A Feedback Control Approach to Mitigating Mistreatment in Distributed Caching Groups[§]

Georgios Smaragdakis[†], Nikolaos Laoutaris[†‡], Ibrahim Matta[†], Azer Bestavros[†], and Ioannis Stavrakakis[‡]

Abstract. We consider distributed collaborative caching groups where individual members are autonomous and self-aware. Such groups have been emerging in many new overlay and peer-to-peer applications. In a recent work of ours, we considered distributed caching protocols where group members (nodes) cooperate to satisfy requests for information objects either locally or remotely from the group, or otherwise from the origin server. In such setting, we identified the problem of a node being *mistreated*, i.e., its access cost for fetching information objects becoming worse with cooperation than without. We identified two causes of mistreatment: (1) the use of a *common caching* scheme which controls whether a node should *not* rely on other nodes in the group by keeping its own local copy of the object once retrieved from the group; and (2) the *state interaction* that can take place when the miss-request streams from other nodes in the group are allowed to affect the state of the local replacement algorithm. We also showed that both these issues can be addressed by introducing two simple additional parameters that affect the caching behavior (the *reliance* and the *interaction* parameters). In this paper, we argue against a *static* rule-of-thumb policy of setting these parameters since the performance, in terms of average object access cost, depends on a multitude of system parameters (namely, group size, cache sizes, demand skewness, and distances). We then propose a *feedback control approach* to mitigating mistreatment in distributed caching groups. In our approach, a node independently emulates its performance as if it were acting selfishly and then adapts its reliance and interaction parameters in the direction of reducing its measured access cost below its emulated selfish cost. To ensure good convergence and stability properties, we use a (Proportional-Integral-Differential) PID-style controller. Our simulation results show that our controller adapts to the minimal access cost and outperforms static-parameter schemes.

Keywords: Cooperative Caching; Feedback Control; Simulation.

[†] Computer Science Dept, Boston University, Boston, Massachusetts, USA. Email: {gsmaragd, nlaout, matta, best}@cs.bu.edu

[‡] Dept of Informatics and Telecommunications, University of Athens, Athens, Greece. Email: istavrak@di.uoa.gr

[§] A. Bestavros and I. Matta are supported in part by NSF grants EIA-0202067, ITR ANI-0205294, CNS-0524477 and CNS-0520166. I. Stavrakakis is supported in part by EU IST project CASCADAS. N. Laoutaris is supported by a Marie Curie Outgoing International Fellowship of the EU MOIF-CT-2005-007230.

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 grouping of nodes is 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 the various overlay protocols [1, 2] and peer-to-peer (P2P) applications. Two distinctive features of such applications are (1) the fact that 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?*

As part of our recent work [3, 4], we studied this question for 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 maximal 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 [3, 4] 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.

Distributed Selfish Caching: Proactive replication strategies such as those studied in [3] 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 [5] and the estimation of the non-stationary demands (or equivalently, the timescale for a stationarity assumption to hold) [6] 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 [4] 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 [3]. In DSC, we adopted an *object caching* model, whereby a node used demand-driven temporary storage of objects, combined with replacement. We examined the operational characteristics of a DSC group that can give rise to mistreatment problems and argued that simple parametric versions of already established protocols and mechanisms are capable of mitigating these problems. In this work we design a control theoretic framework for regulating the value of these parameters and thus adapting to fluid group conditions (varying group size, node capacities, delays and demand patterns). We thus significantly enhance our results from [4] which introduced these new control parameters but did not prescribe a complete method for regulating them in an adaptive manner.

Organization of the Paper: The rest of the paper is organized as follows. In Section 2 we describe our model of a distributed caching group. In Section 3 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 4. In this section, we also evaluate the performance of our adaptive scheme. Section 5 concludes the paper.

2 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 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, $\boldsymbol{p}_j = \{p_{1j}, \ldots, p_{Nj}\}$, 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.[1] 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 $p_i = K/i^{\alpha}$ (such distributions have been observed in many measured workloads [11, 13]).

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 $t_s > t_r > t_l$[2]; 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 [4]).

3 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 [4]: (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 one, and (2) the mutual *state interaction* between replacement algorithms running on different nodes.

3.1 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

[1] The Independent Reference Model (IRM) [7] is commonly used to characterize cache access patterns [8–11]. The impact of temporal correlations was shown in [6, 12] to be minuscule, especially under typical, Zipf-like object popularity profiles.

[2] 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 of our analysis). Our results can be adapted easily to accommodate arbitrary inter-node distances.

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.

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 of each distinct object in the group (e.g. LRU-SC [14]) 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.

3.2 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, 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 it employs LFU replacement, then its frequency will be increased, and so on with other replacement algorithms [15]. 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 users of u from valuable

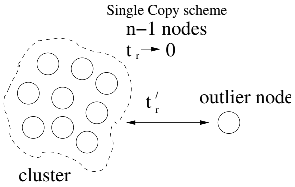

Fig. 1. An example of a group composed of a cluster of $n - 1$ nodes and a unique outlier.

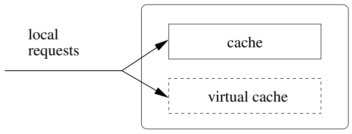

Fig. 2. Block diagram of a node equipped with a virtual cache.

storage space for caching their own objects. This can lead to the mistreatment of u, whose cache is effectively "hijacked" by v.

4 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 [16], not to mention that there are vastly more parameter sets and combinations of schemes that cannot all be investigated exhaustively.

4.1 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 [3], 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. 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 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 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.

4.2 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, 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

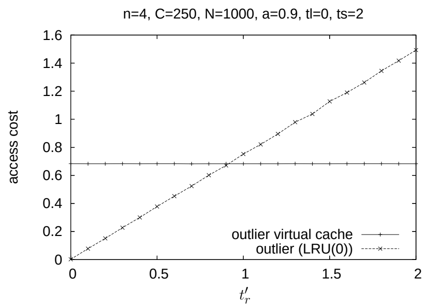

Fig. 3. Simulation results on the effect of the remote access cost t'_r on the access cost of the outlier node under the virtual cache and LRU(0) schemes.

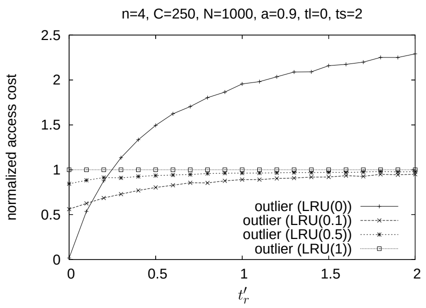

Fig. 4. Simulation results on the effect of the remote access cost t'_r on the normalized (by the virtual cost) access cost of the outlier node under different LRU(q) schemes.

estimate of its threshold cost based on the objects it emulates as present in the virtual cache.[3] 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 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 Fig. 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) [17, 18].

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

[3] 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.

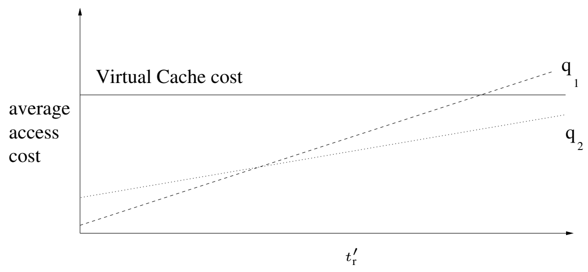

Fig. 5. Representative behavior of average object access cost as a function of the reliance parameter and distance of the outlier from the cluster

Algorithm 1 : mitigation of mistreatment

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

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*; a pseudocode for this process is provided in Algorithm 1. In [19], 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 controller parameters and other system characteristics.

Performance Evaluation: 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($q = 1$)). Thus, we define the following performance metric:

$$minimum \; cost \; reduction \; (\%) \; = \; 100 \cdot \frac{cost_{static} \; - \; cost_{adaptive}}{cost_{static}} \qquad (1)$$

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

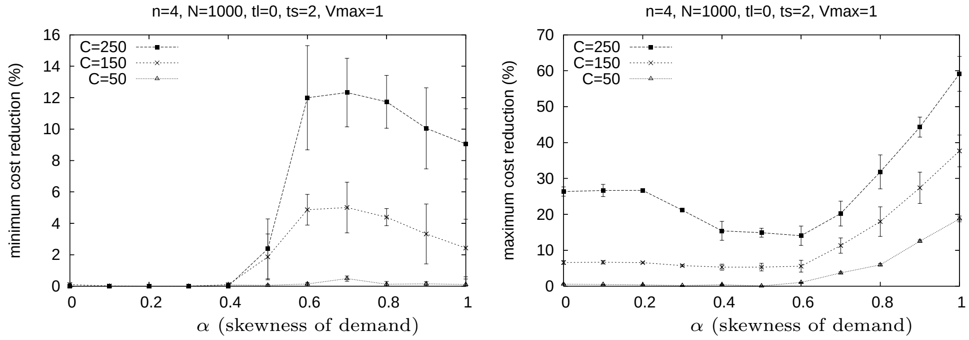

Fig. 6. Simulation results on the cost reduction that is achieved using our adaptive mechanism, (left): The minimum cost reduction, (right): The maximum cost reduction.

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[4] [20]. The motivation for such a scenario comes from a wireless caching application [21]. A detailed description of the design of this experiment is provided in [19]. Figure 6 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, $\alpha = 0$, we get the worst performance of the LRU(1) static scheme, whereas under highly skewed demand, $\alpha = 1$, 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 [11]). 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

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

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.

4.3 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 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).

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.

5 Conclusions

We introduced a feedback control approach to mitigating mistreatment in distributed caching groups. Our approach controls the reliance and interaction parameters (q and p_s) by measuring the node's current access cost under the current scheme and comparing it to the node's emulated selfish access cost. By adapting q and p_s in the direction of moving the current access cost below the selfish cost, our PID-style controller reaps the benefits of cooperation whenever possible under the current system conditions (group size, cache sizes, demand skeweness, distances). Our simulation results confirm the premise of our adaptive (feedback) controller—it effectively adapts to the minimal access cost and even outperforms static controllers (where q and p_s are statically set to zero or one for no- or full-cooperation, respectively) under a wide range of system parameters.

To the best of our knowledge, this is the first attempt to use feedback control to ensure cooperation is always beneficial to users who are autonomous and selfish. Although we considered distributed caching, we believe similar feedback control can be successfully applied to other cooperative applications as well—we intend to investigate this in our future work.

References

1. Byers, J.W., Considine, J., Mitzenmacher, M., Rost, S.: Informed content delivery across adaptive overlay networks. IEEE/ACM Transactions on Networking **12**(5) (2004) 767–780
2. Cohen, E., Shenker, S.: Replication strategies in unstructured peer-to-peer networks. In: Proceedings of ACM SIGCOMM'02 Conference, Pittsburgh, PA, USA (2002)
3. Laoutaris, N., Telelis, O., Zissimopoulos, V., Stavrakakis, I.: Distributed selfish replication. IEEE Transactions on Parallel and Distributed Systems (2005) [accepted for publication].
4. Laoutaris, N., Smaragdakis, G., Bestavros, A., Stavrakakis, I.: Mistreatment in distributed caching groups: Causes and implications. In: Proceedings of IEEE Infocom, Barcelona, Spain (2006)
5. Loukopoulos, T., Lampsas, P., Ahmad, I.: Continuous replica placement schemes in distributed systems. In: Proceedings of the ACM ICS, Boston, MA (2005)
6. Jin, S., Bestavros, A.: Sources and Characteristics of Web Temporal Locality. In: Proceedings of IEEE/ACM Mascots'2000, San Fransisco, CA (2000)
7. Coffman, E.G., Denning, P.J.: Operating systems theory. Prentice-Hall (1973)
8. Arlitt, M.F., Williamson, C.L.: 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) 126–137
9. Cao, P., Irani, S.: Cost-aware WWW proxy caching algorithms. In: Proceedings of USITS. (1997)
10. Young, N.: The k-server dual and loose competitiveness for paging. Algorithmica **11** (1994) 525–541
11. Breslau, L., Cao, P., Fan, L., Philips, G., Shenker, S.: Web caching and Zipf-like distributions: Evidence and implications. In: Proceedings of IEEE Infocom, New York (1999)
12. Psounis, K., Zhu, A., Prabhakar, B., Motwani, R.: Modeling correlations in web traces and implications for designing replacement policies. Computer Networks **45** (2004)
13. Mahanti, A., Williamson, C., Eager, D.: Traffic analysis of a web proxy caching hierarchy. IEEE Network **14**(3) (2000) 16–23
14. Fan, L., Cao, P., Almeida, J., Broder, A.Z.: Summary cache: a scalable wide-area web cache sharing protocol. IEEE/ACM Transactions on Networking **8**(3) (2000) 281–293
15. Podlipnig, S., Böszörmenyi, L.: A survey of web cache replacement strategies. ACM Computing Surveys **35**(4) (2003) 374–398
16. Pan, J., Hou, Y.T., Li, B.: An overview DNS-based server selection in content distribution networks. Computer Networks **43**(6) (2003)
17. Ogata, K.: Modern control engineering (4th ed.). Prentice-Hall, Inc., Upper Saddle River, NJ, USA (2002)
18. Franklin, G.F., Powell, D.J., Emami-Naeini, A.: Feedback Control of Dynamic Systems (5th ed.). Prentice Hall PTR, Upper Saddle River, NJ, USA (2005)
19. Laoutaris, N., Smaragdakis, G., Bestavros, A., Matta, I., Stavrakakis, I.: Distributed Selfish Caching. Technical Report BUCS-TR-2006-003, CS Department, Boston University (2006)
20. Lin, G., Noubir, G., Rajaraman, R.: Mobility models for ad hoc network simulation. In: Proceedings of IEEE Infocom, Hong Kong (2004)
21. Yin, L., Cao, G.: Supporting cooperative caching in ad hoc networks. In: Proceedings of IEEE Infocom, Hong Kong (2004)