Cooperative Replication in Content Networks with Nodes under Churn

Eva Jaho, Ioannis Koukoutsidis, Ioannis Stavrakakis, and Ina Jaho

National & Kapodistrian University of Athens
Dept. Informatics and Telecommunications
Ilissia, 157 84 Athens, Greece
{ejaho,i.koukoutsidis,ioannis,grad0839}@di.uoa.gr

Abstract. In content networks, a replication group refers to a set of nodes that cooperate with each other to retrieve information objects from a distant server. Each node locally replicates a subset of the server objects, and can access objects stored by other nodes at a smaller cost. In a network with autonomous nodes, the problem is to construct efficient distributed algorithms for content replication that decrease the access cost for all nodes. Such a network also has to deal with churn, i.e. random "join" and "leave" events of nodes in the group. Churn induces instability and has a major impact on cooperation efficiency. Given a probability estimate of each node being active that is common knowledge between all nodes, we propose in this paper a distributed churn-aware object placement algorithm. We show that in most cases it has better performance than its churn unaware counterpart, increases fairness and incites all nodes to cooperate.

Keywords: content networks, replication, cooperation, node churn

1 Introduction

In content networks, a *replication group* refers to a set of nodes that cooperate with each other to retrieve information objects (files and other software entities) from a distant server. Nodes in the group usually have a high degree of proximity, and users associated with the nodes usually have common interests. To lower the access cost, each node locally replicates a subset of the server objects, and grants access to these objects to other nodes in the group. The placement of objects in nodes should aim at decreasing the cost for all nodes (or users served by them) to access requested objects. However, in distributed systems where nodes are largely autonomous (e.g. in p2p networks), users may behave selfishly and replicate objects to maximize their own benefit. The problem is to devise a

This work has been supported in part by the following projects funded by the IST FET Program of the European Commission: CASCADAS (Component-ware for Autonomic Situation-aware Communications, and Dynamically Adaptable Services) under contract FP6-027807, BIONETS (BIOlogically-inspired autonomic NETworks and Services) under contract FP6-027748, and NoE CONTENT (IST-384239).

distributed object placement algorithm that ensures that individual nodes would benefit through cooperation. As an additional challenge, such an algorithm must be robust to node misbehavior and general uncertainty conditions.

In this paper, we address this problem in a game-theoretic context. We consider nodes to be the players in the game. Each player implements a *placement strategy* which consists of choosing which objects to replicate locally in its limited storage space. The goal of each player is to minimize the total access cost for all its requested objects. A node may choose not to cooperate with any other node (to act in isolation), in which case the optimal strategy is to replicate a number of most wanted objects, equal to its capacity. This is called the *greedy local* strategy. Naturally, selfish users who are not interested in increasing the social benefit, would want to cooperate with other nodes only if such cooperation could reduce their access cost compared to their incurred cost when all nodes follow the greedy local strategy. If, on the other hand, a node experiences an increase in its access cost, mistreatment occurs as an outcome of cooperation. Ensuring no node mistreatment is key to maintaining a cooperative scheme.

The abstraction of a replication group was initially employed in [1]. We consider that, as in [1], requests for objects are handled in the following manner. First, a user's request is received by the local node the user is associated with. If the requested object is stored locally, it is returned to the requesting user immediately, incurring a minimum 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 can not be located anywhere else in the group, it is retrieved from an origin server – assumed to be outside the group – incurring a maximum access cost. Note that unlike caching, replication refers to storage of objects for a longer term, and no replacement policy (e.g., LRU) is applied for *each* new object request.

In this setting, Laoutaris et al. developed in [2] a distributed algorithm for cooperative object placement that guarantees an access cost reduction for all nodes, when each node knows its local demand pattern and the objects selected for replication by the other nodes in the group. In this algorithm, nodes take turns to replicate objects that minimize their access cost, each node informing all other nodes of its decision. A deficiency of this scheme is that it does not take into account any uncertainty phenomena or node misbehavior. Such phenomena are collectively termed as "node churn" and denote random changes in the set of participating nodes in the group, that may occur due to "join" and "leave" events. For example, in a p2p network, nodes may disconnect themselves from the network as soon as they download their content of interest. It may also occur as a consequence of link failures in networks with wireless links and node mobility. A preliminary study in [3] has confirmed that under such a setting the algorithm in [2] may result in mistreatment for some nodes.

To account for node churn, in this paper we consider a probability estimate of each node to be available for object retrieval in the group, that is common knowledge between all nodes in the group. We propose an object replication strategy wherein each node takes into consideration the probability of other nodes to be available before deciding on its replication strategy. Based on this knowledge, each node makes the number of changes that will give it the greatest access cost reduction. We call this appropriately a *greedy churn-aware* object placement strategy. The performance of the ensuing distributed algorithm is investigated extensively and is shown to mitigate mistreatment, while in most cases reducing the access cost, compared to the churn-unaware and the greedy local strategies.

2 Game Formulation

Hereafter in the paper, nodes that are available shall be said to be "ON", and in the opposite case "OFF". Consider a replication group of N nodes and a set of M objects. Let r_{ij} denote the request rate of node j for object i, where $1 \leq j \leq N$, $1 \leq i \leq M$. All objects are assumed to be unit-sized. Node j has a storage capacity of C_j units and is assumed to be ON with a probability π_j, also called the reliability of a node.

Let P_j denote the placement of node j, defined to be the set of objects replicated at this node. We assume without loss of generality that $|P_j| = C_j$, since a node always has to gain by saving objects locally in its storage. Let $P = \{P_1, P_2, \ldots, P_N\}$ denote the global placement for the replication group and let $P_{-j} = P \setminus P_j$ denote the set that contains the placements of all nodes except for node j.

Assume that the cost for accessing an object from a node's local memory is t_l, from another remote node in the group t_r and from an origin server t_s, with $t_l < t_r < t_s$. These values may denote averages, but are taken the same for all nodes in order to simplify the analysis; in reality, there are differences in these costs for different node pairs, but these are expected to be small, since nodes in a replication group are generally homogeneous devices in geographical proximity.

Given a placement P, the mean total access cost per unit time for node j is given by

$$\begin{aligned}\mathcal{C}_j(P) = &\sum_{i \in P_j} r_{ij} t_l + \sum_{\substack{i \notin P_j, \\ i \notin P_{-j}}} r_{ij} t_s \\ &+ \sum_{\substack{i \notin P_j, \\ i \in P_{-j}}} \left[r_{ij} [t_r (1 - \prod_{\substack{k=1, \\ k \neq j, k: i \in P_k}}^{N} (1 - \pi_k)) + t_s \prod_{\substack{k=1, \\ k \neq j, k: i \in P_k}}^{N} (1 - \pi_k)] \right] .\end{aligned}$$

The scenario under which nodes can cooperatively decide which objects to replicate is as follows. Initially, each node j makes a list of its C_j most popular objects, which constitutes the potential placement of objects in its memory. All nodes are informed of other nodes potential placements. This can be accomplished either by forwarding the information to their neighbors, or in larger networks by uploading these lists to a central database, close to all nodes in the group. In either case, we consider that the cost for disseminating this information is negligible compared to the cost to actually obtain an object. Upon learning the other nodes potential placements, each node can make changes to its list—

exploiting the fact that other nodes may have the same objects, to replicate other ones. We consider that nodes take (possible random) turns to make such changes, and inform other nodes of their decision. We model this as a dynamic game, where players do not form coalitions. Such games, in which there is a certain order of players, and each player's moves have an effect on the utilities of players before or after him, are called *Stackelberg* games, or leaders-followers games (see e.g., [4]). We examine the following replacement strategies.

2.1 Greedy Churn-Unaware Strategy

Under this strategy nodes are assumed to be unaware of the reliability of other nodes; thus they falsely consider other nodes to be always ON when making replacements. It is essentially the strategy proposed in [2] under no node churn. Given the current global placement P, each node considers making replacements so as to minimize its current access cost (hence the term "greedy"). To describe the strategy, we need some further definitions.

For an object $e \in P_j$, define the *eviction loss* $L_{e,j}$ as the increase in access cost if node j would evict the object from its memory. For an object $i \notin P_j$, define the *insertion gain* $G_{i,j}$ as the decrease in access cost if node j would insert the object. In minimizing its current access cost, a node may evict an object from its local memory only if it exists at one or more of the other nodes, and may only insert an object that does not exist in any of the other nodes in the group.

To prove this property, consider that if a node j evicts an object $e \notin P_{-j}$, the eviction cost would be $r_{ej}(t_s - t_l)$. This must be replaced by an object $i \notin P_j$ for which, in the best case that $i \notin P_{-j}$, it would hold that $r_{ij}(t_s - t_l) > r_{ej}(t_s - t_l)$. But there is no such object i, for if there were it should have been in the initial list of most popular objects. Likewise, if node j inserts an object $i \in P_{-j}$, the insertion gain would be $r_{ij}(t_r - t_l)$. Since from the above a node only evicts an object that exists in one or more of the other nodes, i would replace an object e for which $r_{ij}(t_r - t_l) > r_{ej}(t_r - t_l)$. But this also cannot hold, because if $r_{ij} > r_{ej}$ object i would have been inserted in j at the beginning.

With these in mind, for an object e replicated at node j and at one or more of the other nodes in P_{-j}, the eviction cost is

$$L_{e,j} = r_{ej}(t_r - t_l) \ . \qquad (1)$$

For an object i not replicated in any of the nodes in the group, the insertion gain to node j is

$$G_{i,j} = r_{ij}(t_s - t_l) \ . \qquad (2)$$

An object $e \in P_j \cap P_{-j}$ is called an eviction candidate, whereas an object $i \notin P$ an insertion candidate for node j. The set of eviction candidates for node j is denoted as $\mathcal{E}_j$, where $|\mathcal{E}_j| \leq C_j$, and the set of insertion candidates as $\mathcal{I}_j$.

Index the eviction candidates for node j as $e_{1j}, e_{2j}, \ldots, e_{|\mathcal{E}_j|j}$, in order of increasing eviction cost. That is, $L_{e_1,j} \leq L_{e_2,j} \leq \cdots \leq L_{e_{|\mathcal{E}_j|},j}$ (the double subscript j is removed). Accordingly, index the insertion candidates for node j as $i_{1j}, i_{2j}, \ldots, i_{|\mathcal{I}_j|j}$ in order of decreasing insertion gain, i.e., such that $G_{i_1,j} \geq G_{i_2,j} \geq \cdots \geq G_{i_{|\mathcal{I}_j|},j}$ (the double subscript j is likewise removed).

It is evident that to minimize its access cost, node j should make changes in the following way: evict e_{1j} and insert i_{1j}, evict e_{2j} and insert i_{2j}, and so on until the maximum number m_j such that $G_{i_{m_j},j} > L_{e_{m_j},j}$ ($m_j \leq \min(|\mathcal{E}_j|, |\mathcal{I}_j|)$).

For an arbitrary eviction candidate e and insertion candidate i, we call $\Delta G_{(e,i)} \overset{\text{def}}{=} G_{i,j} - L_{e,j}$ the replacement gain for the insertion-eviction pair (e, i). A replacement of object e by object i is also denoted as $e \leftarrow i$.

It was shown in [2] that this strategy results in a Nash equilibrium, i.e., no node can unilaterally change its placement strategy and obtain a benefit, when all other nodes follow the greedy churn-unaware placement strategy. This can be explained from the characteristic property of the strategy mentioned above: since a node only evicts an object that exists in one or more of the other nodes to insert an object unrepresented in the group, each node does its best over all possible moves of subsequent nodes. For a formal proof, interested readers are referred to [2].

Besides showing the Nash equilibrium, the proof leads to a stronger result: in the greedy churn-unaware strategy, no node can gain by playing again at any subsequent epoch in the game. However, as is shown in [3] by calculating the actual expected cost under churn, this algorithm may result in mistreatment for some nodes. Thus, the following object replacement strategy is proposed.

2.2 Greedy Churn-Aware Strategy

In this strategy, nodes have information about the reliability of other nodes, in the form of a common (prior and posterior) distribution of their ON probabilities $\pi_1, \pi_2, \ldots, \pi_N$. This information may be derived and forwarded in a distributed manner, by local interactions between nodes. (For a review of distributed algorithms for deriving reputation measures, readers may refer to [5].) It can also be maintained at a central database that nodes inquire into prior to making their decision. Each node uses this information in replacement decisions and makes a number of changes in its placement that minimize its *expected* access cost.

We can follow the same arguments as in the churn-unaware case to show again that a node may only evict an object that is present in other nodes in the group. However, a node may now insert an object that also exists in one or more of the other nodes in the group (with ON probability smaller that 1).

For an object e replicated at node j, we define the eviction cost as:[1]

$$L_{e,j} = r_{ej}[(t_s - t_l) \prod_{\substack{k=1 \\ (k \neq j, k: e \in P_k)}}^{N} (1 - \pi_k) + (t_r - t_l)(1 - \prod_{\substack{k=1 \\ (k \neq j, k: e \in P_k)}}^{N} (1 - \pi_k))] \,, \tag{3}$$

i.e., it is the increase in the expected access cost if node j would evict the object.

For an object i not replicated at node j, we define the insertion gain as:

$$G_{i,j} = \begin{cases} r_{ij}(t_s - t_l), & \text{if } i \notin P_{-j} \quad \text{(4a)} \\ r_{ij}[(t_s - t_l) \prod\limits_{\substack{k=1 \\ (k \neq j, k: i \in P_k)}}^{N} (1 - \pi_k) + (t_r - t_l)(1 - \prod\limits_{\substack{k=1 \\ (k \neq j, k: i \in P_k)}}^{N} (1 - \pi_k))], & \text{if } i \in P_{-j} \quad \text{(4b)} \end{cases}$$

[1] We maintain the same notation as in Sect. 2.1.

i.e., it is the decrease in the expected access cost if node j would insert the object.

Notice that the right-hand-side of (4b) may be greater than the right-hand-side of (3), which substantiates our previous claim that objects that exist in other nodes in the group may also be inserted.

Considering eviction candidates ordered in increasing average eviction loss, and insertion candidates ordered in decreasing insertion gain, each node j makes a maximum number of replacements m_j with positive average replacement gain, i.e., we have $\Delta G_{(e_k,i_k)} > 0$, for all $k = 1, \ldots, m_j$.

In contrast to the churn-unaware strategy, there is no reason why a node may not benefit by playing again at a subsequent epoch in the game. We therefore formulate a version of the game in which all nodes apply the greedy churn-aware strategy repeatedly for many rounds, until stopping. Although this is not an algorithm requirement, we mostly consider in our study that the same order of play is maintained in each round. For this to be applied in practice, a central coordinating entity is required that, aside from maintaining the database, authorizes each node to access the lists of placements of other nodes only at the specified order.

The algorithm stops when all nodes cannot further improve their placements. In the next section, we show that stopping occurs always.

Remark 1. Note however that we do not study the game as a *repeated game*. If nodes know a priori that they are going to play for a (possibly random) number of rounds, even the churn-unaware strategy may not arrive at a Nash equilibrium.

3 Characteristics of the Greedy Churn-Aware Strategy

We have sought standard Bayesian equilibria by studying the game without communication. (Although communication takes place, we don't consider nodes' placements to be private information they can lie about, and so the game is not treated as a game with communication.)

Unfortunately, the greedy churn-aware strategy may not arrive at an equilibrium. To see this, consider the subgame after player $N-2$ has played. Then, in order to have an equilibrium, player $N-1$ should be *sequentially rational*. In more words, given that the best for the last player N is to follow the greedy churn-aware strategy, would it always be best for $N-1$ to also follow this strategy?

A simple example shows that this may not be the case. Suppose that node $N-1$ evicts an object e, replicated at node N and at one or more of the other nodes $1, ..., N-2$, and replaces it by an object i, such that $G_{i,N-1} > L_{e,N-1}$. Since one or more of the nodes $1, ..., N-1$ still have object e, node N may also at its turn evict it, by inserting an object i' (i' may be equal to i) for which $G_{i',N} > L_{e,N}$. (All of these occurrences are possible since request rates are positive reals.) Since $L_{e,N} > L_{e,N-1}$, we may have that $G_{i,N-1} < L_{e,N}$ and hence it would have been better for player $N-1$ not to make the move $e \leftarrow i$.

Despite this negative result, the strategy does have some good properties, as shown below. Most of these are shown to hold in a homogeneous case in which all nodes have the same request rate for each object and the same storage capacity. Despite the simplicity of this scenario, it is insightful for a real-life system, since nodes in a replication group are likely to have similar interests for objects.

3.1 Mistreatment

As discussed in the Introduction, a rational node is incited to follow the greedy churn-aware strategy if it is not susceptible to mistreatment, i.e. if its incurred access cost when all other nodes follow this strategy is smaller than its incurred cost when all nodes act in isolation. In this subsection, we examine in which cases this constraint can be satisfied for all nodes so that mistreatment is avoided.

First, consider that in the case of two nodes, irrespective of their request rates for objects and their ON probabilities, the greedy churn-aware strategy is always better than the greedy local strategy for both nodes. This happens since the second node only evicts an object that belongs to the first node, and such action does not increase the access cost of the first node.

For more than two nodes, one can easily construct examples with nodes having different request rates and ON probabilities, in which mistreatment occurs against one or more of the nodes. However in the homogeneous case, if less reliable nodes play first, the churn-aware strategy is mistreatment-free under an additional condition stated in the analysis below.

Consider the set of nodes $\mathcal{N} = \{1, 2, \ldots, N\}$. We assume that $\pi_1 \leq \pi_2 \leq ... \leq \pi_N$. Initially, all nodes have the same lists of objects to be placed in their memory. Assume that each node starts making replacements in its list, as dictated by the churn-aware strategy. First notice that because $\pi_j \leq \pi_{j'}$ the cost of each node j to evict an object is smaller than the cost of each node $j' > j$ to evict the same object. Similarly, the gain of each node j to insert an object is greater than the gain of each node $j' > j$ to insert the same object after j. Suppose that each node j makes m_j replacements when playing. It follows that $m_1 \geq m_2 \geq ... \geq m_N$. Consider any subsequence of ℓ nodes, indexed without loss of generality as $1, 2, \ldots, \ell$ where $\pi_1 \leq \pi_2 \leq ... \leq \pi_\ell$ and assume that all nodes in this subsequence make a number of replacements k, where $k \leq m_\ell$. It follows that subsequent nodes have decreasing gains when making the kth replacement.

Denote the objects that node j removed by $e_1, e_2, \ldots, e_{m_j}$, indexed in increasing eviction cost. For its kth replacement, the replacement gain of node j is $G_{i_k,j} - L_{e_k,j}$. Its access cost will be increased if subsequent nodes at their kth replacement remove one or more of the objects $e_1, e_2, \ldots, e_{m_j}$ that node j removed. (Its access cost is not affected if subsequent nodes insert the same objects, and is decreased if they insert some different objects.) We can show however, that under a certain condition it remains lower than its cost before making this replacement.

We generally denote by $L^{(j)}_{e_k,j+n}$ the average increase in access cost incurred to node j by node $j+n$ evicting object e_k. If node $j+n$ is the first node that evicts object e_k after j and if $L_{e_k,j+n}$ is the eviction cost of node $j+n$, it holds that $L_{e_k,j+n} = L_{e_k,j} + L^{(j)}_{e_k,j+n}$.

Let the number of nodes that make a kth replacement after node j has played be n_k, and consider the subsequence $j+1, j+2, \ldots, j+n_k$. The object evicted by node $j+1, j+2, \ldots, j+n_k$ at the kth replacement is denoted by $e_{k(j+1)}, e_{k(j+2)}, \ldots, e_{k(j+n_k)}$, respectively.

Without loss of generality, we assume that at their kth replacement, *all* nodes in this subsequence evict objects evicted by node j (otherwise their induced cost to j is zero and we need not include these nodes). Due to decreasing gains of subsequent nodes for making the kth replacement, the objects evicted by a node $j+\ell$ are a subset of the objects evicted by j and index $k(j+\ell) \geq k \; \forall \ell = 1, \ldots, n_k$. We now impose the critical condition that the objects evicted by node $j+2$ are also evicted by node $j+1$, for each pair $(j+2, j+1)$ in the subsequence.

The replacement gain of node j after these evictions becomes at least (since it may be increased by different inserted objects of other nodes)

$$G_{i_k,j} - L_{e_k,j} - L^{(j)}_{e_{k(j+1)},j+1} - L^{(j)}_{e_{k(j+2)},j+2} - \cdots - L^{(j)}_{e_{k(j+n_k)},j+n_k} \tag{5a}$$

$$\geq G_{i_k,j} - L_{e_{k(j+1)},j} - L^{(j)}_{e_{k(j+1)},j+1} - L^{(j)}_{e_{k(j+2)},j+2} - \cdots - L^{(j)}_{e_{k(j+n_k)},j+n_k} \tag{5b}$$

$$= G_{i_k,j} - L_{e_{k(j+1)},j+1} - L^{(j+1)}_{e_{k(j+2)},j+2} - \cdots - L^{(j)}_{e_{k(j+n_k)},j+n_k} \tag{5c}$$

$$\geq \cdots$$

$$\geq G_{i_k,j} - L_{e_{k(j+n_k)},j+n_k} > 0 \; . \tag{5d}$$

(5c) follows from (5b) because it holds that $L_{e_{k(j+1)},j+1} = L_{e_{k(j+1)},j} + L^{(j)}_{e_{k(j+1)},j+1}$ and $L^{(j+1)}_{e_{k(j+2)},j+2} = L^{(j)}_{e_{k(j+2)},j+2}$ (since both j, $j+1$ have evicted object $e_{k(j+2)}$, the same cost is induced to both by $j+2$ evicting it.) Finally, the last implication holds because $L_{e_{k(j+n_k)},j+n_k} < G_{i_{k(j+n_k)},j+n_k} < G_{i_k,j}$. ($i_{k(j+n_k)}$ denotes the object inserted by node $j+n_k$.)

Since moreover node j makes a higher number of replacements than its subsequent nodes ($m_j \geq m_{j+1} \geq \cdots \geq m_N$), the total gain of node j by making its m_j moves can never become smaller than zero. Therefore node j incurs a smaller access cost under the final placement of all nodes compared to its greedy local placement. Under the same order of play and the same conditions, this holds for any number of rounds in the game.

Remark 2. An important consequence of the previous analysis is that it reveals a mistreatment-free strategy. Consider the greedy churn-aware strategy, with the additional condition that a node only evicts an object if it was also evicted by all previous nodes, or not evicted by any of the previous nodes. In the homogeneous case where less reliable nodes play first this strategy is mistreatment-free.

3.2 Potential Gain of a Node by Playing Again

A useful metric in our model is the potential gain of a node by playing again after a number of steps in the game. This gain is defined as a node's reduction in mean access cost if it were to play again after a number of steps in the game. Intuitively, a good strategy yields small such gains to all nodes.

The analysis below shows that in the homogeneous case studied in the previous subsection, assuming now that more reliable nodes play first ($\pi_1 \geq \pi_2 \geq ... \geq \pi_N$) a node does not have a benefit by playing again at a subsequent epoch, in certain game instances.

A node may have a benefit by playing again by reinserting an object evicted by itself and subsequent nodes, or by evicting an object inserted by subsequent nodes. Consider that at a subsequent epoch after node j has played, a number of players evict an object e that was also evicted by node j and another number of players insert an object i also inserted by j. Say the last node that evicted object e was $j + n_1$, and the last node that inserted object i was $j + n_2$. We can show that if $n_1 = n_2 \; \forall \; (e, i)$, then $L_{i,j} > G_{e,j}$ and node j does not obtain a benefit by playing again. Consider that the insertion gain of node j to reinsert object e equals the eviction cost of the last node after j that evicted the object. So $G_{e,j} = L_{e,j+n_1}$. If the last node that inserted object i was $j + n_2$, since $\pi_j \geq \pi_{j+n_2}$, we also have that $L_{i,j} \geq G_{i,j+n_2}$. If $n_1 = n_2$, then since we necessarily have $G_{i,j+n_1} > L_{e,j+n_1}$ it follows that $L_{i,j} > G_{e,j}$.

If $n_1 \gtrless n_2$ then depending on request rates and the probabilities of nodes to be ON, we may have $L_{i,j} < G_{e,j}$ and node j may obtain a benefit by playing again. This is also the case if some other ordering is applied. However potential gains are usually small, as confirmed by the results in Sect. 4.

3.3 Finiteness of the Algorithm with Multiple Rounds

The following theorem holds:

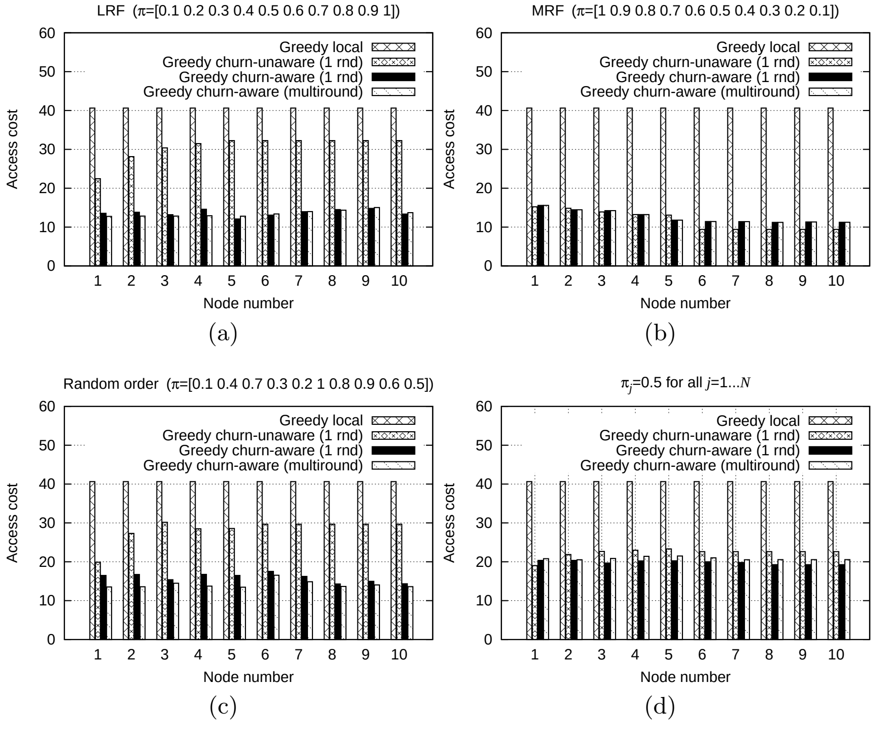
Theorem 1. *The greedy churn-aware algorithm ends in a finite number of rounds, irrespective of the order of play in each round.*

Proof. At each step of the game, each player may evict an object owned by a certain number of other nodes, to insert another object owned by either a) a smaller number of nodes (or none), or b) a larger number of nodes, with smaller probability that at least one of them is ON. Thus, there will come a time when nodes do not have objects in common or no further replacements are possible. □

In practice, the algorithm finishes very quickly. In all test cases that follow, the algorithm finished in 1 to 5 rounds.

4 Numerical Evaluation

In this section we present some numerical examples to show how the different object replacement strategies perform in a replication group under node churn. We are interested in analyzing cases where nodes have similar preferences for objects, so that mutual benefits emerge by cooperation. Here we consider the simplest case where nodes have the exact same request rates. Our major conclusions are similar when we consider different request rates, or when each node has a different preference order for objects. Request rates are drawn from a



(a) (b) (c) (d)

Fig. 1. Access cost for different placement strategies with different node orderings

Zipf distribution with exponent s. The probability of the object with rank k, $k = 1, ..., M$, is

$$f(k; s, M) = \frac{1/k^s}{\sum_{m=1}^{M} 1/m^s} \; .$$

We consider $N = 10$ nodes. Each of the N nodes has a capacity of 10 objects in its local storage, and there exists a total of $M = 50$ objects. We set $t_l = 1$, $t_r = 10$ and $t_s = 100$ cost units. Fittings based on real traces in [6] have shown the value of the exponent s of the Zipf distribution to lie between $0.8 - 0.9$. Here we take $s = 0.9$.

We examine the following orderings of players based on their reliability: (1) Least Reliable First (LRF) where nodes play in increasing order of their ON probabilities, (2) More Reliable First (MRF) where nodes play in decreasing order of ON probabilities, (3) Random order, in which nodes take a selected random order to play.

The (mean) access costs of all nodes under the different orderings are shown in Fig. 1. The vector $\pi = [\pi_1, \pi_2, \ldots, \pi_N]$ of ON probabilities of nodes for each examined order of play is shown in the title of each sub-figure. We also show a case where nodes have the same probability to be ON, $\pi_j = 0.5$ for all $j = 1, ..., N$ (Fig. 1(d)). No mistreatment is shown to occur in these graphs; intuitively, this

Table 1. Potential gains of nodes by playing again after 1 round

	Potential gain									
Node	1	2	3	4	5	6	7	8	9	10
LRF	1.52	1.61	0.83	1.93	0	0	0	0	0	0
MRF	0	0	0	0	0	0	0	0	0	0
Random order	2.81	2.73	0	2.94	2.69	0	0	0	0	0
$\pi_j = 0.5\ \forall j = 1, \ldots, N$	0.35	0.40	0	0	0	0	0	0	0	0

happens here because there exists a number of nodes with relatively high reliability values. (For an example of mistreatment under the churn-unaware strategy, readers are referred to [3].)

It is demonstrated that all group replication strategies perform better than the greedy local. Comparing the churn-unaware and churn-aware greedy strategies, we observe that a significant improvement occurs when the ordering is LRF (Fig. 1(a)), while similar costs are produced for an MRF order (Fig. 1(b)). To understand this, notice that under the LRF order, the churn-unaware strategy results in many high request rate objects being stored at less reliable nodes; more reliable nodes falsely trust high request rate objects to be accessible from the previous nodes, while it would be better to have them stored locally. Such erroneous placements occur less frequently when the order is MRF. The churn-aware strategy can also yield a significant improvement when there is no pre-specified order of play, but nodes take random turns (Fig. 1(c)). On the contrary, a slight improvement is observed when all nodes have the same probability of being ON.

An important observation is that there is no significant access cost reduction by repeating the churn-aware algorithm for multiple rounds. This yields extra benefit only to some nodes; depending on the order of play, some nodes may benefit from the extra rounds, which usually occurs at the expense of others.

We also examine the potential gain of each node by playing again after one round of the churn-aware algorithm, under the various orderings (if there can be no improvement, the potential gain is zero). Results are shown in Table 1. In the MRF case, the algorithms stops in the first round and thus all potential gains are zero. The analysis in Sect. 3.2 partially justifies this result. Generally in the other cases, only the first few nodes attain a benefit by playing again, since they are more likely to be farther from an optimal placement. In any case, benefits are small when compared to the access cost values (see Fig. 1), and hence nodes have a small incentive to play again.

Finally, it is worth discussing the fairness aspects of different orderings. In this way, if some mediator (e.g., a system administrator) could enforce a certain order, we would like to know the fairest one. It is fair that more reliable nodes obtain a greater benefit by participating in the game, than the less reliable ones. We established previously that when all nodes follow the churn-unaware strategy, LRF is an unfair order because it leads more reliable nodes to erroneous placements. Instead, MRF should be followed. Under the churn-aware strategy, such fairness problems are mitigated, as all nodes behave in a rational manner. In the results, both the LRF and MRF orderings tend to give similar benefits.

5 Conclusions

This paper proposed a distributed algorithm for object replication in a content network, in which each node selfishly decides which objects to replicate from a distant server and announces its decision to other nodes, thus allowing all nodes to have a gain by a limited cooperation.

The algorithms accounts for selfish behavior of autonomous nodes, as well as churn phenomena, i.e., nodes that are not always available or operational. It needs minimal information in order to be applied, consisting of the placements of other nodes in the replication group and a probability estimate of their reliability. This information can be forwarded in a distributed manner, or maintained in a central database in the network that nodes query prior to making their decision. While it may not arrive at an equilibrium, we have shown that it possesses good properties: in the majority of cases, it decreases the access cost collectively for all nodes, compared to the greedy local strategy and churn-unaware strategies. Although mistreatment problems may still occur, these are alleviated if all nodes follow the common strategy. Furthermore, if nodes having the same request rates for objects play according to an LRF order, the strategy is near mistreatment-free and provides for a fair treatment of nodes according to their reliability.

A challenging line for future work is the study of conditions under which the strategy arrives at an approximate Nash equilibrium. Based on the analysis in Sect. 3.1, a first conjecture is that an approximate equilibrium exists in the homogeneous case, under an LRF order. A possible extension of this work is to consider communication capabilities of nodes, and the possibility of each one to lie about its placement. Then the game should be studied as a game with communication, and an additional requirement of a good strategy is that nodes do not have an incentive to lie. Finally, the behavior of the algorithm in the cases where there exist objects of different sizes or where each node has a subjective estimate for the reliability of other nodes should be examined.

References

1. Leff, A., Wolff, J., Yu, P.: Replication algorithms in a remote caching architecture. IEEE Trans. Par. and Distr. Systems **4**(11) (November 1993) 1185–1204
2. Laoutaris, N., Telelis, O., Zissimopoulos, V., Stavrakakis, I.: Distributed selfish replication. IEEE Trans. Par. Distr. Systems **17**(12) (December 2006) 1401–1413
3. Jaho, E., Jaho, I., Stavrakakis, I.: Distributed selfish replication under node churn. In: Proc. Med-Hoc-Net. (2007) Poster session.
4. Basar, T., Olsder, G.: Dynamic noncooperative game theory. 2nd edn. Society for Industrial and Applied Mathematics (SIAM) (1999)
5. Avrachenkov, K., Nemirovsky, D., Pham, K.S.: A survey on distributed approaches to graph based reputation measures. In: Proc. of International Workshop on Tools for solving Structured Markov Chains (SMCtools). (2007)
6. Breslau, L., Cao, P., Fan, L., Phillips, G., Shenker, S.: Web caching and Zipf-like distributions: Evidence and implications. In: Proc. IEEE INFOCOM. (1999)