Local Utility Aware Content Replication*

Nikolaos Laoutaris, Orestis Telelis, Vassilios Zissimopoulos, and Ioannis Stavrakakis

Department of Informatics and Telecommunications,
University of Athens, 15784 Athens, Greece
{laoutaris,telelis,vassilis,ioannis}@di.uoa.gr

Abstract. A commonly employed abstraction for studying the object placement problem for the purpose of Internet content distribution is that of a distributed replication group. In this work the initial model of distributed replication group of Leff, Wolf, and Yu (IEEE TPDS '93) is extended to the case that individual nodes act selfishly, i.e., cater to the optimization of their individual local utilities. Our main contribution is the derivation of equilibrium object placement strategies that: (a) can guarantee improved local utilities for all nodes concurrently as compared to the corresponding local utilities under greedy local object placement; (b) do not suffer from potential mistreatment problems, inherent to centralized strategies that aim at optimizing the social utility; (c) do not require the existence of complete information at all nodes. We develop a baseline computationally efficient algorithm for obtaining the aforementioned equilibrium strategies and then extend it to improve its performance with respect to fairness. Both algorithms are realizable in practice through a distributed protocol that requires only limited exchange of information.

1 Introduction

A commonly employed abstraction for studying content distribution systems is that of a distributed replication group [1]. Under this abstraction, nodes utilize their storage capacity to replicate information objects in order to make them available to local and remote users. A request issued by a local user and serviced locally (i.e., involving a locally replicated object), is served immediately, thus incurring a minimal cost. Otherwise, the requested object is searched in other nodes of the group and if not found, it is retrieved from the origin server; the access cost, however, increases with the distance. Depending on the particular application, the search for objects at remote nodes may be conducted through query protocols, succinct summaries, DNS redirection or distributed hash tables.

Several placements problems can be defined regarding a distributed replication group. The proxy (or cache, or mirror, or surrogate) placement problem refers to the selection of appropriate physical network locations (routers) for installing content proxies [2, 3]. Another relevant problem is the object placement problem, which refers to the selection of objects for the nodes, under given node locations and capacities [1, 4, 5]. Joint formulations of the above mentioned problems have also appeared, e.g. in [6, 7], where the proxy placement, proxy dimensioning, and object placement problems are combined into a single problem.

* This work and its dissemination efforts have been supported in part by the IST Programs of the European Union under contracts IST-6475 (ACCA) and FP6-506869 (E-NEXT).

All the aforementioned work has focused on the optimization of the so called *social utility* (sum of the individual *local utilities* of the nodes, defined as the delay and bandwidth gains from employing replication). Optimizing the social utility is naturally the objective in environments where a central authority dictates its replication decisions to the nodes. It suits well applications such as web mirroring and CDNs, which are operated centrally by a single authority (this being the content creator or the content distributor). Applications that are run by multiple authorities, such as web caching networks and P2P networks, may also seek to optimize the social utility. This, however, requires some nodes to act in a spirit of voluntaryism, as the optimization of the social utility is often harmful to several local utilities.

Consider as an example a group of nodes that collectively replicate content. If one of the nodes generates the majority of the requests, then a socially optimal (SO) object placement strategy will use the storage capacity of other nodes to replicate objects that do not fit in the over-active node's cache. Consequently, the users of these other nodes will experience a service deterioration as a result of their storage being hijacked by potentially irrelevant objects with regard to their local demand. In fact, such nodes would be better served if they acted independently and employed a greedy local (GL) object placement strategy (i.e., replicated the most popular objects according to the local demand). A similar situation can arise if caching, rather that replication, is in place: remote hits originating from other nodes may evict objects of local interest in an LRU-operated cache that participates in a web caching network. Concern for such exploitation can prevent rational nodes from participating in such groups, and, instead, lead them to operating in isolation in a greedy local manner. Such a behavior, however, is far from being desirable.

Being GL is often ineffective in terms of performance, not only with respect to the social utility, but with respect to the individual local utilities too. For example, when the nodes have similar demand patterns and the inter-node distances are small, then replicating multiple times the same most popular objects, as done by the same repeated GL placement at all the nodes, is highly ineffective. Clearly, all the nodes may gain substantially in this case, if they cooperate and replicate different objects. In fact, it is even possible that an appropriate cooperation of the nodes can lead to a simultaneous improvement of all local utilities as compared to the GL performance. However, nodes cannot recognize such opportunities for mutually beneficial cooperation, since they are generally unaware of the remote demand patterns. On the other hand, they cannot know the impact (bad or good) that the SO object placement strategy may have on their own local utility.

To address the above mentioned deadlock, we use as reference the object replication problem defined by Leff et al. [1], and extend it to account for the existence of selfishly motivated nodes. We use a strategic game in normal form [8] to model the contention between the selfish nodes and set out to identify pure Nash equilibrium object placement strategies (henceforth abbreviated EQ). There are several advantages in employing EQ strategies. First, by their definition, they can guarantee for each and every node of the group that its local utility under EQ will be at least as good as under GL, and possibly better. The first case ("at least as good") precludes mistreatment problems such as those that can arise under the SO placement which cause the nodes to leave the group in pursuit of GL placement. The second case ("possibly better") is the typical one, and points to the fact that implicit cooperation is induced even by selfishly behaving nodes as they attempt to do better than GL. Consequently,

the EQ strategy is in position to break the above mentioned deadlock, as it forbids the mistreatment of any one node, while it also guards against the disintegration of the group, and the poor performance associated with the GL strategy.

Our main result is that such EQ object placement strategies can be obtained by simple distributed algorithms that do not require the existence of complete information at all the nodes. We describe a two-step local search (TSLS) algorithm for this purpose. TSLS requires each node to know only its local demand pattern and the objects selected for replication by remote nodes, but not the remote demand patterns of other nodes (the demand pattern of a node defines explicitly its utility function, thus in the presented framework it is not assumed that nodes know the utility functions of other nodes). Knowing the remote demand patterns requires the transmission of too much information and thus is seldom possible in large distributed replication groups. On the other hand, knowing the objects selected for replication by remote nodes requires the exchange of much less information, which can be reduced further by employing simple encoding schemes such as Bloom filters [9] (see also [10] for real distributed applications/protocols that utilize such information). Thus in terms of the required information, the proposed EQ strategies fit between the GL strategy that requires only local information, and the SO strategy that requires complete information.

A recent work on game theoretic aspects of caching and replication that is relevant to our work is due to Chun et al. [11]. However, this work does not consider storage capacity limits on the nodes and, thus, differs substantially from our approach.

The remainder of the article is structured as follows. Section 2 describes formally the distributed replication group and the distributed selfish replication (DSR) game. Section 3 describes the baseline TSLS object placement algorithm. Section 4 establishes that the TSLS algorithm produces a pure Nash equilibrium object placement strategy for the DSR game. Section 5 is devoted to the presentation of the TSLS(k) algorithm, whose development is largely motivated by our desire to obtain EQ placements without having to resort to the logical ordering of the nodes. Section 6 describes a distributed protocol for implementing the two algorithms. Section 7 presents some numerical examples. Finally, Section 8 concludes the article. The omitted proofs for the presented Propositions, as well as implementation details and more numerical results, can be found in a longer version of this article [12].

2 Definitions

Let o_i, $1 \leq i \leq N$, and v_j, $1 \leq j \leq n$, denote the ith unit-sized object and 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 a storage capacity for C_j unit-sized objects and a demand described by a rate vector r_j over O, $r_j = \{r_{1j}, \ldots, r_{Nj}\}$, where r_{ij} denotes the rate (requests per second) at which node v_j requests object o_i; let also $\rho_j = \sum_{o_i \in O} r_{ij}$ denote the total request rate from v_j. We follow the access cost model defined in [1] and according to which, accessing an object from a node's local cache costs t_l, from a remote node's cache t_r, and from the origin server t_s, with $t_l \leq t_r \leq t_s$ (Fig. 1 depicts the envisaged distributed replication group).

Let $R_j = \{o_i \in O : r_{ij} > 0\}$ denote the request set of node v_j. Let P_j denote the placement of node v_j, that is the set of objects replicated at this node; $P_j \subseteq O$

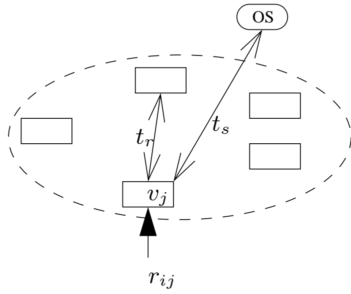

Fig. 1. A distributed replication group.

Step 0 (initialization): $P_j^0 = Greedy_j(\emptyset)$, $1 \leq j \leq n$.
Step 1 (improvement): $P_j^1 = Greedy_j(P_{-j}^{1^-})$, $1 \leq j \leq n$,
where, $P_{-j}^{1^-} = P_1^1 \cup \ldots \cup P_{j-1}^1 \cup P_{j+1}^0 \cup \ldots \cup P_n^0$

Table 1. The TSLS algorithm.

and $|P_j| = C_j$. Let $P = \{P_1, P_2, \ldots, P_n\}$ be referred to as a global placement and let $P_{-j} = P_1 \cup \ldots \cup P_{j-1} \cup P_{j+1} \cup \ldots \cup P_n$ denote the set of objects collectively held by nodes other than v_j under the global placement P. The gain for node v_j under P is defined as follows:

$$G_j(P) = \sum_{o_i \in P_j} r_{ij} \cdot (t_s - t_l) + \sum_{\substack{o_i \notin P_j \\ o_i \in P_{-j}}} r_{ij} \cdot (t_s - t_r) \tag{1}$$

Such a gain definition captures the distance savings when accessing objects from nodes in the group (either the local node or remote ones) instead of the origin server, assumed to be the furthest away.

In the sequel, we define a game that captures the dynamics of distributed object replication under selfishly behaving nodes.

Definition 1 *(DSR game) The distributed selfish replication game is defined by the tuple $\langle V, \{P_j\}, \{G_j\} \rangle$, where:*

- *V is the set of n players, which in this case are the nodes.*
- *$\{P_j\}$ is the set of strategies available to player v_j. As the strategies correspond to placements, player v_j has $\binom{N}{C_j}$ possible strategies.*
- *$\{G_j\}$ is the set of utilities for the individual players. The utility of player v_j under the outcome P, which in this case is a global placement, is $G_j(P)$.*

DSR is a n-player, non-cooperative, non-zerosum game [8]. For this game, we seek equilibrium strategies, and in particular, pure Nash equilibrium strategies.

Definition 2 *(pure Nash equilibrium for DSR) A pure Nash equilibrium for DSR is a global placement P^*, such that for every node $v_j \in V$:*

$$G_j(P^*) \geq G_j((P_1^*, \ldots, P_{j-1}^*, P_j, P_{j+1}^*, \ldots, P_n^*)) \quad \text{for all } P_j \in \{P_j\}$$

That is, under such a placement P^*, nodes cannot modify their individual placements unilaterally and benefit. In the sequel, we develop polynomial time algorithms that, given an instance of the DSR game, can produce several Nash equilibrium placement strategies for it.

3 A Two-Step Local Search Algorithm

In this section we present a two-step local search algorithm that computes a placement for each one of the nodes. In Section 4 we show that these placements correspond to a Nash equilibrium global placement, that is they are EQ strategies. In Section 5, we modify the two-step local search algorithm in order to overcome some of its limitations.

Let P_j^0 and P_j^1 denote the GL placement strategy and the placement strategy identified by TSLS for node v_j, respectively. Let also $Greedy_j(\mathcal{P})$ denote a function that computes the optimal placement for node v_j, given the set $\mathcal{P}$ of distinct objects collectively held by other nodes; we elaborate on this function later on in the section. Table 1 outlines the proposed TSLS algorithm.

At the initialization step (Step 0) nodes compute their GL placements P_j^0. This is done by evaluating $Greedy_j(\emptyset)$ for each v_j, capturing the case in which nodes operate in isolation ($\mathcal{P} = \emptyset$).

At the improvement step (Step 1) nodes observe the placements of other nodes and, based on this information, proceed to improve their own. The order in which nodes take turn in improving their initial placements is determined based on their ids (increasing order). Thus at v_j's turn to improve its initial placement, nodes $v_1, \ldots, v_{j-1}$ have already improved their own, while nodes $v_{j+1}, \ldots, v_n$, have not as yet done so. Node v_j obtains its improved placement P_j^1 by evaluating $Greedy_j(P_{-j}^{1^-})$, where $P_{-j}^{1^-} = P_1^1 \cup \ldots \cup P_{j-1}^1 \cup P_{j+1}^0 \cup \ldots \cup P_n^0$ denotes the set of distinct objects collectively held by other nodes (hence the $-j$ subscript) at the time prior to v_j's turn at Step 1 (hence the 1^- superscript).

We return now to describe how to compute the optimal placement for node v_j, when the set of distinct objects collectively held by other nodes is $\mathcal{P}$; such an optimization is employed twice by the TSLS algorithm: at Step 0 where $\mathcal{P} = \emptyset$, and at Step 1 where $\mathcal{P} = P_{-j}^{1^-}$. To carry it out, one has to select objects according to their relative excess gain, up to the limit set by the storage capacity of the node. Let g_{ij}^k denote the excess gain incurred by node v_j from replicating object o_i at step $k \in \{0,1\}$ of TSLS; g_{ij}^k depends on v_j's demand for o_i and also on whether o_i is replicated elsewhere in the group.

$$g_{ij}^k = \begin{cases} r_{ij} \cdot (t_s - t_l) \text{ ,if} & k = 0 \quad \text{or} \quad k = 1, o_i \notin P_j^0, o_i \notin P_{-j}^{1^-} \\ r_{ij} \cdot (t_r - t_l) \text{ ,if} & k = 1, o_i \notin P_j^0, o_i \in P_{-j}^{1^-} \\ 0 \qquad\qquad\quad \text{ ,if} & k = 1, o_i \in P_j^0 \end{cases} \tag{2}$$

$r_{ij} \cdot (t_s - t_l)$ is the excess gain for v_j from choosing to replicate object o_i that is currently not replicated at any node in V. If o_i is replicated at some other node(s), then v_j's excess gain of replicating it locally is lower, and equal to $r_{ij} \cdot (t_r - t_l)$. Finally, there is no excess gain from choosing to replicate an object that is already replicated locally. Such excess gains are determined by the request frequency for an object, multiplied by the reduction in access cost achieved by fetching the object locally instead from the closest node that currently replicates it (either some other node in V or the origin server).

Finding the optimal placement for v_j given the objects replicated at other nodes ($\mathcal{P}$) amounts to solving a special case of the 0/1 Knapsack problem, in which object

values are given by Eq. (2), object weights are unit, and the Knapsack capacity is equal to an integer value C_j. The optimal solution to this problem is obtained by the function $Greedy_j(\mathcal{P})$. This function first orders the N objects in a decreasing order according to g_{ij}^k ($k = 0$ at Step 0 and 1 at Step 1), and then it selects for replication at v_j the C_j most valuable ones.[1] As the objects are of unit size and the capacity is integral, this greedy solution is guaranteed to be an optimal solution to the aforementioned 0/1 Knapsack problem.

We now proceed to connect the 0/1 Knapsack problem under the g_{ij}^k's, with the gain $G_j(\cdot)$ for v_j under a global placement. We will show that solving the 0/1 Knapsack for v_j under given $P_{-j}^{1^-}$ is equivalent to maximizing $G_j(\cdot)$, given the current placements of nodes other than v_j.

Proposition 1 *The placement $P_j^1 = Greedy_j(P_{-j}^{1^-})$ produced by the TSLS algorithm for node v_j, $1 \leq j \leq n$, satisfies:*

$$G_j(P_1^1,\ldots,P_{j-1}^1,P_j^1,P_{j+1}^0,\ldots,P_n^0) \geq G_j(P_1^1,\ldots,P_{j-1}^1,P_j,P_{j+1}^0,\ldots,P_n^0), \forall P_j \in \{P_j\}.$$

The proof for Proposition 1 and subsequent ones can be found in a longer version of this article [12].

An important observation is that at the improvement step, a node is allowed to retain its initial GL placement, if this is the placement that maximizes its gain given the placements of other nodes. Thus, the final gain of a node will be at least as high as its GL one, irrespectively of the demand characteristics of other nodes; this eliminates the possibility of mistreatment due to the existence of overactive nodes. Regarding the complexity of TSLS, we show in [12] that it is $O(nN \log N)$.

4 Existence of a Pure Nash Equilibrium for DSR

In this section it is shown that the global placement $(P_1^1, P_2^1, \ldots, P_n^1)$ produced by the TSLS algorithm is a pure Nash equilibrium of the distributed replication game. To prove this result we introduce the following additional definitions. Let $E_j^1 = \{o_i \in O : o_i \in P_j^0, o_i \notin P_j^1\}$ denote the eviction set of v_j at Step 1; it is a subset of the initial placement, comprising objects that are evicted in favor of new ones during v_j's turn to improve its initial placement. Similarly, $I_j^1 = \{o_i \in O : o_i \notin P_j^0, o_i \in P_j^1\}$ denotes the insertion set, i.e., the set of new objects that take the place of the objects that belong to E_j^1. At any point of TSLS an object is dubbed a *multiple*, if it is replicated in more than one nodes, and an *unrepresented one* (*represented one*), if there is no (some) node replicating it. Regarding these categories of objects, we can prove the following:

Proposition 2 *(only multiples are evicted) The TSLS algorithm guarantees that the eviction set of node v_j is such that $E_j^1 \subseteq (P_j^0 \cap P_{-j}^{1^-})$.*

Proposition 3 *(only unrepresented ones are inserted) The TSLS algorithm guarantees that the insertion set of node v_j is such that $I_j^1 \subseteq (R_j/(P_j^0 \cup P_{-j}^{1^-}))$.*

[1] Ties are solved arbitrarily at Step 0 and not re-examined at Step 1, i.e., an object that yields the same gain as other objects and is selected at Step 0, is not replaced in favor of any one of these equally valuable objects, later on at Step 1. Thus at Step 1, new objects are selected only if they yield a gain that is strictly higher than that of already selected ones.

The previous two propositions enable us to prove that TSLS finds a pure Nash equilibrium for DSR.

Proposition 4 *The global placement $P^1 = (P_1^1, P_2^1, \ldots, P_n^1)$ produced at the end of Step 1 of the TSLS algorithm is a pure Nash equilibrium for the distributed replication game.*

Assuming that no two g_{ij}^k are the same, then the maximum number of different equilibria that may be identified by the TSLS algorithm is $n!$, i.e., a different equilibrium for each possible ordering (permutation) of the n nodes.

At this point we would like to discuss the subtle difference between the DSR game and the TSLS algorithm, which is just a solution for the DSR game, and not an augmented game that also models the ordering of nodes. The DSR game is a well defined game as it is, i.e., without reference to node ordering, or any other concept utilized by the particular TSLS solution. The ordering of nodes is hence just a device for deriving equilibrium placements, and not a concept of the DSR game itself. The ordering of nodes is not required for defining the DSR game.

The use of a specific ordering of nodes in the improvement step of TSLS is central to the algorithm's ability to find equilibrium placements. It might be the case that algorithms of complete information exist that can find equilibrium placements without requiring the use of such a device. For the case of a distributed replication group, however, it seems that some synchronization mechanism, like the ordering of nodes, is required in order to be able to implement a solution algorithm in a distributed manner without requiring complete information (remote utility functions). In [12] we show that by permitting the nodes to improve their placements without coordination with respect to order, they may never reach a stable placement, but instead loop indefinitely through various transient placements.

5 TSLS(k): Improving on the TSLS Fairness

Consider the case of a homogeneous group, i.e., a group composed of nodes with identical characteristics in terms of capacity, total request rate, and demand distribution. Since such nodes are identical, it is natural to expect that they will be treated equally by a placement strategy. Under the TSLS algorithm, however, the amount of gain that a node receives depends on the node's turn during the improvement step. When the different demand patterns are similar (which is the most interesting case because it allows for higher mutual benefit), then the nodes with the higher turns have an advantage as they can fully utilize the replication decisions of previous nodes [12][2]. This allows for the possibility that small differences in the "merit quantity" of nodes, based on which the turns are decided (to be defined later on in Sect. 6), will translate into large differences in the assigned node turns. This can lead to an unequal treatment of the individual homogeneous nodes.

To address this issue we propose the TSLS(k) algorithm, which is a variation of the baseline TSLS algorithm. Under TSLS(k), *each node may perform only up to k changes during a round of the improvement step*, i.e., evict up to k objects to

[2] In the extended version we also show that a higher turn is not always better under arbitrary demand patterns. This means that the nodes cannot generally act strategically and determine their optimal turn.

insert an equal number of new ones. Note that under TSLS, any number of changes are permitted during the single round of the improvement step. Under TSLS(k), the improvement step might require multiple rounds to reach an equilibrium placement, whereas under TSLS, an equilibrium placement is reached only after a single round. Intuitively, TSLS(k) works in a round-robin fashion based on some node ordering, and allows each node to perform up to k changes of its current placement during a given round, even if the node would like to perform more changes; for additional changes, the node has to wait for subsequent rounds. The effect of this round-robin, k-constrained selection of objects, is that TSLS(k) is at least as and generally more fair than TSLS with respect to the achieved individual gains. By selecting sufficiently small values of k, e.g., $k = 1$, which is an extreme case, *it is possible to almost eliminate the effect of a node's turn on the amount of gain that it receives under the final placement.* Essentially, when k is small, TSLS(k) is able to overcome the inherent limitations with respect to fairness that arise when having to decide a specific node ordering in order to produce an equilibrium placement. For k sufficiently large (approaching the maximum node capacity C^{max}), TSLS(k) reduces to the baseline TSLS. What the TSLS(k) algorithm provides is essentially a tradeoff between an increased fairness and an increased execution time due to the multiple rounds during the improvement step. In [12] we present the full implementation details of TSLS(k) as well as proofs for its convergence in a finite number of rounds upper bounded by $\lceil C^{max}/k \rceil$.

6 A Protocol for Applying the Nash Equilibrium for DSR

In this section we outline a protocol for implementing TSLS or TSLS(k) in a distributed replication group.

6.1 Deciding Turns for the Improvement Step

First, we describe a simple way for deciding turns that can be used with both TSLS and TSLS(k); for TSLS the ordering has an impact on the individual gains, whereas for TSLS(k) under small k, the ordering has a diminishing effect on the gains. Consider an arbitrary labelling of nodes, not related to the ordering in which nodes take turns. Let T_h denote a "merit" quantity associated with node v_h, $1 \leq h \leq n$, based on which v_h's turn is decided. T_h will be defined in such a way that larger ids (turns) will be assigned to nodes having larger values T_h. At the end, a node whose T_h value is the jth largest one, will be re-labelled v_j, thus taking the jth turn. There are many ways to define T_h for a node; among them the following three are of particular interest because they can be naturally associated with a common case in which nodes have similar demand patterns and where a higher turn is better (see Sect. 7):

$$T_h = \begin{cases} C_h & \text{(proportional fairness)} \\ \rho_h & \text{(pro social benefit)} \\ C_h \cdot \rho_h & \text{(hybrid)} \end{cases}$$

The first one caters to *proportional fairness.* It suggests that a node's turn, or equivalently its share of the extra gain produced through the cooperation, be proportional to the amount of resource (storage capacity) that the node contributes. Under such T_h, v_j is the jth largest node.

The second definition is a *socially* inclining one. It favors nodes that generate more requests, as these nodes have the largest influence on the social utility. Under such T_h, v_j is the jth more active node. Notice that following such a criterion for deciding turns is by no means equivalent to the SO strategy. An equilibrium placement under the pro social benefit criterion favors active nodes by allocating them a bigger share of the extra gain produced through the cooperation; this is to say that all other nodes will have (at least) their GL gain intact, whereas under SO, the benefited nodes may cause other nodes to fall below the GL level of gain.

The third expression for T_h is a hybrid way of splitting the gains of the cooperation; it favors nodes that contribute more storage and also produce more requests. Having defined the criterion based on which turns are decided, we move on to defining a protocol for implementing the algorithms and obtaining the equilibrium placement that corresponds to the decided ordering.

6.2 Distributed Protocol

A straightforward centralized implementation would require each node to report r_j and C_j to a central node responsible for executing the TSLS algorithm and sending back the placements P_j. The problem with such a centralized architecture is that it requires transmitting n rate vectors r_j, with each one containing N (object id, request probability) pairs; for large N this can lead to the consumption of too much bandwidth. We, therefore, turn our attention to the development of the following fully distributed protocol which involves three phases:

Phase DT: During this phase, turns are decided.

1. Each node v_h multicasts[3] to the group its value pair (C_h, ρ_h), while listening for, and storing, such pairs from other nodes. The truthfulness of the transmitted pair is crosschecked later on by other nodes during the operation of the distributed group.
2. Having listened to $n-1$ other value pairs, each node may compute its turn j based on a pre-agreed definition of T_h.

Phase 0: In this phase the initial placements according to TSLS are computed and distributed.

1. Each node v_j computes its initial placement P_j^0 and multicasts it to the group. Taking turns is not required at this phase and nodes may transmit their information concurrently.
2. Nodes listen and store the initial placements of other nodes.

Phase 1: In this phase, the initial placements of TSLS are improved.

1. Node v_j waits for its turn (i.e., until v_{j-1} completes its own turn and transmits) and then computes its improved placement P_j^1 as described by TSLS.
2. Following the computation of P_j^1, node v_j transmits E_j^1 and I_j^1 to the group.
3. Nodes $v_{j'}$, $j < j' \leq n$ receive E_j^1 and I_j^1 and use them to produce P_j^1 using also P_j^0, which they have from Phase 0.

[3] Native, or end-system, multicast can be employed.

To implement the TSLS(k) algorithm, Phase 1 needs to be repeated until no node has any more changes to perform. As was mentioned earlier, TSLS(k) provides a tradeoff between the improved fairness and the increased time required to perform multiple rounds at Phase 1. The volume of transmitted information, however, is essentially the same as with the baseline TSLS.

The aforementioned protocol has several advantages. It achieves a degree of parallelism, by permitting nodes to compute their initial placements during Phase 0 independently and concurrently with other nodes. Phase 1 involves a distributed computation too, albeit a sequential one. The major advantage, however, relates to the reduction in the amount of transmitted information as compared to a centralized computation which requires the transmission of $O(nN)$ pairs (object id, request frequency) towards the central point and then $O(\sum_{v_j \in V} C_j)$ object ids sent back from the central point to the nodes carrying the placements P_j^1. Our protocol limits[4] the amount of transmitted information to $O(\sum_{v_j \in V} C_j)$ object ids (initial placements plus eviction and insertion sets). This represents a substantial reduction in the amount of transmitted information, as typically the number of available objects is several orders of magnitude larger than the aggregate storage capacity of the group. Furthermore, lists of object ids can be represented succinctly by employing advanced compression techniques such as Bloom filters [9], whereas rate vectors composed of (object id, request frequency) elements, are much harder to represent and communicate.

With the above protocol, every node is aware of the placements of the other nodes. This information can be used for request routing as well as a means to detect and reveal untruthful nodes, i.e., nodes that try to exploit the group by declaring false placements (see [12] for details).

7 Numerical Examples

In this section we present a simple numerical example for the purpose of demonstrating the operation of TSLS and TSLS(k). Assume there exist two nodes that generate requests following the exact same Zipf-like distribution, i.e., $r_{ij} = \rho_j \cdot K/i^a$, where $K = (\sum_{i'=1}^{N} \frac{1}{i'^a})^{-1}$; the skewness parameter a captures the degree of concentration of requests. The local access cost is, $t_l = 0$, the remote one, $t_r = 1$, and the cost of accessing the origin server, $t_s = 2$; this leads to a hop-count notion of distance. Finally, there are $N = 100$ distinct objects, and each node has a capacity for $C = 40$ objects.

In Table 2 we show the objects replicated under the GL, SO, and EQ replications strategies for fixed $\rho_1 = 1$ and varying ρ_2; here the EQ strategy is produced by the baseline TSLS. The GL strategy selects for each node the first 40 most popular objects, i.e., those with ids in {1:40}, independently of ρ_2. The SO strategy, however, is much different. As the request rate from Node 2 increases, SO uses some of the storage capacity of Node 1 for replicating objects that do not fit in Node 2's cache, thereby depriving Node 1 of valuable storage capacity for its own objects. For $\rho_2 = 10$, Node 1 gets to store only 3 of its most popular objects, while it uses the rest of its

[4] It is worthwhile to emphasize again that although the local placements are multicasted to the group, the amount of information that is gathered at each node is far less than the amount required by centralized algorithms (like the one for the SO placement). Here a node knows only the placements of other nodes, not the entire demand patterns of these nodes.

placement strategy	Node 1 objects	Node 2 objects
GL, $\rho_2 = X$	{1:40}	{1:40}
SO, $\rho_2 = 1$	$\{1:16\} \cup \{41:64\}$	{1:40}
SO, $\rho_2 = 2$	$\{1:12\} \cup \{41:68\}$	{1:40}
SO, $\rho_2 = 3$	$\{1:9\} \cup \{41:71\}$	{1:40}
SO, $\rho_2 = 4$	$\{1:7\} \cup \{41:73\}$	{1:40}
SO, $\rho_2 = 5$	$\{1:6\} \cup \{41:74\}$	{1:40}
SO, $\rho_2 = 6$	$\{1:5\} \cup \{41:75\}$	{1:40}
SO, $\rho_2 = 7$	$\{1:4\} \cup \{41:76\}$	{1:40}
SO, $\rho_2 = 8$	$\{1:4\} \cup \{41:76\}$	{1:40}
SO, $\rho_2 = 9$	$\{1:3\} \cup \{41:77\}$	{1:40}
SO, $\rho_2 = 10$	$\{1:3\} \cup \{41:77\}$	{1:40}
EQ, $\rho_2 = X$	$\{1:23\} \cup \{41:57\}$	{1:40}

Table 2. An example with v_1, v_2 having the same Zipf-like demand pattern with $a = 0.8$. The number of available objects is $N = 100$ and the storage capacity of each node is $C = 40$. Also, $t_l = 0$, $t_r = 1$, $t_s = 2$, $\rho_1 = 1$.

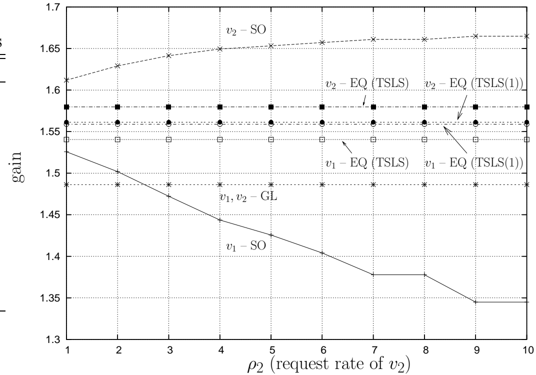

Fig. 2. Individual node gains for the example of Table 2. "v_j – XX" denotes the gain for node v_j under the placement strategy XX.

storage for picking up the next 37 more popular objects for Node 2, starting with the one with id 41. Under the EQ strategy Node 1 (v_1) stores 23 of its most popular objects. Node 2 (v_2) is the second one (i.e., the last one) to improve its placement, and it naturally selects the initial 40 most popular objects.

We now turn our attention to the gain G_j of the two nodes under the various placement strategies (the corresponding access cost can be obtained from the expression $t_s - G_j$). Figure 2 shows that as ρ_2 increases, the gain of v_2 under SO increases as it consumes storage from v_1 for replicating objects according to its preference; v_1's gain under SO decreases rapidly as a result of not being able to replicate locally some of its most popular objects. In fact, for $\rho_2 > 2.5$, v_1's gain becomes worse (lower) that the corresponding one under GL. From this point and onwards, v_1 is being mistreated by the SO strategy and thus has no incentive in participating in it, as it can obviously do better on its own under a GL placement.

By following an EQ strategy, a node's gain is immune to the relative request intensities and this is why the EQ lines are parallel to the x-axis of Fig. 2. v_1's gain under the EQ produced by TSLS is immune to the increasing ρ_2 and strictly higher than its gain under GL. This demonstrates the fact that the EQ strategy avoids the mistreatment problem. Under the EQ produced by TSLS both nodes achieve higher gains than with GL, but it is v_2 that benefits the most, and thus incurs a higher gain than v_1. This owes to the fact that v_2 is the second (last) one to improve its placement and, thus, has an advantage under TSLS. The difference in performance between the two nodes can be eliminated by employing the TSLS(k) algorithm. To show this, Fig. 2 includes the gains of the two nodes under the EQ strategy that is produced by TSLS(1). The corresponding lines almost coincide, which demonstrates the ability of TSLS(k) to be fair and to assign identical gains to v_1 and v_2 (as opposed to TSLS which, in this example, favors v_2).

8 Conclusions

This work has described two algorithms and an efficient distributed protocol for implementing equilibrium object placement strategies in a distributed selfish replication group. Such placement strategies become meaningful when replication nodes cater to their local utilities, as is the case with some content distribution applications that are run under multiple authorities (e.g., P2P, distributed web caching). In such applications, following a socially optimal placement strategy may lead to the mistreatment of some nodes, possibly causing their departure from the group. Our equilibrium strategies on the other hand, guarantee that all nodes are better off participating in the group as opposed to operating in isolation in a greedy local manner. This keeps a distributed group from splitting apart, by creating an excess gain for all (stemming from the cooperation) while forbidding the mistreatment of any one of the nodes. In a longer version of this article, we present several numerical examples for demonstrating the properties of the equilibrium placement strategies.

References

1. Avraham Leff, Joel L. Wolf, and Philip S. Yu, "Replication algorithms in a remote caching architecture," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 11, pp. 1185–1204, Nov. 1993.
2. P. Krishnan, Danny Raz, and Yuval Shavit, "The cache location problem," *IEEE/ACM Transactions on Networking*, vol. 8, no. 5, pp. 568–581, Oct. 2000.
3. Bo Li, Mordecai J. Golin, Giuseppe F. Italiano, Xin Deng, and Kazem Sohraby, "On the optimal placement of web proxies in the internet," in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, New York, Mar. 1999.
4. Madhukar R. Korupolu, C. Greg Plaxton, and Rajmohan Rajaraman, "Placement algorithms for hierarchical cooperative caching," in *Proceedings of the 10th Annual Symposium on Discrete Algorithms (ACM-SIAM SODA)*, 1999, pp. 586 – 595.
5. Thanasis Loukopoulos and Ishfaq Ahmad, "Static and adaptive distributed data replication using genetic algorithms," *Journal of Parallel and Distributed Computing*, vol. 64, no. 11, pp. 1270–1285, Nov. 2004.
6. Nikolaos Laoutaris, Vassilios Zissimopoulos, and Ioannis Stavrakakis, "Joint object placement and node dimensioning for internet content distribution," *Information Processing Letters*, vol. 89, no. 6, pp. 273–279, Mar. 2004.
7. Nikolaos Laoutaris, Vassilios Zissimopoulos, and Ioannis Stavrakakis, "On the optimization of storage capacity allocation for content distribution," *Computer Networks*, vol. 47, no. 3, pp. 409–428, Feb. 2005.
8. Martin J. Osborne and Ariel Rubinstein, *A Course in Game Theory*, MIT Press, 1994.
9. Burton Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, July 1970.
10. Andrei Broder and Michael Mitzenmacher, "Network applications of Bloom filters: A survey," *Internet Mathematics*, 2004, [accepted for publication].
11. Byung-Gon Chun, Kamalika Chaudhuri, Hoeteck Wee, Marco Barreno, Christos H. Papadimitriou, and John Kubiatowicz, "Selfish caching in distributed systems: A game-theoretic analysis," in *Proc. ACM Symposium on Principles of Distributed Computing (ACM PODC)*, Newfoundland, Canada, July 2004.
12. Nikolaos Laoutaris, Orestis Telelis, Vassilios Zissimopoulos, and Ioannis Stavrakakis, "Distributed selfish replication," UoA Technical Report, 2004, available on-line at: `http://www.cnl.di.uoa.gr/~laoutaris/cgame_TECH.pdf`.