A Game Theoretic Approach to the Formation of Clustered Overlay Networks (Extended Version)

Georgia Koloniari and Evaggelia Pitoura

Computer Science Department, University of Ioannina, Greece

{kgeorgia,pitoura}@cs.uoi.gr

Abstract

In many large-scale content sharing applications, participants or peers are grouped together forming clusters based on their content or interests. In this paper, we deal with the maintenance of such clusters in the presence of updates. We model the evolution of the system as a game, where peers determine their cluster membership based on a utility function of the query recall. Peers are guided either by selfish or altruistic motives: selfish peers aim at improving the recall of their own queries, whereas altruistic peers aim at improving the recall of the queries of other peers. We study the evolution of such clusters both theoretically and experimentally under a variety of conditions. We show that, in general, local decisions made independently by each peer enable the system to adapt to changes and maintain the overall recall of the query workload.

1 Introduction

Large content sharing communities such as those present in social networking applications and peer-to-peer file sharing systems have become highly popular. An issue central in such settings is efficiently locating sites, users or nodes, collectively called *peers*, that maintain data relevant to a query and retrieving as many results as possible, i.e. increasing the query recall with a small communication cost. This is an intricate task given the unprecedented scale and dynamicity of such communities.

One of the proposed solutions for locating interesting content is to create groups of peers with similar content or interests. Grouping is achieved by adding links between similar peers thus forming *clustered overlay* networks [3, 5, 14, 19, 11, 8, 4, 6]. Peers in each cluster are highly connected to each other, so that routing queries inside a cluster is very efficient. Clustering has been widely used in databases to reduce I/O costs by placing data items with similar properties together in the disk, since it is expected that similar data is accessed together. The motivation for exploiting clustering in a distributed data sharing system is that through routing queries within each cluster, the performance of locating and accessing the requested content can be greatly improved. This is because, once the appropriate cluster for a query is identified, the peers in the cluster possess relevant content that can be exploited to evaluate and refine the query efficiently.

In fact, measurements from the deployments of many content sharing systems have shown that such clusters are implicit in many distributed interactions. In particular, traces of file-sharing peer-to-peer systems have indicated that peers exhibit the property of interest-based locality, that is, if a peer holds content satisfying some query of another peer, then it is most likely that it also maintains additional content of interest to this other peer [17, 10]. Thus, placing the two peers in the same cluster would increase the recall of their queries. The formation of implicit clusters centered around topics described by common keywords has been observed in the blogosphere [2] as well. In measurements of popular on-line social networks [15], it was also observed that the networks structure is that of a clustered overlay where users form clusters based on common interests, social affiliations or the wish to exploit their shared content. The clusters are loosely connected with each other through a strong central component.

Most previous work mainly focused on the discovery and construction of clusters and ignored the maintenance of the clustered overlay, which is needed for coping with the dynamic nature of the peers. Peers, which join or leave the system constantly and change their content and query workload frequently, may render the original clustered overlay inappropriate under the current system conditions. One solution to the problem is to re-apply the clustering procedure that was used to form the original overlay from scratch taking into account the updated state. However, this incurs large communication costs. It also requires global knowledge about the system state that compromises peer autonomy.

In this paper, we propose a novel approach to clustering by modeling the problem of cluster formation as

a strategic game in which the players are the peers. Each peer/player determines its cluster memberships individually so as to minimize a utility function that depends on the membership cost entailed in belonging to a cluster and the cost of evaluating its query workload at remote clusters. Game-theoretic models have been previously proposed for creating overlays based on the connection cost and radius of the network graph [7, 13, 16]. The originality of our approach lies on the fact that we focus on queries and aim at increasing their recall.

We model both selfish and altruistic behavior of peers as demonstrated in real content-sharing systems by proposing appropriate utility functions. We also introduce global system quality criteria to measure the performance of the system as a whole. In addition, we propose appropriate relocation policies for selfish and altruistic peers and a cluster reformulation protocol that implements the game.

We study both theoretically and experimentally the evolution of clusters under the individual actions of each peer. Our experimental results show that our policies help in coping with the changes in the overlay configuration without compromising its quality. In particular, given the usual assumption for clustering that the underlying data share some similar properties, our policies converge to well formed clusters for most initial system configurations.

The rest of this paper is organized as follows. In Section 2, we present the cluster formation problem as a game and define the utility functions for selfish and altruistic peers along with the corresponding global quality criteria. We also study the stability of the system. In Section 3, we introduce the policies for peer relocation and describe a practical instantiation of the game. Section 4 presents our experimental evaluation and Section 5 refers to related research. We conclude in Section 6.

2 Recall-Based Clustering

We consider a distributed system consisting of highly dynamic nodes (peers) that share content. Usually, such distributed content sharing systems need to scale up to a large number of peers (Internet-scale). Thus, a peer is unable to know and directly communicate with all other peers in the system. Instead, it establishes logical links with only a few, creating logical overlay networks on top of the physical one. Queries are forwarded using these links from peer to peer in the overlay to locate peers that hold any content of interest.

We use P to denote the current set of peers. We do not assume any specific model for the data items shared by the peers, but adopt a rather generic approach where each data item is described by a set of attributes (e.g. keywords for text documents). We denote the number of results for a query q (e.g. keyword query) against the documents of peer p_i as *result(q, p_i)*.

Let Q be the list of all queries in the system. Note that a query q may appear more than once in Q. Let $num(Q)$ be the number of all queries in Q and $num(q, Q)$ be the number of appearances of query q in Q. We characterize the importance of a peer p_i in the evaluation of a query q in Q based on the results that p_i offers for q with regards to the total number of available results (i.e. the recall achieved when q is evaluated solely on p_i). Specifically:

$$r(q, p_i) = \frac{result(q, p_i)}{\sum_{p_k \in P} result(q, p_k)}.$$

We also define as *local workload* of peer p_i, $Q(p_i)$, the list of queries that were issued by peer p_i. Again, $num(Q(p_i))$ stands for the number of all queries in $Q(p_i)$ and $num(q, Q(p_i))$ for the number of appearances of query q in $Q(p_i)$.

The efficiency of query evaluation depends heavily on the structure of the overlay network formed by the peers since that is used to forward queries to interesting content. Thus, efficient overlays improving system performance are required. Typical examples of such overlays are structured overlays such as Chord and CAN with strict topologies in which locating any peer (i.e. any data item) takes $O(log|P|)$ and clustered overlays in which peers are more loosely structured. In clustered overlay networks, peers form sets, called *clusters*. The main motivation for clustering is that inside each cluster, the evaluation of a query is cost efficient. In such overlays, the peers within each cluster are usually highly connected and it is very efficient for each of them to communicate with any other member of the same cluster. Figure 1 shows 8 peers forming a cluster following a fully connected topology (Fig. 1(a)) where each peer can reach any other with one hop, while (Fig. 1(b)) a structured topology in which finding any peer takes $log8$ hops.

Clustering as a Game: We model the problem of cluster formulation as a strategic game. Each peer represents a player in the game and its strategy is defined by the set of clusters it joins. In particular, each peer p_i chooses which clusters to join from the set of C_{max} clusters in the system, $C = \{c_1, c_2, \ldots, c_{C_{max}}\}$, thus, defining its strategy $s_i \subseteq C$. We can describe any cluster configuration by the set of strategies $S = \{s_1, s_2, \ldots, s_{|P|}\}$ that the peers in P have deployed, since from this set we can derive the set of peers belonging to each cluster in C. In this paper, we constraint C_{max} to be equal to $|P|$, i.e. it cannot exceed the number of peers, and assume that some clusters may be empty if needed.

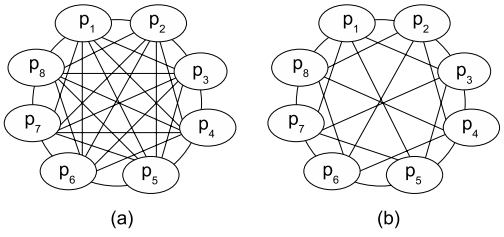

Figure 1: Examples of intra cluster topologies

The goal of the game is for each player (peer) to minimize or maximize a *utility* function. We discern between two types of peers, *selfish* and *altruistic* ones, and define a corresponding utility function for each type.

2.1 Individual Peer Measures

A selfish peer is interested in increasing the recall of its local query workload by joining those clusters whose peers would increase the recall of its local workload the most. Specifically, let $P(s_i)$ be the set of peers belonging to any cluster $c \in s_i$. The gain for a peer p for choosing a strategy s_i is the recall of its local workload achieved by evaluating its queries in the peers $P(s_i)$. Stated differently, the cost associated with a strategy s_i for a peer p_i is the cost (recall) for obtaining query results from peers located in clusters that do not belong to s_i, that is, for peers not in $P(s_i)$.

Clearly, this recall-based cost is minimized, if a peer joins all C_{max} clusters in the system. However, participation in a cluster imposes communication and processing costs. Such costs depend on the size and the topology of the cluster. The larger the size of the cluster, the higher the cost of joining, leaving and maintaining the cluster. Furthermore, a highly connected topology, where each peer maintains links to a large number of other peers, increases the cluster membership cost. To capture this, the cluster membership cost is defined as a monotonically increasing function θ of the number of peers belonging to the cluster, i.e. as a function of the cluster size $|c|$. This function depends on the cluster topology, for instance, when all peers are connected to each other, θ may be linear (Fig. 1(a)), whereas in the case of structured overlays, θ may be logarithmic (Fig. 1(b)).

Definition 1 (Individual Peer Cost) *In a cluster configuration S, the individual cost for a selfish peer p_i for choosing strategy s_i is defined as:*

$$pcost(p_i, S) = \alpha \sum_{c_k \in s_i} \frac{\theta(|c_k|)}{|P|} + \sum_{q in Q(p_i)} \frac{num(q, Q(p_i))}{num(Q(p_i))} \sum_{p_j \notin P(s_i)} r(q, p_j)$$

The first term expresses the cost for cluster membership and the second one the cost (in terms of recall) for obtaining results from peers outside the selected clusters, that is, the average result loss from not participating in all clusters. Parameter α ($\alpha \geq 0$) determines the extent of influence of the cluster membership cost in cluster formation. From a system perspective, parameter α characterizes the ratio between updates and queries in the system. For a given θ, a large value of α means that updates in the system are rather frequent and therefore the cost for cluster maintenance is high, while a small value shows that query evaluation efficiency is more important for determining the overall system performance. Finally, factor $1/|P|$ is used for normalizing the cluster membership cost.

We consider which strategy maximizes the value of each of the two terms contributing into the individual cost function for a peer p_i, i.e., the membership cost and the recall loss. If we consider a scenario in which each peer can join only one cluster, we observe that since θ is a monotonically increasing function, it takes its largest value when all peers form a single cluster of size $|P|$. In this case, the term based on recall takes its lowest value equal to 0. That is, since all peers belong to p_i's cluster, there is no recall loss. In contrast, the recall loss takes its largest value when p_i forms a cluster by its own. If any peer joins its cluster, the loss will either decrease if the new peer has data that p_i is interested in, or remain the same, if it does not, but it can never increase. Evaluating however the membership cost in this case, we observe that it is minimized. Thus, we observe that the two terms tend to guide the peer towards selecting opposing strategies.

Note that in the proposed model, we focus on recall and abstract the cost of processing queries both within each cluster and across clusters. There is a large body of research on routing queries among interconnected peers. By concentrating on recall, we aim at capturing the basic mechanisms underlying clustering independently of any query routing specifics.

While minimizing the individual cost function is appropriate for modeling the behavior of selfish peers that try to maximize the recall of their own queries, we also want to model the behavior of altruistic peers. Altruistic peers are not concerned about their own queries, instead they are interested in offering to other peers. Therefore, we define the corresponding utility function, called *individual peer contribution* (*pcontr*) that an altruistic peer p_i aims at maximizing based on how much p_i improves the recall of the other peers that belong to the clusters of its strategy. Thus, analogously to Def. 1, the individual contribution is defined as follows:

Definition 2 (Individual Peer Contribution) *In a cluster configuration S, the individual contribution of*

an altruistic peer p_i for choosing strategy s_i is defined as:

$$pcontr(p_i, S) = \frac{1}{|P|} \sum_{p_j \in P(s_i)} \sum_{q \in Q(p_j)} \frac{num(q, Q(p_j))}{num(Q(p_j))} r(q, p_i) - \frac{\alpha}{|P|^2} \sum_{c_k \in s_i} |c_k| \theta(|c_k|)$$

The first term of the sum measures the contribution of peer p_i to the peers in the clusters of its strategy, while the second term measures the membership cost these peers pay if p_i joins their cluster. Similarly to the individual cost, the membership cost also takes its lowest value when the peer forms a cluster by its own and its largest when all peers form a single cluster (if we consider that each peer joins only a single cluster), whereas the recall it contributes to other peers takes its largest value in the single cluster and its lowest when it forms its own cluster. Individual contribution is defined from the perspective of each beneficiary peer, that is, the queries frequencies are defined based on their relative frequencies per such peer. While *pcost* measures the cost p_i pays for its query workload and membership to clusters in s_i, *pcontr* is a positive measure showing what other peers gain when p_i chooses strategy s_i.

In general, there is no direct relation between the two measures. We observe the relationship between the membership cost in the two terms. Let us first consider the simple case where each peer p_i joins one cluster c_k. Then it holds that the membership cost in the individual contribution is equal to $|c_k|/|P|$ times the corresponding membership cost in the individual cost. Similarly, when we consider a cluster configuration in which all the clusters have the same size $|c|$, and a peer that has joined k clusters we have for the membership cost in the individual contribution: $\alpha/|P|^2\, k\, |c|\theta(|c|)$, while the same cost in the individual cost is:$\alpha/|P|\, k, \theta(|c|)$, that is the first cost is $|c|/|P|$ times the second.

Besides the pure selfish and the pure altruistic behavior, hybrid behavior can be captured by the following cost function:

$$hpcost(p_i, S) = d\ pcost(p_i, S) - (1 - d)\, pcontr(p_i, S)$$

where $d \in [0, 1]$ captures the degree of selfishness of peer p_i. A hybrid peer considers both its own cost (with degree d) and its contributions to the others (with degree $1 - d$).

2.2 Global Cost Measures

One way to measure the overall quality of a cluster configuration is by evaluating the achieved *social cost* ($SCost$) defined as:

Definition 3 (Social Cost) *The Social Cost of a cluster configuration is defined as the sum of the individual costs of all peers in P.*

$$SCost(S) = \sum_{p_i \in P} pcost(p_i, S)$$

We can also evaluate the overall quality of the configuration from a query workload perspective, by considering the average cost for attaining results for all queries in Q. Then, the *workload cost* ($WCost$) is defined as:

Definition 4 (Workload Cost)

$$WCost(S) = \alpha \sum_{c_k \in C} \frac{|c_k| \theta(|c_k|)}{|P|} + \sum_{q in Q} \frac{num(q, Q)}{num(Q)} \sum_{p_i\ s.t.\ q in Q(p_i)} \frac{num(q, Q(p_i))}{num(q, Q)} \sum_{p_j \notin P(s_i)} r(q, p_j)$$

The first term expresses the cost of maintaining the clusters. The second term expresses the cost of all queries, i.e., the recall of evaluating them outside the clusters of their initiator.

The main difference between the social and the workload cost lies on how they assign weights to the queries. In the social cost, each peer assigns weights to its queries based on their frequency in its local workload, whereas in the workload cost, the weight assigned to each query is based on the frequency of the query in the overall query workload. Intuitively, while the social cost regards all peers as equals, the workload cost considers more demanding peers, i.e. peers that pose more queries, as more important than low demanding ones.

The two cost measures are not equal in the general case.

Proposition 1 *If for all peers p_i, $p_j \in P$, $num(Q(p_i)) = num(Q(p_j)) = \frac{num(Q)}{|P|}$, the social and the workload cost measures are proportional to each other.*

Proof.
Using the definition of individual cost (Def. 1), the social cost can be written as:

$$SCost(S) = \alpha \sum_{p_i \in P} \sum_{c_k \in s_i} \frac{\theta(|c_k|)}{|P|} + \sum_{p_i \in P} \sum_{q in Q(p_i)} \frac{num(q, Q(p_i))}{num(Q(p_i))} \sum_{p_j \notin P(s_i)} r(q, p_j)$$

The membership cost of $SCost$ is equal to the first term of $WCost$. Just consider that each cluster c_k appears in the sum of $SCost$ as many times as the peers that

belong to it, i.e., its size $|c_k|$. The second term differs from the second term of $SCost$ only on how much the workload of each peer is taken into account. It is easy to see, that if peers get an equal part of the query workload, i.e., $num(Q(p_i)) = num(Q(p_j))$, for all peers p_i, $p_j \in P$, the recall parts of the two costs are proportional, that is, improving the social cost improves the workload cost and vice versa.
□

In accordance to the social cost, we can also define the corresponding social contribution as:

Definition 5 (Social Contribution) *The Social Contribution of a cluster configuration is defined as the sum of the individual contributions of all peers in P.*

$$SContr(S) = \sum_{p_i \in P} pcontr(p_i, S)$$

The $WCost$ measure also has a counterpart from an altruistic point of view, $WContr$:

Definition 6 (Workload Contribution) *The Workload Contribution for a cluster configuration S is defined as:*

$$WContr(S) = \sum_{qinQ} \frac{num(q,Q)}{num(Q)} \sum_{p_i \; s.t. \; qinQ(p_i)} \frac{num(q,Q(p_i))}{num(q,Q)} \sum_{p_j \in P(s_i)} r(q,p_j) - \frac{\alpha}{|P|^2} \sum_{p_i \in P} \sum_{c_k \in s_i} |c_k| \theta(|c_k|)$$

Similarly to $SCost$ and $WCost$, the $SContr$ and $WContr$ are also proportional for specific workload distributions, in particular, when the query workload is uniformly distributed among the peers:

Proposition 2 *If for all p_i, $p_j \in P$ and all q in Q, $num(q,Q(p_i))/num(Q(p_i)) = num(q,Q(p_j))/num(Q(p_j)) = num(q,Q)/num(Q)$, the social and the workload contribution measures are proportional to each other.*

Intuitively, social contribution favors queries that are popular to specific peers, whereas its workload counterpart favors overall popular queries.

Let us now examine the relationship between the workload cost and the workload contribution.

Proposition 3 *For $\alpha = 0$, that is, if ignore the cluster membership cost, it holds: $WCost(S) = 1 - WContr(S)$, which means that the two measures are complementary.*

Proof.
It holds that:

$$\sum_{p_j \notin P(s_i)} r(q,p_j) + \sum_{p_j \in P(s_i)} r(q,p_j) = 1, \forall q \, in \, Q, s_i \in S. \quad (1)$$

Thus, $\alpha = 0$ we have: $WCost(S) = 1 - WContr(S)$.
□

Let us consider now, the social cost and the social contribution.

Corollary 1 *For uniform query workload among peers the social cost and social contribution are also complementary: $SContr(S) = 1 - SCost(S)$.*

Proof.
Again, for $\alpha = 0$, we can rewrite $SCost(S)$ using Eq. (1) and if we assume that $\frac{num(q,Q(p_i))}{num(Q(p_i))}$ is the same for all peers p_i, then we have that:$SContr(S) = 1 - SCost(S)$.
□

2.3 Stability

The goal of each player (peer) is to minimize/maximize its individual cost/contribution. Since the two cases are completely symmetric, we will refer in the following to selfish peers, but the same results are applicable for altruistic behavior. The question that arises is: if we leave the players free to play the game to achieve their goal, will the system ever reach a stable state in which no players desire to change their strategy (the set of clusters they belong to)? That is, will the system reach a Nash equilibrium?

Nash Equilibrium: Formally, a (pure) Nash equilibrium is a set of strategies S such that, for each peer p_i with strategy $s_i \in S$, and for all alternative set of strategies S' which differ only in the i-th component (different cluster sets s'_i for peer p_i):

$$pcost(p_i, S) \leq pcost(p_i, S') \quad (2)$$

This means that in a Nash equilibrium, no peer has an incentive to change the set of clusters it currently belongs to, that is, Nash equilibria are stable.

We shall first prove an interesting property of the clustering game. Due to the form of our cost function, the stable states in our system have the following property that constraints the number of possible configurations:

Lemma 1 *In any stable state, there are no clusters c_i, c_j such that $c_i \subseteq c_j$, $i \neq j$.*

Proof. Let S be a cluster configuration, c_i, c_j be two clusters in C such that $c_i \subseteq c_j$. Consider a peer p_k, $p_k \in c_i$. Clearly, $p_k \in c_j$. Let the individual cost of

p_k be: $pcost(p_k, S) = \alpha\gamma + \delta$, where γ is the membership cost for p_k when following strategy $s_k \in S$ and δ the respective recall it loses from the peers that do not belong to $P(s_k)$. Assume for the purposes of contradiction that S describes a stable configuration, then p_k can not select a strategy that would reduce its cost. Let us examine the strategy $s'_k = s_k - \{c_i\}$. Let S' be the configuration resulting by replacing s_k with s'_k in S. Then, $pcost(p_k, S') = \alpha(\gamma - \frac{\theta(|c_i|)}{|P|} + \delta) < pcost(p_k, S)$. The recall part of the cost function remains the same, because $P(s_k) = P(s'_k)$. Thus, p_k can reduce its cost by selecting the strategy s'_k, and therefore S is not a stable state, which contradicts our assumption.
□

Because of Lemma 1, it holds:

Corollary 2 *A cluster configuration in which a peer forms both a cluster of its own and also belongs to another cluster is not stable.*

Another interesting property is the following:

Lemma 2 *For any strictly increasing function θ, there is no stable configuration in which $\exists c_i$, $|c_i| > 1$, $c_i \in s_j$ and $\sum_{q\ in\ Q(p_j)} \sum_{p_k \in c_i} r(q, p_k) = 0$.*
Proof. Let S be a cluster configuration, p_j a peer and c_i a cluster of size $|c_i| > 1$, such that $c_i \in s_j$ and $\sum_{q\ \text{in}\ Q(p_j)} \sum_{p_k \in c_i} r(q, p_k) = 0$. If s_j=$\{c_i\}$, p_i can improve $pcost(p_i, S)$ by moving to a cluster of its own, since $\theta(|c_i|) > \theta(1)$ and the recall loss remains the same. If $c_i \subset s_j$, p_i can improve $pcost(p_i, S)$ by selecting strategy $s'_j = s_j - \{c_i\}$, which reduces its membership cost without affecting its recall loss. Thus, S is not stable.□

It is rather simple to show that for the cluster formation game, a pure Nash equilibrium does not always exist.

Proposition 4 *A pure Nash equilibrium does not always exist for the cluster formation game.*

Proof. Let us consider a simple scenario of two peers p_1 and p_2. Consider also that $Q(p_1)$ consists of a single query q_1 satisfied by p_2 (i.e. $r(q_1, p_2) = 1$) and $Q(p_2)$ consists of q_2 also satisfied by p_2. Let $C = \{c_1, c_2\}$ be the clusters in the system. Using Lemma 1, the following cluster configurations are possible: $p_1 \in c_1$ and $p_2 \in c_2$, described by $S_1 = \{\{c_1\}, \{c_2\}\}$, $p_1 \in c_2$ and $p_2 \in c_1$, described by $S_2 = \{\{c_2\}, \{c_1\}\}$ and both $p_1, p_2 \in c_1$ or c_2 described by $S_3 = \{\{c_1\}, \{c_1\}\}$ and $S_4 = \{\{c_2\}, \{c_2\}\}$, respectively. Let us assume a linear θ function, $\theta(p) = p$. Then, for any value of $\alpha > 0$, we can show that none of the possible configurations is a Nash equilibrium. In particular, since the first two configurations are symmetric, let us examine the first one. The individual costs of the two peers are: $pcost(p_1, S_1) = \alpha\frac{1}{2} + 1$ and $pcost(p_2, S_1) = \alpha\frac{1}{2}$. If p_1 moves to cluster c_2, then the system configuration becomes $\{\{c_2\}, \{c_2\}\}$, that is, configuration S_4, and the cost for p_1 becomes $pcost(p_1, S_4) = \alpha \leq pcost(p_1, S_1)$. Thus, configuration S_1 is not a Nash equilibrium, since p_1 can reduce its cost by moving to c_2. Let us consider now the configuration S_3 (S_4 is symmetric) in which both peers belong to the same cluster. Their individual costs are now: $pcost(p_1, S_3) = \alpha$ and $pcost(p_2, S_3) = \alpha$. Peer p_2 can reduce its cost by moving to the (empty) cluster c_2 (resulting in configuration S_1) and therefore S_3 is not a Nash equilibrium. Table 1 summarizes the payoff (cost) table for this two-player game.
□

Table 1: Payoff Table

	p_2 joins c_1	p_2 joins c_2
p_1 joins c_1	α,α	$\frac{\alpha}{2}$+1,$\frac{\alpha}{2}$
p_1 joins c_2	$\frac{\alpha}{2}$+1,$\frac{\alpha}{2}$	α,α

2.4 Social Optimum

Even if the system does eventually reach a stable state (Nash equilibrium), it is not necessary that this stable state has a satisfying cost. A measure widely used for evaluating how far from the best possible outcome a stable state is, is the *price of anarchy* defined as the ratio between the social cost of the worst Nash equilibrium and the "social optimum". The social optimum is obtained by minimizing the social cost measure over all possible configurations, even for those configurations that do not correspond to a stable state. In accordance to the price of anarchy, another measure often used is the *price of stability* defined by the ratio between the cost of the best Nash equilibrium and the social optimum. We can acquire a rough bound of the cost of the social optimum by considering each peer separately and evaluating its individual cost over all possible configurations. Then, by selecting for each peer the configuration that yields the minimum individual cost and adding these values, we obtain a bound for the minimum value of the social cost in the system, i.e., for the social optimum. Note that we are adding together individual costs that may correspond to different configurations, thus, the estimated social cost may refer to a configuration that cannot exist and may be very far from the actual value of the social optimum that we can achieve in a given system.

In general, our game is an asymmetric game. A game is *asymmetric*, if the value of the utility function or payoff differ if different players select the same strategy. However, there are cases in which the content and query distribution among the peers is such that the game is symmetric. For symmetric games, the following holds.

Observation 1 *Let $p_i \in P$ be a peer and s_i its optimal*

strategy, that is, the strategy that minimizes the value of its individual cost. If the game is symmetric for the peers in P, then s_i is also optimal for all peers in P.

2.5 Load Balance

First, we define *size-based load balance*.

Definition 7 *A configuration S is (δ_s)-size balanced if and only if $\forall c_i, c_j \in C$, it holds that: $(|c_i| - |c_j|)/|P| \leq \pm\delta_s$.*

We also consider *cluster-centric* load balance, where we require the load to be distributed evenly among the clusters. Load includes both the query workload generated and the content offered by the peers of each cluster. Specifically, for each cluster c_i, we define the *cluster query load*, denoted $L_q(c_i)$, as the sum of the number of queries issued by the peers in c_i to the total number of queries:

$$L_q(c_i) = \sum_{p_k \in c_i} num(Q(p_k))/num(Q)$$

Similarly, we define, for each cluster c_i, the *cluster content load*, denoted $L_r(c_i)$, as the sum of the number of results offered by the peers in c_i to the total number of results:

$$L_r(c_i) = \sum_{p_k \in c_i} \sum_{q \text{ in } Q} r(q, p_k) / \sum_{p_k \in P} \sum_{q \text{ in } Q} r(q, p_k)$$

For a configuration to be considered load balanced, we require that all clusters have similar load.

Definition 8 *A configuration S is (δ_q, δ_r)-load balanced, if and only if, $\forall$ c_i, $c_j \in C$, it holds that:*

$$L_q(c_i) - L_q(c_j) \leq \pm\delta_q$$
$$L_r(c_i) - L_r(c_j) \leq \pm\delta_r$$

where $0 \leq \delta_q, \delta_r \leq 1$.

It holds that:

Observation 2 *If we assume a uniform distribution of the global query workload and content among all peers, then a (δ)-size load balanced configuration is also (δ, δ)-load balanced.*

3 Case Studies

Although in the general case a Nash equilibrium does not always exist, there are cases in which the data and query workload distribution lead to the formation of stable clusters. Next, we present three simple scenarios.

3.1 Stability

Case I: No Underlying Clustering

We consider first the case in which all peers in P are similar in the following sense:

$\forall p_i, p_j \in P, num(Q(p_i)) = num(Q(p_j)) = num(Q)/|P|$ and $\forall q \in Q, r(q, p_i) = r(q, p_j) = 1/|P|$.

This scenario corresponds to a data and query distribution for which no physical grouping among the peers exist. All peers can be viewed as belonging to a single cluster, since they all have the same data and query distribution. Note that in this case, our game becomes a symmetric one, since all players yield the same payoffs when applying the same strategy. We consider three different configurations for this case and examine whether they constitute a Nash equilibrium.

Case(I.a): A single cluster

Let us assume as our first configuration the one in which all peers form a single cluster. Let us call this configuration S. The only way a single peer p_i can change its own strategy in S is by forming a cluster by its own. Let us denote this as configuration S'. Note that Corollary 2 does not allow a configuration in which a peer belongs both to a cluster with all other peers and form a cluster by its own. For configuration S to correspond to a Nash equilibrium, it must hold $pcost(p_i, S) \leq pcost(p_i, S')$, $\forall p_i \in P$ and by evaluating these costs, we get that this is true for:

$$\alpha \leq \frac{|P| - 1}{\theta(|P|) - \theta(1)}. \quad (3)$$

If we assume that the θ function corresponds to a linear function of the form: $\theta(p) = \lambda p$, $\lambda \leq 1$, we deduce that S (i.e. a single cluster) is an equilibrium for $\alpha \leq 1/\lambda$. Recall that large values of α mean that maintenance costs are more important than query recall. Thus, for the same θ, for values of α larger than this threshold, the maintenance cost would surpass those gained by recall and would lead to splitting the cluster. Note also, that whether a single cluster is stable or not depends also on the topology as captured through function θ. For instance, when λ is small (less connected topology), a single cluster remains stable for larger values of α.

If we consider altruistic peers that change strategies according to their individual contribution, then configuration S is a Nash equilibrium when:

$$\alpha \leq \frac{|P| - 1}{|P|\theta(|P|) - \theta(1)}, \quad (4)$$

and, if θ is the linear function, then: $\alpha \leq 1/\lambda(|P| + 1)$.

Case(I.b): Each peer forms its own cluster

Let us assume a second initial configuration S, in which each peer forms a cluster by its own. The only way for p_i to change its strategy is to leave its own cluster and join k other clusters, where $1 \leq k \leq |P| - 1$ (configuration S''). Evaluating the individual cost for S'', we see that the initial configuration corresponds to an equilibrium for:

$$\alpha \geq \frac{k}{k\theta(2) - \theta(1)}. \quad (5)$$

If we consider a linear θ function, S is an equilibrium for $\alpha \geq \frac{k}{(2k-1)\lambda}$. As expected, if α is small, the initial configuration is not stable, since peers would tend to cluster with each other to improve recall versus maintenance.

Again, for altruistic peers for S to be a Nash equilibrium it should hold:

$$\alpha \geq \frac{2k-1}{2k\theta(2)-\theta(1)} \tag{6}$$

And for linear θ functions: $\alpha \geq (2k-1)/\lambda(4k-1)$.

Case(I.c): m non-overlapping clusters

Finally, let us now assume an initial configuration S in which the peers form m non-overlapping clusters of the same size $|c| = |P|/m$. Consider a peer $p_i \in c_j$. The available options for p_i for changing its own strategy are to: form a cluster by its own; additionally to c_j, join k other clusters, where $1 \leq k < m$; or leave c_j and join k other clusters.

Thus, for S to be a Nash equilibrium, the following must hold, for all k, $1 \leq k < m$:

$$\alpha \leq \frac{|c|-1}{\theta(|c|)-\theta(1)}, \quad \alpha \geq \frac{|c|}{\theta(|c|+1)},$$
$$\alpha \geq \frac{|c|(k-1)+1}{k\theta(|c|+1)-\theta(|c|)}. \tag{7}$$

The above case can be easily generalized for clusters of different sizes.

Finally, for altruistic peers we have that S is a Nash equilirium when:

$$\frac{1}{\theta(|c|)} \leq \alpha \leq \frac{|c|-1}{|c|\theta(|c|)-\theta(1)} \tag{8}$$

Therefore, we showed that even if there is no underlying clustering according to the data and query workload distribution among the peers, a system can still reach a stable state that depends each time on the cluster maintenance costs and the portions of data and query workload each peer offers or demands.

Table 2(line 3) presents the conditions under which these configurations are stable for selfish peers.

Case II: Symmetric Clusters

The second case we consider refers to a scenario in which the data and query distribution is such that an underlying clustering/grouping exists among the peers. In particular, we examine a scenario in which the peers in P belong to g ($g \geq 1$) different groups of the same size $|c| = |P|/g$. For $g = 1$ we fall back to the case where there is no underlying clustering among the peers. The members in each group offer and demand data only within their group. All peers in the same group are equal. Formally, for all pairs of peers p_i, p_j in the same group, it holds $num(Q(p_i)) = num(Q(p_j))$ and $\sum_{q\text{in } Q(p_j)} \frac{num(q,Q(p_j))}{num(Q(p_j))} r(q,p_i) = g/|P|$, whereas for all pairs of peers p_i, p_j not in the same group, the lists $Q(p_i)$ and $Q(p_j)$ have no queries in common and $\forall q$ in $Q(p_j)$, $r(q,p_i) = 0$.

Case(II.a): A single cluster

We consider first a configuration S, in which all peers form a single cluster. The only other strategy for a peer as in the previous symmetric case is to form a cluster by its own. Thus, for S to correspond to a stable state, it must hold:

$$\alpha \leq \frac{|P|-g}{\theta(|P|)-\theta(1)}. \tag{9}$$

Case(II.b): Each peer forms its own cluster

For an initial configuration S in which every peer forms a cluster on its own, the only other option again is for a peer p_i to join k other peers p_j, $1 \leq k \leq |P|-1$ We can discern between two cases: (a) p_j is in the same group with p_i (S'), and (b), p_j belongs to a different group (S''). Case (b) is the same, whatever group out of the $m-1$ we consider, since all such peers are symmetric for p_i, i.e., they do not satisfy any of its local query workload. Since joining peers from different groups does not reduce the recall loss or the membership cost, a peer only considers joining k peers from its own group, thus, $k < |P|/g$. Since $pcost(p_i, S') < pcost(p_i, S'')$, for S to correspond to a stable state, we have:

$$\alpha \geq \frac{kg}{k\theta(2)-\theta(1)}. \tag{10}$$

Particularly for a linear θ and $k = 1$ it should hold: $\alpha \geq g/\lambda$.

Case(II.c): m non-overlapping clusters

We consider another initial configuration in which the peers form $m = g$ clusters of equal size $|P|/g$, with each cluster containing peers of a single group. Then, the individual peer cost for each peer $p_i \in P$ is equal to its cluster membership cost, since the cost for computing queries outside its cluster is zero (there are no results for $Q(p_i)$ in peers not in $P(s_i)$).

If p_i wants to change its strategy s_i then one possibility is to move to a cluster of its own, since joining any other clusters would not improve its recall loss or its membership cost. For S to correspond to an equilibrium in this case, it should hold for α:

$$\alpha \leq \frac{|P|-g}{\theta(|P|/g)-\theta(1)} \tag{11}$$

The conditions for stability are presented in Table 2(line 6). Similarly to the first case, we can perform the same analysis when we are dealing with altruistic peers by considering the individual contribution instead of the individual cost.

By comparing Case I (no underlying clustering) with $k = 1$ and Case II (perfect underlying clustering), we see that in Case II, configuration (B) where each peer forms a cluster of its own (no clustering) is stable for larger values of α. For example, for a linear θ, it is stable for $\alpha \geq g/\lambda$ as opposed to $\alpha \geq 1/\lambda$. Note also that in the clustered configuration (C), small values of g (which correspond to a smaller number of larger groups) result in smaller values of α.

Case III: Asymmetric Scenario

In this case, the shared content belongs to t ($t > 1$) different categories and the content and query distributions are such that each peer has content belonging to one category but poses queries for content belonging to a single different category. All categories have the same number of peers (i.e., $|P|/t$ each) maintaining their content and querying their content. Furthermore, all the peers that maintain (query) content from a category maintain (query) the same portion of data of this category (i.e., $t/|P|$).

The cluster configurations are:

CASE(III.A): A SINGLE CLUSTER. We consider again a configuration S, in which all peers form a single cluster. The only other strategy for a peer as in the previous cases is to form a cluster by its own. Then, for S to be stable we have:

$$\alpha \leq \frac{|P|}{\theta(|P|) - \theta(1)} \tag{12}$$

CASE(III.B): EACH PEER FORMS A CLUSTER OF ITS OWN. Similarly with Case (II.B), the only alternative for a peer is to join k other peers, $1 \leq k \leq |P|/t$, that maintain content that belongs to the same category as its queries. Then, for this case to constitute a Nash equilibrium it should hold:

$$\alpha \geq \frac{kt}{k\theta(2) - \theta(1)} \tag{13}$$

CASE(III.C): m NON-OVERLAPPING CLUSTERS. In this case, the peers form $m = t(t-1)/2$ clusters of the same size $2|P|/t(t-1)$ such that half of the peers in each cluster maintain content belonging to category t_i and pose queries for category t_j and the other half content of t_j and queries of t_i. The options for a peer are to: (1) leave its cluster and form a cluster of its own, (2) leave its cluster and join k other clusters or (3) additionally to its own cluster, join k other clusters. Since the only clusters a peer may consider are the ones maintaining content that it queries, $1 \leq k < (t-1)/2$. Thus, for this case to be stable it should hold:

$$\alpha \leq \frac{|P|}{(t-1)(\theta(\frac{2|P|}{t(t-1)}) - \theta(1))},$$

$$\alpha \geq \frac{|P|}{(t-1)\theta(\frac{2|P|}{t(t-1)} + 1)},$$

$$\alpha \geq \frac{(k-1)|P|}{(t-1)(k\theta(\frac{2|P|}{t(t-1)} + 1) - \theta(\frac{2|P|}{t(t-1)})}$$

Table 2(line 9) summarizes the conditions for Case III for selfish peers. Similar results can be attained for altruistic ones.

Compared to the symmetric scenario, configuration (C) is stable for a very limited range of α values. For a linear θ and $k = 1$, (B) is stable for $\alpha \geq t/\lambda$ similarly to $\alpha \geq g/\lambda$ which holds for the symmetric scenario, while configuration (A) is stable for smaller values of α.

3.2 Social Optimum

We examine whether any of the configurations we presented for the three case studies also corresponds to a social optimum. For a configuration to correspond to a social optimum then it must have the lowest social cost among all possible configurations (stable or not). We consider the case of using the linear θ function and selfish peers. The results can be easily adapted for altruistic peers if we consider the respective contribution measures.

It holds that:

Proposition 5 *Assume a linear θ function, $\theta(p) = \lambda p$, $0 < \lambda \leq 1$. Let S be a configuration where a peer p_i belongs to more than one cluster, then $pcost(p_i, S)$ is not the smallest possible for p_i.*

Proof. Let S be a configuration where a peer p_i belongs to $k > 1$ clusters of sizes $|c_1|, |c_2|, \ldots, |c_k|$. For a linear θ, the membership cost of p_i for S is always larger than the membership cost for a configuration S' where p_i belongs to a single cluster that includes exactly the peers of all k clusters and has size $|c| \leq |c_1| + |c_2| + \cdots + |c_k|$ - $k + 1$. Since the recall loss of p_i for S' is the same as for S, $pcost(p_i, S') < pcost(p_i, S)$. □

Based on this, we compare our stable configurations with all possible configurations in which a peer p_i belongs to one cluster of size $|P| - |P_0|$, where $0 < |P| - |P_0| \leq |P|$. In particular, we consider the case where $|P| - |P_0| = |P|$ (all peers in a single cluster) (S_i^{opt}), $|P| - |P_0| = 1$ (p_i forms a cluster by itself) (S_{ii}^{opt}), and $1 < |P| - |P_0| < |P|$ (S_{iii}^{opt}).

Case I: No Underlying Clustering

Since the game is symmetric, based on Observation 1, it suffices to minimize the individual cost of any peer.

From Table 2, Case (I.A) is stable for $\alpha \leq 1/\lambda$. Comparing the cost of Case (I.A) with all possible configurations, we conclude that when Case (I.A) is stable, its cost corresponds to the social optimum. Analogously, Case (I.B) is stable and has a cost equal to the social optimum for $\alpha \geq 1/\lambda$. Case (I.C) has a cost equal to the social optimum for $\alpha = 1/\lambda$. For this α, all three configurations have the same optimal cost.

CASE(I.A): A SINGLE CLUSTER

This configuration is equivalent to S_i^{opt}. Note that if Eq. (3) holds and θ is the linear function, then this Nash equilibrium corresponds also to a state with social cost equal to the social optimum. We already proved that any peer has larger cost if it forms its own cluster (S_{ii}^{opt}). By also comparing Case (I.A) to S_{iii}^{opt}, we conclude that (I.A) has the lowest cost for $\alpha \leq 1/\lambda$. Thus, the value of the cost of Case (I.A) corresponds to the social optimum.

CASE(I.B): EACH PEER FORMS ITS OWN CLUSTER

In this case, we have a equilibrium when $a \geq k/(2k-1)\lambda$ and based on Proposition 5 $k = 1$. By applying a similar analysis, we conclude that the cost of this case corresponds to the social optimum as in the previous scenario for $a \geq 1/\lambda$.

We observe that for $\alpha = 1/\lambda$ both case (I.A) and (I.B) correspond to equilibria with the same social cost, equal to the social optimum.

CASE(I.C): m NON-OVERLAPPING CLUSTERS

Similarly, by comparing CASE(I.C) to the possibly optimal configurations we conclude that it has a cost equal to the social optimum for $\alpha = 1/\lambda$. For this α, all three configurations have the same optimal cost.

Case II: Symmetric Clusters

In this case, the peers belonging to each group are symmetric to each other and each group is symmetric to the others. Thus, to determine the social optimum it again suffices to find the configuration that minimizes the individual cost for any of the peers.

CASE(II.A): A SINGLE CLUSTER

For the first configuration we can easily discern that for any value of $\alpha > 0$ a configuration with a lower social cost is one where the g groups form separate clusters. When all the peers form a single cluster each peer connects to $|P| - |P|/g$ peers which do not offer any recall to its queries thus the cost can easily be reduced by not connecting to them.

In particular, any configuration that includes clusters with peers from more than one group does not have an optimal cost, since its cost can be reduced if the cluster is split by separating the peers from the different groups. Therefore, in this case we consider only configurations with $0 < |P| - |P_0| \leq |P|/g$.

CASE(II.B): EACH PEER FORMS ITS OWN CLUSTER

In this case we need to consider cases S_i^{opt} and S_{iii}^{opt} in which p_i forms a cluster with peers of its own group (i.e. at the most —P—/g-1 as we already have shown) since peers from other groups only increase the membership cost without reducing the recall factor. If we consider $a \geq g\lambda$, then we conclude that the cost for a peer of joining a cluster with more peers is higher than forming a cluster by its own. So again, this configuration corresponds to a social optimum.

CASE(II.C): m NON-OVERLAPPING CLUSTERS

We have already considered the case of a peer moving to its own cluster which has a higher cost for $alpha \leq g/\lambda$ or a different existing cluster which cannot improve its cost. Thus, the only case we need to consider is S_{iii}^{opt}. We conclude that (II.C) has a cost equal to the social optimum for $alpha \leq g/\lambda$.

Case III: Asymmetric Clusters

Similar to Case II, the peers belonging to each group (t) are symmetric to each other and each group is symmetric to the others. Thus, our analysis is similar to the symmetric scenario.

CASE(III.A): *A single cluster*

This case does not have a cost equal to the social optimum for any $\alpha > 0$, because as in Case (II.A), separating the different groups results in a configuration with lower social cost.

CASE(III.B): EACH PEER FORMS ITS OWN CLUSTER

Similarly to (II.B), this configuration has cost equal to the social optimum for $\alpha \geq t/\lambda$.

CASE(II.C): m NON-OVERLAPPING CLUSTERS

Finally, Case (III.C) does not have a cost equal to the social optimum for any $\alpha > 0$, since, if, we remove from the cluster of peer p_i the $\frac{|P|}{t(t-1)} - 1$ peers that maintain content of the same category as the content of p_i, its individual cost is improved.

3.3 Load Balance

All three cases are by their definition (0)-size balanced, since all clusters contain the same number of peers.

No Underlying Clustering. From Observation 2, Case I is also $(0, 0)$-load balanced, since the content and the query workload is distributed uniformly among all peers. Let us study whether a configuration with non equal sized clusters is stable. To this end, we consider Case (I.D), that extends Case (I.C), by assuming a configuration of m non-overlapping clusters each with a different size $|c_k|$, $1 < |c_k| < |P|$, $1 \leq k \leq m$. For this case to correspond to a Nash equilibrium, we have $\forall p_i \in c_k$, $\forall k' \neq k$: $\alpha(\theta(|c_k|) - \theta(|c_{k'}|)) \leq |c_k| - |c_{k'}|$. Since θ is increasingly monotonous, it also holds that:

$$\frac{\theta(|c_k|) - \theta(|c_{k'}|)}{|c_k| - |c_{k'}|} = 1/\alpha$$

In the general case, there may be stable configurations with non equal sized clusters depending on the value of α and θ. Thus, we consider different θ functions. For a

constant θ: $\theta(p) = \lambda$, $\lambda > 0$, we have: $|c_{k'}| - |c_k| \leq 0$. Since the inequality holds $\forall c_{k'}, c_k$, we have $|c_{k'}| = |c_k|$. Thus, only (0)-size balanced configurations are stable. For a linear θ ($\theta(p) = \lambda p$, $0 < \lambda \leq 1$), the configuration is stable for $\alpha = 1/\lambda$. For this α, all configurations with m non-overlapping clusters are stable, including configurations with non equal sized clusters.

Symmetric and Asymmetric Scenario. Case (II.C) and (III.C) are (0)-size balanced but not $(0, 0)$-load balanced, unless the total amount of query workload and content of each of the g and t categories respectively is the same. For Case (II.C), since each cluster maintains content from a single category and queries content only of the same category, we may consider each cluster separately. By partitioning each cluster to m non-overlapping clusters, we obtain results similar to Case (I.D). For Case (III.C), we derive the same results, if we split each of the $t(t-1)/2$ clusters in m sub-clusters, so that in each of the created sub-clusters half of the peers maintain content belonging to category t_i and pose queries for category t_j and the other half content of t_j and queries of t_i.

4 Cluster Evolution

Assume some initial cluster configuration. As the system evolves, the recall achieved by this cluster configuration may deteriorate. Changes that affect the quality of clustering include topology updates as peers enter and leave the system, as well as changes of the peer content and the query workload. We propose a suite of protocols to keep the clustered overlay up-to-date with respect to these changes. Our protocols are based on local relocation policies that each peer follows so that it moves to the most appropriate cluster under the given system conditions. Such protocols can also be used to bootstrap the system, for example, by applying them on an initial configuration in which all peers belong to a single cluster or each peer forms a cluster by its own. We describe first the relocation policies followed by each peer, and then how such policies are applied for creating a new cluster configuration. For simplicity, in the rest of this paper, we focus on the case where each peer belongs to a single cluster: $s_i = \{c_j\}$.

4.1 Relocation Policies

Unlike most network creation games, our game is not a one-shot game but a repeated one, where the peers re-examine their strategy selection through time to cope with the system dynamics.

We assume that each cluster has a unique identifier, *cid*, and that all peers in the cluster are aware of this unique id. We also assume that the results of each query are annotated with the corresponding *cid*s of the clusters that provided them. Cluster ids are assigned based on peer IPs and timestamps. For example, when the first peer joins a cluster, the cluster id is formed by the IP of the peer concatenated with a timestamp. When another peer joins the cluster, it is informed of this *cid*. Every peer in the system does not need to know all *cid*s, but it gradually learns them, as results annotated with new *cid*s are returned for its queries. Therefore, when all peers leave a cluster, its *cid* just becomes unused. Recycling *cid*s is beyond the scope of this paper.

Since a peer can not be aware of all available results for a query, we define a measure called *cluster recall* as the fraction of results returned to peer p for query q by a cluster c_i to the total number of results returned for the query. The number of these results depends on the routing algorithm used, and if a query is evaluated against all clusters, it is equal to the total number of results for q in the system.

Based on the behavior of each peer, we consider two types of relocation policies: *selfish* and *altruistic*.

In the selfish relocation policy, a peer determines that its current cluster is no longer suitable, when it observes that its queries receive a low recall. Since, all query results received by a peer are annotated with the *cid* of the cluster they came from, each peer can keep track of its recall with respect to all clusters in the system. In particular, each peer p_i incrementally updates its individual cost, $pcost(p_i, S)$, considering all different strategies sets $\mathbf{S} = S_1, S_2, \ldots, S_{|P|}$ which differ only at their s_i component, one for each of the clusters c_i currently in the system. When the peer needs to decide whether to relocate or not, it selects the S_{new} with $s_i = \{c_{new}\}$ for which:

$$S_{new} = \arg\min_{S_j \in \mathbf{S}} pcost(p_i, S_j) \tag{14}$$

The motivation behind this policy is that a peer chooses to move to the cluster that has yielded the most results for its query workload.

In the altruistic relocation policy, the peers decide to move to the cluster whose recall would be improved the most by this movement. To this end, each peer keeps track of the number of results it sends to queries coming from a particular cluster. In particular, it incrementally updates its individual contribution, $pcontr(p_i, S)$, considering different s_i components for each of the clusters c_i in the system. When it is its turn to play, each peer selects the S_{new} defined as:

$$S_{new} = \arg\max_{S_j \in \mathbf{S}} pcontr(p_i, S_j) \tag{15}$$

The motivation behind this policy is that a peer chooses to move to the cluster (say c_{new}) for which it has provided the most results.

Table 2: Conditions for stability, $\surd$ indicates optimality for linear θ

Case I: No Underlying Clustering					
Case (I.a)		Case (I.b)		Case (I.c)	
$\alpha \leq \frac{\lvert P\rvert-1}{\theta(\lvert P\rvert)-\theta(1)}$	$\surd$	$\alpha \geq \frac{k}{k\theta(2)-\theta(1)}$	$\surd$	$\alpha \leq \frac{\lvert P\rvert/m-1}{\theta(\lvert P\rvert/m)-\theta(1)}$, $\alpha \geq \frac{\lvert P\rvert}{m\theta(\lvert P\rvert/m+1)}$, $\alpha \geq \frac{\lvert P\rvert/m(k-1)+1}{k\theta(\lvert P\rvert/m+1)-\theta(\lvert P\rvert/m)}$	$\surd$
Case II: Symmetric Scenario					
Case (II.a)		Case (II.b)		Case (II.c)	
$\alpha \leq \frac{\lvert P\rvert-g}{\theta(\lvert P\rvert)-\theta(1)}$		$\alpha \geq \frac{kg}{k\theta(2)-\theta(1)}$	$\surd$	$\alpha \leq \frac{\lvert P\rvert-g}{\theta(\lvert P\rvert/g)-\theta(1)}$	$\surd$
Case III: Asymmetric Scenario					
Case (III.a)		Case (III.b)		Case (III.c)	
$\alpha \leq \frac{\lvert P\rvert}{\theta(\lvert P\rvert)-\theta(1)}$		$\alpha \geq \frac{kt}{k\theta(2)-\theta(1)}$	$\surd$	$\alpha \leq \frac{\lvert P\rvert}{(t-1)(\theta(\frac{2\lvert P\rvert}{t(t-1)})-\theta(1))}$, $\alpha \geq \frac{\lvert P\rvert}{(t-1)\theta(\frac{2\lvert P\rvert}{t(t-1)}+1)}$, $\alpha \geq \frac{(k-1)\lvert P\rvert}{(t-1)(k\theta(\frac{2\lvert P\rvert}{t(t-1)}+1)-\theta(\frac{2\lvert P\rvert}{t(t-1)})}$	

A hybrid relocation policy that uses the hybrid peer cost, *hpcost*, is also feasible.

4.2 Cluster Reformulation Protocol

The relocation policies are deployed by the peers and applied to form the cluster reformulation protocol.

First, we define a new measure the *gain* which is defined as the absolute difference of the value of the utility function of a peer if we consider its current strategy from the value of the utility function when we consider the new strategy the peer wants to assume. That is, the gain measure shows how much the peer will benefit from its move to a new cluster.

Coordinated Protocol

If we assume the existence of some entity that has global view and control of the system, then this entity can coordinate the peers in their application of the policies resulting in a *coordinated cluster reformulation protocol*.

To implement such a coordinated protocol, we may use cluster representatives. The cluster representative does not need to remain the same for a cluster. Representative selection is local within each cluster and may be random or based on specific properties of the peers. When a peer stops acting as representative, it suffices to redirect all requests at the new representative. Practically, for a peer to join a cluster it just needs to know one of its members. It then sends a relocation request to that member, which forwards the request to the current cluster representative that takes over.

The clusters representatives gather and exchange information about their clusters so as to determine when and where a move is recommended. The protocol is initiated according for all peers at the same time. Each peer evaluates its relocation policy and determines the cluster it needs to move to. Then, all relocation requests are gathered and ordered according to non-increasing value of gain. A portion of these requests is granted in this order (i.e., the x% of all the requests) (Alg. 1).

Algorithm 1 CoordinatedEvent

$|P|$: number of peers
n:number of clusters
$C = \{c_1, \ldots, c_n\}$: number of clusters
$R = \{r_1, \ldots, r_n\}$: cluster representatives

```
1: for all global events do
2:   for all r_i ∈ R do
3:     send a game initialise request to all p_j ∈ c_i
4:     for all p_j ∈ c_i do
5:       evaluate gain_{j,i-i'}
6:       send relocation request with gain_{j,i-i'}
7:     end for
8:     send all relocation requests to all other r_i ∈ R
9:     sort relocation requests in non-increasing order of
       gain
10:  end for
11:  for all gain_{j,i-i'} within x% of the list do
12:    p_j moves from c_i to c_{i'}
13:  end for
14: end for
```

Uncoordinated Protocol

We argue that the use of coordination is not necessary and instead propose an *uncoordinated cluster reformulation protocol* in which each peer determines when to play locally and independently from the other peers in the system.

Each peer, when it determines that it is its turn to play, applies the relocation policy it follows. If the relocation policy indicates that moving to a new cluster (c_{new}) improves the peer's utility function, the peer leaves its current cluster and moves to the new one.

To facilitate the movements between clusters and reduce the overhead required when a new member joins one, we may again use cluster representatives as in the coordinated protocol.

Game Types

Based on how often the peers determine that there is their turn to play in the uncoordinated protocol or the coordinated protocol is initiated, we discern between two types of game: an *event-based* and a *trigger-based* game.

For the coordinated protocols, when we consider an event-based game, the protocol is initiated after each event in the system, i.e., a query issued or an update in content or topology. In the uncoordinated event-based game, a peer determines whether it needs to relocate after it is made aware of a certain type of event. The relevant events are related to the queries and differ for selfish and altruistic peers. Selfish peers re-examine their strategy after a query from their local query workload has been evaluated, while altruistic peers after a query for which they maintain results reaches them. Selfish peers consider their own queries as relevant events because it is their recall they aim to increase, while altruistic ones find the queries for which they can provide results for relevant because they are interested in contributing the most to their cluster. A hybrid peer may choose either or both types of event as relevant.

We also consider a variation of the event-based game the*batch-based* game. In the coordinated version, the game is initiated not after each global event but after a predefined number of such events (batch). Similarly, for the uncoordinated version the peers determine whether they need to relocate to a new cluster after a batch of relevant events.

In the coordinated trigger-based game, the social or workload cost (or corresponding contributions) are constantly updated and the protocol is initiated when the respective global gain becomes greater than zero. For the uncoordinated version, each peer continuously updates its gain measure and decides to play whenever its value becomes greater than zero, i.e, when it has something to win from moving to another cluster. To continuously update the measures there is a need to monitor the system or perform some type of polling.

4.3 Controlling Parameters

The gains that individual peers attain from relocation may not always worth the re-organization cost. Thus, there is a need to control the overheads of cluster reorganizations. To this end, we consider different auxiliary parameters to the basic reformulation protocol, which act as overhead control mechanisms. The presented mechanisms can be applied in all three types of game we have described and both for the uncoordinated and coordinated protocols. We can apply either one of them or a combination of them globally in the case of coordination or locally at each peer.

Stopping Condition

Since each move imposes considerable overheads, we want to reduce the number of movements that occur by allowing a move only if the gain we will get is considerable. Thus, after applying its relocation policy and before issuing a relocation request, each peer evaluates its gain and compares it with a system-defined threshold ϵ. The relocation request is issued only if the gain value is is larger than ϵ. Consequently, the protocol stops without the system reaching an equilibrium, but rather an ϵ-stable state.

In the coordinated protocol, the stopping condition on ϵ may also be applied on a global level if we measure the gain with the respect to the social cost or contribution instead of the corresponding individual measures.

Playing Probability

Instead of allowing a peer to play (i.e. re-evaluate its strategy) every time it determines that it is time to do so, for example after an event in the event-based game, we introduce the use of a *playing probability* Pr. The basic protocol is altered such that each time the peer determines it needs to play, it plays with a probability Pr. This probability determines how aggressive a player is, i.e., how high is the chance of a peer to play, and the larger its value the more often a player is allowed to play. When $Pr = 1$, then the protocol is the basic reformulation protocol where peers play every time it is their turn according to the type of game. By decreasing Pr, we reduce the number of moves caused by the protocol.

The playing probability can either be the same for all peers, so as to treat all peers as equals or it may be different for each peer. For example, one may choose to give a higher probability to peers that change their content or workload often so that they can adapt faster to these changes. Also, another factor we can take into account is the size or the level of demand of a peer. Peers with more content or a bigger query workload may be given a higher probability for playing since these peers are the ones that most influence the workload cost and contribution.

Movement Quota

Another mechanism to limit the number of moves the peers make is to enforce a movement quota policy. In particular, a quota of n possible moves (relocations) is assigned to each peer when the peer joins the system. This is the maximum number of moves a peer is allowed for a specified time.

The peer can spend this quota in a time period of T_q. If a peer makes n relocations before T_q expires, it is not allowed to make any other moves. At the end of the of

T_q, the quota is replenished and the peer has again n available moves.

The quota policy is fair, since it treats each peer the same and does not penalize one in favor of another as it may be the case with the locking protocol. If a peer is very active, then it quickly uses up its quota and is forced to remain in its current cluster.

Since all peers do not join the system simultaneously, each peer has its own starting point for the T_q time periods.

For the event-based game, T_q can be either an independent time period at each peer or it can be also represented as a number of relevant events. That is, for a number of k relevant events that reach a peer it can only realize n moves ($n < k$). After each k events, the quota is replenished. For the trigger-based game, again a time period T_q is required.

The value of n, i.e. the number of quota, expresses a trade-off between consuming system resources for reclustering and tolerating low recall values from a poor clustering. Moreover, a very large quota allows all movements and thus becomes obsolete as a mechanism for preventing excessive movements. Whereas, a very small quota would prevent peers from making movements that would improve the social cost of the system.

All controlling parameters that we introduced for the uncoordinated protocols can also be used in the coordinated versions, in addition to the parameter inherent in this version that determines how many of the relocation requests are granted. The stopping condition (ϵ) is applied in a global level by examining the corresponding global gain.

5 Experimental Evaluation

We model a system of peers sharing data belonging to different semantic categories. Each peer in the system is associated with a data category j and maintains documents belonging to it. The local query workload of each peer is generated by first selecting a document category with probability $P(j)$ following a zipf distribution, and then a document d from that category with probability $P(d, j)$ following another zipf distribution within each category. We define $P_{x \in l}(d, j)$ as the probability of peer x associated with category l posing a query about document d of category j as:

$$P_{x \in l}(d, j') = \begin{cases} (1-m)P(d,j), l \neq j \\ ((1-m) + m/P(j))P(d,j), l = j \end{cases}$$

Parameter m is a measure of the interest-based locality ([17]) the users exhibit. The presence of interest-based locality, i.e., of the peers property to maintain data similar to their local query workload is derived from measurements in real traces of p2p system found in ([17]). For our evaluation we consider three specific scenarios. For the first, $m = 1$. This is the *symmetric scenario* in which both queries and data of each peer belong to the same category. In the second scenario, we consider again $m = 1$, but for a $j \neq l$ which is selected randomly from the remaining categories. This is the *asymmetric scenario*, in which each peer has data from one category but poses queries for a single different category. Thus, the symmetric scenario exhibits maximum interest-locality, while the asymmetric none. Finally, in the third scenario, the *random scenario*, $m = 0$. Again, there is no interest-based locality and each peer has both data and queries uniformly distributed from all categories.

We used as data Newsgroup articles belonging to 10 different categories. The articles were pre-processed, stop words were removed from the text, lemmatization was applied and the resulting words were sorted by frequency of appearance. The texts are distributed among 10000 peers. Queries are generated by choosing a random word from the texts such that each query is satisfied by documents only in a single category. Table 3 summarizes our parameters.

We present four sets of experiments. In the first set, we evaluate the event-based and the trigger-based games and the influence of the tuning parameters on them. Furthermore, we compare the uncoordinated versions we propose to corresponding coordinated protocols. In the second set, we examine how our protocols perform when we start from various initial configurations. In the third set, we focus on how well our protocols enable the system adapt to changes. Finally, in the forth set we compare our protocols adjusting capability to a caching scheme.

5.1 Comparison with coordinated protocols

We compare our two types of game, the event-based and trigger-based, as well as the batch-based variation of the event-based game with corresponding coordinated protocols. Our goal is to demonstrate that our protocols work efficiently and competitively to the coordinated ones, despite the lack of global control.

Tuning Parameters Influence

We compare uncoordinated and coordinated versions of the event-based, trigger-based and batch-based protocols with batches of 20, 50 and 100 events, which correspond to the 1/10th, 1/4th and 1/2 of the queries in the average local workload of a peer respectively, for different tuning parameters. We consider asymmetric peers since this type of peers pose the greatest challenge when applying clustering. The peers are all selfish in this experiment. We explore the influence of strategy

selection later. We assume that clusters are organized in a chord-like topology (logarithmic θ). We start with an initial configuration in which each peer forms its own cluster, which is similar to a system in which no clustering algorithm has been applied yet. We measure the social cost, the number of movements and the average number of turns per peer in the system until we reach a stable state. With the term turn, we refer to the turn of each player in the game when it needs to decide whether to play or not. The more playing turns the protocol needs to complete, the slower it reaches stability.

Stopping Condition. We first examine how the various protocols behave for different values of the stopping condition ϵ. We compared all approaches for different values of ϵ. For the uncoordinated protocols we used different probabilities for playing (1, 0.5, 0.25). Figure 2(left) and Figure 2(left-center) report our results for probability $Pr = 0.5$. For the coordinated protocols, we set the playing probability of all peers to 1 and regulate how aggressively the peers play by changing the percentage of peers in the ordered list that are allowed to play at each round. We experimented with setting this percentage to 25% 50% and 100% of all the peers and report our results in Figure 2(center-right) and Figure 2(right) when 50% of the peers in the list are allowed to play.

We observed that for the same value of ϵ each variation presents an approximate fixed value for the social cost regardless of the value of the playing probability or the percentage of peers allowed to play. Among the uncoordinated protocols the trigger-based is the one that displays the largest overhead. Since our experiments so far concern only updates in the local query workload of the peers (queries are being issued gradually), we observe that the coordinated versions of the trigger-based and the event-based protocols behave identically. That is, the event-based protocol is initiated after each query, and since after each query the social cost also changes, the trigger-based protocol is initiated as well. Both protocols require the largest overhead we encountered yet, especially when 50% and more peers are allowed to play. However, the final value of the social cost is similar to that achieved by the corresponding uncoordinated protocols with respect to the selected ϵ.

For each of the protocols we can detect a range of values ϵ for which the protocols exhibit the best trade off between social cost and number of movements and turns. For values greater than this value, the protocols have increased social cost, while for lower values the additional number of movements required is not justified by the reduction in the social cost it achieves. For the event-based protocol this critical range is around 10^{-4}. The social cost decreases significantly as ϵ decreases up to 10^{-4}, but when $\epsilon < 10^{-4}$, more than $1/2|P|$ of movements are required for the social cost to decrease by a factor of about 10^{-2}. For the uncoordinated trigger-based approach the critical value range is larger and around 10^{-3}. Due to the large overhead the trigger-based approach entails, lower values render it useless (almost all players play after each event that occurs in the system). Finally, when we use batches of events, we observe that the system works better for lower values of epsilon around 10^{-5} to 10^{-6}, where with a small number of movements we still achieve a significant reduction in the social cost. For the larger values of the parameter this variation performs much worse than the other two, especially for larger batches of events like 50 and 100. For the coordinated event-based and trigger-based variations, the value of ϵ is even greater than the uncoordinated trigger-based one and around 10^{-2} or larger. Even then, the overhead is larger with respect to the uncoordinated trigger-based game. The corresponding value for the individual cost in this case is at most around 10^{-3}. For the batch-based version we observe that we have much better behavior for lower values of ϵ and the number of moves is only marginally larger than that in the uncoordinated version. When using large batches, the moves become even the same as we have almost no unnecessary moves and unlike the event and trigger-based version, the batch-based ones work better with larger percentages of playing peers.

Playing Probability. We now select the value of ϵ for each protocol according to the previous results, i.e., 10^{-4} for the event-based, 10^{-3} for the trigger-based, 10^{-6} for all the batch-based variations and 10^2 and 10^{-4} for the coordinated ones.

The playing probability does not influence the value of the social cost we achieve considerably, as for each protocol it depends mostly on the selected value of ϵ and is around 922 for the values we selected for this experiment. Thus, we measure the number of required moves and turns.

We first discuss the uncoordinated protocols. The trigger-based approach has the largest overhead both in terms of moves and turns for all values of probability. For all the protocols, while a lower playing probability decreases the number of required movements by avoiding unnecessary ones (Fig. 3(center-left)), it increases the number of turns required for stability. The peers do not play every time they could, since the probability check fails frequently, thus resulting in more required turns (Fig. 3(center-right)). For the batch-based approaches, we observe that for smaller batches of events (such as 20) the approach behaves similarly to the event-based one. When the batch sizes are larger, then low playing probabilities cause even larger increases in the number of required turns without reducing the number of movements more significantly. Thus, the batch-based approaches are more sensitive

Table 3: Tuning Parameters

Parameter	Range	Default Value
Topology and Strategy		
number of peers ($\|P\|$)	-	10000
parameter α	1-100	1
membership cost function (θ)	-	log,linear
strategy	-	self-alt-hybrid-mixed
Data-Query Distribution		
number of categories	-	10
interest locality degree (m)	-	0-1
Tuning Parameters		
stopping condition (ϵ)	0-10^{-8}	$10^{-4}, 10^{-3}, 10^{-6}$
playing probability (Pr)	0-1	0.5
movement quota (n)	1-15	∞
quota period (k)	-	20, (5*batch size)
% of allowed moves (x)	0.1-1	1

to the probability value when it falls under some value (around 0.5).

To compare with the coordinated protocols, we consider the relationship between the percentage of peers we allow to play at each round and the playing probability. This percentage of peers can be viewed as another way to control the overhead by reducing the number of peers that play at each turn in a way similar to the probability. Furthermore, by ordering the peer requests according to their gain value, we implicitly associate their playing probability to their gain value. Thus, we set the playing probability to 1 for all peers and instead experiment with different percentages of playing peers. The results (Fig. 3)(center-right) and Fig. 3)(right)) are similar to the corresponding results for the uncoordinated version when we varied the playing probability. We notice that the coordinated version of the protocols perform better for lower values of playing probabilities than the corresponding uncoordinated versions.

Quota. To evaluate the influence of the quota protocol we consider the same values of ϵ we used in the previous experiment and a playing probability of $0.25, 0.5 and 1$. In Figure 4 we report our results for probability equal to 0.5 and the uncoordinated protocols. We consider that each peer is given a quota of n moves which is replenished after k events. We set $k = 20$ for the event-based and trigger-based approaches. For the batch-based approaches, k is defined as a product of the size of the batch. We use 5 times the size of the batch as our replenishing period. We vary n from 1 to 15.

The social cost is again the same regardless the quota used (Fig. 4(left)). When n is relatively small $n < 6$ then for all values of probability we observed a large number of checks with the worst behaviour for the smallest probability value (Fig. 4(center)). There is a small gain in the number of movements that is slightly decreased especially in the case of the trigger-based approach where the communication overhead is considerably larger. We observed that there is an area of quota for each approach and each probability value in which our protocols behave the best. For example, for the event-based approach for $n > 6$ and $n < 12$ and probability of 0.5 we observe the best system performance (a balance between the number of movements and the number of checks). In general, for larger probabilities and protocols that make more movements, the use of quota combined with the playing probability can significantly improve the performance. For example, for the trigger-based approach with probability 0.5, while with quota $n = 10$ we have a decrease of about 10% in the number of movements compared to not using any quota, while the number of checks is not increased. This proves that we mostly avoided unnecessary movements.

Similarly to the observation regarding the playing probability, the use of a low movement quota is also beneficial for the coordinated protocols. While the trigger-based protocol in the uncoordinated version required a quota of about 10 moves, in the coordinated version the protocol works better when the quota is around 12. Similar conclusions apply for the batch-based approaches.

Progress Per Round

It appears from our experiments so far that the batch-based approaches perform the best. They reach the same social cost as the other two while requiring less movements and less turns, thus they are the ones with the lowest overhead. However, we need to consider that the reported social cost value is reached at the end of the protocol when we reach stability. While the protocol is still active, since the batch-based approach is the one that acts the last to correct the increase in the cost caused by updates, we expect that it will have the worst behaviour.

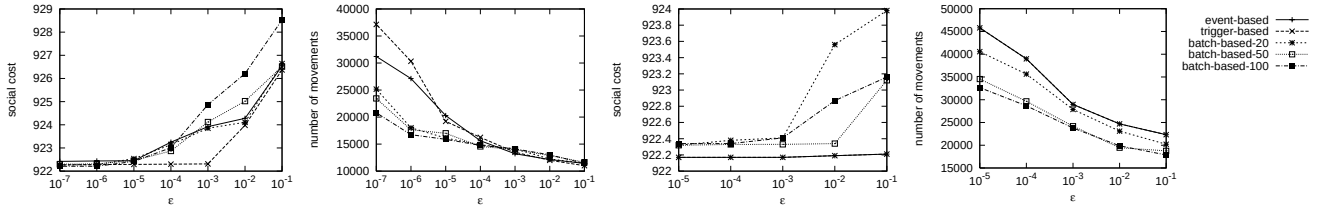

Figure 2: (left) Social cost and (left-center) movements with no coordination and (right-center) social cost and (right) movements with coordination

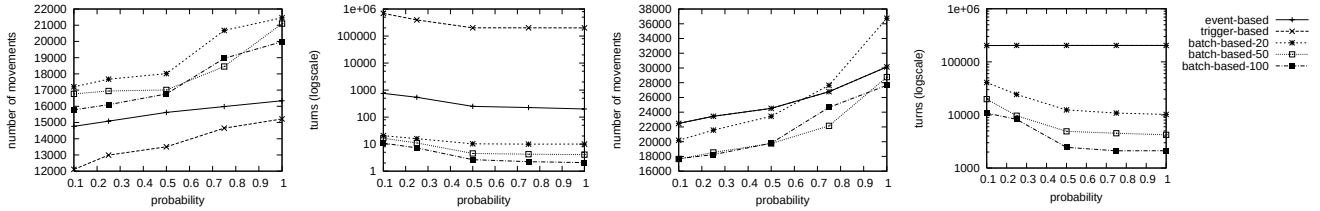

Figure 3: Varying probability and (left) movements and (center-left) turns with uncoordinated protocol and (center-right) movements, (right) turns with coordinated protocol

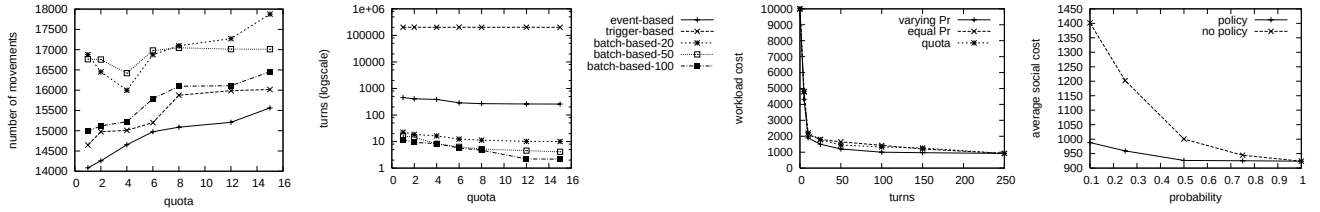

Figure 4: (left) Movements and (center-left) turns with varying quota and (center-right) workload cost with different probabilities for each peer and (right) social cost with policy and no policy

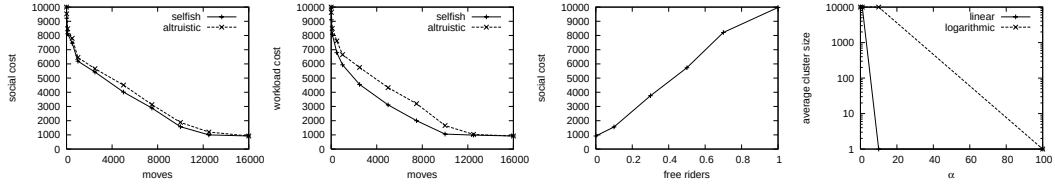

Figure 5: (left) Social and (center-left) workload cost through progressing rounds, (center-right) influence of free riders and (right) influence of α

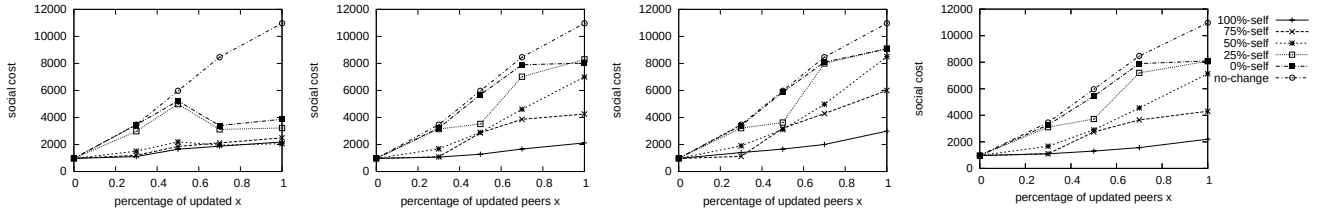

Figure 6: Social cost for different percentages of updated peers (query workload) for (left) scenario 1, (center-left) scenario 2, (center-right) scenario 3 and (right) scenario 4.

Table 4: Social Cost Per Round

	Uncoordinated					Coordinated				
	Event-based	Trigger-based	Batch-20	Batch-50	Batch 100	Event-based	Trigger-based	Batch-20	Batch-50	Batch 100
SCost	989.32	945.56	1207.05	1876.15	2876.45	926.41	926.41	1100.25	1543.55	2454.67

To demonstrate this we measure the average social cost per turn (Table 4). Among the uncoordinated protocols the trigger-based has the lowest average social cost (945.56), with the event-based approach coming a close second (989.32). All batch-based approaches perform considerably worse, and for 100 batch size, we have the worst performance. The main advantage of the coordinated version seems to be the very fast reaction to changes. Both the trigger-based and the event-based version work more efficiently than the uncoordinated trigger-based approach with an average cost of 926.41. The advantage is explained because of the policy the coordinated protocol follows by applying ordering in the relocation requests. By granting the requests with the larger gain, the protocol seems to avoid some unnecessary moves that would have to be retracted in the future. This is also the only difference between the two versions of the trigger-based protocol as they are both triggered by the same events that affect the social cost. The batch-based approaches also perform much better, since they take into account global rather than just local events.

Protocol Variations

Varying Probabilities. So far in our experiments, we use the same value of probability for all the peers. This does achieve a sense of fairness since we give all players an equal chance of playing. However, one may need to consider that all players are not equal, they have different sizes and different levels of demand. Also, some may update their content or workload more frequently than others. Thus, we may want to tune their playing probability accordingly. Besides tuning the playing probability, another way to determine the number of times each player plays is with the moving quota. The quota in contrast to tuning the probability according to the levels of demand ensures fairness. In this experiment, we increase the level of demand of 20% of the peers to double the average level of demand of all and we set their playing probability to 1 while the probability for the rest of the peers is set to 0.5. We compare it to an approach where all peers have the same probability 0.5 regardless their demand level and another approach where all players have playing probability 1, but movement quota is utilised. We used the setting from the previous experiment and set $n = 10$ and $k = 20$. We measure the average workload cost each peer observes at its turn.

Giving the more demanding peers larger playing probability improves the workload cost much faster than when all players have the same probability. The more impressive result is that this configuration also decreases the number of movements significantly. While with equal probability we require about 15000 movements, the varying probability scheme reduces them to about 11500, more than 1/4 of the peers save one move. The movement quota approach performs similarly to tuning the probability. Since the quota is replenished after a number of events and the more demanding peers are influenced by more events, their quota is replenished faster than that of the other peers thus allowing them to play more often. Thus, quota can be used when no information about the level of the demand of the peers is known and we cannot tune the probability accordingly. The additional overhead is that we need to tune the quota policy instead.

Ordering of Relocation Requests. The coordinated protocols besides reacting faster to changes also have the advantage of being able to order the relocation requests and granting first the ones that result in the largest reduction in the social cost. To evaluate how much they benefit from this ordering we compare the event-based coordinated protocol for different percentages of playing peers with a corresponding protocol in which no ordering among the requests is utilised. That is, x% of the requests are selected randomly to be granted. We measure the average social cost per round (global event) achieved by both approaches. While the ordering does impose an additional cost, it also improves social cost significantly (up to 15%). The improvement is better observed when a small number of requests is only granted (small x), while their performance becomes identical when almost all requests are satisfied.

Events not affecting the cost. Thus far, our experiments involved events in the type of issuing queries that always yielded to a change (improvement or not) of the cost of some peers. To differentiate between the trigger-based and the event-based protocols, we performed an experiment which we added events that did not affect any cost functions. Such events include updates of content that is not queried by anyone so far, or insertion of new peers that have no content of interest to any of the other peers so far. In this case, the trigger-based protocol outperforms the event-based one in term of turns, while they still require the same number of movements and yield the same social cost. The difference is the number of times the event-based protocol is initiated without any reason causing the excessive evaluation of cost functions at each peer. The batch-based events are also influenced by these types of events, but since they consider them in batches, the problem is not so significant, especially if the rate of such events is not that large.

Summary:

- The value of ϵ is the main factor controlling the value of the achieved social cost. For each approach we can define an appropriate value lower than which the improvement in the social cost does not justify the number of required moves. The

playing probability and quota reduce the number of movements while increasing the number of turns.

- The trigger-based approach is the most expensive among the uncoordinated approaches. Its main advantage is that it reacts much faster to any change. The batch-based approaches present the lowest overhead but also react much slower to changes. The event-based approach seems the most reasonable compromize between overhead and fast reaction to changes.

- Variations that adjust the probability according to the peer's level of demand manages to reduce the movements and increase the workload cost faster (in less turns).

- The coordinated protocols achieve approximately the same social cost value while imposing a much larger overhead, especially with regards to required turns. Their main advantage is that they react to changes even faster than the uncoordinated trigger-based approach and use requests ordering yielding the best average social cost. The event-based approach is initiated by events that do not affect the cost, thus making it even more expensive in terms of turns with respect to the trigger-based version.

5.2 Cluster Formation

In this set of experiments, we start from a random peer configuration and examine whether the peer reformulation protocol leads to the desired cluster configuration. In this and the rest of the experiments, we use the uncoordinated event-based protocol with Pr equal to 0.5 for all the peers and $\epsilon = 10^{-4}$ as it offers the best trade-off between overhead and performance. Let M be the number of peer categories in the system for each scenario, i.e., for the symmetric scenario $M = 10$. We consider five different cases for the initial system configuration: (i) each peer forms its own cluster; (ii) all peers form a single cluster; (iii) peers are randomly distributed to n groups and we discern for different values of n the subcases: (a) $n = M$, (b) $n < M$ and (c) $n > M$; (iv) peers are clustered according to their content and (v) peers are clustered according to their workload.

The peer that issues a query each time is selected uniformly at random from all peers in the system. We allow the system to run multiple queries and check whether the system reaches an equilibrium and if so, what is the total number of moves the peers made, the number of clusters they formed and the average size of those clusters (Table 5). We also provide the achieved social and workload cost as well as the respective social and workload contribution (Table 6).

Both in the symmetric (Table 5 lines 1-7) and asymmetric scenario (Table 5 lines 8-15), all strategies reach a Nash equilibrium and form the desired number of clusters. Furthermore, for the symmetric peers both social and workload cost only depend on the cluster membership cost; the cost for the recall is zero, since all relevant data are located within the cluster (Table 6 lines 1-7). Similarly the recall term in the social and workload contribution has its maximum value and the peers only pay for their membership cost. Moreover, according to our case studies for the given $\alpha = 10$ the social cost achieved is very close to the social optimum. If we consider that all our peers are perfectly symmetric and all clusters have the same number of peers (1000) as in Case Study II, then the lowest individual cost for a peer is equal to: $10^{-3}\alpha$ and the social cost is: $10\alpha = 10$ which is about the same with the cost our approach achieves. The value is not exact because the peers and the clusters are not perfectly symmetrical as in the study case.

For the asymmetric scenario, the recall factor in this case is not zero, and we observe higher social cost and lower social contribution. Also, since the queries are not uniformly distributed among the peers, the social cost differs from the workload cost (similarly for the respective contribution measures). When the symmetric peers are clustered according to their content or workload, there is no need for change since the appropriate clusters are already formed (Table 5 lines 6-7). For the asymmetric peers, both configurations are not stable, i.e., the peers can improve their cost, though they require less moves to reach stability than the other initial configurations. Thus, we deduce that relying only on content or workload information is not enough to provide the appropriate clustering and thus, our policies take into account both.

Finally, for the third scenario since no clear number of desired clusters can be deduced, we consider all peers as single-membered clusters (Table 5 line 16) and all peers in one cluster (Table 5 line 17). This scenario is similar to our first case study where no underlying clustering exists. Due to the small value of the membership cost (logarithmic) and the α parameter that is set to 10, the peers converge towards a single cluster in both cases as we showed in our case studies and achieve a social cost around the optimum. In particular, the second case requires no moves since all peers are already in one cluster. For the same α, if we use the linear function for the θ, then the single cluster is no longer the best configuration and the peers tend to split into smaller clusters. Similarly, a larger value of α would force the peers to form single membered clusters. For example, we repeated the experiment setting $\alpha = 100$. In this case, the first configuration consisting of single member clusters was the one both initial

Table 5: Cluster Formation

	Moves			Clusters			Cluster Size		
	Self	Alt	Mix	Self	Alt	Mix	Self	Alt	Mix
Symmetric Scenario									
i	15358	15436	14900	10	10	10	1087.5	1100	1100.5
ii	14786	14365	14657	10	10.5	10	1079	1109.5	1125.5
iii(a)	9654	9812	9867	10	10	10.5	1112	1085.5	1119.5
iii(b)	10211	10210	11270	10	10.5	10.5	1010.5	1056.25	1011.75
iii(c)	9841	9554	9456	10	10.5	10	1012.25	1009.5	1016.5
iv	0	0	0	10	10	10	1100	1100	1100
v	0	0	0	10	10	10	1100	1100	1100
Asymmetric Scenario									
i	16875	16758	17008	44.55	43.88	45.08	110.75	110.9	111.15
ii	16257	16109	16431	45.67	44.78	45.04	111.9	111.75	111.45
iii(a)	9405	9115	9650	45.17	45.2	45.01	111.05	110.5	110.33
iii(b)	9180	9320	9125	44.19	45.18	44.92	111.15	110.9	110.67
iii(c)	8502	8745	8790	44.67	44.78	44.87	110.5	110.24	111
iv	5865	6120	6275	45.07	45.15	44.18	111.4	111.75	111.05
v	6015	6104	6436	45.12	44.85	45.05	110.9	111.09	110.89
Random Scenario									
i	19450	19710	19340	1	1	1	10000	10000	10000
ii	0	0	0	1	1	1	10000	10000	10000

Table 6: Cluster Formation Utility Functions

	SCost			WCost			SCont			WCont		
	Self	Alt	Mix	Self	Alt	Mix	Self	Alt	Mix	Self	Alt	Mix
Symmetric Scenario												
i	10.07	10.23	10.15	9.97	10	10.15	9.67	9.85	9.84	9.32	9.45	9.65
ii	10.42	10.67	10.93	10.2	10.28	10.65	9.45	9.63	9.58	9.52	9.35	9.61
iii(a)	10.4	10.97	11.35	10.55	10.86	10.93	9.98	9.64	9.69	9.22	9.28	9.53
iii(b)	11.45	10.81	11.59	10.75	10.65	10.9	9.82	9.36	9.42	9.29	9.22	9.22
iii(c)	10.65	10.83	11.72	10.35	10.65	11.93	9.32	9.22	9.29	9.13	9.16	9.14
iv	10.09	10.09	10.09	10.09	10.09	10.09	9.94	9.94	9.94	9.94	9.94	9.94
v	10.09	10.09	10.09	10.09	10.09	10.09	9.94	9.94	9.94	9.94	9.94	9.94
Asymmetric Scenario												
i	919.9	922.35	924.85	914.77	920.01	925.86	987.45	974.15	992.09	982	989.76	985.05
ii	924.75	929.25	926.05	919.87	925.5	931.43	974.01	959.25	964.26	971.17	948.89	963.25
iii(a)	922.86	926.41	941.02	917.9	921.12	937.09	947.05	961.67	961.08	957.26	954.25	958.25
iii(b)	922.65	921.15	924.66	918.07	917.78	924.67	967.33	965.09	964.23	962.25	954.57	962.44
iii(c)	923.08	924.44	926.12	920.17	922.99	924	978.02	967.15	976.45	971.67	962.32	969.35
iv	929.05	926.10	924.57	926.48	922.33	923.08	966.14	967.86	968.1	964.56	963.47	963.18
v	917.95	919.05	920.69	915.05	919.7	918.6	952.21	954.33	956	952.39	951.05	952.75
Random Scenario												
i	11.33	11.33	11.33	11.33	11.33	11.33	11.33	11.04	11.04	11.04	11.04	11.04
ii	11.33	11.33	11.33	11.33	11.33	11.33	11.04	11.04	11.04	11.04	11.04	

configurations converged to. Consequently, according to the importance we want to give to the membership cost, we can accordingly tune the α parameter to suit our needs. Figure 5(right) reports the average size of the clusters that are created for different values of α when we start with an initial configuration of all peers in a single cluster for the linear and the logarithmic θ function.

In all three scenarios, we did not observe significant differences between selfish and altruistic peers. When a mixed strategy is used, we see that, in general, more moves are required so as to reach a stable state. However, the social cost in the resulting state is not significantly different.

Social vs Workload Cost. We also measured the progress of the social and workload cost as the peers realised more moves, i.e., during the game and until reaching a stable state. To show the difference in the two measures we used varying probabilities for each peer for the playing probability Pr. Furthermore, we selected a percentage of peers (20%) as demanding peers and doubled their query workload, and did the same with respect to content to a different 20%.

The more demanding peers are the ones that move first to accommodate their query workload. As Fig. 5(center-left) shows, the workload cost decreases faster as the demanding peers are the ones that dominate the moves. After the demanding peers find the appropriate cluster, even if they do have a higher probability to move they do no longer have anything to gain from the move. The social cost that treats all peers equally decreases linearly through all rounds (Fig. 5(left)). When we consider altruistic peers in this experiment we observe that the workload cost is not reduced much faster. The peers that have large size (i.e., more data) do not have an accordingly higher probability to move. The demanding peers that do move more move towards clusters that they can offer to, not caring if their own workload will be satisfied. To have similar results for the altruistic peers as the ones for the selfish we would have to increase the probability Pr for those peers with large size.

Hybrid Peers Besides the selfish and altruistic strategy we also have hybrid peers that take into account both individual cost and contribution when moving to another cluster. From repeating the same experiments for hybrid peers, we observe behaviors similar to the selfish and altruistic ones. The number of moves required to reach stability is slightly greater ($\approx$ 10% greater) than when using the purely selfish or altruistic strategies, but the number of clusters formed and their size is not influenced. Another difference is that

hybrid peers reduce the social cost of the system faster at first, compared to selfish ones, but since they have more options than purely selfish or altruistic peers, they require more fine-tuning moves.

Free Riders Free-riders are peers that use data items offered by others, but never contribute any content. We model a free-rider p as a selfish peer with $r(q, p) = 0$, for all q in Q. We want to examine how the appearance of free riders that is a phenomenon often met in p2p [1] influences the behaviour of the system. As the percentage of free riders increases, the social cost also increases (Fig. 5(center-right)), and finally the system degenerates to a system where each peer forms a cluster by its own since it has nothing to gain from connecting to other peers.

Summary:

- Applying the relocation policies enables the system to reach a stable state and the peers to form the desired clusters for a variety of different starting system configurations and both symmetric and asymmetric peers.
- For symmetric peers the social cost achieved depends only on the membership cost and is close to the social optimum (≈ 10) for the given setting. Similarly, when no underlying clusters exist, the social cost of the system again is very close to the social optimum.
- Clustering based solely on the content or workload distribution may be enough for dealing with symmetric peers, but both need to be considered for more general cases (i.e., asymmetric peers).
- The value of α determines the significance we put in the membership cost and is the main factor determining the number and size of the produced clusters.
- When the playing probability of the most demanding peers is increased, the workload cost is reduced faster than the social cost that treats all peers equally. (Similarly for peers with more content when dealing with altruistic peers).
- Hybrid peers require about 10% more moves to reach stability but reduce the social cost faster during the first turns compared to purely selfish or altruistic strategies.
- The presence of free riders affects the social cost of the system negatively.

5.3 Cluster Adaptation

In this set, we start from a "good" cluster configuration for given content and workload, and examine how well the reformulation protocol adapts to changes in the system conditions. We consider content, workload and topology updates (peers join/leave the system). The initial system configuration consists of clusters of peers complying to the symmetric scenario, i.e., maintaining data and posing queries belonging to a single category. We use mixed populations of both selfish and altruistic peers with different ratios, i.e., from all selfish peers ($100\% - selfish$) to all altruistic ($0\% - selfish$). We consider four different update scenarios:

- Popular existing category ($Sc1$): $x\%$ of peers randomly distributed across all clusters change their query workload to the specific category.
- New popular category ($Sc2$): $x\%$ of the peers become interested in a data category that does not have a corresponding cluster. We assume that content regarding this data category was already distributed among peers but no queries were issued concerning it.
- k Popular existing categories ($Sc3$):k data categories become more popular and $x\%$ of the peers are assigned one of those categories at random.
- k categories deletion ($Sc4$): k categories seize to be popular, i.e., different percentages of peers from k clusters change their query workload to one of the other existing categories.

We compare the social cost our policies achieve for all scenarios to not applying any changes at all. We assume the logarithmic as the theta function, $Pr = 0.5$ and $\epsilon = 10^{-4}$.

In all scenarios, our policies significantly reduce the social cost compared to the social cost the system would exhibit if no measures were taken to cope with the changes, i.e., if we applied no changes in the cluster memberships and maintained a static overlay. For example, the cost is reduced up to 1/3 of the cost of the static overlay for $Sc1$ (Fig. 6(left)). Also, the more the selfish peers in the system the better our policies perform. This is because the changes occur in the workload and not the content of the peers. For the altruistic peers to issue a relocation request, a large percentage of peers needs to change its workload. This percentage is even larger than expected, since the changes affect peers across all clusters, and for an altruistic peer to have a gain to move to another cluster, a large number of peers in both its current and the new cluster should change their workload. Also, we have to note that the social cost never reaches its initial value since a cluster of bigger size is created by this update.

The second scenario performs better than the first (Fig. 6(center-left)), as the relocation requests are more evenly distributed among the clusters, since the

peers maintaining data of the new category are also distributed across the clusters unlike the first scenario in which they were all concentrated in one. Figure 6(center-right) shows the third scenario for $k = 4$, which is similar to the second. The slightly worst social cost is due to the existence of more asymmetric peers. Figure 6(right) shows the corresponding results for the last scenario when $k = 4$.

Content Changes. A conclusion from all four scenarios is that when only the workload of the peers change, selfish peers contribute to better performance since altruistic peers take more time to react to such changes. We repeated the same experiment for all four update scenarios, when the peers change their content instead of their query workload. As Figure 7 shows, when the content changes, then the more altruistic peers exist in the system, the better the system performance. In particular, the behaviour of the social cost when the altruistic peers increase is completely analogous to that of the social cost when selfish peers increased in the case of changes in the query workload.

Hybrid Peers. While selfish peers react fast to workload changes, altruistic ones react faster to content changes. Hybrid peers consider both selfish and altruistic criteria to determine their cluster membership. We evaluate system performance when all peers are hybrid with different β for update scenario 2, when the changes refer to workload (Fig. 8(left)) and content (Fig. 8(left-center)). When β is small, the peers act more like altruistic peers being affected from content changes more than from workload changes. For β closer to 1 their behavior is similar to selfish peers. When both selfish and altruistic criteria are taken into account equally ($\beta = 0$), hybrid peers react faster than selfish peers to content changes but slower than altruistic ones and vice versa for workload changes. That is, if changes in both content and workload happen with the same frequency then adapting a hybrid strategy.

Workload and Content Change. We consider changes in the workload that also cause changes in the content of the peers. If we assume that a peer becomes interested in a topic, i.e., changes its query workload to a specific data category, then this peer will also acquire data of this category. We consider that 50% of the peers change their query workload to a category that has not any data in the system so far. Then, we measure the social cost as the number of turns increases, while considering that these same peers change incrementally 10% of their content at each turn to that particular category. Figure 8(right-center) reports our results. In the first rounds, the social cost increases as there are not enough data to satisfy the query workload of the updated peers. As the number of turns increases and the peers change more of their data, the updated peers tend to form a new cluster, effectively reducing the social cost.

Peers Joining and Leaving the System. Besides content and query workload updates that peers perform locally, another type of changes concerns peers joining and leaving the system. We examine how the system adapts when a varying percentage of peers joins the system. The peers that are selected to join are selected uniformly from all the categories and are all selfish. When peers first join, they form a cluster by their own. As they pose queries and are gradually informed about the other clusters, they move to the one that matches their workload. Thus, there is initially a considerable increase in the social cost because of the burst of new peers, but as the rounds progress and the peers join their cluster this is corrected (Fig. 8(right)). However, the social cost remains slightly larger than its original value since the system now has more peers. When peers leave the system, the social cost is actually reduced as the number of peers is reduced. In this case, no immediate adaptation is required from the reformulation protocols. If a large percentage of the members of a cluster leave it, then the other member may require to move to other clusters to find more results to their queries, or peers from other clusters might be motivated to move to this cluster to exploit the low membership cost.

Cluster Reformulation vs Clustering from Scratch Another alternative to cope with changes, rather than applying the reformulation policies, would be to apply the clustering procedure from scratch so as to take into account the new data and query workload distributions. Such an approach is expected to result in a better clustering scheme with a lower social cost than the reformulation protocols. However, we argue that this would entail a much larger communication cost with respect to the gain in the social cost it would offer. To demonstrate this we consider the second update scenario where a new category of data becomes popular. To apply clustering from scratch, we first "decluster" the peers, considering each as forming its own cluster and then apply our policies.

We measure the social cost in this case after the system has reached stability for various percentages of updates peers (Fig. 11(right)). Compared to the case where we apply the reformulation policies for adaptation only (Fig. 6(center)), the social cost in this case is lower up to 10% for all selfish peers. Furthermore, the re-clustering is able to perform well even in the case of all altruistic peers in which adaptation does not work when only the workload changes. However, the re-clustering technique is much more expensive. We measure the number of turns for both cases to achieve a stable state and also the number of movements to different clusters that take place. The re-clustering technique requires for both all selfish and all altruistic peers

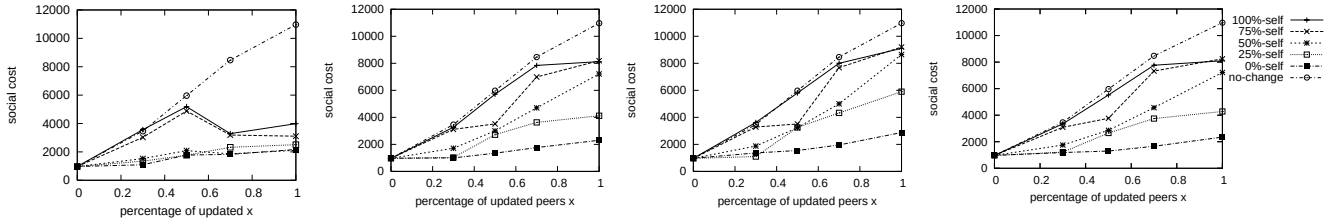

Figure 7: Social cost for different percentages of updated peers (content) for (left) scenario 1, (center-left) scenario 2, (center-right) scenario 3 and (right) scenario 4.

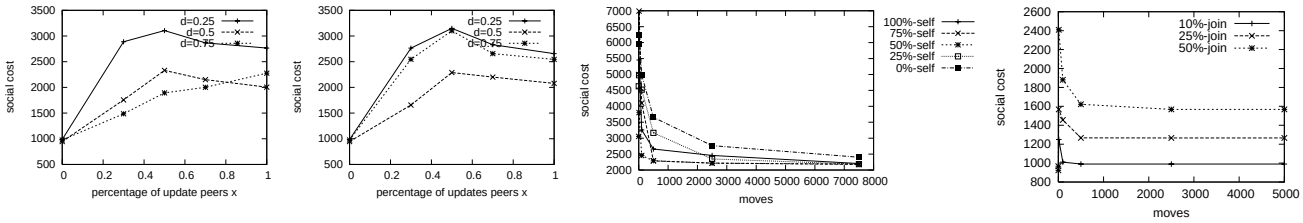

Figure 8: Social cost for (left) workload and (left-center) content changes for hybrid peers, (center-right) different percentages of updating both workload and content, and (right) peers joining the system.

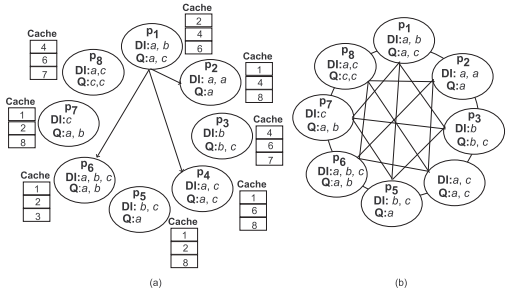

Figure 9: (left) A caching scheme and (b) the corresponding clustered scheme

about 250 turns, while the reformulation policies reach stability in only 10 turns. As far as movements, the reclustering realises $O(N)$ movements, where N is the number of peers. Even in the case where only 10% of the peers make the change in their workload, it still requires over 10000 movements, whereas the number of movements in reformulation, depends on the number of the updated peers and is around 1100 when 10% of the peers update their workload, while it remains below 10000 even for 100% of updated peers.

Summary:

- The relocation policies are able to cope with changes efficiently for a wide variety of update scenarios affecting the content, workload and population of peers. Their application reduces the social cost of the system up to 1/3 compared to a static overlay in which the changes are ignored.

- Selfish peers react faster to workload changes, while altruistic ones are more sensitive to content changes. Hybrid peers perform according to their b value and provide a compromise between sensitivity to workload and content changes.

- When peers join the system the social cost is initially increased but the policies are able to fast accommodate the new peers to their appropriate cluster.

- Re-clustering from scratch taking into account the changed conditions yields a system with a social cost up to 10% lower than applying the relocation policies and is not affected by the existence of selfish or altruistic peers that do not react to changes in the content or workload respectively. However, re-clustering imposes a considerable overhead in both turns (250 turns instead of 10 in cluster adaptation) and moves (always $\geq |P|$, while in cluster adaptation it depends on the number of affected by the changes peers).

5.4 Comparison with a Caching Scheme

Caching is widely used to improve performance in distributed systems. It either refers to caching the results of previous queries to facilitate future similar queries or caching the peers that provided answers to previous queries and forwarding future queries to them. We consider a system based on the second idea and similar to [17]. The cache entries form an implicit overlay network in which peers that are considered to share similar interests are connected. Figure 9 shows a simple example with six peers and their content and query

workloads, and how the peers would be connected (a) if a caching scheme was used or (b) if clustering was applied. Queries are first forwarded to the peers in the query's origin cache and if the results are not satisfying, then the peer resorts to other more inefficient methods of search, i.e., flooding.

In an interesting variation of this strategy (*transitive*), a peer sends its queries to the peers in its cache as a message with a $TTL = 2$ prompting the peers that receive them to also forward them to the peers in their own caches. This is based on the idea that a peer has common interests with the peers that have common interests with the peers in its own cache, and enables the peers to discover more peers with results to their queries without flooding.

After each query, the peer that issued it updates its cache. For the peers already in the cache, their recall value (or any measure indicating their usefulness) is updated according to the results for the latest query. For the peers which have returned results and are not in the cache an update cache policy is deployed. We discern between two variations. In the first variation (*update(1)*), the peer with the highest recall from the ones not in the cache is selected. If there is enough space in the cache then it is just added. Otherwise, it replaces the peer with the lowest recall value that was found in the cache. The alternative $update(x)$, selects the x new peers with the highest recall for either adding them or replacing them in the cache. This alternative is able to adapt faster to changes in the interests of a peer as it maintains a fresher cache.

We want to compare our clustering scheme to a cache-based scheme like the one we described above. We model the cache-based system by considering a random graph where each node representing a peer has a maximum out-degree d of which a constant number of 3 links is devoted for the formation of the graph, while the rest $d-3$ links are allocated as spaces in the peer's cache. To accomplish a fair comparison with our scheme, we set d equal to the number of links a peer establishes in its cluster. We first consider a linear θ function for the topology within each cluster, i.e., a fully connected graph and then, a chord-like topology in which the number of links is equal to $log(|c_i|)$.

Symmetry. The cache-based scheme is expected to work better than clustering when the interests among peers are not symmetric. That is, when the peers offer content different from their query workload. Each peer has a cache with the peers that maintain its content of interest, while it belongs to the caches of peers that are interested in its own content. In contrast, for symmetric peers, while the caching scheme needs for both peers to discover each other, in clustering, it suffices for one of them to discover the other to establish the bidirectional link between them (i.e., for them to become members of the same cluster).

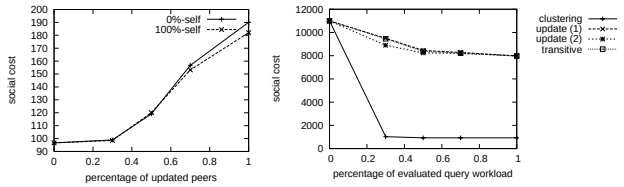

Figure 11: (left) Social cost with re-clustering and (right) clustering vs caching for changes in the workload.

To demonstrate this we repeat the first experiment of our first set of experiments, for both symmetric and asymmetric peers. We measure the social cost for all caching scheme variations, setting x equal to 2, after intervals in which about 1/5 of the local query workload is issued by each peer and compare it to that of the clustering scheme. As the membership cost for the caching scheme we consider the cost of the d links and as the loss in the recall, the percentage of results a peer obtains through flooding rather than its cache. We assume same size clusters and compare the caching scheme for $d = 1000$ with fully connected clusters, and for $d = 10$ with chord-like ones.

Our results confirm our expectation but also indicate that the topology within the clusters that determines their membership cost also plays an important factor. In particular, when θ is logarithmic yielding a low membership cost, clustering outperforms caching for both symmetric (Fig. 10(center-left)) and asymmetric peers (Fig. 10(right)). For symmetric peers, with $d = 10$ links, the efficiently structured clustering scheme reaches all 1000 peers of the same category, while in caching only 10 such peers are reached. In contrast, when θ is linear, the difference between the performance of clustering and caching becomes much smaller for symmetric peers (Fig. 10(left)), while caching outperforms clustering for the asymmetric ones (Fig. 10(center-right)). Both replace policies behave similarly since the peers do not change the category of their workload or content. The transitivity property is only useful when the peers are symmetric, and again it does not suffice to achieve the performance we have with clustering.

Coping with Changes. We consider the asymmetric scenario and the peers organised in a chord-like topology. We repeat scenario 3 of the third set of experiments and measure the social cost after different percentages of the local queries workload have been processed (Fig 11(right)). Clustering adapts to changes more efficiently. While caching changes one to two neighbours, clustering changes all neighbours at once, thus, achieving lower social cost faster.

Summary:

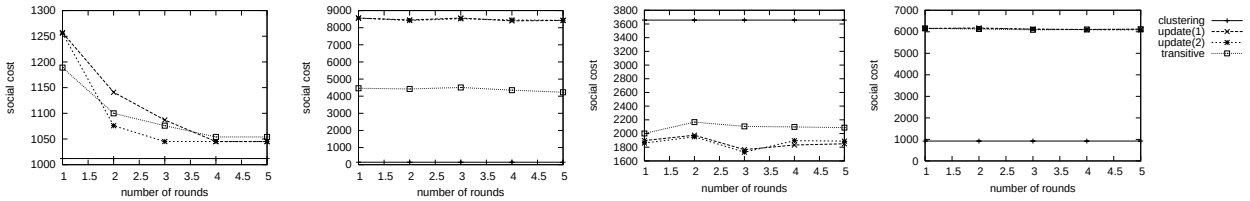

Figure 10: Caching for symmetric peers vs clustering (left) with linear and (center-left) logarithmic θ function, and caching for asymmetric peers with (center-right) linear and (right) logarithmic θ.

- Caching is more appropriate for asymmetric peers, while clustering is more suitable for symmetric ones. However, deploying efficient topological structures that significantly reduce the membership cost within the clusters, such as chord-like topologies, enable clustering to outperform caching even with asymmetric peers.

- Clustering reacts faster and more effectively to changes. While caching changes a subset of neighbours (one to two) after each query, when the cluster membership changes the entire neighbour set is changed.

6 Related Work

Game theoretic approaches have been applied to model the behavior of peers in p2p systems. In [7], the creation of an Internet-like network is modeled as a game with peers acting as selfish agents without central coordination. The aim of the game is for each peer to choose the peers with which to establish links. The peers pay for the creation of a link, but gain by reducing the shortest distance to any other peer in the system. In our approach, instead of establishing links randomly, we consider content and query workload for creating clusters of peers with similar properties. [13] considers a more sophisticated model, in which strict bounds are enforced on the out-degree of the peers, links are directed and peers are allowed to express preferences regarding the choice of their neighbors. Our approach can be viewed as setting these preferences based on recall benefits. In [16], the authors show that allowing peers to act completely freely performs much worse than collaboration, and prove that even a static p2p system of selfish peers may never reach convergence. This result is in accordance to our findings that show that in only specific scenarios, we reach a Nash equilibrium. In [20], altruistic peers determine the level of their contribution to the system based on a utility function that depends on a variety of parameters such as the amount of data they upload and download, whereas in our altruistic policy, the choice of the cluster a peer joins depends on its contribution to this cluster.

Recent research efforts are focusing on organizing peers in clusters. In most cases, the focus is on cluster formation and query processing using the clusters and the adaptation of the overlay to changing conditions is not addressed. In [9], a superpeer-based architecture is proposed under which peers with common interests are organized based on their caches. The paper exploits the idea of [17], and since it is based on caches it implicitly addresses the issue of cluster adaptation, but does not focus on it. Furthermore, as a cache-based scheme it is better suited for selfish symmetric peers, while our model can encompass more types of peers. In [3], peers are partitioned into topic segments based on their documents. A fixed set of M clusters is assumed, each one corresponding to a topic segment. Knowledge of the M centroid is global. Clusters of peers are formed in [19] based on the semantic categories of their documents; the semantic categories are predefined. Similarly, [5] assumes predefined classification hierarchies based on which queries and documents are categorized. Instead of predefined categories, [8] use a learning approach that based on generalizing the data the peers share, learns the semantic categories they belong to and then uses those to form the appropriate clusters. Clustering in [14] is based on the schemes of the peers and on predefined policies provided by human experts. Besides clustering based on peers content, clustering based on other common features, such as the interests of peers [11], is possible. In [6], clustering is first applied on the documents of each peer, and then recursively on the derived feature vectors by selected peer representatives. While this approach does not assume predefined categories, it still requires the use of cluster representatives unlike our uncoordinated protocol. In [4], peers maintain sets of guide rules, which are formed by the users either explicitly based on their interests, or implicitly through query history, and point to other peers in the system, thus defining semantic clusters. Queries are then first routed through relevant guide rules. A somewhat different approach to clustering is taken in pSearch [18]. pSearch is a system that maps the documents of the peers on a DHT, based on their term vectors and exploiting only the most important terms. Thus, semantically related documents are "clustered" in the DHT, enabling the search process to limit the

space it has to consider.

A preliminary, short version of this paper has been presented in [12].

7 Conclusions

In this paper, we have modelled peers in a clustered overlay as players that dynamically change the set of clusters they belong to according to an individual utility function, which is based on a cluster membership cost and query recall. We modelled both selfish peers that aim at minimizing their individual cost, i.e., maximizing their recall, and altruistic peers that try to maximize their contribution to other peers. In addition, we have defined measures for evaluating global system quality. We have proposed corresponding relocation policies for both selfish and altruistic peers. Our experimental results showed how by following these policies, the peers can change the clustered overlay to reflect the current system conditions thus, gradually correct system performance. Furthermore, our results indicated that the proposed policies can also be used for the initial construction of clusters, when the underlying data distribution permits it.

There are many open issues for future work. One issue is to derive theoretical results regarding convergence perhaps by considering more restricted forms of the cluster formation problem. Also, practical issues, such as the maximum number of clusters that a realistic system can support and the expected look-up cost with respect to the number of clusters and their sizes, are worth exploring.

References

[1] E. Adar and B. A. Huberman. Free riding on gnutella. *First Monday*, 5(10), 2000.

[2] N. Bansal, F. Chiang, N. Koudas, and F. W. Tompa. Seeking stable clusters in the blogosphere. In *VLDB*, 2007.

[3] M. Bawa, G. Manku, and P. Raghavan. Sets: Search enhanced by topic segmentation. In *SIGIR*, 2003.

[4] E. Cohen, A. Fiat, and H. Kaplan. Associative search in peer to peer networks: Harnessing latent semantics. In *INFOCOM*, 2003.

[5] A. Crespo and H. Garcia-Molina. Semantic overlay networks for p2p systems, technical report, computer science department, stanford university, 2002.

[6] C. Doulkeridis, K. Norvag, and M. Vazirgiannis. Desent: decentralized and distributed semantic overlay generation in p2p networks. *JSAC*, 25(1):25–34, 2007.

[7] A. Fabrikant, A. Luthra, E. Maneva, C. H. Papadimitriou, and S. Shenker. On a network creation game. In *PODC*, 2003.

[8] A. Fast, D. Jensen, and B. N. Levine. Creating social networks to improve peertopeer networking. In *KDD*, 2005.

[9] P. Garbacki, D. H. J. Epema, and M. van Steen. Optimizing peer relationships in a super-peer network. In *ICDCS*, 2007.

[10] S. B. Handurukande, A.-M. Kermarrec, F. L. Fessant, L. Massouli, and S. Patarin. Peer sharing behaviour in the edonkey network, and implications for the design of server-less file sharing systems. In *EuroSys*, 2006.

[11] M. Khambatti, K. Ryu, and P. Dasgupta. Efficient discovery of implicitly formed peer-to-peer communities. *IJPDSN*, 5(4):155–164, 2002.

[12] G. Koloniari and E. Pitoura. Recall-based cluster reformulation by selfish peers. In *NetDB*, 2008.

[13] N. Laoutaris, G. Smaragdakis, A. Bestavros, and J. W. Byers. Implications of selfish neighbor selection in overlay networks. In *INFOCOM*, 2007.

[14] A. Loser, F. Naumann, W. Siberski, W. Nejdl, and U. Thaden. Semantic overlay clusters within super-peer networks. In *DBISP2P*, 2003.

[15] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *IMC*, 2007.

[16] T. Moscibroda, S. Schmid, and R. Wattenhofer. On the topologies formed by selfish peers. In *PODC*, 2006.

[17] K. Sripanidkulchai, B. Maggs, and H. Zhang. Efficient content location using interest-based locality in peer-to-peer systems. In *INFOCOM*, 2003.

[18] C. Tang and Z. X. andMallik Mahalingam. psearch: information retrieval in structured overlays. *Computer Communication Review*, 33(1):89–94, 2003.

[19] P. Triantafillou, C. Xiruhaki, M. Koubarakis, and N. Ntarmos. Towards high performance peer-to-peer content and resource sharing systems. In *CIDR*, 2003.

[20] D. K. Vassilakis and V. Vassalos. Modelling real p2p networks: The effect of altruism. In *P2P*, 2007.