} - H_{C_j+k}^{(a)}), \tag{4}$$

$$cost_j = \sum_{i=1}^{N} p_{ij} \cdot [\pi_{ij} \cdot t_l + (1-\pi_{ij}) \cdot \pi_{i-j} \cdot t_r + (1-\pi_{ij}) \cdot (1-\pi_{i-j}) \cdot t_s] \quad \text{where} \quad \pi_{i-j} = 1 - \prod_{\forall j' \neq j} (1-\pi_{ij'}) \tag{6}$$

$$p_{ij}^{(k)} = \frac{\lambda_j \cdot p_{ij} + \sum_{j'=1, j'\neq j}^{n} \lambda_{j'} \cdot p_{ij'} \cdot (1-\pi_{ij'}^{(k-1)}) \cdot \left[\frac{\left(\pi_{ij}^{(k-1)}\right)^2}{\sum_{j''=1, j''\neq j'}^{n} \pi_{ij''}^{(k-1)}} \right]_{\pi_{ij}^{(k-1)}}^{+}}{\sum_{i'=1}^{N} \left(\lambda_j \cdot p_{i'j} + \sum_{j'=1, j'\neq j}^{n} \lambda_{j'} \cdot p_{i'j'} \cdot (1-\pi_{i'j'}^{(k-1)}) \cdot \left[\frac{\left(\pi_{i'j}^{(k-1)}\right)^2}{\sum_{j''=1, j''\neq j'}^{n} \pi_{i'j''}^{(k-1)}} \right]_{\pi_{i'j}^{(k-1)}}^{+} \right)} \tag{7}$$

---

where K is the normalization constant of the power-law distribution $p_i = K/i^a$. The generalized harmonic number $H_C^{(a)}$ can be approximated by its integral expression (see [35]) $H_C^{(a)} = \sum_{i=1}^{C} \frac{1}{i^a} \approx \int_1^C 1/l^a dl = \frac{C^{1-a}-1}{1-a}$. Plugging this into Eq. (4) we obtain:

$$Loss = K\left(2\frac{{C_j}^{1-a}-1}{1-a} - \frac{(C_j-k)^{1-a}-1}{1-a} - \frac{(C_j+k)^{1-a}-1}{1-a} \right) \tag{5}$$

¿From Eq. (5) it is clear that as C_j increases, both $C_j - k$ and $C_j + k \rightarrow C_j$ thus leading to $Loss \rightarrow 0$. Combining our observations from Eq. (3) and Eq. (5) we conclude that the *occurrence* of mistreatment is fostered by small C_n and large C_j. Its magnitude, however, decreases with C_j. So, practically, it is in intermediate values of C_j that mistreatment can arise in a substantial manner.

4.2 Analysis of Mistreatment Under LRU Replacement

In the remainder of this section, our objective will be to derive the steady-state hit probabilities $\vec{\pi}_j = \{\pi_{1j}, \ldots, \pi_{Nj}\}$, where π_{ij} denotes the steady-state probability of finding object o_i at node v_j that operates under LRU replacement. We will then use these results for studying mistreatment in the context of LRU.

Let $\vec{\pi} = LRU(\vec{p}, C)$ denote a function that computes the steady-state object hit probabilities for a single LRU cache in isolation, given the cache size and the demand distribution. Due to the combinatorial hardness of analyzing LRU replacement, it is difficult to derive an exact value for $\vec{\pi}$; there are, however, several methods for computing approximate values for it (see for example [18] and references therein). In this paper, we employ the approximate method of Dan and Towsley in [7] that provides an accurate estimation of $\vec{\pi}$ through an iterative algorithm that incurs $O(NC)$ time complexity. Having computed $\vec{\pi}_j$, $\forall v_j \in V$, we can obtain the per-node access cost, $cost_j$, as well as the social cost of the entire group, $cost_{soc} = \sum_{\forall v_j} cost_j$, by using Eq. (6). In this equation π_{i-j} denotes the probability of finding o_i in any node of the group other than v_j.

We can obtain $\vec{\pi}_j$ by using the $LRU(\cdot,\cdot)$ function for isolated caches as our basic building block and taking into consideration the impact on the local state of the hits caused by remote requests. Deriving an exact expression for these added hits based on the involved cache states is intractable as it leads to state-space explosion. Therefore, we turn to approximate techniques and, in particular, to techniques that consider the expected values of the involved random variables, instead of their exact distributions. The basic idea of our approach is to capture these added hits by properly modifying the input to the $LRU(\cdot,\cdot)$ function.

Remote hits can be considered simply as additional request that augment the local demand, thereby creating a new aggregate demand for the $LRU(\cdot,\cdot)$ function as explained later. The idea of modifying the input of a simpler system to capture a policy aspect of a more complex system and then using the modified simpler system to study the more complex one has been employed frequently in the past [17]. Since the remote hits are shaped by the cache states, which are coupled due to the exchanges of miss-streams, an iterative procedure is followed for the derivation of the per-node steady-state vectors and access costs. The uncoupled solution (corresponding to nodes operating in isolation) is obtained first, and is refined progressively by taking into account the derived states and the cooperative servicing of the misses. The resulting approximate analytical model for predicting the average access cost in a distributed caching group is described below. In the next section, we show that the results produced from this model match quite well the results obtained through simulations.
The iterative procedure follows:

(1) For each node v_j compute $\vec{\pi}_j^{(0)} = LRU(\vec{p}_j, C_j)$, *i.e.*, assume no state interaction among the different nodes.

(2) Initiate Iteration: At the kth iteration the aggregate demand distribution for v_j, $\vec{p_j}^{(k)} = \{p_{ij}^{(k)}\}$, $1 \le i \le N$, is given by Eq. (7) (see top of page). The function $[x]_y^+$ returns 0 if $y = 0$ and x otherwise.[5] The steady state vector of object hit probabilities for v_j at

[5]This function is used to ensure correctness when the denominator $\sum_{j''=1,j''\neq j'}^{n} \pi_{ij''}^{(k-1)}$ becomes zero. Notice that the nominator $\pi_{ij}^{(k-1)}$ is included in the denominator, so when $\pi_{ij}^{(k-1)} > 0$, the denominator is guaranteed to be non zero.

iteration k can be obtain from: $\vec{\pi}_j^{(k)} = LRU(\vec{p_j}^{(k)}, C_j)$

(3) Convergence Test: if $|\vec{\pi}_j^{(k)} - \vec{\pi}_j^{(k-1)}| < \vec{\epsilon}$ for all v_j, $1 \leq j \leq n$, then set $\vec{\pi}_j = \vec{\pi}_j^{(k)}$ and compute the per node access costs from Eq. (6); $\vec{\epsilon}$ denotes a user-defined tolerance for the convergence of the iterative method. Otherwise, set $\vec{\pi}_j^{(k-1)} = \vec{\pi}_j^{(k)}$ and $\vec{p}_j^{(k-1)} = \vec{p}_j^{(k)}$ and perform another iteration by returning to step 2.

The nominator of Eq. (7) adds the requests generated by the local population of v_j for object o_i, to the requests for the same object due to the $n-1$ miss streams from all other nodes that create hits at v_j. The explanation of the circumstances under which such hits exist, goes as follows (see also Fig. 2): a request for o_i received at the *contributor* node $v_{j'}$ (prob. $p_{ij'}$) affects the *tagged* node v_j, if the request cannot be serviced at the contributor node (prob. $(1 - \pi_{ij'}^{(k-1)})$), can be serviced at the tagged node (prob. $\pi_{ij}^{(k-1)}$), and is indeed serviced by the tagged node and not by any other *helper* node $v_{j''}$ that can potentially service it (prob. $\pi_{ij}^{(k-1)} / \sum_{j''=1, j'' \neq j'}^{n} \pi_{ij''}^{(k-1)}$, *i.e.*, the model assumes that when more than one helper nodes can offer service, then the request is assigned uniformly to any one of them).

Before we conclude this section, we note that our aforementioned analysis could be construed as providing a lower bound of the intensity of mistreatment assuming that the proxy is configured such that only *one* peer (proxy cache) replies to a remote request. Mistreatment could be more severe if, upon a local miss, requests are routed to more than one proxy, which is the case in many real systems [8].

5 Mistreatment Due to State Interaction: Evaluation

In this section, we use a combination of simulation experiments and numerical solutions of the analytical model developed in the previous section to explore the design space of distributed caching with respect to its vulnerability to the on-set of mistreatment as a result of the state interaction phenomenon. We start by validating the accuracy of the analytical model of Section 4.2 and follow that with an examination of various dimensions of the design space for distributed caching, including a comparative evaluation of mistreatment in caching versus replication.

It is important to note that throughout this section, we use a number of settings to gain an understanding of state interaction in distributed caches and its consequences on local and group access costs. Most of these settings are *intentionally* very simple (*i.e.*, small "toy" examples) so that they can be possible to track.

Also, it is important to note that the various parameterizations of our analytical and simulation models are not meant to represent particular content networking applications. Examining specific incarnations of the state interaction phenomenon is, after all, not our intention in this paper—which is the first to identify and analyze the problem. Rather, our exploration of the extent of mistreatment is meant to help us gain insights into the fundamental aspects of state interactions in distributed caching, such as its dependence on the request rate imbalance and the nodes' relative storage capacities.[6] In most of the following numerical results we assume that nodes follow a common power-law demand distribution with skewness a as reported by several measurement studies [10]. We relax the common demand distribution assumption in Sect. 5.3 where we study the effect of non-homogeneous demand on mistreatment and the social cost of the group. Overall we pay greater attention to the case of homogeneous demand, since it is under such demand that cooperative caching becomes meaningful and effective (the benefits from employing cooperative caching diminish when the similarity of demand patterns is small).

5.1 Validation of the Analytic Model

The analytical model presented in Section 4.2 included a number of approximations—namely: (i) the basic building block, the $LRU(\cdot, \cdot)$ function, is itself an approximation; (ii) the mapping of the effect of remote hits on the local state through Eq. (7) is approximate; the solution of the model through the iterative method is approximate.

In this section, we show that despite these approximations, the analytical model presented in Section 4.2 is able to produce fairly accurate results. We do so by comparing the model predictions with simulation results in Fig. 3. As evident from these results, the aforementioned approximations have a very limited effect on the model's prediction accuracy. We have obtained similar results across a wide variety of parameter sets. Thus, in the remainder of this section, we use this model to study several aspects of mistreatment due to state interaction.

[6]With respect to storage capacities, it is important to note that performance results depend on the relative size of the cache to the object space–*i.e.*, the ratio C/N, but not on the particular values of C and N, *i.e.*, our results are immune to scale.

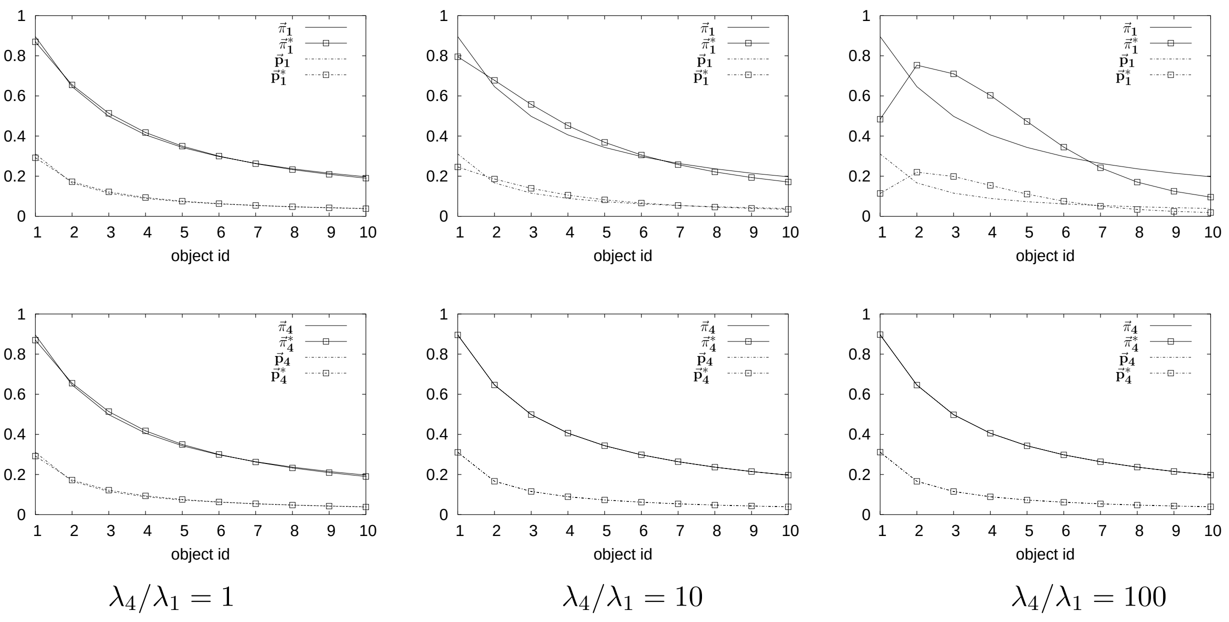

Figure 4: Analytical results on the effect of request rate imbalance on the per object request and hit probabilities under LRU (values with "*" superscript) and LRU without state interaction. $\vec{p}$ denotes the demand and $\vec{\pi}$ the steady-state hit probabilities. Other parameters: $N = 10,\ n = 4,\ C = 4,\ \alpha = 0.9$.

5.2 Understanding State Interaction

Fig. 4 provides a microscopic view of state interaction by showing its effect at the object level. The results are from an illustrative example involving a group of $n = 4$ nodes, each of which has storage for up to $C = 4$ objects, in a universe of $N = 10$ objects (other parameters are shown in the caption and legends of the figure). Nodes $v_1, \ldots, v_3$ have the same fixed request rate $\lambda_1 = 1$, whereas the overactive node v_4 has request rate $\lambda_4 = 1, 10, 100$ (*i.e.*, we have three sets of results that correspond to different λ_4; each set is depicted on a different column of Fig. 4). The three graphs on the top depict the demand and the steady-state vector for node v_1 (which will be used as a representative for all three non-overactive nodes), while the ones on the bottom depict the corresponding quantities for node v_4. Each graph includes four curves. The bottom two curves indicate the local demand distribution $\vec{p}$ and the aggregate demand distribution $\vec{p}^*$, which includes the effect of the other nodes' miss streams (each of these curves sums up to 1). The top two curves $\vec{\pi}$ and $\vec{\pi}^*$ show the steady-state vectors of a node when the input is $\vec{p}$ (no miss-stream present) and $\vec{p}^*$ (miss-stream present), respectively, as obtained from the analytical method of Section 4.2 (each of these curves sums up to C).

Looking at the three graphs on the bottom of Fig. 4, we see that overactive node v_4 is not affected by the miss streams of other nodes. For $\lambda_4 = 10$ and 100, its aggregate demand and

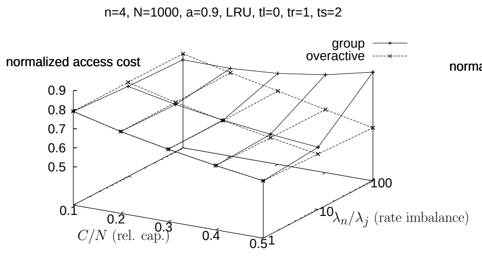

Figure 5: Analytical results on the effect of the state interaction on the normalized access cost of the overactive node and the remaining nodes of the group under different relative storage capacities and request rate imbalances. No mistreatment occurs in this case.

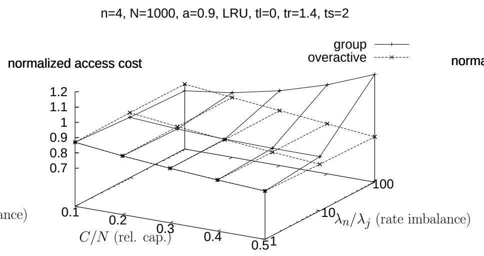

Figure 6: Analytical results on the state interaction effect on the normalized access cost of the overactive node and the remaining nodes of the group under different relative storage capacities and request rate imbalances. Mistreatment occurs due to larger t_r.

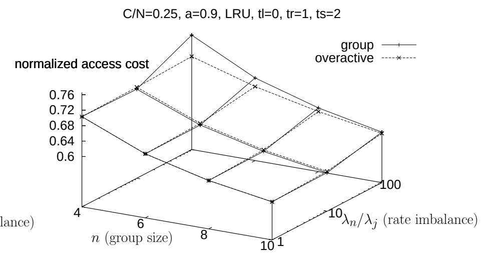

Figure 7: Analytical results on the effect of the state interaction on the normalized access cost of the overactive node and the remaining nodes of the group under different group sizes and request rate imbalances.

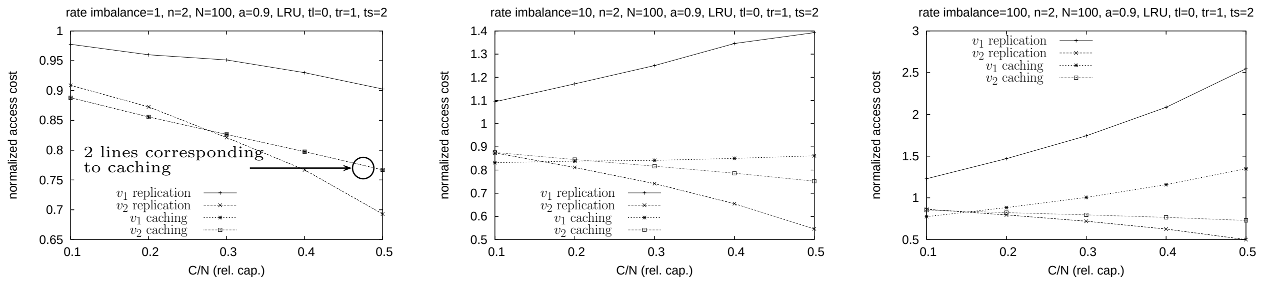

Figure 8: Analytical results on the comparison of replication and caching under three cases of request imbalance (1,10 and 100).

its steady state vector are identical to the corresponding ones without state interaction, *i.e.*, $\vec{p}_4^* \approx \vec{p}_4$ and $\vec{\pi}_4^* \approx \vec{\pi}_4$. For $\lambda_4 = 1$, there is a very slight effect due to the presence of the miss streams of the other three nodes, but this has almost no effect on the steady-state vector $\vec{\pi}_4^*$.

Looking at the top left graph in Fig. 4, which corresponds to $\lambda_4 = 1$, we see that the same slight effect exists for node v_1 due to the reception of the other three miss streams. The situation, however, changes radically when increasing λ_4 (second and third graphs of the top row). In that case, $\vec{p}_1^*$ and $\vec{p}_1$, and as a consequence $\vec{\pi}_1^*$ and $\vec{\pi}_1$ also become distinctively different. The intense miss stream from v_4 increases the popularity of some objects from the middle part of $\vec{p}_1$, thereby making them the most popular objects in $\vec{p}_1^*$. For example, when $\lambda_4 = 100$, objects 2,3 and 4, become more popular than object 1. This change in the profile of $\vec{p}_1^*$ is then reflected in $\vec{\pi}_1^*$, thereby affecting its access cost (Eq. (6)), as we explain below.

5.3 Effect on Performance

Fig. 5 provides a macroscopic view of state interaction by considering its effects on the normalized access cost of each node. The normalized cost of node v_j under the aggregate demand $\vec{p}_j^*$ is defined as follows:

$$\hat{cost}_j(\vec{p}_j^*, \vec{p}_j) = \frac{cost_j(\vec{p}_j^*)}{cost_j^{iso}(\vec{p}_j)}, \tag{8}$$

where $cost_j^{iso}(\vec{p}_j) = \sum_{i=1}^{N} p_{ij} \cdot [\pi_{ij} \cdot t_l + (1 - \pi_{ij}) \cdot t_s]$ is the cost that would be incurred by v_j if it operated in isolation (outside the group) and received only its local demand $\vec{p}_j$. If $\hat{cost}_j < 1$, the node benefits from its participation in the group, otherwise, it is being mistreated. When considering two nodes, v_j and $v_{j'}$, then the fact that $1 > \hat{cost}_j > \hat{cost}_{j'}$, means that although both are better off by participating in the group, v_j gets a relatively larger benefit.

There are two main points to be concluded from Fig. 5. First, *it requires a very strong imbalance of request rates in order to create a substantial difference in the incurred normalized access costs*. In the presented example, the overactive node v_4 has a 30% reduction of its normalized cost, only when it produces a 100-fold more intense request stream. Even such a strong imbalance, is not enough to mistreat the other nodes ($v_1, \ldots, v_3$ have normalized access costs < 1). For the occurrence of mistreatment, remote accesses have to be even more expensive (this is shown in Fig. 6, where t_r increases from 1 to 1.4, thereby making the normalized access cost of the group nodes > 1). Second, *the nodes must have large storage capacity to be affected by state interaction related phenomena*. In the presented example, the nodes must have at least 20% relative storage capacity C/N to be affected by the overactive node. Surprisingly, for small C/N, *e.g.*, less than 15%, the group nodes actually benefit more than the overactive node, *i.e.*, they achieve a smaller normalized access cost. In [19] we explained this peculiar phenomenon by arguing that the miss-stream from the overactive node actually helps the other nodes, in this case by creating more skewed demands for them, which lead to higher hit ratios. Figure 7) shows that increasing the size of the group, reduces the effects of the state interaction. This occurs as the miss stream of the overactive node(s) (here just one) weakens by being divided across more nodes.

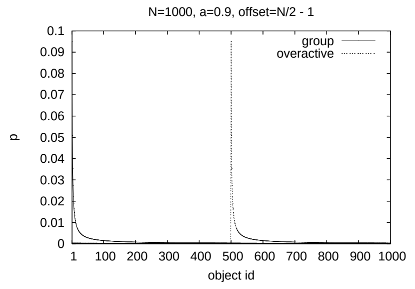

Figure 9: The object demand distribution for the overactive, when the popularity ranking has been shifted by an offset O, and the remaining nodes of the group.

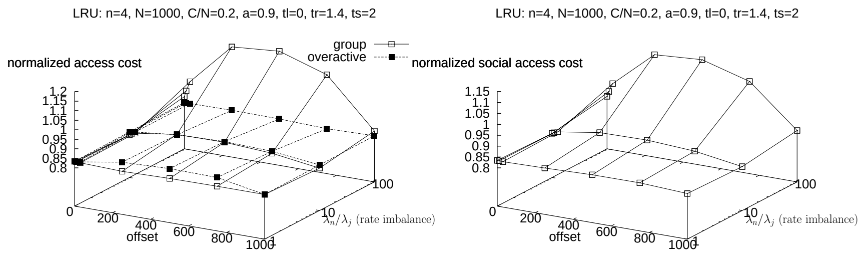

Figure 10: Analytical results of the state interaction on the normalized access cost of the outlier and the remaining nodes in the group, and the normalized social cost of the nodes in the caching group under non-homogeneous demand distributions, where overactive's node popularity ranking has been shifted rightwards by an offset O.

5.4 The Case of Non-homogeneous Demand Patterns

What we described so far is fairly optimistic as we assumed that all participants in the distributed caching group exhibit similar access patterns. If this assumption does not hold, then intensity of the mistreatment could be much higher, even for small C/N. To underscore this point, in this section, we will deviate from our course so far, and examine mistreatments and the social cost of the group under non-homogeneous demand distributions. For non-overactive nodes we will maintain the popularity ranking of objects as it was $(o_1, o_2, \dots)$. For the overactive node, however, we will shift the popularity ranking by an *offset* O, $0 \leq O \leq N$, therefore making object $o_{1+(O+i-1) \bmod N}$ be the ith most popular object. We assign request probabilities taken from the same generalized power-law profile with skewness $a = 0.9$ that is used for the non-overactive nodes. Figure 9 depicts the demand distribution for the overactive for $O = N/2 - 1$. The two graphs of Fig. 10 depict the normalized individual cost for the overactive and the non-overactive nodes as well as the social cost (normalization is obtained by dividing by the corresponding cost obtained when remote hits are not allowed to affect the local caching state). As it is obvious, mistreatments can occur even under non-homogeneous demand distributions. The concave profile with respect to O occurs as with high O the popularity ranking starts to look like the original one due to "wrapping" after N. We have obtained similar results with several other perturbations of the popularity ranking [34].

5.5 Caching versus Replication

In this section we will consider both replication and caching and compare their relative robustness to mistreatment. For replication we will consider the socially optimal (SO) replication algorithm of Leff et al. [22]. For simplicity of exposition, and also to be able to compare with our previous numerical results from [20], we will consider a group with only $n = 2$ nodes and a universe of $N = 100$ objects. The three graphs of Fig. 8 depict the normalized access costs[7] for nodes v_1 and v_2 (overactive), for three cases of request imbalance, 1, 10, and 100. When there is no request imbalance (first graph), no node is mistreated. Caching yields the exact same performance for both nodes (the two curves for v_1, v_2 coinciding), while replication might unintentionally favor one of them (there are several optimal solutions, and the particular one chosen has to do with the specific solution algorithm that is employed, here an LP relaxation of an integer problem solved via Simplex).

The different sensitivity to mistreatment becomes apparent as soon as request imbalance is introduced, *i.e.*, with $\lambda_2/\lambda_1 = 10$ and 100 (second and third graphs of Fig. 8). By observing these figures, we see that the curves for caching are always contained within the angle specified by the curves for replication (except for very small C/N, where we have the peculiar behavior of caching discussed in the previous section). The point to be kept from these results is that *replication is much more sensitive to mistreatment than caching.* Under replication, the slightest imbalance of request intensities is directly reflected in the outcome of the replication algorithm. In contrast, the state interaction that takes place in caching is a much weaker catalyst for mistreatment. This fortunate weakness owes to the stochastic nature of caching, and to the requirement for the concurrent occurrence of two independent events: An unpopular object must first be brought to the cache due to the local demand, and then the miss-stream must feed it with requests, if it is to lock it in the cache (and thereafter beat the local request stream that tries to push it out and reclaim the storage space).

5.6 LRU versus LFU

Fig. 11 shows analytical results under LRU replacement, as well as simulation results under perfect LFU replacement [29] (two group sizes, $n = 2$ and $n = 4$, are considered). We plot the absolute, instead of the normalized access costs, as we are considering different replacement algorithms. Looking first at LRU, we notice the following. The effects of state interaction (reflected in the width of the angle between group and overactive curves, after $\lambda_n/\lambda_j = 10$)

[7]For the case of replication, the normalization is conducted by dividing with the performance of the greedy local (GL) replication strategy. See [20] for details.

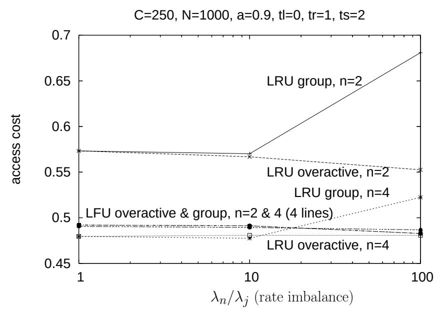

Figure 11: Analytical results on the comparison LRU vs LFU.

decrease as the group grows larger, as also noted in the previous section. Moreover, the absolute access costs for both the group and the overactive node also decrease with the size of the group. The reason is that a bigger group has more aggregate storage capacity and thus succeeds in caching more distinct objects, which in turn benefits all the nodes.

Turning our attention to the LFU curves, we see a completely different behavior. For a given n, both the overactive node and the rest of the group, have the same access cost, *i.e.*, *the request imbalance has no affect on the nodes under LFU.* This happens because once in steady-state, perfect LFU avoids replacement errors, thus does not give any opportunities for locking unpopular objects and losing storage due to the miss-stream of remote overactive nodes. Thus LFU has an advantage over LRU in terms of its immunity to request imbalance. What is even more interesting, however, is that the access cost under LFU remains the same under different n, *i.e.*, increasing the group size does not help in reducing the access costs. This happens because under LFU and common demand patterns, all the nodes end up caching exactly the same sets of objects. In such a group, a local miss is bound to miss also in the group. In other words, *LFU eliminates all the cooperation gain in groups of similar nodes.* This does not occur when the group operates under LRU: the replacement errors committed by the individual nodes in this case, create a healthy amount of noise that increases the distinct objects held in the group, thereby decreasing the access cost of all the nodes. Thus, in large groups under small inter-node distances, LRU is more appropriate than LFU (see for example the access cost for small t_r in Fig. 12 in Section 6.1, where $n = 10$). When the inter-node distances increase, then the perfect ranking of objects under LFU becomes more important than the cooperation gain and, thus, LFU becomes better for the group (see Fig. 12 for large t_r).

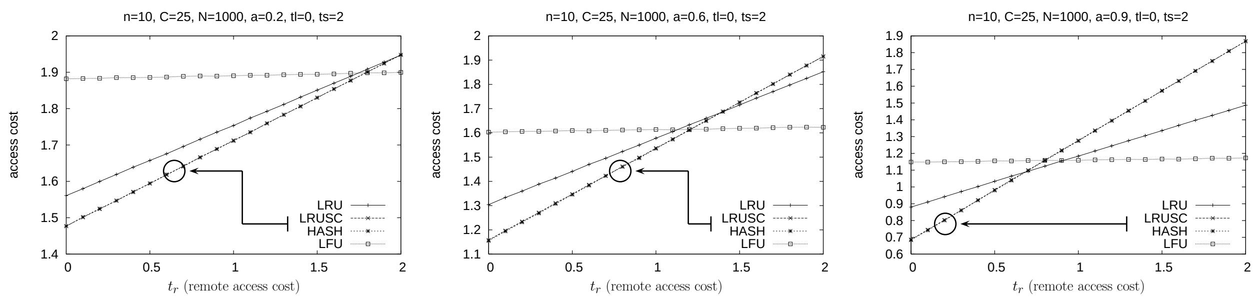

Figure 12: Simulation results on the effect of the remote access cost t_r on the performance ranking of different SC and MC schemes for three cases of skewness of demand ($a = 0.2$, $a = 0.6$, $a = 0.9$). MC schemes (LRU, LFU) perform better when $t_r \rightarrow t_s$.

5.7 L2 versus Non-L2 Caching

When a cache operates in Level-2 (L2) mode, it fetches and maintains a copy from the origin server for every request that it receives from a remote node (whether it hits or misses locally). In [19] we showed that *L2 caching eliminates the robustness to mistreatment of non-L2 caching, leading to a vulnerability level similar to the one under replication.* To understand this, one has to observe that in L2-caching and replication, locally irrelevant objects may occupy the local cache without the intervention of the local demand, whereas in non-L2 caching, the local demand has first to bring the objects in the cache, and thus give other nodes the opportunity to maintain them there by feeding them with requests.

6 Mistreatment Due to Use of a Common Scheme

In this section we study cases of mistreatment due to the use of a common scheme vis-a-vis the object admission control algorithm. Specifically, we consider Single Copy (SC) schemes, like, HASH and LRU-SC [8], *i.e.*, schemes that allow for the existence of up to one copy of each object in the group and, Multiple Copy (MC) schemes, *i.e.*, schemes that allow for the existence of multiple copies of the same object at different nodes of the group. All the replacement algorithms when combined with a non-SC object admission control fall into the MC category.

[8]Under HASH, requests are received by the local node, which employs a hash function to identify the node that is responsible for the requested object. The responsible node returns the object immediately if it already caches it, or contacts the origin server, and then returns it, also keeping a local copy in this case. The local node does not keep a local copy, unless it is the one responsible for that object according to the employed hash function. Under LRU-SC (single copy), a local copy is maintained at the local node only for objects that were fetched from the origin server. When an object is fetched from elsewhere in the group, no local copy is kept. In both cases, the number of copies of each object in the group is limited to at most one.

6.1 Single Versus Multiple Copy Schemes

Fig. 12 depicts simulation results showing the average access cost of a group (social cost) under different SC and MC schemes, and for different values of t_r representing different levels of "tightness" of the group. Three types of demand are considered: lightly (a=0.2), moderately (a=0.6), and highly skewed (a=0.9) demand. The following observations apply. *Single copy schemes, (*i.e.*, HASH and LRU-SC, whose curves overlap almost completely in these figures, as the two have very similar caching behavior) perform better when the access cost between the nodes is small.* In such cases the cost of local and remote accesses is similar, so it pays to eliminate multiple copies of the same object at different nodes and instead make room for storing a larger number of distinct objects. *Multiple copy schemes, (*i.e.*, LRU and LFU) perform better when the access cost between the nodes is high.* In such cases, a much higher cost is incurred when an object is fetched from the group, so it becomes imperative to maintain some of the most popular objects locally (thereby creating multiple copies at different nodes). The threshold value of t_r at which the performance ranking between SC and MC changes depends on the skewness of the demand: the higher the skewness, the lower the value of t_r and the earlier the MC schemes become better.

It is also worthwhile noting that the curves for LFU are parallel to the x-axis, *i.e.*, the access cost is immune to the inter-node distance under LFU and identical demand. This happens because, as noted earlier, under LFU all the nodes store the same objects, and this has the consequence of eliminating all remote hits. In that case, the exact value of the remote access cost does not affect the LFU curves, since there are no remote hits. Regarding the comparison between LRU and LFU, the figure shows that LFU is better when the remote access cost is high (see the discussion in Section 5.6 for an explanation of this).

The above observations highlight the fact that "fixed schemes" operate efficiently only under specific parameter sets. If these parameter sets are common to all the nodes, then good design choices can be made among the different schemes. However, when some of the parameters (*e.g.*, inter-node distances) are not common to all nodes, then *it may well be the case that no single scheme is appropriate for all the nodes. Enforcing a common scheme under such conditions is bound to mistreat some of the nodes.* The following section illustrates such an example.

6.2 Relaxing the Common Scheme Requirement

So far, we have assumed that all group nodes employ the same (common) caching scheme. In this section, we look at the advantages to be gotten from relaxing this constraint.

Consider the group depicted in Fig. 13 in which $n-1$ nodes are clustered together, meaning that they are very close to each other ($t_r \to t_l$), while there's also a single "outlier" node at distance t'_r from the cluster. The $n-1$ nodes would naturally employ the LRU-SC scheme in order to capitalize on their small remote access cost. From the previous discussion it should be clear that the best scheme for the outlier node would depend on t'_r. If $t'_r \to t_l$, the outlier should obviously follow LRU-SC and avoid duplicating objects that already exist elsewhere in the group. If $t'_r \gg t_l$, then the outlier should follow a MC scheme, e.g., LRU.

To permit the outlier to adjust its caching behavior according to its distance from the group, we introduce the LRU(q) scheme, under which, objects that are fetched from the origin server are automatically cached locally, but objects that are fetched from the group are cached locally only with probability q. For $q = 0$, LRU(q) reduces to LRU-SC, while for $q = 1$ it reduces to the multiple copy LRU scheme. One may think of q as a *reliance parameter*, capturing the confidence that a node has in its ability to fetch objects efficiently (*i.e.*, "cheaply") from other members of the group.

Figure 14 presents the performance of LRU(q), for $q = 0, 0.1, 0.5, 1$ under different t'_r. The results are normalized by dividing the access cost of each LRU(q) scheme by the corresponding access cost of the LRU($q = 1$) scheme. The later can be seen as a basis for what a node can achieve by operating greedily, *i.e.*, when it always keeps a copy of each incoming object. Such a behavior corresponds to a node that wants to avoid relying on other nodes for fetching objects. As with the state interaction case, mistreatment is signified by a normalized access cost greater than 1.

Figure 14 shows that for the considered scenario, always keeping local copies of all incoming objects (*i.e.*, employing LRU(1) and incurring a normalized access cost of 1) is a reasonably good choice across most values of t'_r. The only case that LRU(1) performs poorly is when t'_r becomes very small, which corresponds to the case in which the node ceases to be an outlier, and actually becomes part of the cluster. As discussed earlier, in this case maintaining multiple object copies within the group becomes wasteful, with the optimal scheme being the single copy LRU(0) scheme.

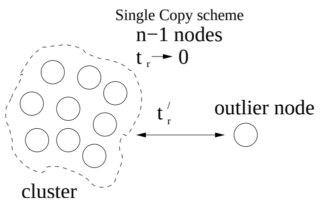

Figure 13: An example of a group composed of a cluster of $n-1$ nodes and a unique outlier.

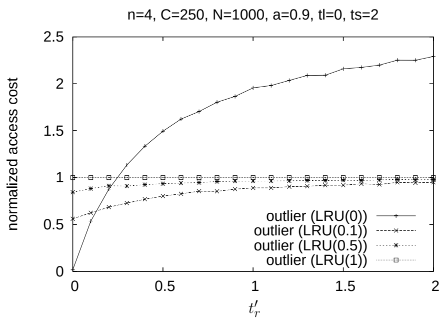

Figure 14: Simulation results on the effect of the remote access cost t'_r on the normalized access cost of the outlier node under different LRU(q) schemes.

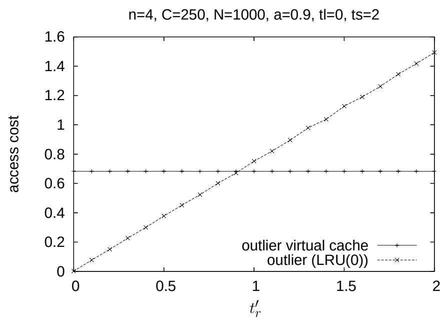

Figure 15: Simulation results on the effect of the remote access cost t'_r on the access cost of the outlier node under the virtual cache and LRU(0) schemes.

Another interesting observation from the above results is that there is a noticeable performance differential between the single copy LRU(0) scheme, and any other multiple copy LRU(q) scheme with $q > 0$. A non-zero LRU(q) scheme, even one where q is small, is capable of eventually caching locally the most popular objects, even if this requires several misses. LRU(0), on the other hand, has almost no chance of bringing a globally popular object locally since it is much more likely for such an object to be cached in the cluster before being requested by the outlier node (which means that it won't be cached locally). When this happens for several popular objects, the performance degradation for the outlier node becomes very serious. That is why LRU(0) performs poorly for large values of t'_r.

7 Towards Mistreatment-Resilient Caching

From the exposition so far, it should be clear that there exist situations under which an inappropriate, or enforced, scheme may mistreat some of the nodes. While we have focused on detecting and analyzing two causes of mistreatment which appear to be important (namely, due to cache state interactions and the adoption of a common cache management scheme), it should be evident that mistreatments may well arise through other causes. For example, we have not investigated the possibility of mistreatment due to request re-routing [28], not to mention that there are vastly more parameter sets and combinations of schemes that cannot all be investigated exhaustively.

To address the above challenges, we first sketch a general framework for designing mistreatment-resilient schemes. We then apply this general framework to the two types of mistreatments that we have considered in this work. We target "open systems" in which group settings (*e.g.*, number of nodes, distances, demand patterns) change dynamically. In such systems it is not possible to address the mistreatment issue with predefined, fixed designs (*e.g.*, using the results of the previous section for selecting a fixed value for the reliance parameter q). Instead, we believe that *nodes should adjust their scheme dynamically so as to avoid or respond to mistreatment if and when it emerges*. To achieve this goal we argue that the following three requirements are necessary.

Detection Mechanism: This requirement is obvious but not trivially achievable when operating in a dynamic environment. *How can a node realize that it is being mistreated?* In our previous work on replication [20], a node compared its access cost under a given replication scheme with the guaranteed maximal access cost obtained through GL replication. This gave the node a "reference point" for a mistreatment test. In that-game theoretic framework, we considered nodes that had *a priori* knowledge of their demand patterns, thus could easily compute their GL cost thresholds. In caching, however, demand patterns (even local ones) are not known *a priori*, nor are they stationary. Thus in our DSC setting, the nodes have to estimate and update their thresholds in an on-line manner. We believe that a promising approach for this is *emulation*. Figure 16 depicts a node equipped with an additional *virtual cache*, alongside its "real" cache that holds its objects. The virtual cache does not hold actual objects, but rather object identifiers. It is used for emulating the cache contents and the access cost under a scheme *different from* the one being currently employed by the node to manage its "real" cache under the same request sequence (notice that the input request stream is copied to both caches). The basic idea is that *the virtual cache can be used for emulating the threshold cost that the node can guarantee for itself by employing a greedy scheme*.

Mitigation Mechanism: This requirement ensures that a node has a mechanism that allows it to react to mistreatment—a mechanism via which it is able to respond to the onset of mistreatment. In the context of the common scheme problem, the ability to adjust the *reliance parameter* q acted as such a mechanism. In the context of the state interaction problem, one may define an *interaction parameter* p_s and the corresponding LRU(p_s) scheme, in which a remote hit is allowed to affect the local state with probability p_s, whereas it is denied such access with probability (1-p_s). As it will be demonstrated later on, nodes may avoid mistreatment by selecting appropriate values for these parameters according to the current operating conditions.

Control Scheme: In addition to the availability of a mistreatment mitigation mechanism (*e.g.,* LRU(q)), there needs to be a programmatic scheme for adapting the control variable(s) of that mechanism (*e.g.,* how to set the value of q). Since the optimal setting of these control variables depends heavily on a multitude of other time-varying parameters of the DSC system (*e.g.*, group size, storage capacities, demand patterns, distances), it is clear that there cannot be a simple (static) rule-of-thumb for optimally setting the control variables of the mitigation mechanism. To that end, dynamic feedback-based control becomes an attractive option.

To make the previous discussion more concrete, we now focus on the common scheme problem and demonstrate a mistreatment-resilient solution based on the previous three principle requirements. A similar solution can be developed for the state interaction problem.

Resilience to Common-Scheme-Induced Mistreatments: We start with a simple "hard-switch" solution that allows a node to change operating parameters by selecting between two alternative schemes. This can be achieved by using the virtual cache for emulating the LRU(1) scheme, *i.e.*, the scheme in which the reliance parameter q is equal to 1 (capturing the case that the outlier node does not put any trust on the remote nodes for fetching objects and, thus, keeps copies of all incoming objects after local misses). Equipped with such a device, the outlier can calculate a running estimate of its threshold cost based on the objects it emulates as present in the virtual cache.[9] By comparing the access cost from sticking to the current scheme to the access cost obtained through the emulated scheme, the outlier can decide which one of the two schemes is more appropriate. For example, it may transit between the two extreme LRU(q) schemes–the LRU($q = 0$) scheme and the LRU($q = 1$) scheme. Figure 15 shows that the relative performance ranking of the two schemes depends on the distance from the group t'_r and that there is a value of t'_r for which the ranking changes.

A more efficient design can be obtained by manipulating the reliance parameter q at a finer scale. Indeed, there are situations in which intermediate values of q, $0 < q < 1$, are better than either $q = 0$ and $q = 1$ (see the LRU(0.1) and LRU(0.5) curves in Fig. 14). Consider two different values of the reliance parameter q_1, q_2 such that $q_1 < q_2$. Figure 17 illustrates a typical development of the average object access cost under q_1 and q_2 as a function of the distance

[9]The outlier can include in the emulation the remote fetches that would result from misses in the emulated cache contents; this would give it the exact access cost under the emulated scheme. A simpler approach would be to disregard the remote fetches and thus reduce the inter-node query traffic; this would give it an upper bound on the access cost under the emulated scheme.

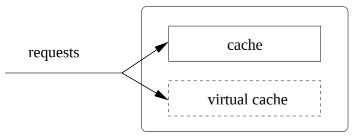

Figure 16: Block diagram of a node equipped with a virtual cache.

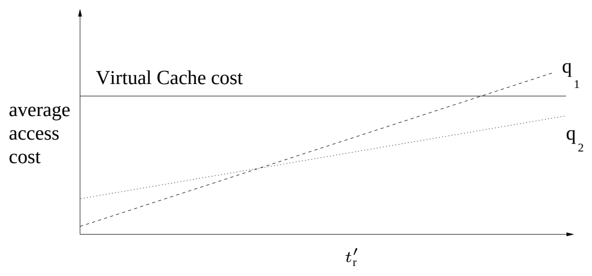

Figure 17: Representative development of average object access cost as a function of the reliance parameter and distance of outlier from the cluster

t'_r of the outlier node from its cooperative cluster. As discussed in the previous section, q_1 (q_2) will perform better with small (large) t'_r. In the remainder of this section, we present and evaluate a Proportional-Integral-Differential (PID) controller for controlling the value of q. This type of controller is known for its good convergence and stability properties [27, 11].

A node equipped with the PID controller maintains an Exponential Weighted Moving Average (EWMA) of the object access cost ($cost_{virtual}$) for the emulated greedy scheme. The virtual cache emulates an LRU($q = 1$)-scheme in which no remote fetches are considered, so as to avoid doubling the number of queries sent to remote nodes. Let $cost_q$ denote the EWMA of the object access cost of the employed LRU(q)-scheme in the actual cache of the node. Let $dist$ denote the difference between the virtual access cost and the actual access cost, and let $diff$ be the difference between two consecutive values of $dist$ at different points in time.

The PID controller adapts q proportionally to the magnitude of $diff$; a pseudo-code for this process is provided in Algorithm 1. In Appendix 1, we argue that the access cost of a node equipped with this controller converges to a value which is optimal (as compared with the cost of any scheme that employs a fixed q) and we also provide an estimation of the converged value. Our algorithm has two parameters that need to be tuned. The first one, denoted by α_c, is the *gain* of the controller, which determines the rate (speed) with which the value of q is changed in a single control period. The second parameter, denoted by β_c, is the *update weight* of the difference in the cost that is observed in the last (two) updates. The above mechanism does not introduce additional communication overheads since it runs solely on local information. The memory and computational overheads are small too. Since the virtual cache stores only object identifiers and not actual objects, the amount of memory required for implementing it is negligible compared to the amount of memory used for storing the actual objects. Furthermore, since both LRU and LFU can be implemented

Algorithm 1 : mitigation of mistreatment

```
$dist(t) = cost_{virtual}(t) - cost_q(t)$
$dist(t-1) = cost_{virtual}(t-1) - cost_q(t-1)$
$diff(t) = dist(t) - dist(t-1)$
$\sigma = sign(diff(t))$
**if** $q(t-1) \geq q(t-2)$ **then**
    $q(t) \leftarrow q(t-1) + \sigma \cdot \alpha_c \cdot |diff(t)| + \sigma \cdot \beta_c \cdot |\,|diff(t)| - |diff(t-1)|\,|$
**else**
    $q(t) \leftarrow q(t-1) - \sigma \cdot \alpha_c \cdot |diff(t)| - \sigma \cdot \beta_c \cdot |\,|diff(t)| - |diff(t-1)|\,|$
```

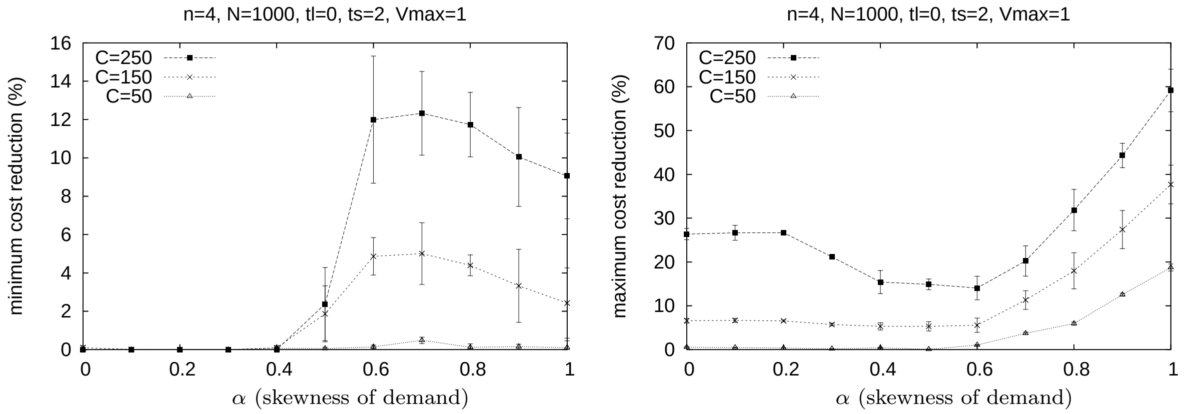

Figure 18: Simulation results on the cost reduction that is achieved using our adaptive mechanism, (left): The minimum cost reduction, (right): The maximum cost reduction.

efficiently ($O(1)$ time required for updating the caching state after each request), the added computational burden from updating the virtual cache alongside the actual one is small.

Performance Evaluation: In order to evaluate our adaptive scheme, we compare its cumulative average access cost to the corresponding cost of one of the two extreme static schemes (LRU($q = 0$),LRU($q = 1$)). Thus, we define the following performance metric:

$$minimum\ cost\ reduction\ (\%) \ = \ 100 \cdot \frac{cost_{static} \ - \ cost_{adaptive}}{cost_{static}} \tag{9}$$

where $cost_{adaptive}$ is the access cost of our adaptive mechanism, and $cost_{static}$ is the minimum cost among the two static schemes: $cost_{static} = min\,(cost(LRU(q = 0), LRU(q = 1))$. This metric captures the minimum additional benefit that our adaptive scheme has over the previous static schemes. To capture the maximum additional benefit of our adaptive scheme (the optimistic case), we similarly define *maximum cost reduction* as in Eq. (9), where $cost_{static} = max\,(cost(LRU(q = 0), LRU(q = 1))$.

We evaluate the performance of our PID-style feedback controller experimentally by considering a scenario in which the distance between the outlier node and the cooperative group

(t'_r) changes according to the Modified Random Waypoint Model [23]. The motivation for such a scenario comes from a wireless caching application [37]. A detailed description of the design of this experiment is provided in Appendix 2. Figure 18 summarizes results we obtained under different cache sizes, demand skewness, and movement speed $V_{max} = 1$ distance units/time unit (similar results are observed under higher speeds as well). All experiments were repeated 10 times and we include 95^{th}-percentile confidence intervals in the graphs.

By employing our adaptive scheme, the outlier achieves a maximum cost reduction that can be up to 60% under skewed demand. The depicted profile of the maximum cost reduction curve can be explained as follows. The worst performance of the static schemes appears at the two extremes of skewness. Under uniform demand, $a = 0$, we get the worst performance of the LRU(1) static scheme, whereas under highly skewed demand, $a = 1$, we get the worst performance of the LRU(0) static scheme. In the intermediate region both static schemes provide for some level of compromise, and thus the ratio of the cost achieved by either schemes to the corresponding cost of the adaptive scheme becomes smaller than in the two extremes.

Turning our attention to the minimum cost reduction, we observe that it can be substantial under skewed demand, and disappears only under uniform demand (such demand, however, is not typically observed in measured workloads [2]). The explanation of this behavior is as follows. At the two extreme cases of skewness, one of the static scheme reaches its optimal performance—under low skewed demand, the best static scheme is the LRU(0) and under high skewed demand the best static scheme is the LRU(1). Thus, the ratio of the cost achieved by the best static scheme and the corresponding cost of our adaptive scheme gets maximized in the intermediate region, in which neither of the static schemes can reach its best performance.

Resilience to State-Interaction-Induced Mistreatments: Immunizing a node against mistreatments that emerge from state interactions could be similarly achieved. The interaction parameter p_s can be controlled using schemes similar to those we considered above for the reliance parameter q. It is important to note that one may argue for *isolationism* (by permanently setting $p_s = 0$) as a simple approach to avoid state-interaction-induced mistreatments. This is not a viable solution. Specifically, by adopting an LRU($p_s = 0$) approach, a node is depriving itself from the opportunity of using miss streams from other nodes to improve the accuracy of LRU-based cache/no-cache decisions (assuming a uniform popularity profile for group members). This was highlighted in the results shown in Fig. 5.

To conclude this section, we note that the approaches we presented above for mistreatment resilience may be viewed as "passive" or "end-to-end" in the sense that a node infers the onset of mistreatment *implicitly* by monitoring its utility function. As we alluded at the outset of this paper, for the emerging class of network applications for which grouping of nodes is "ad hoc" (*i.e.*, not dictated by organizational boundaries or strategic goals), this might be the only realistic solution. In particular, to understand "exactly how and exactly why" mistreatment is taking place would require the use of proactive measures (*e.g.*, monitoring/policing group member behaviors, measuring distances with pings, *etc.*), which would require group members to subscribe to some common services or to trust some common authority—both of which are not consistent with the autonomous nature (and the mutual distrust) of participating nodes.

8 Summary and Concluding Remarks

Distributed on-demand caching enables loosely coupled groups of nodes to share their (storage) resources to achieve higher efficiencies and scalability. In addition to its traditional use in content distribution/delivery networks, distributed caching is also used as an important building block of many emerging applications and protocols, including its use in route caching in ad-hoc networks [26] and in P2P content replication [6, 15].

Summary: This paper has uncovered the susceptibility of nodes participating in a distributed on-demand caching group to being *mistreated*. We have identified two causes of mistreatments– namely mistreatment due to cache *state interactions* between various members of the group, and due to the use of a *common scheme* for cache management across all members of the group. We have backed up our findings by analytic models, numerical solutions of these models, as well as simulations in which assumptions (necessary for analysis) have been relaxed.

The results of our analysis and evaluation suggest that on-demand distributed caching is fairly resilient to the onset of mistreatment as long as proxying (a.k.a. L2 caching) is not enabled, and as long as intra-group access costs do not include outliers. More constructively, we have outlined an efficient emulation-based approach that allows individual nodes to decide autonomously (i.e., without having to trust any other node or service) whether they should stick to, or secede from a caching group, based on whether or not their participation is beneficial to their performance compared to a selfish, greedy scheme.

Other Incarnations of Mistreatment in On-Line Distributed Resource Management Problems: In this paper, we focused on distributed caching as an instance of an on-line protocol for the management of a distributed resource—namely the limited storage available at each node. While our exposition has focused on the well-known problem of caching "retrievable" content (*e.g.*, web pages and media objects), it should be evident that our results extend to *any* other type of cached content, including non-retrievable content used as part of the control plane of a distributed protocol or application (*e.g.*, route paths stored in routing tables of group members). Clearly, given the different nature of the workloads that such distributed resources must support, a more specific examination of potential mistreatments in such settings is warranted, and is a current subject of inquiry of ours.

Coincidental versus Adversarial Mistreatment: In this paper we focused on the onset of mistreatment due to benign operating conditions of a caching group. For instance we identified rate imbalance (of local versus remote requests streams) conditions as well as cache sizing conditions that are necessary for mistreatment to occur. As such, the cases of mistreatment we have uncovered could be considered "coincidental". Another possible source of mistreatment, however, could be adversarially motivated, in the sense that one (or more) of the group members collude to negatively impact the performance of other members. While we did not consider adversarial mistreatments *per se*, our results suggest that distributed caching is fairly immune to *high potency exploits* [12] (a.k.a. low rate attacks) by non-clairvoyant adversaries. More work is needed to characterize the vulnerability of distributed caching to more elaborate adversarial exploits, including those from more powerful agents (*e.g.*, those with knowledge of a victim's cache contents).

References

[1] Martin F. Arlitt and Carey L. Williamson. Web server workload characterization: the search for invariants. In *Proceedings of the 1996 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 126–137, 1996.

[2] Lee Breslau, Pei Cao, Li Fan, Graham Philips, and Scott Shenker. Web caching and Zipf-like distributions: Evidence and implications. In *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, New York, March 1999.

[3] John W. Byers, Jeffrey Considine, Michael Mitzenmacher, and Stanislav Rost. Informed content delivery across adaptive overlay networks. *IEEE/ACM Transactions on Networking*, 12(5):767–780, October 2004.

[4] Byung-Gon Chun, Kamalika Chaudhuri, Hoeteck Wee, Marco Barreno, Christos H. Papadimitriou, and John Kubiatowicz. Selfish caching in distributed systems: A game-theoretic analysis. In *Proc. ACM Symposium on Principles of Distributed Computing (ACM PODC)*, Newfoundland, Canada, July 2004.

[5] E. G. Coffman and P. J. Denning. *Operating systems theory.* Prentice-Hall, 1973.

[6] E. Cohen and S. Shenker. Replication strategies in unstructured peer-to-peer networks. In *Proceedings of ACM SIGCOMM'02 Conference*, Pittsburgh, PA, USA, August 2002.

[7] Asit Dan and Dan Towsley. An approximate analysis of the LRU and FIFO buffer replacement schemes. In *Proceedings of ACM SIGMETRICS*, pages 143–152, Boulder, Colorado, United States, 1990.

[8] Duane Wessels and k claffy. ICP and the Squid Web Cache. http://www.ircache.net/∼wessels/Papers/icp-squid.ps.gz.

[9] Özgür Erçetin and Leandros Tassiulas. Market-based resource allocation for content delivery in the internet. *IEEE Transactions on Computers*, 52(12):1573–1585, December 2003.

[10] Li Fan, Pei Cao, Jussara Almeida, and Andrei Z. Broder. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM Transactions on Networking*, 8(3):281–293, 2000.

[11] Gene F. Franklin, David J. Powell, and Abbas Emami-Naeini. *Feedback Control of Dynamic Systems (5th ed.)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[12] Mina Guirguis, Azer Bestavros, and Ibrahim Matta. Exploiting the transients of adaptation for RoQ attacks on Internet resources. In *Proceedings of the 12th IEEE International Conference on Network Protocols (ICNP'04)*, Berlin, Germany, October 2004.

[13] Wendi R. Heinzelman, Anantha P. Chandrakasan, and Hari Balakrishnan. An application-specific protocol architecture for wireless microsensor networks. *IEEE Transactions on Wireless Communications*, 1(4):660–670, October 2002.

[14] Shudong Jin and Azer Bestavros. Sources and Characteristics of Web Temporal Locality. In *Proceedings of Mascots'2000: The IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, San Fransisco, CA, August 2000.

[15] Jussi Kangasharju, Keith W. Ross, and David A. Turner. Optimal Content Replication in P2P Communities. Manuscript, 2002.

[16] Madhukar R. Korupolu and Michael Dahlin. Coordinated placement and replacement for large-scale distributed caches. *IEEE Transactions on Knowledge and Data Engineering*, 14(6):1317–1329, 2002.

[17] Randall Landry and Ioannis Stavrakakis. Queueing study of a 3-priority policy with distinct service strategies. *IEEE/ACM Transactions on Networking*, 1(5):576–589, 1993.

[18] Nikolaos Laoutaris, Hao Che, and Ioannis Stavrakakis. The LCD interconnection of LRU caches and its analysis. *Performance Evaluation*, 63(7):609–634, 2006.

[19] Nikolaos Laoutaris, Georgios Smaragdakis, Azer Bestavros, and Ioannis Stavrakakis. Mistreatment in Distributed Caching Groups: Causes and Implications. In *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, Barcelona, Spain, April 2006.

[20] Nikolaos Laoutaris, Orestis Telelis, Vassilios Zissimopoulos, and Ioannis Stavrakakis. Distributed selfish replication. *IEEE Transactions on Parallel and Distributed Systems*, 2006. [accepted for publication, preliminary version available from the authors' home-pages].

[21] Nikolaos Laoutaris, Vassilios Zissimopoulos, and Ioannis Stavrakakis. On the optimization of storage capacity allocation for content distribution. *Computer Networks*, 47(3):409–428, February 2005.

[22] Avraham Leff, Joel L. Wolf, and Philip S. Yu. Replication algorithms in a remote caching architecture. *IEEE Transactions on Parallel and Distributed Systems*, 4(11):1185–1204, November 1993.

[23] Guolong Lin, Guevara Noubir, and Rajmohan Rajaraman. Mobility models for ad hoc network simulation. In *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, Hong Kong, China, March 2004.

[24] Thanasis Loukopoulos, Petros Lampsas, and Ishfaq Ahmad. Continuous replica placement schemes in distributed systems. In *Proceedings of the 19th ACM International Conference on Supercomputing (ACM ICS)*, Boston, MA, June 2005.

[25] Anirban Mahanti, Carey Williamson, and Derek Eager. Traffic analysis of a web proxy caching hierarchy. *IEEE Network*, 14(3):16–23, May 2000.

[26] M. Marina and S. Das. Performance of Route Caching Strategies in Dynamic Source Routing. In *Proceedings of the Int'l Workshop on Wireless Networks and Mobile Computing (WNMC) in conjunction with Int'l Conf. on Distributed Computing Systems (ICDCS)*, Washington, DC, USA, 2001.

[27] Katsuhiko Ogata. *Modern control engineering (4th ed.)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2002.

[28] Jianping Pan, Y. Thomas Hou, and Bo Li. An overview DNS-based server selection in content distribution networks. *Computer Networks*, 43(6), December 2003.

[29] Stefan Podlipnig and Laszlo Böszörmenyi. A survey of web cache replacement strategies. *ACM Computing Surveys*, 35(4):374–398, 2003.

[30] Konstantinos Psounis, An Zhu, Balaji Prabhakar, and Rajeev Motwani. Modeling correlations in web traces and implications for designing replacement policies. *Computer Networks*, 45, July 2004.

[31] Pablo Rodriguez, Christian Spanner, and Ernst W. Biersack. Analysis of web caching architectures: Hierarchical and distributed caching. *IEEE/ACM Transactions on Networking*, 9(4), August 2001.

[32] Keith W. Ross. Hash-routing for collections of shared web caches. *IEEE Network*, 11(6), November 1997.

[33] Swaminathan Sivasubramanian, Guillaume Pierre, and Maarten van Steen. A case for dynamic selection of replication and caching strategies. In *Proceedings of the 8th International Workshop on Web Caching and Content Distribution (WCW)*, New York, September 2003.

[34] Georgios Smaragdakis. Notes on the Effect of Different Access Patterns on the Intensity of Mistreatment in Distributed Caching Groups. Technical Report BUCS-TR-2006-023, CS Department, Boston University, September 10 2006. http://www.cs.bu.edu/techreports/2006-023-effect-different-patterns.ps.Z.

[35] Xueyan Tang and Samuel T. Chanson. Adaptive hash routing for a cluster of client-side web proxies. *Journal of Parallel and Distributed Computing*, 64(10):1168–1184, October 2004.

[36] Alec Wolman, M. Voelker, Nitin Sharma, Neal Cardwell, Anna Karlin, and Henry M. Levy. On the scale and performance of cooperative Web proxy caching. *SIGOPS Oper. Syst. Rev.*, 33(5):16–31, 1999.

[37] Liangzhong Yin and Guohong Cao. Supporting cooperative caching in ad hoc networks. In *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, 2004.

APPENDIX 1: Convergence of the Controller

In this appendix we argue that the access cost of a node equipped with our adaptive mechanism converges to a value that is lower than the one under any other static scheme, and we analytically estimate this value. We consider the scenario with the outlier node that was presented in *Section* 6.2.

Claim: When Algorithm 1 is used for controlling the probability q of caching an incoming object at the outlier node, then its average access cost will converge to a single value which is upper-bounded by the minimum average cost of any static scheme (*i.e.* a scheme employing a fixed q).

Justification: As illustrated in Figure 17, the average access cost of the outlier increases linearly with its distance from the group (t'_r). When $t'_r \rightarrow t_l$, the optimal value of $q \rightarrow 0$. When $t'_r \rightarrow t_s$, the optimal value for $q \rightarrow 1$. We would like our controller to exhibit this behavior while tuning the value of q. We can approximate the "desired" average access cost of the outlier as a linear function of q. In Figure 19 we illustrate a representative behavior of the average access cost of two static schemes, LRU($q = 0$) and LRU($q = 1$). We also illustrate the "desired" behavior of the controller as a linear function with slope θ. It is clear that the average access cost of the controller will be a lower bound on the average access cost of every static scheme, as $\theta < \psi$ and although $\chi < \theta$ the average access cost of $LRU(q = 1)$ will converge to the access cost of the controller only when $t'_r = t_s$, because the initial access cost for LRU($q = 1$) is higher than that of LRU($q = 0$). In our analysis we assume that the capacity C and the skewness of the demand α are constants; their exact values affect only the slopes.

We define two operating regions for the controller: Region A and Region B, as denoted in Figure 19. In Region A, the cost of the outlier under the LRU($q = 1$)-scheme,is always higher than the one under the LRU($q = 0$)-scheme and vice versa for Region B.

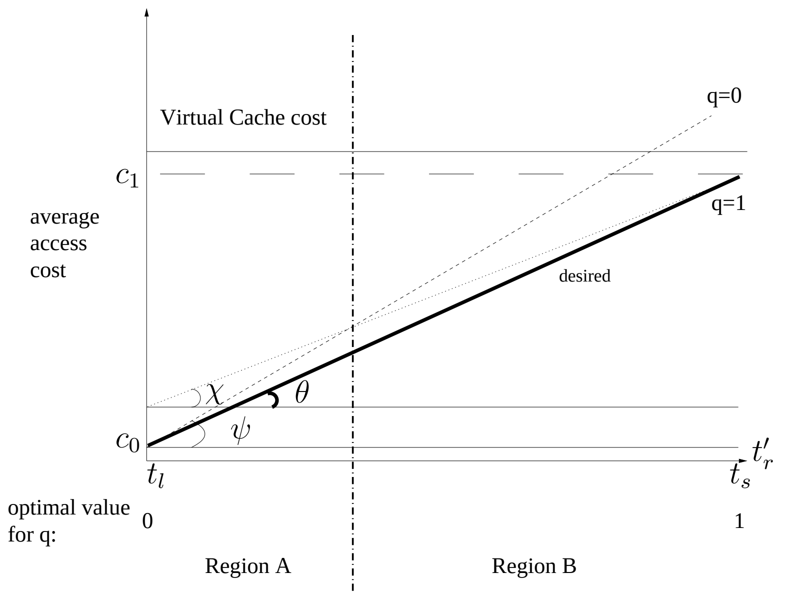

Figure 19: Representative behavior of static schemes ($LRU(q = 0)$, $LRU(q = 1)$) and the "desired" behavior of the controller.

We now proceed to analyze the behavior of our adaptive scheme in these two regions. We can design the controller such that the control update rate is higher that the rate at which t'_r changes (see Appendix 2). Thus, for the purpose of our analysis, let us assume that t'_r is fixed for a short period that includes few control updates.

In Region A, we consider two cases:

case A1: If $q(t) \geq q(t-1)$ then $cost(t) \geq cost(t-1)$, $dist(t) \leq dist(t-1)$ and as a result $diff(t) \leq 0$, thus our controller switches course and decreases the value of q at the adaptation point $t+1$.

case A2: If $q(t) < q(t-1)$ then $cost(t) < cost(t-1)$, $dist(t) > dist(t-1)$ and as a result $diff(t) > 0$, thus the controller will keep decreasing the value of q at the adaptation point $t+1$.

In both cases, our adaptive scheme examines locally the possible values of q and updates its value towards the direction that reduces the average access cost.

In Region B, we consider the same two cases:

case B1: If $q(t) \geq q(t-1)$ then $cost(t) \leq cost(t-1)$, $dist(t) \geq dist(t-1)$ and as a result

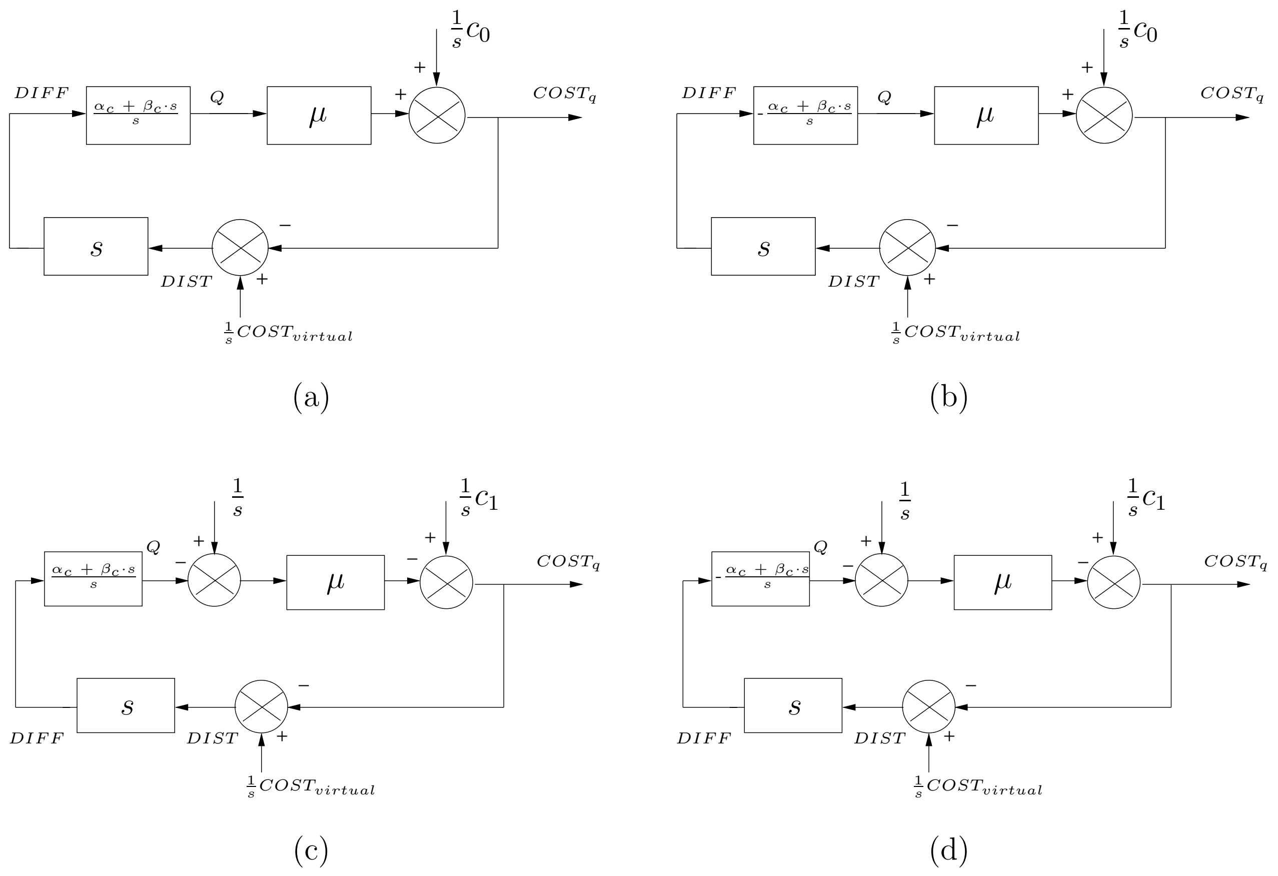

Figure 20: Laplace-Transform of the control loop for cases: (a) A1; (b) A2; (c) B1; (d) B2.

$diff(t) \geq 0$, thus the controller will increase the value of q at the adaptation point $t+1$.

case B2: If $q(t) < q(t-1)$ then $cost(t) > cost(t-1)$, $dist(t) < dist(t-1)$ and as a result $diff(t) < 0$, thus the controller will change course and increase the value of q at the adaptation point $t+1$.

As in the previous cases, our adaptive scheme updates the value of q in the direction of reducing the access cost of the outlier.

We follow a control-theoretic approach to show the convergence properties of our controller. We start by providing the proof for case **A1**:

From Algorithm 1, we derive the following continuous-time equations:

$$\frac{\partial dist(t)}{\partial t} = diff(t) \tag{10}$$

where

$$dist(t) \;=\; cost_{virtual}(t) \;-\; cost_q(t) \tag{11}$$

The value of q is updated as follows:

$$\frac{\partial q(t)}{\partial t} \;=\; \alpha_c \cdot diff(t) \;+\; \beta_c \cdot \frac{\partial diff(t)}{\partial t} \tag{12}$$

We can approximate the average access cost under our adaptive scheme as follows:

$$cost_q(t) \;\approx\; c_0 \;+\; \mu \cdot q(t) \tag{13}$$

where $\mu \;=\; tan(\theta)$ and c_0 is the cost of the outlier for $q = 0$ and $t'_r = t_l$.

Next, we take the Laplace-transform of (10), (11),(12) and (13) and draw the block diagram that describes the flow of the signals (Figure 20(a)). We can now derive the relation between $DIFF$ and $COST_q$ (in the s-domain):

$$\frac{c_0}{s} \;+\; DIFF(s) \cdot \frac{\alpha_c \;+\; \beta_c \cdot s}{s} \cdot \mu \;=\; COST_q(s)$$

Furthermore, assuming that $cost_{virtual}$ is constant [10], we have:

$$DIFF(s) \;=\; s\left(\frac{1}{s}COST_{virtual} \;-\; COST_q(s)\right)$$

After some algebraic manipulations we have:

$$COST_q(s) = \frac{c_0 \;+\; (\alpha_c \;+\; \beta_c \cdot s) \cdot \mu \cdot COST_{virtual}}{s \cdot (1 \;+\; (\alpha_c \;+\; \beta_c \cdot s) \cdot \mu)}$$

From the above equation we can conclude that the system is stable and overdamped since the pole $s \;=\; -\frac{1}{\beta_c} \cdot (\frac{1}{\mu} \;+\; \alpha_c)$ is negative [11, 27]. In order to find the steady-state value of the average cost, we use the Final Value Theorem [11, 27]:

$$cost_q(\infty) \;=\; \lim_{s \to 0} \; s \cdot COST_q(s) \;=\; \frac{c_0 \;+\; \alpha_c \cdot \mu \cdot COST_{virtual}}{1 \;+\; \alpha_c \cdot \mu}$$

[10]Notice that the value of the virtual cache cost is independent of t'_r

If we can calculate the "desired" slope μ, we can estimate the optimal value for α_c. In principle, for small c_0 the smaller the value of $\alpha_c < 1$ the lower the access cost is, even when the value of μ is not known in advance.

In case **A2**:

We follow the same analysis that was provided for the **A1** case, but with a new expression in place of Equation (12):

$$\frac{\partial q(t)}{\partial t} \;=\; -\,\alpha_c \cdot diff(t) \;-\; \beta_c \cdot \frac{\partial diff(t)}{\partial t}$$

The block diagram for this case is illustrated in Figure 20(b). It is easy to show that the steady-state value of the average access cost is given by:

$$cost_q(\infty) \;=\; \frac{c_0 \;-\; \alpha_c \cdot \mu \cdot COST_{virtual}}{1 \;-\; \alpha_c \cdot \mu}$$

In case **B1**:

We follow the same analysis that was provided for the A1 case, but with a new expression for Equation (13):

$$cost_q(t) \;\approx\; c_1 \;-\; \mu \cdot (1 \;-\; q(t))$$

where c_1 is the cost of the outlier for $q = 1$ and $t'_r = t_s$. Furthermore, notice that $c_1 \;\approx\; c_0 + \mu$. The block diagram for this case is illustrated in Figure 20(c). After some algebraic manipulations, we can show that the steady-state value of the average access cost is given by:

$$cost_q(\infty) \;=\; \frac{c_0 \;+\; \alpha_c \cdot \mu \cdot COST_{virtual}}{1 \;+\; \alpha_c \cdot \mu}$$

Notice that this steady-state value is the same as in case A1.

In case **B2**:

The initial analysis for A1 case is used, but Equation (12) is not valid, as q is updated as in case A2, and Equation (13) is not valid, as the cost of the outlier follows the same relation

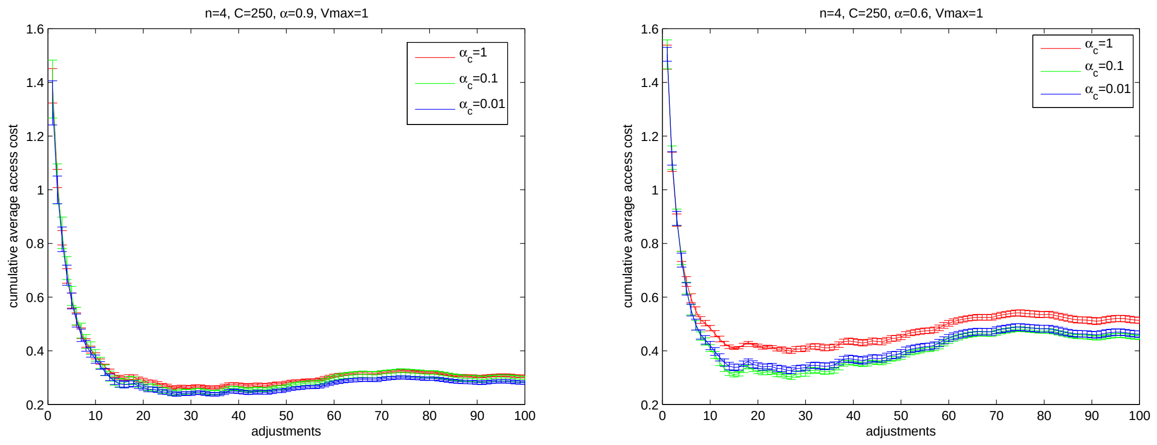

Figure 21: The effect of α_c on average access cost under different demand skewness

as in case B1. The block diagram for this case is illustrated in Figure 20(d). Following the same analysis, it is easy to show that the steady-state value of the average access cost is given by:

$$cost_q(\infty) \;=\; \frac{c_0 \;-\; \alpha_c \cdot \mu \cdot COST_{virtual}}{1 \;-\; \alpha_c \cdot \mu}$$

Note that this steady-state value is the same as for case A2.

To investigate the effect of the choice of α_c on the average access cost of the outlier, we repeat 10 experiments with the setup that was described in *Section 5.2*, for different values of α_c. As it is illustrated in Figure 21, when the demand is very skewed the effect of the choice of α_c on the access cost of the outlier is minimal as there is overlap of the 95^{th}-percentile confidence intervals for values of α_c that range from 0.01 to 1. For less skewed demands, the access cost of the outlier is very sensitive to the choice of α_c. By setting the value of α_c close to 1, yields significant increase in the access cost of the outlier. For values of $\alpha_c \leq 0.1$, our adaptive scheme outperformed the most cost effective static scheme (LRU($q = 0$) or LRU($q = 1$), for different cache sizes and demand skewness. For a detailed description of the parameters of this experiment the reader is refer to Appendix 2.

APPENDIX 2: Experimental Performance Evaluation of the Adaptive Scheme

In this appendix we provide a detailed description of the design and parameterization of the experiment used to evaluate the performance of our adaptive scheme. We consider the outlier scenario described in *Section* 6.2. Motivated by a realistic scenario from a wireless caching application [37], we capture the dynamics in t'_r by having the outlier move according to the Modified Random Waypoint Model (MRWP) [23]—this recent version fixes the non-stationarity of the original model, and thus provides better statistical confidence. Under the MRWP model, the outlier (mobile) node picks an initial distance D_0 according to the distribution:

$$F_D^0(r) = \frac{3}{2R_{max}}\left(r - \frac{r^3}{3R_{max}^2}\right)$$

for the first time period X_0. R_{max} is the maximum distance that a mobile node can travel in a given chosen direction. Moreover, the node picks a velocity V_0 uniformly from $[V_{min}, V_{max}]$, where V_{min} and V_{max} denote the minimum and maximum speed of the outlier node, respectively. For the following time periods $X_i,\ i > 0$, the outlier node picks distance D_i according to the distribution:

$$F_D(r) = \left(\frac{r}{R_{max}}\right)^2$$

and speed according to the distribution:

$$F_V(v) = \frac{v^2 - V_{min}^2}{V_{max}^2 - V_{min}^2}$$

Upon reaching the randomly chosen destination point, the outlier node pauses for a time period P, and the process repeats itself until the end of the simulation.

In order to map the distance (determined by this mobility model) to an associated cost in the mobile environment, we use the energy cost function which is proportional to the square of that distance [13], *i.e.* the instantaneous outlier's cost is given by $t'_r(t) = \left(\frac{r(t)}{R_{max}}\right)^2 t_s$, where $r(t)$ is the current distance of the outlier from other group members.

Modified RWP parameters			Controller parameters		
Outlier's Speed	mean(t'_r)	stdv(t'_r)	update control period	α_c - C=250/150/50	β_c
low: $V_{max} = 1$ du/tu	0.67 du	0.49 du	250 local requests	0.1/0.1/1	0.01
moderate: $V_{max} = 5$ du/tu	0.62 du	0.45 du	50 local requests	0.1/0.1/1	0.01
high: $V_{max} = 20$ du/tu	0.65 du	0.46 du	13 local requests	0.1/0.1/1	0.01

Table 1: The characteristics of the Modified RWP model and the controller that we used for our simulation setup.

Unless otherwise specified, for the modified RWP mobility model, we set V_{min} and P to zero, and the dimensions of the space inside which the outlier node moves are given by a circle of radius R_{max}=1000 distance units (du) centered around other (non-mobile) nodes in the cooperative group. We also take the time between successive requests for objects as our basic time unit (tu).

The rate at which our feedback controller updates q should depend only on the access cost rate of local requests. Given that the latter is determined by the rate of change in distance traveled by the outlier (which is reflected to how fast its access cost changes) the control update period must be set accordingly. The average distance that the mobile object travels is given by:

$$E[D] \;=\; \sum_{D_i} D_i \cdot pmf(D_i) \;=\; \sum_{D_i} D_i \cdot (F_D(D_{i+1}) - F_D(D_i)) \;=\; 2/3\ R_{max}$$

Following the same analysis it can be shown that $E[V] \;=\; 2/3\ (V_{max} \;-\; V_{min})$ and as a result the average travel time of the outlier is $E[X] \;=\; \frac{E[D]}{E[V]} \;=\; \frac{R_{max}}{V_{max} \;-\; V_{min}}$.

Under the assumption that each node in the group (of n nodes) generates on average the same number of requests, the embedded controller in a node updates q at least every $E[X]/n$ local requests.

We consider three different values of speed for the outlier (mobile) node: low, moderate, and high. We generate $100,000$ requests uniformly initiated from the peers in the group. We consider a group size of four nodes, of which one is the outlier node. Without further optimization, following the design disciplines that were sketched in Appendix 1, we set $\alpha_c \;=\; 0.1$ and $\beta_c \;=\; 0.01$. To match the aggressiveness of LRU under small cache sizes (C=50), we set $\alpha_c = 1$. We compare our PID-style controller with two static schemes (LRU($q = 0$), LRU($q = 1$)), corresponding to no- or full-cooperation, respectively. A summarization of the Modified Random Waypoint Model and Controller parameters we used in our experiments is presented in Table 1.

We repeated the experiment 10 times under each setting (of cache size, access demand skewness and maximum speed of the outlier) under the adaptive and the static schemes using the same random seed for a single run of the experiment. In all experiment we report the virtual cache cost as well as the actual access cost using EWMA where the weight of the history w was set to 0.875.