Mistreatment-Resilient Distributed Caching ⋆

Georgios Smaragdakis[1], Nikolaos Laoutaris[1,2], Azer Bestavros[1]
Ibrahim Matta[1], Ioannis Stavrakakis[2]

Abstract

The distributed partitioning of autonomous, self-aware nodes into cooperative groups, within which scarce resources could be effectively shared for the benefit of the group, is increasingly emerging as a hallmark of many newly-proposed overlay and peer-to-peer applications. Distributed caching protocols in which group members cooperate to satisfy local requests for objects is a canonical example of such applications. In recent work of ours we identified *mistreatment* as a potentially serious problem for nodes participating in such cooperative caching arrangements. Mistreatment materializes when a node's access cost for fetching objects worsens as a result of cooperation. To that end, we outlined an emulation-based framework for the development of mistreatment-resilient distributed selfish caching schemes. Under this framework, a node opts to participate in the group only if its individual access cost is less than the one achieved while in isolation. In this paper, we argue against the use of such *static "all or nothing"* approaches which force an individual node to either join or not join a cooperative group. Instead, we advocate the use of a smoother approach, whereby the level of cooperation is tied to the benefit that a node begets from joining a group. To that end, we propose a distributed and easily deployable feedback-control scheme which mitigates mistreatment. Under our proposed adaptive scheme, a node independently emulates its performance as if it were acting in a greedy local manner and then adapts its caching policy in the direction of reducing its measured access cost below its emulated greedy local cost. Using control-theoretic analysis, we show that our proposed scheme converges to the minimal access cost, and indeed outperforms any static scheme. We also show that our scheme results in insignificant degradation to the performance of the caching group under typical operating scenaria.

Key words: Cooperative Caching, Service Overlay Networks, Peer-to-Peer Networks, Control Theory, Performance Evaluation.

1 Introduction

Background and Scope: Network applications often rely on distributed resources available within a cooperative grouping of nodes to ensure scalability and efficiency. As typical in many applications such as web server farms or content distribution networks, the grouping of nodes is dictated by a common strategic objective, and as such, the payoff from cooperation is assessed by the overall benefit to the group as opposed to the benefit reaped by individual nodes in the group (which in this case are not presumed to be selfish). 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 the various overlay and peer-to-peer (P2P) applications [2,3]. For such applications, the grouping of nodes is not governed by a common objective, but rather by the individual (selfish) objectives of the constituent nodes. Under such a setting, and as we have shown in prior work of ours [4,5], it is possible for a node (or a set of nodes) to be "mistreated" in the sense that its participation in the group (while advantageous to the group) would not be advantageous to its own objective. In this paper we show how to design mistreatment-resilient cooperative applications.

Mistreatment in Distributed Selfish Replication and Caching Systems: As part of our recent work on mistreatment in distributed cooperative settings [4,5], we focused on content networking applications, whereby the distributed resource being shared amongst a group of nodes is *storage*. In particular, we considered 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 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 max-

* 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. A preliminary version of this work appeared in the proceedings of 2006 IFIP Networking Conference [1].

[1] Computer Science Dept, Boston University, Boston, Massachusetts, USA. Email: {gsmaragd, nlaout, best, matta}@cs.bu.edu
[2] Dept of Informatics and Telecommunications, University of Athens, Athens, Greece. Email: istavrak@di.uoa.gr

imal access cost. Contrary to most previous work in the field, we considered *selfish nodes*, *i.e.*, nodes that cater strictly and only to the minimization of the access cost for their local client population (disregarding any consequences for the performance of the group as a whole).

In [4,5] we established the vulnerability of many *socially optimal* (SO) object replication/caching schemes to *mistreatment* problems. A mistreated node was defined as a node whose access cost under some cooperative scheme is higher than the corresponding minimal access cost that the node can guarantee for itself by being uncooperative. 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.

Proactive replication strategies such as those studied in [4] 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 for 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 load), 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 [6] and the estimation of the non-stationary demands (or equivalently, the timescale for a stationarity assumption to hold) [7] 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 [5] we considered the problem of *Distributed Selfish Caching* (DSC), which could be seen as the *on-line* equivalent of the *Distributed Selfish Replication* (DSR) problem [4]. In DSC, we adopted an *object caching* model, whereby a node used demand-driven temporary storage of objects, combined with replacement. Based on that model, we uncovered the operational characteristics of a DSC group that can give rise to mistreatment problems. And, while we argued that under stationary conditions, simple parametric versions of already established protocols and mechanisms are capable of mitigating these problems, we did not prescribe an integrated approach for regulating these parameters in a manner that adapts to the constantly changing conditions of the group (*e.g.*, varying group size, node capacities, delays, and demand patterns).

Mistreatment-Resilient Distributed Selfish Caching: In this paper, we take our work one significant, constructive step further by proposing a general control-theoretic framework, which enables the parametrization of DSC protocols so as to make these protocols resilient to mistreatment, even under the aforementioned fluid group conditions. Through extensive analysis and simulation experiments, we show that our adaptive scheme not only mitigates mistreatment that may evolve in a distributed caching group, but it also guarantees an access cost for each individual node that is lower than the one achieved by any static scheme. We also show that the impact of this adaptive scheme on the performance of the distributed caching group is minimal under typical operating scenaria.

Organization of the Paper: The rest of the paper is organized as follows. Section 2 summarizes related work in cooperative caching and dynamic schemes used to optimize caching. In Section 3 we describe our model of a distributed caching group. In Section 4 we demonstrate the causes of mistreatment in distributed caching groups. The design of a generic feedback controller for the mitigation of mistreatment is covered in Section 5. In Section 6 we argue that a node equipped with our feedback controller achieves the minimum access cost (compared to any other static scheme). In Section 7, we study the impact of such a controller on the overall performance of the distributed group. We evaluate the performance of our controller with extensive simulations in Section 8. Section 9 concludes the paper.

2 Related Work

Cooperative caching [8,9] allows multiple caches to cooperate in servicing each others' requests. Monitoring and controlling the number of copies for the same documents across different caches has been studied in [10] for the web and [11] for wireless ad hoc networks. All aforementioned studies were concerned with the minimization of the aggregated cost of the group. Apart from our previous work [5,1], we are aware of only two additional works that deal with dynamic schemes to optimize caching: According to [12] different caching algorithms may be employed dynamically to best serve the demand; in [13], a control-theoretic approach is proposed for achieving performance differentiation in proxy caches. Our study differs from these two works, by focusing on characterizing the impact of dynamic schemes on cache groups, as opposed to individual caches. 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 Model of a Distributed Caching Group

In this section we present the model of a distributed caching group that we consider in our study. Let o_i, $1 \leq i \leq N$, and v_j, $1 \leq j \leq n$, denote the i^{th} unit-sized object and the j^{th} 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 requested by the local users of node v_j. Successive requests are assumed to be independent and identically distributed. [3] For our numerical examples in later sections we will assume that the i^{th} most popular object is requested according to a generalized power-law distribution, i.e., with probability $p_i = \Lambda / i^{\alpha}$ (such distributions have been observed in many measured workloads [18,20]).

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] these costs can be interpreted either as delay costs for delivering an object to the requesting user or as bandwidth consumption costs for bringing the object from its initial location. User requests are serviced by the closest node that stores the requested object along the following chain: local node, group, origin server. Each node employs an object admission algorithm for storing (or not) objects retrieved remotely either from the group or from the origin server. Furthermore, each node employs a replacement algorithm for managing the content of its cache. In this work we focus on the Least Recently Used (LRU) replacement algorithm but we can obtain similar results under other replacement algorithms, such as Least Frequently Used (LFU) replacement algorithm (see also our previous work in [5]).

[3] The Independent Reference Model (IRM) [14] is commonly used to characterize cache access patterns [15–18]. The impact of temporal correlations was shown in [7,19] 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 (those values can also be assumed as upper bounds in our analysis). Our results can be adapted easily to accommodate arbitrary inter-node distances.

4 Mistreatment in Distributed Caching Groups

The examination of the operational characteristics of a group of nodes involved in a distributed caching solution enabled us to identify two key culprits for the emergence of mistreatment phenomena [5]: (1) the use of a *common caching scheme* across all the nodes of the group, irrespectively of the particular capabilities and characteristics of each individual node, and (2) the mutual *state interaction* between replacement algorithms running on different nodes.

4.1 Mistreatment Due to Common Scheme

The *common caching scheme* problem is a very generic vehicle for the manifestation of mistreatment. To understand it, one has first 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 caching 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 since otherwise, they would have to be fetched from the origin server at a high access cost.

Enforcing a common scheme under such a setting is bound to mistreat either the outlier node or the rest of the group. Consider the group depicted in *Figure* 1 in which $n-1$ nodes are clustered together, meaning that they are very close to each other ($t_r \to t_l \approx 0$), while there's also a single "outlier" node at distance t_r' from the cluster. The $n-1$ nodes would naturally employ a *Single Copy* (SC) scheme, *i.e.*, a scheme where there can be at most one copy

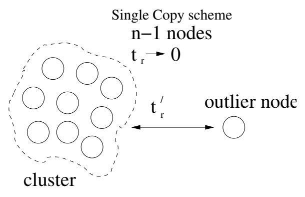

Fig. 1. An example of a group composed of a cluster of $n-1$ nodes and a unique outlier.

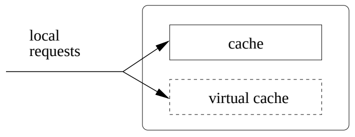

Fig. 2. Block diagram of a node equipped with a virtual cache.

of each distinct object in the group (e.g. LRU-SC [10]) 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 $t_r' \to t_r$, the outlier should obviously follow LRU-SC and avoid duplicating objects that already exist elsewhere in the group. If $t_r' \gg t_r$, then the outlier should follow a *Multiple Copy* (MC) scheme, *i.e.*, a scheme where there can be multiple copies of the same object at different nodes — an example of an MC scheme is the LRU-MC. Under LRU-MC, if a node retrieves an object from a remote node in the group (or the origin server), then it stores a copy of it locally replacing an existing object if the cache is full, according to the LRU policy.

4.2 Mistreatment Due to State Interaction

The *state interaction* problem 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[5], then the remote hit for object o will affect the state of the replacement algorithm in effect at u. If u is employing LRU replacement, then o will be brought to the top of the LRU list. If u employs LFU replacement, then the frequency of o will be increased, and so on with other replacement algorithms [22]. 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 u: u's cache may get invaded by objects that follow v's demand, thereby depriving the users 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.

[5] The Internet Cache Protocol (ICP)/Squid web cache [21] and other systems (e.g., Akamai Content Distribution Network, IBM Olympic Server Architecture), by default, do not discriminate between local and remote requests.

5 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 the adoption of a common cache management scheme and cache state interactions), 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 [23], not to mention that there are vastly more parameter sets and combinations of schemes that cannot all be investigated exhaustively.

5.1 Design Disciplines

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. 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 [4], a node compared its access cost under a given replication scheme with the guaranteed maximal access cost obtained through greedy local (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 2 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[6]. It is used

[6] Since the virtual cache stores object identifiers as opposed to the actual objects, the memory it uses is insignificant (compared to the memory required to store the actual objects). The processing cost is also trivial (i.e., incurring $O(1)$ update cost for each request under both LRU and LFU replacement policies).

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 local 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 (non-cooperative) 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 outlier should adjust its caching behavior according to its distance from the group. For this purpose, we introduce the LRU(q)-scheme, under which, objects that are fetched from the group are cached locally only with probability q; q will hereafter be referred to as the *reliance parameter*, capturing the amount of reliance that the node puts into being able to fetch objects efficiently from other nodes. 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.

5.2 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(q =1) scheme,

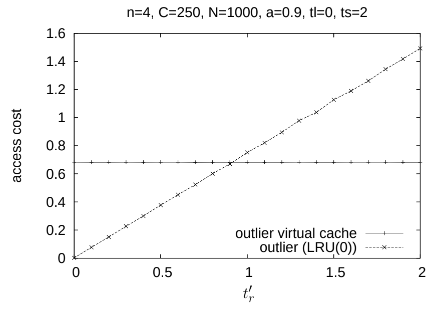

Fig. 3. Simulation results on the effect of the remote access cost t'_r on the access cost of the outlier node under the virtual cache and LRU(0) schemes.

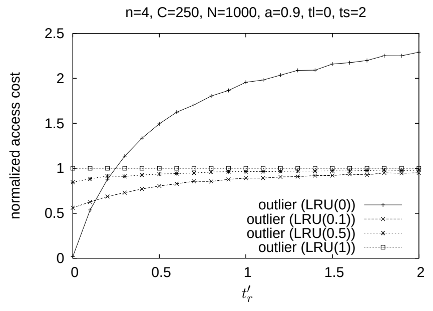

Fig. 4. Simulation results on the effect of the remote access cost t'_r on the normalized (by the virtual cost) access cost of the outlier node under different LRU(q) schemes.

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. [7] By comparing the access cost of 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 3 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 Figure 4). Consider two different values of the reliance parameter q_1 and q_2 such that $q_1 < q_2$. Figure 5 illustrates a typical behavior of the average object access cost under q_1 and q_2 as a function of the distance 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 (converges to a target value with zero error) [24,25].

[7] The outlier can include in the emulation the cost of 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 replace the access cost of remote fetches by that from the origin server and thus reduce the inter-node query traffic; this would give it an upper bound on the access cost under the emulated scheme.

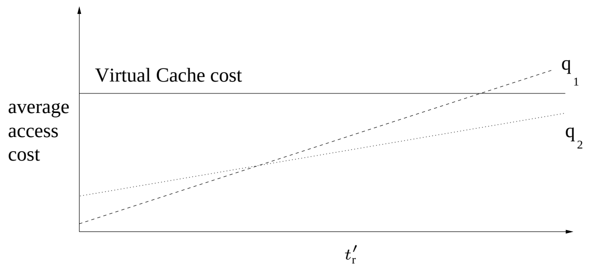

Fig. 5. Representative behavior of average object access cost as a function of the reliance parameter and distance of the outlier from the cluster, $q_1 < q_2$.

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*.

The PID controller adapts q proportionally to the magnitude of *diff*; if the magnitude of *diff* is small then consecutive observations of the performance of the system are very close, thus the value of q should be changed smoothly until the *diff* is zeroed. On the other hand, if the magnitude of *diff* is high, then the value of q has to be updated significantly. A pseudo-code for this process is provided in Algorithm 1. In the forthcoming section (Section 6), we argue that the access cost of a node equipped with this controller converges to a value which is lower than that of any scheme that employs a fixed q. We also provide an estimation of the converged value as a function of controller parameters and other system characteristics.

Our algorithm has two parameters. The first one, denoted by α_c, is the *gain* of the controller, which determines the rate 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. A methodology on how to tune these parameters and how sensitive is the performance of the controller to these parameters will be presented in Section 8 (evaluation section).

5.3 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 con-

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** $q(t) \ \geq \ q(t-1)$ **then**
    $q(t+1) \ \leftarrow \ q(t) \ + \ \sigma \cdot \alpha_c \cdot |diff(t)| + \ \sigma \cdot \beta_c \cdot | \ |diff(t)| \ - \ |diff(t-1)| \ |$
**else**
    $q(t+1) \ \leftarrow \ q(t) \ - \ \sigma \cdot \alpha_c \cdot |diff(t)| - \ \sigma \cdot \beta_c \cdot | \ |diff(t)| \ - \ |diff(t-1)| \ |$
```

trolled 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).

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.

6 Convergence of the Controller

In this section 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* 4.1.

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

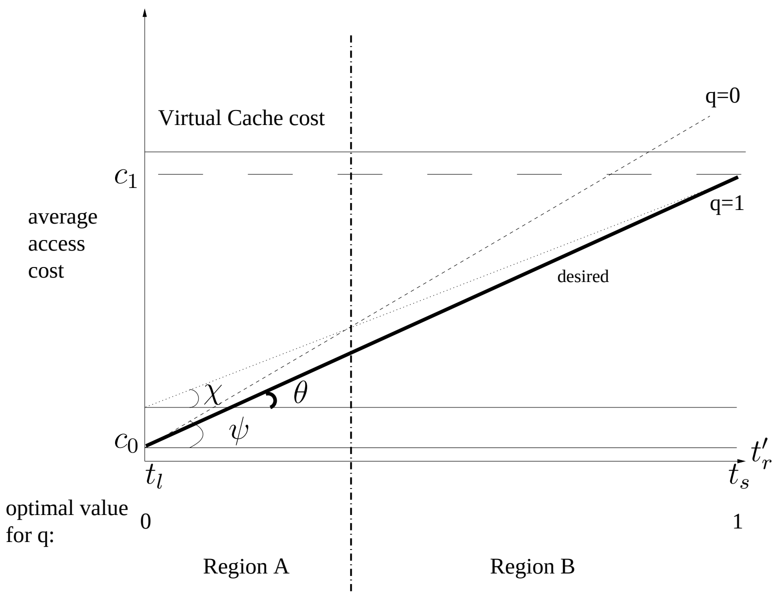

Fig. 6. Representative behavior of static schemes ($LRU(q = 0)$, $LRU(q = 1)$) and the "desired" behavior of the controller.

of any static scheme (*i.e.* a scheme employing a fixed q).

Justification: As illustrated in Figure 5, 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 6, 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 cache 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 6. 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.

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. Thus, for the purpose of our analysis, let us

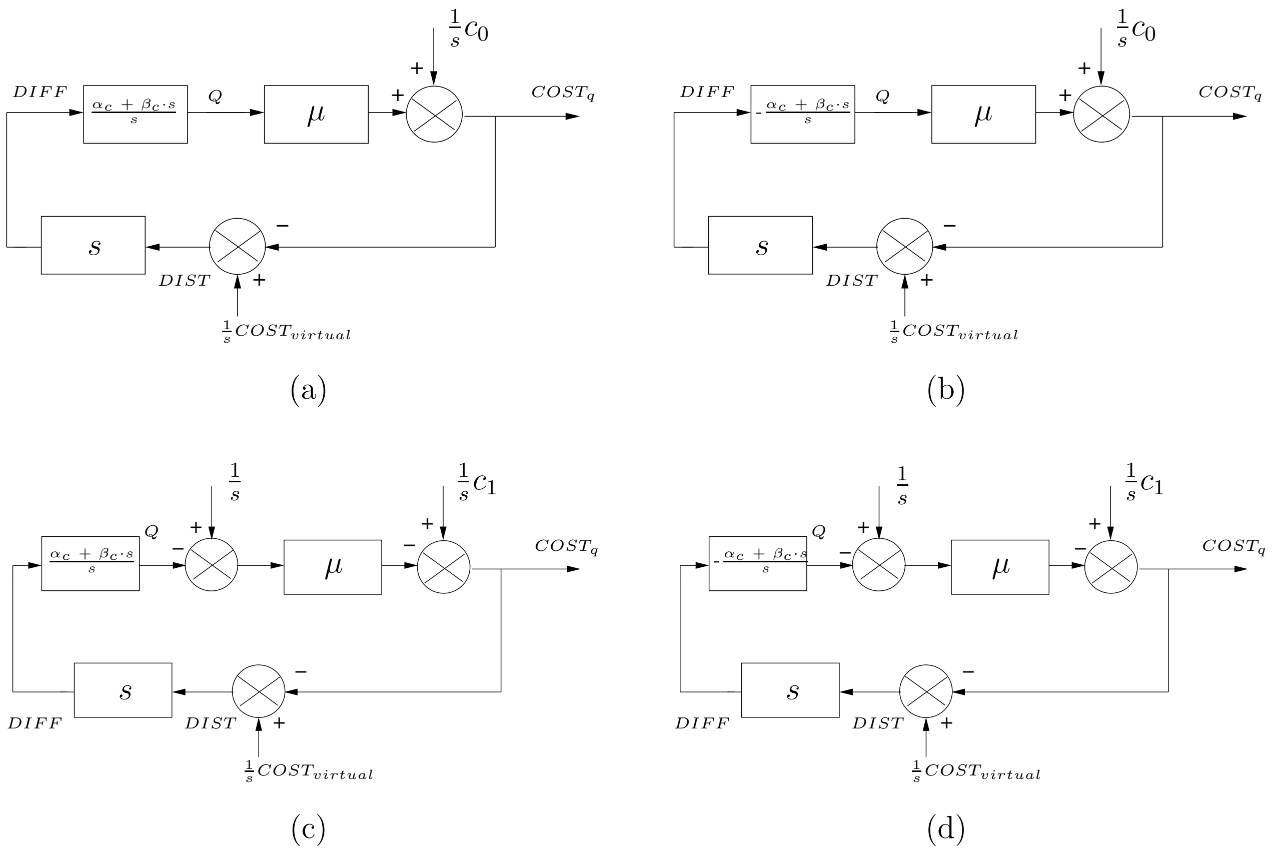

Fig. 7. Laplace-Transform of the control loop for cases: (a) A1; (b) A2; (c) B1; (d) B2.

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 $diff(t) \geq 0$, thus the controller will keep increasing 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 $\mathit{diff}(t) \ < \ 0$, 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:

$$\frac{\partial dist(t)}{\partial t} \ = \ diff(t) \tag{1}$$

where

$$dist(t) \ = \ cost_{virtual}(t) \ - \ cost_q(t) \tag{2}$$

The value of q is updated as follows:

$$\frac{\partial q(t)}{\partial t} \ = \ \alpha_c \cdot diff(t) \ + \ \beta_c \cdot \frac{\partial diff(t)}{\partial t} \tag{3}$$

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

$$cost_q(t) \ \approx \ c_0 \ + \ \mu \cdot q(t) \tag{4}$$

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 Equations (1), (2), (3) and (4), and draw the block diagram that describes the flow of the signals (Figure 7(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 [8], 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 over-damped since the pole $s \;=\; -\frac{1}{\beta_c} \cdot (\frac{1}{\mu} \;+\; \alpha_c)$ is negative [24,25]. In order to find the steady-state value of the average cost, we use the Final Value Theorem [24,25]:

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

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. We show that this condition on α_c holds for all other cases as well.

In case **A2**:

We follow the same analysis that was provided for the **A1** case, but with a new expression in place of Equation (3):

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

The block diagram for this case is illustrated in Figure 7(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 (4):

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

[8] Note that the value of the virtual cache cost is independent of t'_r.

where c_1 is the cost of the outlier for $q = 1$ and $t'_r = t_s$. Furthermore, note that $c_1 \approx c_0 + \mu$. The block diagram for this case is illustrated in Figure 7(c). After some algebraic manipulations, we can show that the steady-state value of the average access cost is given by:

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

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

In case **B2**:

We follow the same analysis that was provided for the A2 case, but with a new expression for the Equation (4). The block diagram for this case is illustrated in Figure 7(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.

7 The Effect of Individual Controllers on the Overall Performance of the Group

In this section we turn our attention to the performance implications resulting from the use of individual-cost minimizing controllers at different nodes. In particular, we look at the impact on the (global) group's performance by comparing the *aggregated steady-state access cost* of the distributed caching group (AAC_{PID}) when all constituent nodes are equipped with the PID controller described in *Section* 5.2 with the corresponding cost of the distributed caching group (AAC) when nodes are not equipped with the PID controller. We refer to the ratio AAC_{PID}/AAC as the *controller's impact ratio* (CIR) on group performance.

7.1 Analysis

We consider a setting similar to the one described in Section 4.1, but with $n - k$ nodes clustered together and k "outlier" nodes at distance t'_r from the cluster. Nodes u_j, $1 \leq j \leq n - k$, are cluster nodes whereas nodes u_j, $n - k + 1 \leq j \leq n$, are outlier nodes. Each node has storage capacity up to C

unit-sized objects, and has the same request rate λ as all other nodes in the caching group. In our analysis, we assume that the i^{th} most-popular object has a request probability drawn from a generalized power-law with skewness a, *i.e.*, $p_i = \Lambda/i^a$, $1 \leq i \leq N$.

Closed-form analytic models for prediction of cache performance do not exist (see [26] for the development of numerical models). A simulation study is more tractable, but given the size of the parameter space we are considering, it is not clear that simulation studies would be useful. Thus, the alternative we adopt in this paper relies on the observation that the content of a cache could be seen as the result of superimposing an offline-optimal replication strategy with the "noise" caused by replacement errors. Using this approach we are able to draw some basic qualitative conclusions for caching by studying replications [27].

Under this framework, the clustered group of nodes can be abstracted by a single cache with storage capacity of $(n-k)C$ unit-sized objects, whereas each outlier node represents a single cache with storage capacity of C unit-sized objects. Assuming that the size of the cluster is larger than the population of the outliers (*i.e.*, $n > 2k$), the $(n-k)C$ most popular objects will be stored in the clustered nodes, this would occur under LRU ($q = 0$) or any other SC-scheme (which assumes full collaboration in caching decisions, e.g., as with the case of hash-based caching). Objects that are less popular–namely objects with ids $(n-k)C+1, .., nC$– would be stored in the outlier nodes. Without loss of generality, we assume that objects $(j-1)C+1, .., jC$, will be stored in the node u_j, $n-k+1 \leq j \leq n$.

The aggregated access cost of the distributed caching group is the summation of the individual costs of the clustered and the outlier nodes, and is equal to:

$$\begin{aligned}
AAC &= \sum_{j=1}^{n-k} cost_j + \sum_{j=n-k+1}^{n} cost_j \\
&= \sum_{j=1}^{n-k} \left(\sum_{i=1}^{(n-k)C} p_i t_l + \sum_{i=(n-k)C+1}^{nC} p_i t'_r + \sum_{i=nC+1}^{N} p_i t_s \right) \\
&\quad + \sum_{j=n-k+1}^{n} \left(\sum_{i=1}^{(n-k)C} p_i t'_r + \sum_{i=(n-k)C+1}^{nC} p_i t_{ij} - \sum_{i=(j-1)C+1}^{jC} p_i t_{ij} + \sum_{i=nC+1}^{N} p_i t_s \right)
\end{aligned}$$

where t_{ij}, $(n-k)C+1 \leq i \leq nC$, $n-k+1 \leq j \leq n$ is the cost of fetching object o_i by outlier node u_j from a different outlier node. In the worst case scenario, this object will be stored in another outlier which cannot be within a distance greater than $2t'_r$. Thus, for the aforementioned values of i, j, $t_{ij} \leq \min(2t'_r, t_s)$. For the rest of our analysis, we will assume that $t'_r < 0.5t_s$, in order to study a more realistic setting, in which outliers are not extremely far from the cluster.

Substituting in the above equation, with $t_{ij} \leq 2t'_r$ and $t_l = 0$, we get:

$$\begin{aligned}
AAC \leq & \sum_{j=1}^{n-k} \left(\sum_{i=1}^{(n-k)C} p_i t_l + \sum_{i=(n-k)C+1}^{nC} p_i t'_r + \sum_{i=nC+1}^{N} p_i t_s \right) \\
& + \sum_{j=n-k+1}^{n} \left(\sum_{i=1}^{(n-k)C} p_i t'_r + \sum_{i=(n-k)C+1}^{nC} p_i \; 2t'_r - \sum_{i=(j-1)C+1}^{jC} p_i \; 2t'_r + \sum_{i=nC+1}^{N} p_i t_s \right) \\
= & (n-k) \left(\sum_{i=(n-k)C+1}^{nC} p_i t'_r + \sum_{i=nC+1}^{N} p_i t_s \right) \\
& + k \left(\sum_{i=1}^{(n-k)C} p_i t'_r + 2 \sum_{i=(n-k)C+1}^{nC} p_i t'_r + \sum_{i=nC+1}^{N} p_i t_s \right) \\
& - 2 \sum_{j=n-k+1}^{n} \sum_{i=(j-1)C+1}^{jC} p_i t'_r \\
= & (n-k) \left(\sum_{i=(n-k)C+1}^{nC} p_i t'_r + \sum_{i=nC+1}^{N} p_i t_s \right) \\
& + k \left(\sum_{i=1}^{(n-k)C} p_i t'_r + 2 \sum_{i=(n-k)C+1}^{nC} p_i t'_r + \sum_{i=nC+1}^{N} p_i t_s \right) \\
& - 2 \sum_{i=(n-k)C+1}^{nC} p_i t'_r \\
= & \Lambda[(n-k) \left(\left(H_{nC}^{(a)} - H_{(n-k)C}^{(a)} \right) t'_r + \left(H_N^{(a)} - H_{nC}^{(a)} \right) t_s \right) \\
& + k \left(\left(H_{(n-k)C}^{(a)} - H_0^{(a)} \right) t'_r + 2 \left(H_{nC}^{(a)} - H_{(n-k)C}^{(a)} \right) t'_r + \left(H_N^{(a)} - H_{nC}^{(a)} \right) t_s \right) \\
& - 2 \left(H_{nC}^{(a)} - H_{(n-k)C}^{(a)} \right) t'_r] \\
= & \Lambda \left[n t_s H_N^{(a)} - (n-2) t'_r H_{(n-k)C}^{(a)} + (n+k-2)\, t'_r H_{nC}^{(a)} - n t_s H_{nC}^{(a)} \right]
\end{aligned}$$

The generalized harmonic number $H_C^{(a)}$ can be approximated by its integral expression [28]:

$$H_C^{(a)} = \sum_{i=1}^{C} \frac{1}{i^a} \approx \int_1^C \frac{1}{l^a} dl = \frac{C^{1-a} - 1}{1-a}$$

where $H_0^{(a)} = 0$.

Using this approximation we obtain:

$$AAC \le \frac{\Lambda}{1-a}\left[\left(\left((n+k-2)t'_r - nt_s\right)n^{1-a} - (n-2)t'_r(n-k)^{1-a}\right)C^{1-a} + nt_sN^{1-a} - kt'_r\right] \quad (5)$$

In the presence of a controller at each node, the analysis is more involved. In particular, we note that the outlier nodes are located at distance t'_r from the cluster, and as a consequence, the cost to fetch a popular object from the group becomes much greater than the corresponding cost if the object was stored locally. Therefore, some of the objects (the most popular ones) may end up being cached locally, which is precisely the goal of the PID controller of Section 5.2. Thus, although the $(n-k)C$ most-popular objects would still be stored in the clustered nodes, the object placement in the outlier nodes may result in having duplicate (local) copies of these objects at the outliers as well.

The aggregated access cost for this setting is equal to:

$$\begin{aligned}
AAC_{PID} &= \sum_{j=1}^{n-k} cost_j + \sum_{j=n-k+1}^{n} cost_j \\
&= \sum_{j=1}^{n-k}\left(\sum_{i=1}^{(n-k)C} p_i t_l + \sum_{i=(n-k)C+1}^{N} p_i t_s\right) \\
&\quad + \sum_{j=n-k+1}^{n}\left(\sum_{i=1}^{C} p_i t_l + \sum_{i=C+1}^{(n-k)C} p_i t'_r + \sum_{i=(n-k)C+1}^{N} p_i t_s\right) \\
&= (n-k)\sum_{i=(n-k)C+1}^{N} p_i t_s + k\left(\sum_{i=C+1}^{(n-k)C} p_i t'_r + \sum_{i=(n-k)C+1}^{N} p_i t_s\right) \\
&= \Lambda\left[(n-k)\left(H_N^{(a)} - H_{(n-k)C}^{(a)}\right)t_s + k\left(\left(H_{(n-k)C}^{(a)} - H_C^{(a)}\right)t'_r + \left(H_N^{(a)} - H_{(n-k)C}^{(a)}\right)t_s\right)\right] \\
&= \Lambda\left[nt_sH_N^{(a)} + \left(kt'_r - nt_s\right)H_{(n-k)C}^{(a)} - kt'_rH_C^{(a)}\right] \\
&\approx \frac{\Lambda}{1-a}\left[\left((kt'_r - nt_s)(n-k)^{1-a} - kt'_r\right)C^{1-a} + nt_sN^{1-a}\right]
\end{aligned}$$

Using the harmonic approximation given in Equation (5) and substituting in the above, we can lower-bound the controller's impact ratio as follows.

$$\begin{aligned}
CIR &\ge \frac{\left((kt'_r - nt_s)(n-k)^{1-a} - kt'_r\right)C^{1-a} + nt_sN^{1-a}}{\left(\left((n+k-2)t'_r - nt_s\right)n^{1-a} - (n-2)t'_r(n-k)^{1-a}\right)C^{1-a} + nt_sN^{1-a} - kt'_r} \\
&= \frac{\left((kt'_r - nt_s)(1-\frac{k}{n})^{1-a} - \frac{k}{n}t'_r\right)\left(\frac{C}{N}\right)^{1-a} + t_s}{\left(\left((1+\frac{k}{n}-\frac{2}{n})t'_r - t_s\right)n^{1-a} - (n-2)t'_r(1-\frac{k}{n})^{1-a}\right)\left(\frac{C}{N}\right)^{1-a} + t_s - \frac{k}{nN}t'_r}
\end{aligned}$$

Figure 8 shows how the above lower-bound on CIR relates to the skewness of the power-law demand under a range of values for the relative population of the outliers.

7.2 Observations

The above bound and illustration allow us to make three important observations. First, given some relative storage capacity C/N, the controller's impact ratio is constant, and as one would expect, it is more pronounced for smaller values of C/N. Second, under skewed demands ($a \to 1$), the impact of the controller on the performance of the caching group is small. This could be explained by noting that for workloads with less skewed (*i.e.*, more uniform) demand distributions the marginal utility of the outlier caches to the caching group becomes higher, resulting in a more pronounced impact ratio. However, for highly-skewed demand, the collective caches of the caching group (excluding those at the outliers) would be able to accommodate most of the working set in the request stream, rendering the impact ratio insignificant. The aforementioned explanation is also supported by our numerical results. Figure 9 depicts the cluster node's cost increase ratio and the lower bound of outlier node's cost reduction ratio (cost of a node equipped with the PID controller compared to its cost when it is not equipped with the PID controller). Under skewed demands, the cluster node's cost increases in a linear fashion, but the outlier node's cost decreases exponentially fast. This is an important observation because most reference streams characterized in the literature exhibit highly-skewed demand distributions (see [18] for example). Third, when the relative population of the outliers is small ($k/n \to 0$), the controller's impact on the performance of the caching group is insignificant [9]. This observation is important because in a typical setting, one would expect that the clustering of nodes in a caching group would result in the presence of a small ratio of outlier nodes (k/n). Moreover, in a typical setting, a large k/n ratio may signify an anomalous condition (*e.g.*, the onset of congestion that effectively partitions the cluster), which may trigger a regrouping.

To summarize, under typically skewed demand distributions and for well-formed caching groups featuring a small proportion of outliers, our adaptive scheme guarantees minimal access cost for each individual selfish node, with insignificant degradation to the group's performance.

[9] Both cluster and outlier node's cost increases in a linear fashion with k/n (see Figure 9).

$n = 100,\ C = 50,\ N = 10000,\ t_l = 0,\ t_s = 2,\ t'_r = 0.3ts$

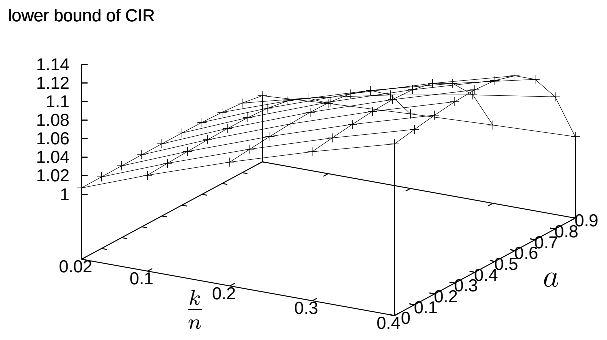

Fig. 8. The behavior of the lower bound of CIR, under power-law demand and different values for the relative population of outliers.

$n = 100,\ C = 50,\ N = 10000,\ t_l = 0,\ t_s = 2,\ t'_r = 0.3ts$

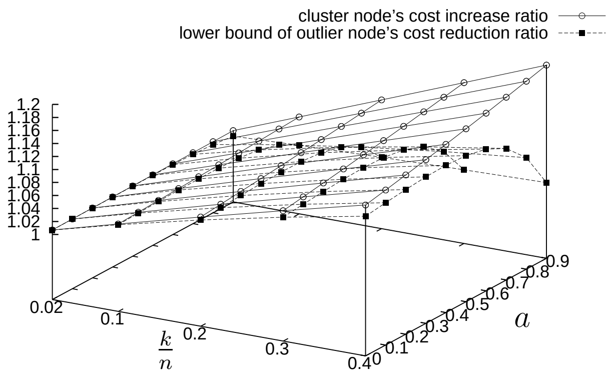

Fig. 9. The behavior of the cluster node's cost increase ratio and the lower bound of outlier node's cost reduction ratio, under power-law demand and different values for the relative population of the outliers.

8 Evaluation Of the Controller

In order to evaluate our adaptive scheme, we compare its steady-state average access cost to the corresponding cost of one of the two extreme static schemes (LRU($q = 0$) or LRU($q = 1$)), corresponding to full- or no-cooperation, respectively. To that end, we define the following performance metric:

$$minimum\ cost\ reduction\ (\%) \ = \ 100 \cdot \frac{cost_{static} \ - \ cost_{adaptive}}{cost_{static}} \tag{6}$$

where $cost_{adaptive}$ is the access cost of our adaptive mechanism, and $cost_{static}$ is the minimum cost of the two static schemes: $cost_{static} = min(\ cost(LRU(q =$

0)), $cost(LRU(q = 1))$). 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 Equation (6), where $cost_{static} = max(\ cost(LRU(q = 0)), cost(LRU(q = 1))\ )$.

8.1 Experimental Setting

In this subsection 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* 4.1.

Motivated by a realistic scenario from a wireless caching application [11], we capture the dynamics in t'_r by having the outlier move according to the Modified Random Waypoint Model (modified RWP) [29].[10] Under the modified RWP 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.

[10] This recent version fixes the non-stationarity of the original model, and thus provides better statistical confidence.

Modified RWP parameters		
Outlier's Speed	mean(t'_r)	stdv(t'_r)
low: $V_{max} = 1$ du/tu	0.67 du	0.49 du
moderate: $V_{max} = 5$ du/tu	0.62 du	0.45 du
high: $V_{max} = 20$ du/tu	0.65 du	0.46 du

Table 1
The characteristics of the Modified RWP model used in our simulation study.

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 [30], *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.

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 rate of change of the access cost of local requests. Given that the latter is determined by the rate of change in distance traveled by the outlier, the control update period must be set taking this distance change into consideration. 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.

8.2 Experimental Results

For the aforementioned setting, we consider a group of $n = 4$ nodes composed of a cluster of 3 nodes and a unique outlier. The size of the object universe was set to $N = 1000$. Each node is assumed to have the same storage capacity C and the same request rate as all the other nodes in the group. All the

	Controller parameters		
Outlier's Speed	update control period	α_c , C=250/150/50	β_c
low: $V_{max} = 1$ du/tu	250 local requests	0.1/0.1/1	0.01
moderate: $V_{max} = 5$ du/tu	50 local requests	0.1/0.1/1	0.01
high: $V_{max} = 20$ du/tu	13 local requests	0.1/0.1/1	0.01

Table 2
The characteristics of the PID Controller used in our simulation study.

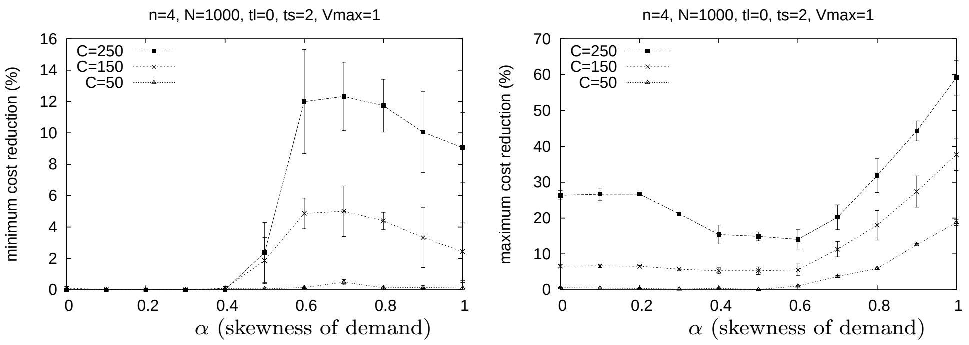

Fig. 10. Simulation results on the cost reduction that is achieved using our adaptive mechanism, (left): The minimum cost reduction, (right): The maximum cost reduction.

nodes in the caching group have the same Zipf (with parameter a) object popularity profile. We consider three different values for C such that the total capacity of the caching group, nC, is smaller than or equal to the size of the object universe. We also 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. For our adaptive controller, 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 $\alpha_c = 1$. Tables 1 and 2 summarize the parameters of the modified Random Waypoint Model and the controller used in our experiments respectively.

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 experiments we report the virtual cache cost as well as the actual access cost averaged using EWMA where the weight of the history w was set to 0.875.

Figure 10 summarizes results (along with 95^{th}-percentile confidence intervals) 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).

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 scheme 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 [18]). The explanation of this behavior is as follows. At the two extreme cases of skewness, one of the static scheme reaches its best 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.

8.3 Sensitivity of the Controller

Figure 11 depicts the effect of the choice of α_c on the average access cost of the outlier. 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.

9 Conclusion

Recent work in the literature [5] has uncovered the susceptibility of nodes participating in a distributed caching group to being mistreated, when a node's access cost for fetching objects while participating in the group is higher than that accrued when operating in isolation. Mistreatment emerges as a result of the adoption of a common caching scheme or as a result of the mutual state interaction between replacement algorithms used by members of the group. Preliminary mistreatment-resilient mechanisms either relied on decisions based on

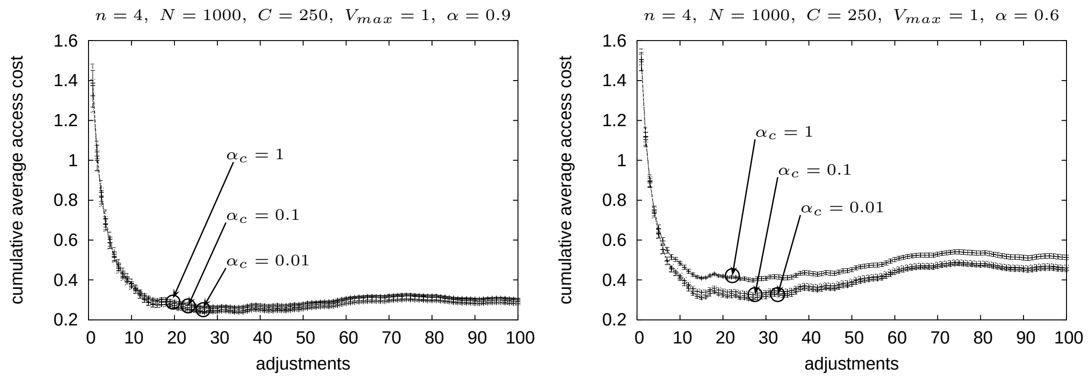

Fig. 11. Simulation results on the effect of the choice of α_c on average access cost under different demand skewness.

the emulated individual cost that can be achieved by the node while not participating in the group, or using probabilistic caching schemes by assigning values to the so-called reliance or interaction parameters, respectively [5]. The first approach was able only to provide a hard "all or nothing" switch between full-participation and no-participation in the caching group. The latter affords more flexibility to each node, but did not propose any concrete methodology for assigning values to the aforementioned reliance or interaction parameters, which are crucial especially in a dynamic environment.

In this paper, we have proposed a distributed and lightweight mechanism to mitigate mistreatment by optimally tuning the reliance and interaction parameter in a dynamic environment. Both the computational and memory requirements for developing such an adaptive mechanism are minimal. Through detailed analysis and extensive simulations we showed that a node equipped with this mechanism achieves a minimal access cost (compared to any other static scheme), without increasing substantially the access cost for the group, for a large spectrum of operational settings.

Our future research agenda, includes a tolerance study of our adaptive scheme against orchestrated adversarial mistreatment that may target the adaptation of our scheme. This type of mistreatment may be more intense [31] than the coincidental mistreatment that was the subject of the work presented in this paper. We are also interested in examining the applicability of our adaptive scheme on other networking applications, where a join/no-join decision is too conservative, including but not limited to admission control and load balancing where the resources being shared amongst a group of nodes can be cpu cycles, time slots or wireless spectrum, to give some examples.

References

[1] G. Smaragdakis, N. Laoutaris, I. Matta, A. Bestavros, I. Stavrakakis, A Feedback Control Approach to Mitigating Mistreatment in Distributed Caching Groups, in: Proceedings of IFIP Networking 2006, Coimbra, Portugal, 2006.

[2] J. W. Byers, J. Considine, M. Mitzenmacher, S. Rost, Informed content delivery across adaptive overlay networks, IEEE/ACM Transactions on Networking 12 (5) (2004) 767–780.

[3] E. Cohen, S. Shenker, Replication strategies in unstructured peer-to-peer networks, in: Proceedings of ACM SIGCOMM'02 Conference, Pittsburgh, PA, USA, 2002.

[4] N. Laoutaris, O. Telelis, V. Zissimopoulos, I. Stavrakakis, Distributed selfish replication, IEEE Transactions on Parallel and Distributed Systems 17 (12) (2006) 1401–1413.

[5] N. Laoutaris, G. Smaragdakis, A. Bestavros, I. Stavrakakis, Mistreatment in Distributed Caching Groups: Causes and Implications, in: Proceedings of the Conference on Computer Communications (IEEE Infocom), Barcelona, Spain, 2006.

[6] T. Loukopoulos, P. Lampsas, I. Ahmad, Continuous replica placement schemes in distributed systems, in: Proceedings of the 19th ACM International Conference on Supercomputing (ACM ICS), Boston, MA, 2005.

[7] S. Jin, A. 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, 2000.

[8] P. Rodriguez, C. Spanner, E. W. Biersack, Analysis of web caching architectures: Hierarchical and distributed caching, IEEE/ACM Transactions on Networking 9 (4) (2001) 401–418.

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

[10] L. Fan, P. Cao, J. Almeida, A. Z. Broder, Summary cache: a scalable wide-area web cache sharing protocol, IEEE/ACM Transactions on Networking 8 (3) (2000) 281–293.

[11] L. Yin, G. Cao, Supporting cooperative caching in ad hoc networks., in: Proceedings of the Conference on Computer Communications (IEEE Infocom), Hong Kong, China, 2004.

[12] S. Sivasubramanian, G. Pierre, M. 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, NY, 2003.

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

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

[15] M. F. Arlitt, C. 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, 1996.

[16] P. Cao, S. Irani, Cost-aware WWW proxy caching algorithms, in: Proceedings of USITS, Monterey, California, United States, 1997.

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

[18] L. Breslau, P. Cao, L. Fan, G. Philips, S. Shenker, Web caching and Zipf-like distributions: Evidence and implications, in: Proceedings of the Conference on Computer Communications (IEEE Infocom), New York, NY, 1999.

[19] K. Psounis, A. Zhu, B. Prabhakar, R. Motwani, Modeling correlations in web traces and implications for designing replacement policies, Computer Networks 45 (4) (2004) 379–398.

[20] A. Mahanti, C. Williamson, D. Eager, Traffic analysis of a web proxy caching hierarchy, IEEE Network 14 (3) (2000) 16–23.

[21] D. Wessels, K. Claffy, ICP and the Squid web cache, IEEE Journal on Selected Areas in Communications 16 (3) (1998) 345–357.

[22] S. Podlipnig, L. Böszörmenyi, A survey of web cache replacement strategies, ACM Computing Surveys 35 (4) (2003) 374–398.

[23] J. Pan, Y. T. Hou, B. Li, An overview DNS-based server selection in content distribution networks, Computer Networks 43 (6) (2003) 695–711.

[24] K. Ogata, Modern control engineering (4th ed.), Prentice-Hall, 2002.

[25] G. F. Franklin, D. J. Powell, A. Emami-Naeini, Feedback Control of Dynamic Systems (5th ed.), Prentice-Hall, 2005.

[26] N. Laoutaris, H. Che, I. Stavrakakis, The LCD interconnection of LRU caches and its analysis, Performance Evaluation 63 (7) (2006) 609–634.

[27] A. Leff, J. L. Wolf, P. S. Yu, Replication algorithms in a remote caching architecture, IEEE Transactions on Parallel and Distributed Systems 4 (11) (1993) 1185–1204.

[28] X. Tang, S. T. Chanson, Adaptive hash routing for a cluster of client-side web proxies, Journal of Parallel and Distributed Computing 64 (10) (2004) 1168–1184.

[29] G. Lin, G. Noubir, R. Rajaraman, Mobility models for ad hoc network simulation, in: Proceedings of the Conference on Computer Communications (IEEE Infocom), Hong Kong, China, 2004.

[30] W. R. Heinzelman, A. P. Chandrakasan, H. Balakrishnan, An application-specific protocol architecture for wireless microsensor networks, IEEE Transactions on Wireless Communications 1 (4) (2002) 660–670.

[31] M. Guirguis, A. Bestavros, I. 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, 2004.