Computer Networks 54 (2010) 2056–2070

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

# Affinity P2P: A self-organizing content-based locality-aware collaborative peer-to-peer network

Juan M. Tirado [a,*], Daniel Higuero [a], Florin Isaila [a], Jesús Carretero [a], Adriana Iamnitchi [b]

[a] *Computer Architecture Group, Carlos III University, Madrid, Spain*
[b] *Computer Science and Engineering Department, University of South Florida, Tampa, FL, USA*

ARTICLE INFO

*Article history:*
Available online 1 June 2010

*Keywords:*
Peer-to-peer networks
Content-aware network
Data locality

ABSTRACT

The last years have brought a dramatic increase in the popularity of collaborative Web 2.0 sites. According to recent evaluations, this phenomenon accounts for a large share of Internet traffic and significantly augments the load on the end-servers of Web 2.0 sites. In this paper, we show how collaborative classifications extracted from Web 2.0-like sites can be leveraged in the design of a self-organizing peer-to-peer network in order to distribute data in a scalable manner while preserving a high-content locality. We propose Affinity P2P (AP2P), a novel cluster-based locality-aware self-organizing peer-to-peer network. AP2P self-organizes in order to improve content locality using a novel affinity-based metric for estimating the distance between clusters of nodes sharing similar content. Searches in AP2P are directed to the cluster of interests, where a logarithmic-time parallel flooding algorithm provides high recall, low latency, and low communication overhead. The order of clusters is periodically changed using a greedy cluster placement algorithm, which reorganizes clusters based on affinity in order to increase the locality of related content. The experimental and analytical results demonstrate that the locality-aware cluster-based organization of content offers substantial benefits, achieving an average latency improvement of 45%, and up to 12% increase in search recall.

## 1. Introduction

The last years have shown an unprecedented increase in the Internet traffic, boosted by the emergence of Web 2.0 sites such as social networking, wikis and blogs. Web 2.0 facilitates the publishing of user-generated content and the creation of social networks of people sharing common interests. According to Alexa [3], YouTube is the largest and one of the fastest growing user-generated content sites for video sharing. Due to its large popularity and increasing traffic, YouTube offers slow response times to user requests [7,31]. This fact is mostly due to the limits on scalability placed by the server-based architecture of YouTube-like sites.

On the other hand, the peer-to-peer (P2P) architecture has become a powerful paradigm for the design of large-scale content-sharing systems. P2P architectures offer a scalable platform for applications such as file sharing or video streaming. Existing P2P video streaming protocols assume that the videos are long and served independently from each other [8]. However, a Web 2.0 site such as YouTube is characterized by short video sizes, collaborative content sharing and recommendation-based surfing. These characteristics make interest-based content locality of critical importance for the efficient searching and retrieving of videos in a scalable P2P system.

In this paper, we propose and evaluate Affinity P2P (AP2P), a novel scalable system for the management of collaboratively-generated content in YouTube-like networks. The system targets to increase content locality and to improve search latency and recall in a large-scale distributed

* Corresponding author. Tel.: +34 91 624 62 60.
*E-mail address:* jtirado@inf.uc3m.es (J.M. Tirado).

1389-1286/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2010.04.016

system. AP2P is a cluster-based structured P2P network that self-organizes in order to increase content locality, while offering a high degree of availability.

The main contributions of this paper are the following. First, AP2P exploits Web 2.0 collaborative classification of content in order to automatically improve content locality at two levels. At the first level, peers autonomously join or leave clusters according to their content. At the second level, the clusters of peers self-organize on the overlay targeting to maximize the global inter-cluster locality. Second, we propose a novel affinity-based metric for estimating the distance between clusters of interests. Third, we use a logarithmic-time parallel flooding algorithm that aims to achieve high data recall, high tolerance to node failure, and no redundant communication. Fourth, we propose a greedy cluster placement algorithm, which reorganizes clusters based on affinity in order to increase content locality.

The remainder of this paper is organized as follows. Section 2 presents an overview of our system. In Section 3 we discuss related work. Section 4 presents in detail the system architecture. Section 5 presents the experimental results. Finally, Section 6 concludes and discusses our future plans.

## 2. System overview

In this paper, we target two main objectives. The first is to investigate to what extent collaborative classifications can be useful in designing an efficient P2P system. The second is to design a P2P network that self-organizes in order to increase content locality and to achieve a high data recall with low latency and low network traffic. The AP2P system has the following major features:

- AP2P network is a structured Chord-like [35] overlay with three main differences when compared to Chord. First, AP2P logical nodes correspond to Chord nodes, with the difference that Chord identifiers are generated through uniform hashing, while AP2P identifiers are a concatenation of a cluster identifier and a node identifier generated through uniform hashing. While Chord generates a global uniform distribution of node identifiers, AP2P locally distributes node identifiers uniformly inside each cluster. Second, resource placement is not done based on node identifiers: each peer stores the resources it shares, as in many real applications scenarios. Third, interest-based clusters of peers are supported by extending the identifier space with a cluster identifier field. All the other overlay maintenance operations of Chord are preserved with the same functionality.
- Each cluster of peers is labeled with a category. Labeling a cluster with a category does not mean that all the resources of its peers are classified in the same category, but that the peers have a sufficient number of resources classified in that category.
- Each peer joins at least one primary cluster, corresponding to the category to which the highest number of its resources are classified. Additionally, a peer may join one or several secondary clusters, if a sufficiently large number of its resources are classified in the category corresponding to the secondary clusters. Peers automatically change their primary and secondary clusters over time, adapting to changes in the resources they make available.
- The semantic closeness between any two clusters is estimated by an affinity matrix. The affinity matrix is leveraged in order to place clusters with close semantic content close to each other in the overlay. The affinity matrix is not an exact global measure of content distribution, but rather an approximation periodically updated.
- The lookup algorithm localizes first the cluster of interest for the search and, subsequently, performs a fast parallel logarithmic flooding based on a recursive doubling strategy. The parallelism of the algorithm assures that if a node fails, the search can continue on alternative parallel branches.
- Periodically, clusters are reorganized in the overlay based on the changing affinities of the peers. This approach ensures that affine clusters are placed close to each other, thus increasing the locality of related content.

We note that AP2P offers a general solution in several respects. First, a resource can be classified in more than one category. Second, the classification is not limited to Web 2.0 based collaborative classifications: any clustering algorithm can be used in order to classify resources. Third, the categories can be split into subcategories without affecting the correctness of the algorithms presented in this paper.

## 3. Related work

P2P networks can be classified into structured and unstructured ones. Structured networks consist of a set of nodes interconnected in a regular topology. Examples of structured networks include Chord [35], Pastry [30], CAN [26], and Bamboo [27]. They are typically built based on the Distributed Hash Table (DHT) abstraction, which guarantees the location of information in a small number of hops, while incurring the cost of maintaining the overlay network topology. For unstructured networks, such as Gnutella [5] or Freenet [9], nodes can connect in an arbitrary way and the maintenance costs are reduced. On the other hand, the search relies on a flooding mechanism and the localization of content is not guaranteed.

Table 1 provides a comparison of AP2P with two well known P2P networks: the structured Chord [35] and the unstructured Gnutella. AP2P is structured in a ring topology as Chord, and allows natural replication of contents as Gnutella. Unlike Chord and Gnutella, AP2P is structured in overlapping clusters and self-organizing for high-content locality. AP2P uses informed search [21] based on semantic cluster information: the search process locates first the cluster of interest and then performs a logarithmic parallel flooding. Like in Chord, and unlike Gnutella, no redundant messages are generated for searching in AP2P.

**Table 1**
Comparison of AP2P with Chord and Gnutella.

| | Chord | Gnutella | AP2P |
|---|---|---|---|
| Overlay characterization | Structured | Unstructured | Structured |
| Topology | Ring | Random | Ring |
| Content replication support (implicit) | No | Yes | Yes |
| Cluster support | No | No | Yes |
| Locality oriented | No | No | Yes |
| Search mechanism | Logarithmic keylookup | Non-informed flooding | Logarithmic lookup + informed flooding |
| Communication redundancy | No | Yes | No |

### *3.1. Interest-based self-organizing P2P networks*

Sripanidkulchai et al. [34] propose an improvement of Gnutella scalability by allowing peers to connect among each other via shortcuts selected based on a comparison between shared resources. Similarly, REMINDIN' [37] exploits previous successful searches in order to improve the efficiency of future queries by adaptive semantic-based routing. Our approach differs in that we target to increase the content locality at both local and global level (nodes and clusters). Löser et al. [20] propose a semantic overlay where super-peers act as centroids of semantic clusters. New nodes join the network by connecting themselves with super-peers that match node information contents attending to different clustering and matching policies. iCluster [25] uses a rewiring method to connect similar peers with different interests throughout the network. In turn, the aforementioned solutions focus on locally optimizing the routing based on content at node level.

The common difficulty in the aforementioned approaches is to determine which metric or estimator is the most suitable for connecting peers. Data sharing graphs [16] capture common user interest in data at resource granularity. These graphs can be used to dynamically organize files in clusters of interest [15]. Condie et al. [11] propose an adaptive topology based on local trust scores. We propose a new metric for estimating the degree of interest sharing based on user-agreed classification: we assume the existence of social-based classification and show that this knowledge can be leveraged in order to improve the content locality and, therefore, the probability of information finding.

### *3.2. Semantic overlay P2P networks*

Crespo and Garcia Molina [12] propose a cluster-based semantic overlay network (SON). When a node joins the P2P network, a classifier determines the most suitable clusters according to the resources that the node shares. P2PDating [22] extends the SON idea to a P2P overlay network whose peers have complete autonomy. The nodes may group themselves based on several estimators such as level of overlap between documents of two peers, the similarity between their documents, history of the peer, level of trust. Our approach complements the SON and P2PDating work in at least three main directions: we introduce a metric for semantic closeness of clusters, we present algorithms for self-organization of content at both node level and cluster level based on collaborative classifications, and we describe a fast parallel flooding search algorithm.

Cohen et al. [10] assume that searching some items implies searching other related items with a certain probability. Peers perform blind flooding searches and store information about peers that share both the searched items and related resources. A later search of related items is faster, as peers can use previously gathered information. AP2P differs from this approach as it places affine nodes close to each other using a clustering approach over a structured network.

Aberer et al. [2] propose a semantic overlay network layered on top of P-Grid overlay. The semantic information described by RDF (Resource Description Framework) triples is stored in the overlay network. Unlike in AP2P, the semantic information is leveraged neither in the network construction nor in resource placement.

Rostami et al. [29] propose a system with the objective of reducing P2P traffic by employing an external ontology, which filters only the semantically-relevant queries. Peers consult the ontology service for P2P routing. In contrast, AP2P reduces traffic by self-organizing for higher locality and eliminating redundant messages by using an efficient parallel flooding algorithm.

In pNear [33] every peer chooses its neighbors based on semantic similarity evaluated by a customizable metric. This selfish approach trades off content locality for peer independence, and may reduce the search efficiency, as the quality of the routing process depends on how effective the peers cluster themselves. AP2P leverages the common knowledge to global benefit of all peers by global cluster reordering.

Scholosser et al. [32] propose an improvement of HyperCuP overlay by mapping an ontology onto an hypercube topology. Using this hierarchical hypercubes, a query can be routed through the network using semantic links. Triantafillou et al. [38] target high performance in a cluster-based P2P system by distributing the load among peers. Cluster leaders monitor request statistics in order to adapt the assignation of categories to non-overloaded clusters. Our approach changes the order of the clusters in the network to increase content locality, and not to balance the load.

Anwar et al. [4] evaluate the Orkut social network and propose to leverage common user interests for setting up an overlay topology. They show that this approach reduces the latency in group communication compared to other application-level multicast techniques, but do not provide any formal definition of how such a network can be created. Pouwelse et al. [24] propose a social P2P network in which communities are built based on similar interests of peers. In order to locate content, they use an algorithm called Buddycast that uses a flooding algorithm to share preferences with other peers. Our approach places users into semantically-related clusters according to the resources they share in the network.

Voulgaris et al. [40] propose a proactive gossip-based management network that uses a semantic proximity function to define the semantically closest neighbors of a node. Nodes try periodically to find new neighbors that improve this function in order to increase the similarity between nodes. Koloniari and Pitoura [19] follow a similar approach based on selecting clusters depending on the rate of successful queries. Nodes analyze the recall obtained from previous queries and decide which clusters are more suitable using a selfish or an altruistic relocation strategy. Lodi et al. [23] present a strategy that permits nodes to choose among several SONs using a semantic similarity function. We share with these solutions the target to dynamically increase the similarity among neighbors. However AP2P permits a node to belong simultaneously to several clusters and we propose a dynamic reorganization at both node and cluster level.

## 4. System architecture

This section describes the AP2P system architecture, which builds on the general architecture [1] for overlay networks depicted in Fig. 1. The P2P layer offers basic interconnection services at clusters level such as joining or leaving a cluster and overlay maintenance. The semantic layer provides the management of content-based information and offers high level services to the applications such as strategic node joins, efficient content-based searches.

### 4.1. Concepts and formal definitions

The AP2P network interconnects nodes in a ring topology in a similar manner as other overlays, such as Chord [35] or Pastry [30]. However, there are two main departures from the Chord and Pastry approaches. First, like in CAN [26] and SON [12] the presence of the nodes in the overlay is *logical*, i.e. a *physical* node may have several appearances at various locations of the network. Second, our architecture provides support for clusters of nodes. Consequently, a physical node may have several logical presences in different clusters. For example, Fig. 2 shows a ring with three clusters $C_3$, $C_5$ and $C_9$, corresponding to Music, Movies and Games categories. The clusters may

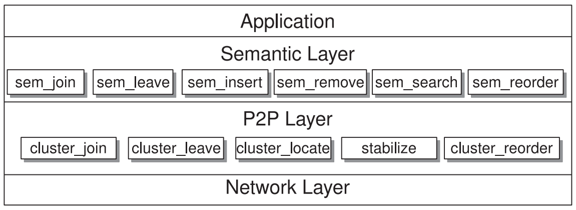


**Fig. 1.** AP2P architecture layers.

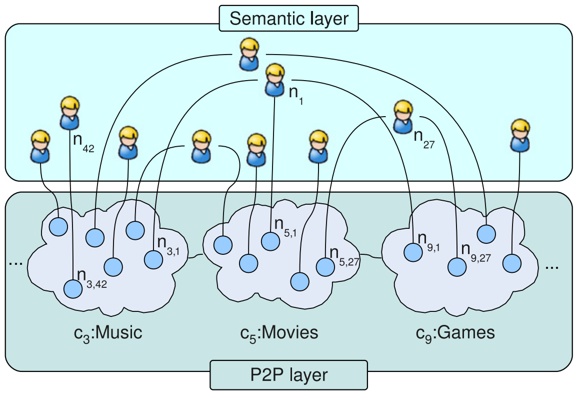


**Fig. 2.** Example of physical node joining several clusters as logical nodes.

share physical nodes, e.g. node $n_1$ participates in all three clusters.

A *physical node* represents a user contributing a collection of resources in the network. Any physical node joining AP2P is assigned a unique identifier $n_j$, where $j = 0, 1, \ldots, (N - 1)$, and $N$ is the number of physical nodes in the network. Each node has a node identifier that is computed as in Chord, by hashing the IP address of the node.

A *resource* is a persistent object characterized by a set of attributes, including a unique identifier. Unlike in Chord, the resource identifier space is different from the peer identifier space. A resource identifier does not dictate on which node the resource will be placed. Each resource can be classified in one or several categories (e.g., Music, Movies, etc.). A user can share resources classified in various categories. AP2P leverages this classification for computing the degree of interest of a certain user in a cluster. Based on this information, peers self-organize into *clusters* by joining or leaving different parts of the overlay. A cluster is denoted by $C_k$, where $k = 0, 1, \ldots, (c - 1)$ and $c$ is the total number of clusters. The purpose of introducing a cluster organization of nodes is twofold. First, we want to increase locality of related content in the same cluster. Second, we want to leverage this content locality in the search process to reduce latency, and improve recall without increasing the number of messages.

In the general case, one node shares resources classified in different semantic categories, e.g. Music and Movies. In order to address this scenario, a physical node may join several clusters as a *logical node*. A logical node represents the presence of a physical node inside a cluster. AP2P logical node identifier is denoted by $n_{kj}$, whose value is the concatenation of cluster identifier and node identifier: $n_{kj} = [C_k|n_j]$. Fig. 2 shows several physical nodes in the semantic layer. Physical nodes $n_1$, $n_{27}$ and $n_{42}$ join different clusters as logical nodes in the P2P layer. For instance, $n_1$ is interested in music, movies and games, thus joins clusters: $C_3$, $C_5$ and $C_9$ with logical nodes $n_{3,1}$, $n_{5,1}$ and $n_{9,1}$, respectively.

The number of resources of a node $n_j$ classified in the category associated to cluster $k$ is denoted by $r_{kj}$. Consequently, a cluster $C_k$ can be formally defined as:

$$C_k = \{n_{kj} | n_j \text{ is a physical node sharing } r_{kj} \text{ resources, where } r_{kj} > 0 \text{ and the node meets membership conditions}\}. \quad (1)$$

AP2P network can be defined as the union of all clusters:

$$AP2P = \bigcup_{k=0}^{c-1} C_k. \quad (2)$$

The definition of a node identifier as a concatenation of cluster and physical node identifiers implies that each cluster is a segment of the AP2P ring. According to the previous definition, we can model the YouTube content network as *YouTube* = $\{C_0, \ldots, C_{14}\}$ = {*Music*,*Entertainment*,..., *Comedy*}.

Each physical node joins at least one cluster, called *primary cluster*. The primary cluster $p_j$ of a node $n_j$ corresponds to the category in which the highest number of node resources are classified. Formally, the index of the primary cluster can be calculated as:

$$p_j = t \text{ such that } r_{tj} = \max_{k=0}^{c-1}(r_{kj}), \quad (3)$$

where $r_{kj}$ is the number of resources of node $n_j$ classified to the category associated to cluster $C_k$ and $c$ is the number of clusters. A physical node may join additional clusters, called *secondary clusters*, depending on absolute and relative thresholds, as it will be explained in Section 4.3.1.

Each logical node maintains one *finger table* of size $2 \times m$, where $m$ is the number of table entries. The entry $i$ in the finger table points to the logical node at distance $2^i$ in both clockwise and counterclockwise directions, where distance is the path length between two nodes following exclusively ring edges. Unlike in Chord where the $i$th entry in the finger table of node with identifier $n$ points to the first node that succeeds $n + 2^i$, in AP2P the $i$th entry points to the logical node at distance $2^i$. Fig. 3 shows an example of a clockwise finger table with four entries.

AP2P clusters are mapped on the ring topology in an order that is decided based on content locality (more details about cluster ordering on the ring are given in Section 4.3.4). Each node keeps the order of cluster placement on the ring in a *cluster placement order table*, which is updated each time a new cluster ordering is computed. This table is used in the locate operation for identifying the cluster, as described in Section 4.2.

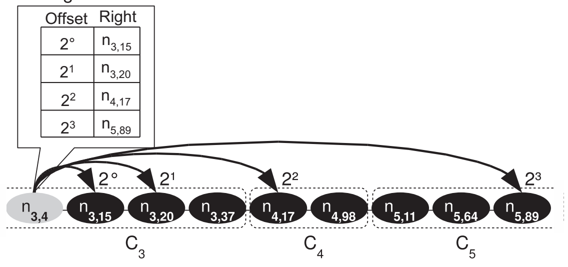


**Fig. 3.** Example of a finger table with four entries. Entry $i$ in the finger table points to the node at distance $2^i$. Note that fingers are cluster-agnostic. For instance, the last finger of node $n_{3,4}$ from cluster $C_3$ points to a node in cluster $C_5$.

One goal of our architecture is to achieve high locality for related resources. In a real world scenario a node may share resources classified in different categories, given that a user may have different interests. The *affinity matrix* approximates the level of interest sharing among all pairs of clusters in the network and therefore, the probability that a user from a cluster may share content classified in the category of an other cluster. This global approximation of the network content can be used to dynamically adapt its configuration to maximize the content locality. More formally, the affinity matrix $(A)$ is a square $(c-1) \times (c-1)$ matrix of the form:

$$\mathbf{A} = \begin{bmatrix} a_{00} & \cdots & a_{0(c-1)} \\ \vdots & \ddots & \vdots \\ a_{(c-1)0} & \cdots & a_{(c-1)(c-1)} \end{bmatrix},$$

where each element $a_{kt}$ is computed as:

$$a_{kt} = \frac{R_{kt} + R_{tk}}{\sum_{s=0}^{c-1}(R_{ks} + R_{sk})}, \quad (4)$$

where $R_{kt}$ is the number of resources accessible through cluster $C_k$ and classified in the category of cluster $C_t$. The $a_{kt}$ values represent the sum of the resources stored by the nodes with primary cluster $C_k$ and classified in the category of cluster $C_t$ and the resources stored in nodes with primary cluster $C_t$ and classified in the category of cluster $C_k$ divided by the total number of resources accessible on cluster $C_k$. The affinity matrix is used to determine the placement of the clusters in the overlay: affine clusters should be placed near each other in order to improve cross-cluster locality. This algorithm is described in Section 4.3.4. Placing clusters with affine content close to each other is an heuristic to minimize the distance between them. Searches from one cluster to another are with high probability targeted to affine clusters; in this situation the latency to locate the target cluster is reduced.

The affinity matrix is not an exact value of the system state, but an approximation. Our solution is similar to the one used in [13] and employs agents, which traverse the whole overlay in order to compute $R_{kt}$ values and to distribute them. The agents are spawned periodically from nodes bordering the clusters in clockwise directions. Each agent travels using the successor links and sum up the contribution of each node to all clusters $(R_{kt})$ values. These values are an approximation, as nodes may leave and join the network, while the agent traverses the overlay. In the second pass, the agent distributes the $R_{kt}$ values to all nodes, which can compute the new affinity matrix. New nodes joining the overlay receive the current affinity matrix from their neighbors. The period between launches of new agents to recalculate the affinity matrix should be adapted based on the success rate of agents. This problem is outside the scope of this paper and it is the subject of future work.

### *4.2. P2P layer*

The P2P layer depicted in Fig. 1 provides mechanisms for the construction and maintenance of the P2P overlay network. This layer is not semantics-aware. The main operations of the P2P layer are:

- *cluster_join*($n_{kj}, n_i$) receives as parameters a logical node $n_{kj}$ and an arbitrary physical node $n_i$ and attaches the physical node $n_j$ to the cluster $C_k$ through the intermediation of $n_i$. Subsequently, the successor and predecessor nodes are informed and the Chord-like stabilize method is called in order to update the corresponding finger tables. This operation is called by the upper content-aware join operation after deciding the primary and secondary clusters of a physical node.
- *cluster_leave*($n_{kj}$) receives as parameter a logical node $n_{kj}$ and removes the physical node $n_j$ from the cluster $C_k$. When leaving, the successor and predecessor nodes are informed and the Chord-like stabilize function is called in order to update the finger tables of all involved nodes.
- *cluster_locate*($C_k$) locates and returns an arbitrary node from the cluster $C_k$. This operation works similarly to the key retrieve algorithm from Chord. The invoking node looks at its finger table for an entry from the target cluster. If the node does not find the target cluster, the locate operation consults the cluster placement order table and delegates the locate operation to a logical node closer to the target cluster. This operation is used by join and stabilize operations from P2P layer and the search operation from the semantic layer.
- *cluster_reorder*($C_0, C_1, C_2, \ldots, C_{c-1}$) is called by the upper layer for changing the order of cluster placement on the overlay. The parameter is an ordered list of clusters. If the parameter order is different from the current order, each involved cluster is cut from the present location and reinserted in the new location. This is an expensive operation, as it may involve a substantial stabilization process. However, it is not expected to be invoked frequently, as cluster cross-locality evolves slowly, given that the interests of users do not change fast. Additionally, when a new placement is necessary, it likely involves a small number of cluster reorderings.
- *stabilize()* is similar to the method with the same name from Chord: brings the finger table of invoking nodes to a coherent state, i.e. the entries point clockwise and counterclockwise to $2^i$ hops away.

### *4.3. Semantic layer*

The semantic layer contains operations that leverage collaborative classifications in order to improve content locality via a self-configuring topology.

#### *4.3.1. Node join*

The *sem_join*($n_j, n_i$) operation connects the physical node $n_j$ to the network through the intermediation of an arbitrary physical node $n_i$, as described in Algorithm 1. First, $n_j$ contacts a known node $n_i$ and receives the affinity matrix $A$. Second, the node computes and joins its *primary cluster*. Finally, the node joins its *secondary clusters*. The intuition behind the algorithm for choosing secondary clusters can be summarized as follows: a node is replicated in secondary clusters only if it stores enough resources classified in the category of the secondary clusters. The number of such resources should be larger than an absolute and a relative thresholds. Both thresholds are necessary in order to avoid nodes with a negligible amount of relevant resources from joining secondary clusters, increasing the overhead with practically no benefit. The absolute threshold is computed as the multiplication of an absolute factor of logical node presence ($\alpha$) and an estimation of the total number of resources classified to the category corresponding to cluster $C_k$, denoted $T_k$. The relative threshold is given by the affinity matrix values ($a_{kt}$) multiplied by a relative factor of logical node presence ($\beta$). Both $\alpha$ and $\beta$ values are network wide constant parameters and can be chosen empirically based on application requirements.

**Algorithm 1**. sem_join ($n_j$, $n_i$)

```
Request and receive affinity matrix $A$ from $n_i$
$p_j = t$ such that $r_{tj} = \max_{k=0}^{c-1}(r_{kj})$/* Eq. (3) */
cluster_join ($n_{p_j j}$,$n_i$)/* Join the primary cluster */
for $k \in \{0,1,\ldots,c-1\}$ do
  /* $T_k$: total number of resources classified in the
  category of cluster $C_k$ */
  $T_k = \sum_{s=0}^{c-1} R_{ks}$
  if $r_{jk} > \alpha \times T_k$ then
    /* $r_{jk}$ documents stored in the node $n_j$ and
  classified to the category of cluster $C_k$ */
    $ratio \leftarrow r_{jk} / \sum_{t=0}^{c-1} r_{tk}$
    if $ratio > \beta \times a_{pk}$ then
      /* Join secondary clusters */
      cluster_join($n_{kj}$, $n_i$)
    end if
  end if
end for
```

#### *4.3.2. Resource insertion*

The *sem_insert*(*resource*) function adds a resource to the local library of a physical node. Given that the physical node can have several points of logical presence in the P2P network, the inserted resource becomes visible in the primary and all secondary clusters. Before insertion, a resource must be classified in at least one category.

The insertion of resources may cause changes in the relative contribution of a node to different clusters. The node decides autonomously to join a new secondary cluster, to leave a secondary cluster or to choose a different primary cluster, by using the same procedure described in Algorithm 1.

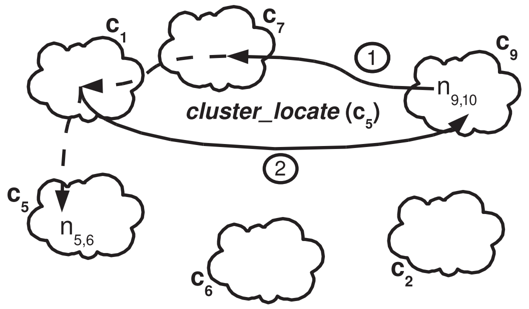
#### *4.3.3. Resource search*

The *sem_search*(*search terms*, $C_k$) operation (Algorithm 2) receives a set of terms and the cluster where the resources are supposed to be. The set of terms can contain wildcards, exact filenames, etc. The search consists of two phases. First, a node initiates the search, using the P2P layer operation *cluster_locate* to find any node in the target cluster $C_k$. Second, a parallel flooding algorithm is used in the target cluster $C_k$ in order to locate the nodes that store the searched resources.

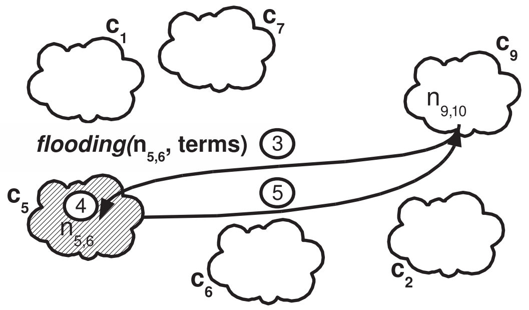
**Algorithm 2**. sem_search(*search terms*, $C_k$)

```
/* Locate the target cluster */
$n_{kj} \leftarrow cluster\_locate(C_k)$
/* Initiate parallel flooding */
$r \leftarrow 0$
while r < max_relaunches and insufficient results returned do
    $flooding(n_{kj}, search\ terms)$
    $n_{kj} \leftarrow$ last node of the previous flooding
    $r \leftarrow r + 1$
end while
```

Fig. 4 shows a simplified search operation in five steps. In step 1 node $n_{9,10}$ invokes *cluster_locate* for finding a node in cluster $C_5$. A pointer to node $n_{5,6}$ is found in cluster $C_1$ and is returned to $n_{9,10}$ in step 2. In step 3, $n_{9,10}$ contacts $n_{5,6}$ which starts the flooding search (step 4). Finally, the nodes with matching resources answer directly to the source node $n_{9,10}$ (step 5).



(a) Node $n_{9,10}$ locates a node ($n_{5,6}$) in cluster $C_5$ using *cluster_locate* operation. The *cluster_locate* identifies in the finger table of node $n_{9,10}$ a node in a cluster closer to destination according to the cluster order table. The operation is repeated in subsequent nodes until cluster $C_5$ is reached.



(b) Node $n_{5,6}$ starts flooding search in cluster $C_5$. Those nodes having the requested resource reply to $n_{9,10}$. In case the flooding is not successful, the negative answer is returned by the last node reached by flooding.

**Fig. 4.** Different phases involved in the search process.

Unstructured P2P systems such as Gnutella [28], use flooding for content location. One of the main disadvantages of Gnutella flooding is the generated redundant communication, which reduces the scalability of the system. In this paper we propose a parallel flooding algorithm based on recursive doubling strategy. We define the flooding horizon (*FH*) as the maximum distance from the source to the farthest node reached by a flooding message. In our algorithm, at step *i*, all the peers that have received a search message forward it in parallel to peers at distance $2^i$. Therefore, the flooding horizon is reached in $log_2FH$ parallel steps. Each node that receives a search message and has a match replies to the source. Negative responses are reported to the source only from the nodes reaching the boundary of the flooding horizon. The source can subsequently decide to relaunch the search beyond the current flooding horizon in order to locate more replicas.

Fig. 5 shows an example of a flooding search for $FH = 8$. For clarity, the figure does not show replies from visited nodes. In the initial step (Fig. 5(a)) node 0 sends a message to node 1 (finger table entry 0th). In step 1 (Fig. 5(b)), 0 and 1 send messages in parallel to nodes 2 and 3 (finger table entry 1). In step 2 (Fig. 5(c)) nodes 0, 1, 2 and 3 send messages in parallel to 4, 5, 6 and 7 (finger table entry 2). Nodes 4, 5, 6 and 7 reach the flooding horizon and the search is stopped. This flooding generates no redundant messages, visiting seven nodes and requiring seven messages.

For reaching the flooding horizon, the flooding algorithm needs $s = log_2FH$ steps and generates $1 + 2 + 2^2 + \cdots + 2^{s-1} = FH - 1$ requests followed either by a maximum number of $FH - 1$ replies or $FH/2$ negative acknowledgements. Therefore, assuming that $FH > 1$, the upper bound on number of messages involved in the search can be calculated in the following way:

$$N_{max} = FH - 1 + max\left(FH - 1, \frac{FH}{2}\right) = 2 \times FH - 2. \quad (5)$$

Given that the search is started simultaneously clockwise and counterclockwise, the upper bound is $4 \times FH - 4$. Eq. (6) gives the total number of messages generated in the worst case for AP2P. We assume a limited *FH*, therefore relaunching searches is needed to cover the whole cluster.

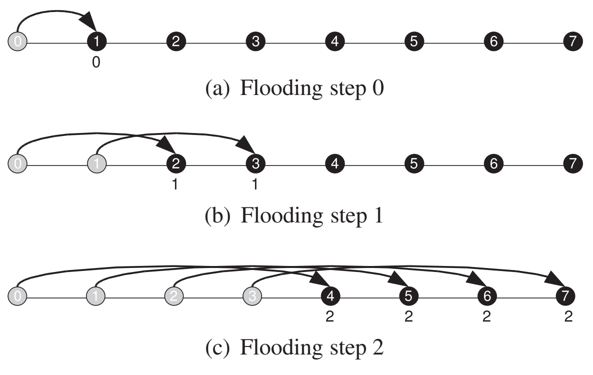


**Fig. 5.** Example of a flooding search with $FH = 8$ on eight neighbors on the ring. In step *i* the current node communicates with a node at distance $2^i$.

$$msg_{AP2P} = effective\ msg + relaunch\ msg = |C_k| + \frac{|C_k|}{FH}. \quad (6)$$

### 4.3.4. Cluster reordering

AP2P self-organizes the placement of the clusters on the ring topology in order to improve the inter-cluster locality. The reorganization of clusters is infrequent, as it is not likely that the interests of users change very quickly. The function *sem_reorder()* is automatically invoked by any node of the system at regular intervals in order to check if cluster reorderings may improve locality and to effectively perform the reordering.

**Algorithm 3.** sem_reorder()

```
/* Greedy */
cluster_order_table ← greedy_order (A)
/* Reorder clusters */
cluster_reorder(cluster_order_table)
```

The reorder operation consists of two steps. In the first step, the greedy algorithm described below is used for computing a new cluster placement order table. The second step reorders the clusters based on the result of the greedy algorithm.

The greedy algorithm (Algorithm 4) works as follows. First, it computes the maximum non-diagonal element of the affinity matrix $a_{row,col}$. The *row* and *col* values are used for placing the first cluster $C_{row}$ and the second cluster $C_{col}$ to the right of $C_{row}$. Second, the value of the column *col* becomes the new *row*; the next cluster is decided by choosing the maximum value of the affinity matrix on the row, given that the column does not correspond to a cluster, which has already been chosen. The second step of the Algorithm 4 is repeated until all the clusters are placed. We call this algorithm GREEDY-MAX, as it greedily targets to maximize the total affinity as a sum of affinities of all pairs of neighbors.

**Algorithm 4.** greedy_order(A)

```
for $k \leftarrow 0$ to $c-1$ do
    cluster_order_table[$k$] ← not_assigned
end for
(row, col) ← $(k,t)$ such that $a_{kt} = \max a_{ij}$, where
    $0 \leqslant i,j < c$ and $i \neq j$
cluster_order_table[0] ← row
for $k \leftarrow 1$ to $c-1$ do
    cluster_order_table[$k$] ← col
    row ← col
    col ← $j$ such that $a_{row,col} = \max a_{row,j}$, where $0 \leqslant j < c$
     and $j \notin$ cluster_order_table[$h$] for $h = 0,1,\ldots,k$
end for
return cluster_order_table
```

For instance, for the affinity matrix in Fig. 6, the GREEDY-MAX algorithm returns the following cluster order: 0, 3, 1, 4, 2. The total affinity is $0.25 + 0.20 + 0.20 + 0.15 + 0.15 = 0.95$. If we apply the same greedy algorithm, but choose the minimum affinity between pairs of clusters (GREEDY-MIN), a possible cluster order is 3, 4, 2, 0, 1 with a total affinity of 0.75, representing 78% of the total affinity returned by GREEDY-MAX.

$$\mathbf{A} = \begin{bmatrix} 0.30 & 0.10 & 0.15 & 0.25 & 0.20 \\ 0.10 & 0.35 & 0.15 & 0.20 & 0.20 \\ 0.15 & 0.15 & 0.35 & 0.20 & 0.15 \\ 0.25 & 0.20 & 0.20 & 0.30 & 0.05 \\ 0.20 & 0.20 & 0.15 & 0.05 & 0.40 \end{bmatrix}$$

**Fig. 6.** Example of affinity matrix for five clusters.

Finally, in the second step of the reordering algorithm (Algorithm 3), the solution of the GREEDY-MAX algorithm is used in order to update the cluster placement order in the overlay network.

As a final observation, we note that the accuracy of affinity matrix and the placement order of the clusters on the ring do not affect the correctness of the search, but only its performance in terms of latency reaching the target cluster. Therefore, the update of the affinity matrix and the reorder frequency can be relaxed, trading off reorganization overhead for search efficiency.

### 4.3.5. Other operations

Apart from the operations previously described, there are other two operations worth mentioning: *sem_leave*($n_j$) and *sem_remove*(*resource_id*). The *sem_leave*($n_j$) operation disconnects the logical node from the primary and secondary clusters by using *cluster_leave* operation. The *sem_remove*(*resource_id*) operation removes the resource identified by *resource_id* from the node. As in the case of insert, this operation may cause a node to change its primary cluster, or to leave or join secondary clusters.

## 5. Experimental results

This section contains the evaluation of AP2P system and it is organized as follows. In Section 5.1 we present the methodology used for data collection and a workload characterization. Section 5.2 describes the evaluation setup. Section 5.3 provides an evaluation of the small-world properties of AP2P. In Section 5.4 we evaluate the benefits of cluster reordering in terms of latency and recall. In Section 5.5 we investigate how different data replica distributions affect latency. Finally, in Section 5.6 we evaluate AP2P scalability for different network sizes and replication distributions.

### 5.1. Workload characterization

Our YouTube traces are based on the traces collected in [39], that contain information about YouTube traffic captured from June 2007 to March 2008 on a campus network. Using the video identifiers and the YouTube public API [42], we collect information about the user who uploaded each video, and all the videos uploaded by that user until June 2008. We also extract the number of views per video and the category of each video. The videos appearing in [39] but not available at the moment of harvesting were discarded. The obtained traces contain 23,076 users sharing a total of 952,718 distinct videos.

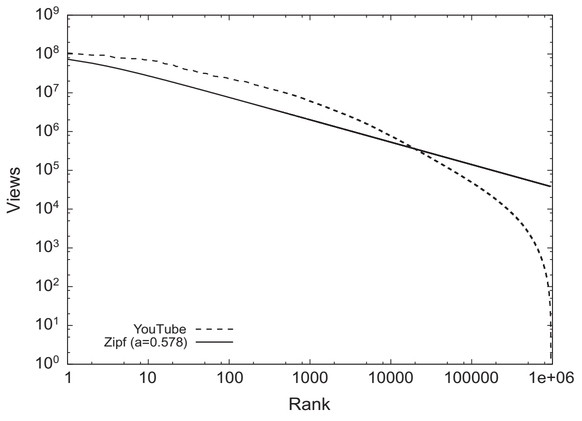


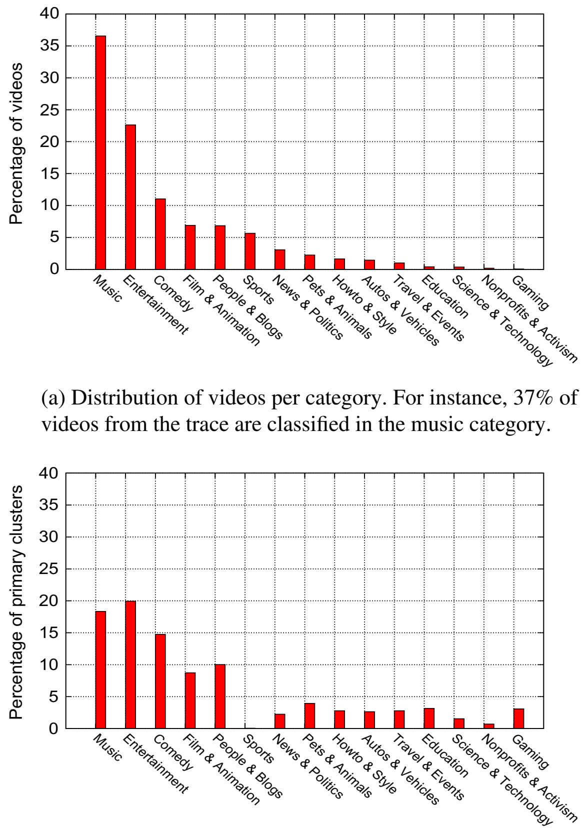
**Fig. 7.** YouTube videos rank ordered by number of views.

Fig. 7 shows the video popularity distribution of the YouTube trace. First we note that this distribution is similar to those obtained in other studies [6–8]. From our trace, only 0.08% of the videos have more than a million views, and 50% of the videos have less than 2300 views. The Zipf distribution overestimates the number of views for 98% of the videos and underestimates the 2%. This situation is also discussed in [8], where the authors note that the Zipf distribution does not predict the existence of many popular videos. Additionally, we observe that the top 2% most popular videos aggregate 35,603 million views, which represent 70% of the total number of views.

In our YouTube trace each video was classified by the user who uploaded it into exactly one of 15 video categories. Fig. 8(a) shows the percentage of videos classified in each category. Videos classified in *Music* and *Entertainment* represent more than 50% of the sample. Other categories such as *Education*, *Gaming* or *Nonprofits and Activism* have a small representation. These values confirm the recreational character of YouTube. Recent studies [7,14] also obtain a similar distribution of videos per category.

Fig. 8(b) shows the primary cluster distribution calculated for the YouTube trace. We note that the distribution of videos per category does not directly correspond to the distribution of primary clusters: e.g., music videos represents 37% of the videos in the system, but only 18% of the users joined Music as their primary cluster. This phenomenon corresponds to the fact that the Music videos are popular to the large majority of users, including users having their primary interests in other clusters.

Users interests are determined by the number of categories they are interested in. Fig. 9 shows the percentage of nodes that share videos in one or more categories. We notice that most users are interested in more than three categories, with an average of 3.3. Interest in more than five categories is unusual, whereas users interested in just one category represent 15% of the total.



(a) Distribution of videos per category. For instance, 37% of videos from the trace are classified in the music category.

(b) Classification of nodes based on their maximum contribution. For instance, 20% of the node have the majority of their videos classified in the Entertainment category.

**Fig. 8.** Distribution of users and videos according to YouTube categories.

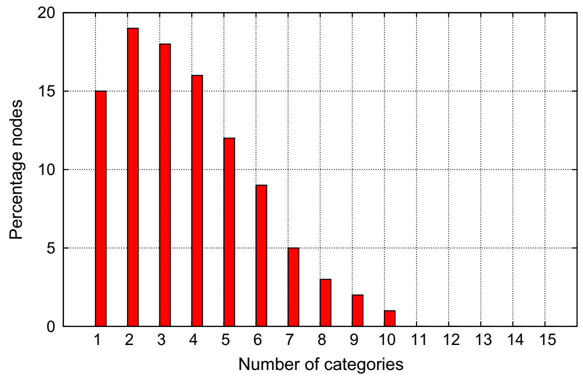


**Fig. 9.** Number of categories a user is interested in. For instance, 15% of the users have videos classified only in one category.

## 5.2. Experimental setup

Simulated networks are composed of different numbers of physical nodes, which additionally join the network as logical nodes according to the procedure explained in Section 4.3.1.

We have implemented a multi-threaded discrete event simulator in Java. The simulator uses the YouTube trace for constructing the P2P network. Each AP2P physical node corresponds in our experiments to a YouTube user. The simulations were performed on two Sun SPARC Enterprise T1000 with 8 GB of RAM and eight multi-threaded cores.

Three common metrics are used in this evaluation: rate of successful searches, latency and recall. The rate of successful searches measures the ratio of searches returning at least one resource. Latency is defined as the number of hops until the first replica of a resource is encountered. Latency results are presented in the paper as Cumulative Distribution Functions (CDF). Recall is the number of replicas returned by the search divided by the total number of replicas in the network.

The flooding horizon value was $FH = 64$, i.e. six entries from the finger table were utilized. The search was performed with no relaunches in clockwise and counterclockwise directions in parallel over the ring topology. The number of logical nodes appearing in the simulations are controlled by two parameters $\alpha$ and $\beta$ used in the join operation (Algorithm 1). The absolute threshold was $\alpha = 0$ and the relative threshold $\beta = 0.5$.

The construction of the simulated network consists of the following steps. First, $n$ users and their shared resources are read from the trace. Each user corresponds to a new physical node. Second, each video is replicated on the physical nodes according to its popularity shown in Fig. 7. The nodes on which replicas are placed are chosen randomly with a uniform distribution from the nodes already sharing resources in the same category as the candidate video. The number of replicas is proportional to the video popularity as depicted in Fig. 10 for the three replica distributions used for evaluation: *rd*1, *rd*2 and *rd*3. The shape of these distributions is similar to the popularity distribution depending on the video age showed in [6]. Third, the affinity matrix is computed according to Eq. (4). Fourth, the physical nodes are inserted one by one as logical nodes in the AP2P network by calling Algorithm 1. Fifth, the affinity matrix is updated to reflect the new network state.

For each simulation described in this paper 100,000 search events are generated. The number of generated search events for a particular video is proportional to the popularity (the number of views). The origin node of the search is chosen randomly with a uniform distribution.

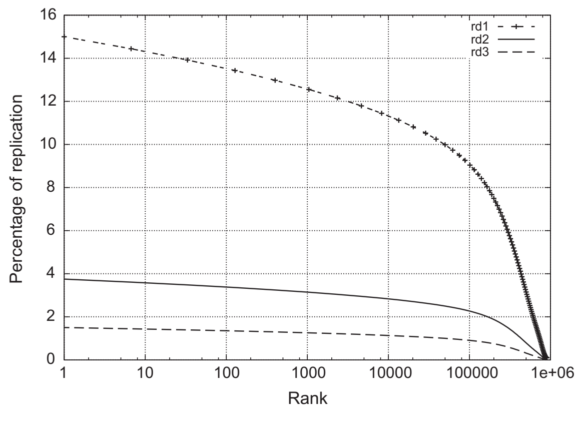


**Fig. 10.** The three replica distributions used in the simulation. Each video is replicated over nodes storing videos from the same category with the probability shown on the $y$-axis.

Table 2 shows a summary of parameter values used in the experiments.

## 5.3. Small-world analysis

This section presents a small-world analysis of AP2P network. This analysis targets to highlight network characteristics related to the potential efficiency of content lookup both at local level (AP2P cluster) and global level (AP2P network). Locally, a small world with a low average path length inside a cluster will indicate that a flooding is likely to reach all nodes storing relevant content in an efficient way. Globally, the small-world property will suggest that cluster lookup is likely to be quickly directed to the target cluster.

A small world is defined in [41] based on two metrics: clustering coefficient ($\gamma$) and characteristic path length ($L$). The *clustering coefficient* represents the probability that the neighbors of a vertex are connected. It is calculated as the average of the number of connections among the neighbors of all vertices of a graph. The *characteristic path length* is the mean of the path lengths between all pairs of vertices of a graph. A graph is *a small world* if, compared with a random graph with the same number of vertices and edges, it has roughly the same characteristic path length and a significantly larger clustering coefficient.

Previous studies [18,36,41] demonstrate that Gnutella and many real networks (e.g., actors, power grid and C. Elegans) exhibit small-world properties. Table 3 shows $\gamma$ and $L$ computed in these studies along with the values for AP2P logical and physical networks corresponding to the workload described in the previous subsection. We note that the characteristic path length for $AP2P_{logical}$ network is 14.06 which is roughly $log(n)$, corresponding to the characteristic path length of a Chord-like ring. However, in AP2P each physical node has several logical presences in the network. The $\gamma$ and $L$ values for the $AP2P_{physical}$ show that AP2P

**Table 2**
Parameter values used in the experiments.

| Parameter | Values |
|---|---|
| Physical network size | 3000–5000 nodes |
| Logical network size | 8897–14,718 nodes |
| Number of clusters ($c$) | 15 |
| Finger table size ($m$) | 6 |
| $\alpha$ | 0 |
| $\beta$ | 0–1 |

**Table 3**
Comparison of clustering coefficient and characteristic path length for various real networks.

| | $L$ | $L_{random}$ | $\gamma$ | $\gamma_{random}$ |
|---|---|---|---|---|
| $AP2P_{logical}$ | 14.06 | 3.57 | 0.16 | 0.003 |
| $AP2P_{physical}$ | 2.57 | 2.48 | 0.15 | 0.0262 |
| Gnutella 0.6 [36] | 4.17–4.23 | 3.75 | 0.018 | 0.00038 |
| Gnutella 0.4 [18] | 3.30–4.42 | 3.66 | 0.02 | 0.002 |
| MovieActors [41] | 3.65 | 2.99 | 0.79 | 0.00027 |
| PowerGrid [41] | 18.7 | 12.4 | 0.08 | 0.005 |
| C. Elegans [41] | 2.65 | 2.25 | 0.28 | 0.05 |

satisfies the small-world conditions. This fact shows that by using logical nodes, two physical nodes can be connected with less hops acting like shortcuts.

AP2P is structured as a collection of interconnected clusters. In order to investigate the local characteristics of each cluster, we computed the values for local characteristic path length of a cluster $L_{local}$ and clustering coefficient of a cluster $\gamma_{local}$. This approach was inspired by the Watts' model [41] used for analyzing the local and global length scale of a small-world graph. Fig. 11 shows the values for $\gamma_{local}$, $L_{local}$ and the number of logical nodes of each cluster. The clustering coefficients of all clusters are between 0.16 and 0.26. Nevertheless, the clustering coefficients of popular clusters are within a small range, i.e. 0.16 and 0.18 and do not appear to depend on the cluster size; for instance, the difference between sizes of clusters Music and Auto and Vehicles is one order of magnitude. $L_{local}$ values appear to depend on the cluster size. Fig. 12 shows that this dependence is sublogarithmic. These characteristics show that the content is highly clustered inside each AP2P cluster with a low average path among nodes. Consequently, the flooding inside each cluster is highly probable to reach relevant content with a low latency (number of hops).

### 5.4. Cluster placement sensitivity analysis

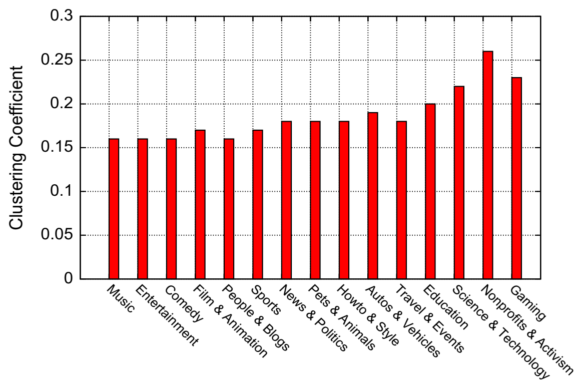
In this subsection, we evaluate the benefits that can be achieved from a proper order of cluster placement on the AP2P ring. A good order translates into a high-content locality across clusters. For the evaluation we have computed two possible orders by applying the GREEDY-MAX and GREEDY-MIN algorithms on the same network.

Fig. 13 compares search latency and recall for 4000 nodes. We note that the order produced by GREEDY-MAX results in significant improvements in both latency and recall. In terms of latency, for GREEDY-MAX order more than 80% of the searches return the first replica in 40 hops. For the same number of hops GREEDY-MIN order finds the first replica only for less than 20% of the searches. The recall scales almost linearly with the number of messages for both orders. However, for GREEDY-MAX the slope is significantly greater. For instance, for 300 messages the recall is almost 17% for GREEDY-MAX and approximately 6% for GREEDY-MIN. GREEDY-MAX and GREEDY-MIN can be seen as an approximation of the upper bound and a lower bound on a spectrum of different possible cluster orders. The locality of content is improved by GREEDY-MAX, as affine clusters are placed near each other.

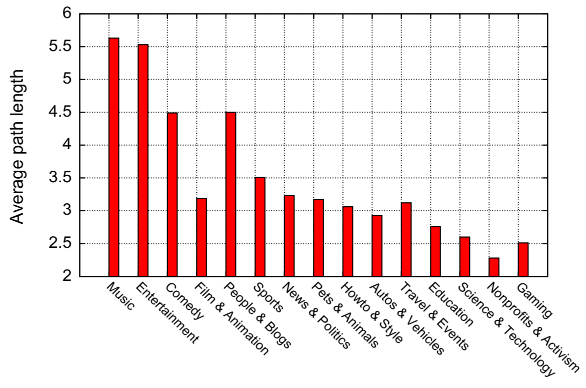
### 5.5. Effect of replication

In this subsection, we present an evaluation of the effect of replication on the search latency. We have used the three replication distributions introduced in Section 5.1 for a network composed of 4000 physical nodes after applying GREEDY-MAX reordering algorithm. Fig. 14 shows the latency results obtained from simulations.

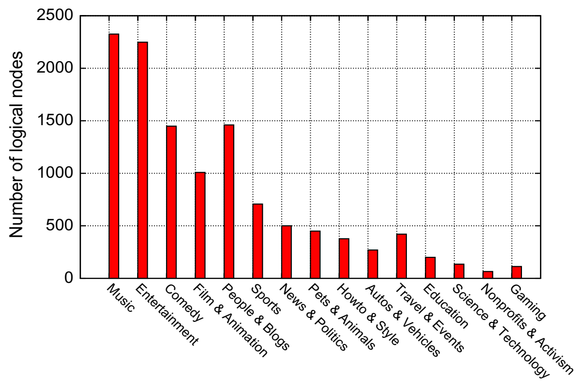
We note that a higher replication implies a lower latency. For instance, for $rd1$ more than 95% of the searches find the first replica in at most 20 hops. For comparison, in the same number of hops 65% and 40% of the searches return the first result for $rd2$ and $rd3$, respectively.



(a) $\gamma_{local}$



(b) $L_{local}$



(c) Clustersize (number of logical nodes)

**Fig. 11.** $\gamma_{local}$, $L_{local}$ and cluster size for $AP2P_{logical}$.

### 5.6. Sensitivity to network size

In this section, we evaluate the sensitivity of latency and recall to the AP2P network size. Fig. 15(a) and (b) show the results when the network size increases from 3000 to 5000 physical nodes, while maintaining unchanged all the other parameters. The replication distribution used in this experiment was $rd2$. The sizes of AP2P network in

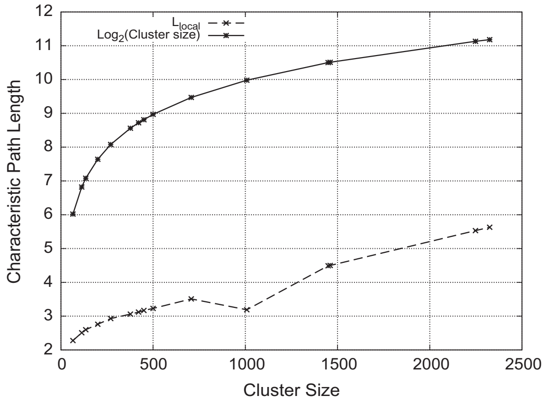


**Fig. 12.** Variation of cluster-local characteristic path length ($L_{local}$) with the the cluster size.

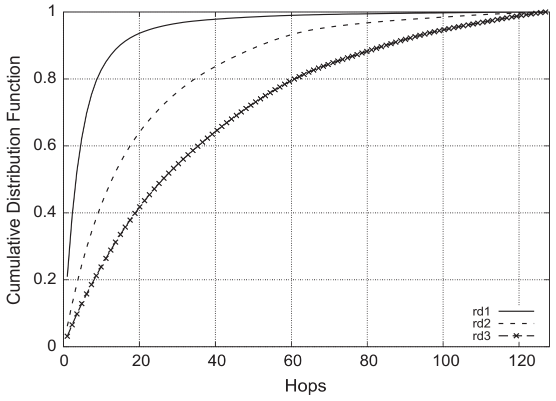


**Fig. 14.** Latency comparison for a 4000 nodes network using different replica distributions.

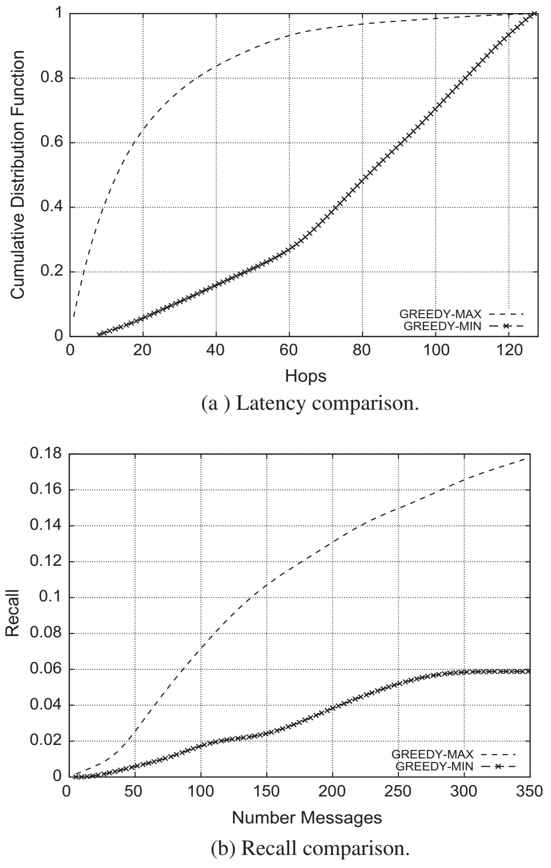


**Fig. 13.** Comparisons of GREEDY-MIN and GREEDY-MAX algorithms in a 4000 nodes network using replica distribution *rd*2.

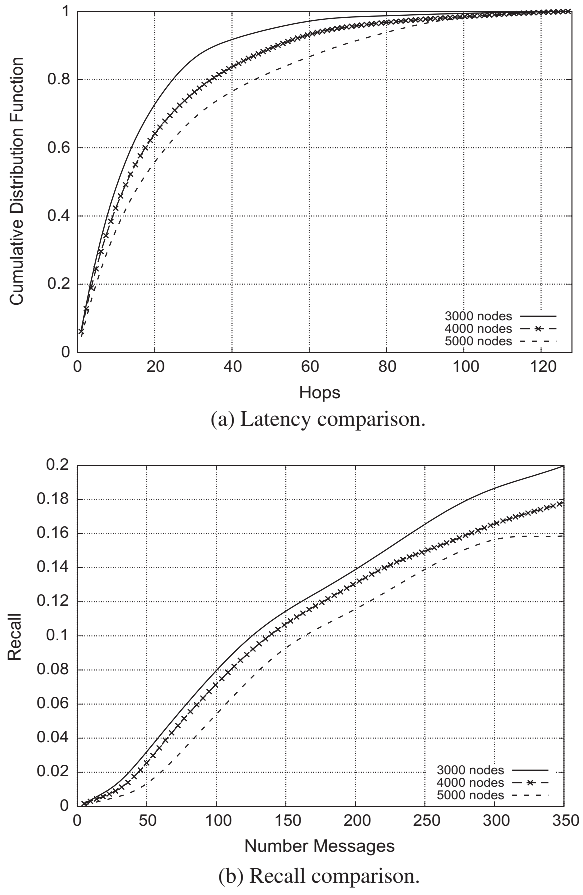


**Fig. 15.** Evaluation of scalability for different network sizes using replica distribution *rd*2.

terms of logical nodes were 8897, 11,732 and 14,718, for 3000, 4000 and 5000 physical nodes, respectively.

It can be noticed that the variation of both latency and recall are small, when increasing the number of nodes by 66% (from 3000 to 5000 physical nodes). For instance, in numerical terms, for 20 hops, 17% more searches return the first replica for 3000 nodes than for 5000 nodes.

The recall (Fig. 15(b)) also decreases with the increase in the number of nodes. These results are intuitive, as the distance between replicas is expected to increase with the number of nodes and the FH is unmodified. Recall can be improved by increasing the FH as a function of the network size. AP2P recall results are good compared with other solutions. For example, the efficient search algorithm from [17] shows a recall value of 14% for a 5000 nodes network, whereas AP2P obtains a recall value of 16%.

Table 4 shows a comparison of the search success rate for different network configurations. As expected, the success rate increases with the degree of replication. The success rate decreases slowly with the number of nodes showing good scalability with network size. For instance, increasing by 66% the number of physical nodes (from 3000 to 5000), the success rate is reduced by 0.5%, 2.7%, and 11.2% for the three replication distributions. It can also be noticed that the decreasing of the success rate is slower for higher degrees of replication. However, in our simulations the flooding horizon (FH) was kept constant at 64. The success rate directly depends on FH and, it can be increased by choosing a higher FH.

### 5.7. Locality evaluation

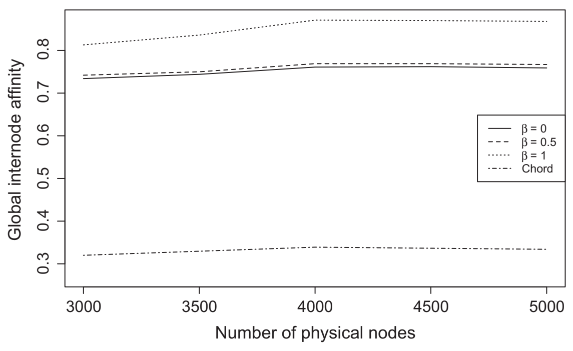
In this section, we present a study of impact of physical network size and relative factor of logical nodes presence ($\beta$) on the average inter-node affinity.

In order to estimate the global locality we compute the average inter-node affinity for all the connected peers in AP2P and Chord. For each node $n$, we compute the vector $v$ of size $k$, where $v_k$ is the percentage of resources of node $n$ in classified in the category of cluster $C_k$. Given two nodes $n_1$ and $n_2$ with vectors $u$ and $v$, we define inter-node affinity as $MAX(MIN(u_i,v_i))\forall i \in \{0,1,\ldots,c-1\}$. Intuitively, MIN operation computes the taste overlap per category for two users, while MAX computes the category with highest taste overlap. For instance, the inter-node locality is 1 for two nodes storing resources only in Music category, and 0 if one node stores exclusively resources classified in Music and another one exclusively resources classified in Movies.

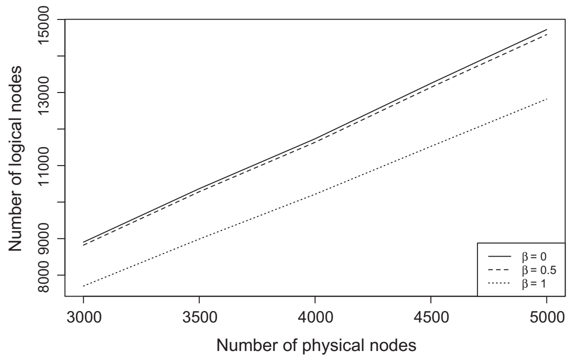
In order to compare Chord and AP2P locality we build a Chord network with the same number of physical nodes as AP2P and with the same finger table size and insert the same documents in the network. We then compute average inter-node affinity for both Chord and AP2P networks.

Fig. 16 plots (a) the average inter-node affinity and (b) logical node size for physical network size ranging from 3000 to 5000 nodes and for $\beta$ from Algorithm 1 ranging from 0 to 1. If $\beta = 0$ a node is added to secondary clusters if it has at least one resource in that cluster. This value is expected to reduce the affinity and to increase the logical network size. If $\beta = 1$ a node is added to secondary clusters if the number of its resources classified in that cluster is larger than the affinity between node's primary cluster and target cluster. This value is expected to increase global affinity and decrease logical network size (as nodes are added to secondary clusters selectively based on a high affinity). The value of $\alpha$ was fixed to 0, meaning that a node is considered to be added in secondary clusters based only on the relative number of resources classified in the respective category.

The average inter-node affinity for Chord represents less than half of the minimum value for AP2P and less than a third of the maximum value. As expected, the highest locality is obtained in all cases for $\beta = 1$, that is when secondary clusters are joined selectively. In this case the affinity in each cluster is increased, minimizing the probability of finding less affine resources. When decreasing $\beta$, the global locality decreases, the logical network size increases (Fig. 16(b)), while the probability of finding resources with the lower degree of affinity faster increases.



(a) Average inter-node affinity sensitivity to the physical network size and relative factor of logical node presence ($\beta$).



(b) Logical network size sensitivity to physical network size and relative factor of logical node presence ($\beta$).

**Fig. 16.** Locality evaluation of AP2P network.

**Table 4**
Rate of successful searches for different number of nodes and different replica distributions.

| | | Number of nodes | | |
|---|---|---|---|---|
| | | 3000 | 4000 | 5000 |
| Replica distribution | *rd*1 | 0.9968 | 0.9884 | 0.9910 |
| | *rd*2 | 0.9783 | 0.9553 | 0.9520 |
| | *rd*3 | 0.9134 | 0.8624 | 0.8014 |

## 6. Conclusions and future work

The recent huge increase of popularity of collaborative user-generated content sites has exposed the scalability limitations of server-based systems, especially due to the high Internet traffic. One of the challenges of distributing shared data from server-based systems to scalable P2P networks is the preservation of content locality in order to achieve efficiency of searches for related content. In this paper we have presented AP2P, a cluster-based locality-aware self-organizing peer-to-peer network that leverages collaborative classifications in order to self-organize for a higher content locality. AP2P self organizes at two levels: node and cluster level. Each node may decide autonomously to join or leave clusters based on the own composition of content or interests. Clusters may change their positions in order to achieve a high inter-cluster locality.

Based on analytical and experimental results, our paper makes the following claims. AP2P shows small-world characteristics, and, therefore, facilitates the localization of content. By joining the network at different points as logical nodes, physical nodes offer high content locality in different clusters, substantially improving the search performance. Additionally, the dynamic reorganization at cluster level provides an improvement in both search latency and recall. We also demonstrate empirically that replication of content according to popularity may be leveraged for achieving a lower latency with a relative small amount of replication. Finally, we show that the search latency, recall and rate of success scale smoothly with the number of nodes.

Several future research lines open up from this work. The search complexity in AP2P is logarithmic in number of clusters. As new nodes join the network, this complexity can be maintained by proportionally adapting the number of clusters. We plan to study this aspect in more detail in future work. We plan to evaluate alternative topologies that may further improve the locality of content. This goal makes necessary the construction of new linear optimization algorithms for cluster placement, alternative to the ring-based greedy algorithm proposed in this paper. Regarding the $\alpha$ and $\beta$ parameters that determine which nodes join a cluster, we plan to further investigate which are the optimal values based on different aspects of the application scenario: number of users, number of documents, expected join rate, etc. Finally, we aim to support and optimize multi-attribute multi-range queries based on the self-organization of content at both node and cluster level.

## Acknowledgements

This research has been partially funded by the Spanish Ministry of Education under Project TEALES (TIN2007-63092), FPU program AP2007-03530 (Juan M. Tirado) and by the United States National Science Foundation under Grant No. CNS-0831785 (Adriana Iamnitchi).

## References

[1] K. Aberer, L.O. Alima, A. Ghodsi, S. Girdzijauskas, S. Haridi, M. Hauswirth, The essence of P2P: a reference architecture for overlay networks, in: Fifth IEEE International Conference on Peer-to-Peer Computing (P2P 2005), IEEE Computer Society, Konstanz, Germany, 2005, pp. 11–20.

[2] K. Aberer, P. Cudré-Mauroux, M. Hauswirth, T.V. Pelt, Gridvine: building internet-scale semantic overlay networks, in: International Semantic Web Conference, pp. 107–121.

[3] Alexa, Alexa Search Engine, 2008. <http://www.alexa.com/>.

[4] Z. Anwar, W. Yurcik, V. Pandey, A. Shankar, I. Gupta, R.H. Campbell, Leveraging Social-network Infrastructure to Improve Peer-to-peer Overlay Performance: Results from Orkut, CoRR abs/cs/0509095, 2005.

[5] R. Bolla, R. Gaeta, A. Magnetto, M. Sciuto, M. Sereno, A measurement study supporting p2p file-sharing community models, Computer Networks 53 (2009) 485–500.

[6] M. Cha, H. Kwak, P. Rodriguez, Y.Y. Ahn, S. Moon, I tube, you tube, I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system, in: IMC '07: Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement, ACM, 2007, pp. 1–14.

[7] X. Cheng, C. Dale, J. Liu, Statistics and social network of youtube videos, in: 16th International Workshop on Quality of Service, IWQoS 2008, 2008, pp. 229–238.

[8] X. Cheng, J. Liu, NetTube: exploring social networks for peer-to-peer short video sharing, in: Proceedings of INFOCOM 2009, 2009, pp. 1152–1160.

[9] I. Clarke, O. Sandberg, B. Wiley, T.W. Hong, Freenet: a distributed anonymous information storage and retrieval system, Lecture Notes in Computer Science 2009 (2001) 46–56.

[10] E. Cohen, A. Fiat, H. Kaplan, Associative search in peer to peer networks: harnessing latent semantics, Computer Networks 51 (2007) 1861–1881.

[11] T. Condie, A.D. Kamvar, H. Garcia-molina, Adaptive peer-to-peer topologies, in: International Conference on Peer-to-Peer Computing, IEEE, 2004, pp. 53–62.

[12] A. Crespo, H. Garcia-Molina, Semantic overlay networks for P2P systems, Agents and P2P Computing, vol. 3601, Springer, 2005, pp. 1–13.

[13] A. Forestiero, C. Mastroianni, M. Meo, Self-chord: a bio-inspired algorithm for structured P2P systems, in: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid-Volume 00, IEEE Computer Society, 2009, pp. 44–51.

[14] P. Gill, M. Arlitt, Z. Li, A. Mahanti, Youtube traffic characterization: a view from the edge, in: IMC '07: Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement, ACM, New York, NY, USA, 2007, pp. 15–28.

[15] A. Iamnitchi, I. Foster, Interest-aware information dissemination in small-world communities, in: HPDC '05: Proceedings of the High Performance Distributed Computing, HPDC-14. Proceedings, 14th IEEE International Symposium, IEEE Computer Society, Washington, DC, USA, 2005, pp. 167–175.

[16] A. Iamnitchi, M. Ripeanu, I. Foster, Small-world file-sharing communities, in: The 23rd Conference of the IEEE Communications Society (InfoCom 2004), vol. 2, Hong Kong, pp. 952–963.

[17] H. Jin, X. Ning, H. Chen, Efficient search for peer-to-peer information retrieval using semantic small world, WWW '06: Proceedings of the 15th International Conference on World Wide Web, ACM, New York, NY, USA, 2006, pp. 1003–1004.

[18] M. Jovanovic, F. Annexstein, K. Berman, Modeling peer-to-peer network topologies through small-world models and power laws, in: TELFOR.

[19] G. Koloniari, E. Pitoura, Recall-based cluster reformulation by selfish peers, in: IEEE 24th International Conference on Data Engineering Workshop, ICDEW 2008, 2008, pp. 200–205.

[20] A. Löser, F. Naumann, W. Siberski, W. Nejdl, U. Thaden, Semantic overlay clusters within super-peer networks, Databases, Information Systems, and Peer-to-Peer Computing (2004) 33–47.

[21] E. Meshkova, J. Riihijärvi, M. Petrova, P. Mähönen, A survey on resource discovery mechanisms, peer-to-peer and service discovery frameworks, Computer Networks 52 (2008) 2097–2128.

[22] J.X. Parreira, S. Michel, G. Weikum, P2PDating: real life inspired semantic overlay networks for web search, Information Processing & Management 43 (2007) 643–664.

[23] W. Penzo, S. Lodi, F. Mandreoli, R. Martoglia, S. Sassatelli, Semantic peer, here are the neighbors you want!, in: EDBT '08: Proceedings of the 11th International Conference on Extending Database Technology, ACM, 2008, pp 26–37.

[24] J.A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D.H.J. Epema, M. Reinders, M.R. van Steen, H.J. Sips, Tribler: a social-based peer-to-peer system: research articles, Concurrency and Compututation: Practice and Experience 20 (2008) 127–138.

[25] P. Raftopoulou, E. Petrakis, iCluster: a self-organizing overlay network for P2P information retrieval, in: Advances in Information Retrieval: 30th European Conference on IR Research, ECIR 2008, p. 65.

[26] S. Ratnasamy, P. Francis, M. Handley, R.M. Karp, S. Shenker, A scalable content-addressable network, in: SIGCOMM, pp. 161–172.

[27] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, H. Yu, OpenDHT: a public DHT service and its uses, SIGCOMM Computer Communication Review 35 (2005) 73–84.
[28] M. Ripeanu, I. Foster, A. Iamnitchi, Mapping the gnutella network: properties of large-scale peer-to-peer systems and implications for system design, IEEE Internet Computing Journal 6 (2002) 50–57.
[29] H. Rostami, J. Habibi, E. Livani, Semantic routing of search queries in P2P networks, Journal of Parallel Distributed Computing 68 (2008) 1590–1602.
[30] A. Rowstron, P. Druschel, Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems, Lecture Notes in Computer Science 2218 (2001) 329–350.
[31] M. Saxena, U. Sharan, S. Fahmy, Analyzing video services in web 2.0: a global perspective, in: NOSSDAV '08: Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video, ACM, New York, NY, USA, 2008, pp. 39–44.
[32] M. Schlosser, M. Sintek, S. Decker, W. Nejdl, HyperCuP-hypercubes, ontologies and efficient search on P2P networks, in: Lecture Notes in Computer Science, Springer, 2002, pp. 112–124.
[33] R. Siebes, pNear: Combining Content Clustering and Distributed Hash Tables, in: P2P Knowledge Management.
[34] K. Sripanidkulchai, B.M. Maggs, H. Zhang, Efficient content location using interest-based locality in peer-to-peer systems, in: INFOCOM, vol. 3, pp. 2166–2176.
[35] I. Stoica, R. Morris, D.R. Karger, M.F. Kaashoek, H. Balakrishnan, Chord: a scalable peer-to-peer lookup service for internet applications, in: SIGCOMM, pp. 149–160.
[36] D. Stutzbach, R. Rejaie, S. Sen, Characterizing unstructured overlay topologies in modern p2p file-sharing systems, IEEE/ACM Transactions on Networking 16 (2008) 267–280.
[37] C. Tempich, S. Staab, A. Wranik, Remindin': semantic query routing in peer-to-peer networks based on social metaphors, in: WWW '04: Proceedings of the 13th International Conference on World Wide Web, ACM Press, New York, NY, USA, 2004, pp. 640–649.
[38] P. Triantafillou, C. Xiruhaki, M. Koubarakis, N. Ntarmos, Towards high performance peer-to-peer content and resource sharing systems, in: Conference on Innovative Data Systems Research, pp. 120–132.
[39] Umass Trace Repository, YouTube traces, 2008. <http://traces.cs.umass.edu/index.php/Network/Network>.
[40] S. Voulgaris, M. van Steen, K. Iwanicki, Proactive gossip-based management of semantic overlay networks, Concurrency and Computation 19 (2007) 2299–2311.
[41] D.J. Watts, Small Worlds: The Dynamics of Networks between Order and Randomness (Princeton Studies in Complexity), Princeton University Press, 2003.
[42] YouTube, YouTube API, 2008. <http://code.google.com/apis/youtube>.

**Juan M. Tirado** is a Ph.D. candidate at the Universidad Carlos III of Madrid where he received his MSc in Computer Science in 2007. Since 2009 he is part of the Spanish program for predoctoral students FPU. His research interests are in data distribution over dynamic and heterogeneous systems, self-organizing networks and cloud computing.

**Daniel Higuero** is a Ph.D. student at the University Carlos III of Madrid where he received his MSc in Computer Science in 2007. His research interest are in peer-to-peer networks, distributed systems, autonomous and self-organizing networks and cloud computing.

**Florin Isaila** has been an Assistant Professor of the University Carlos III of Madrid since 2005. Previously, he was teaching and research assistant in the Departments of Computer Science of Rutgers University and University of Karlsruhe. He was a visiting scholar at Northwestern University in 2006 and at Argonne National Lab in 2007–2008. His primary research interests are parallel computing and distributed systems. He is currently involved in various projects on topics including parallel I/O, parallel architectures, peer-to-peer systems and social networking. He received a Ph.D. in Computer Science from University of Karlsruhe in 2004 and a MS from Rutgers The State University of New Jersey in 2000.

**Jesús Carretero** is Full Professor of Computer Architecture and Technology at Universidad Carlos III de Madrid (Spain), where he is responsible for that knowledge area since 2000. He is also Director of the Master in Administration and Management of Computer Systems, that he found in 2004. He serves as a Technology Advisor in several companies. His major researches are in parallel and distributed systems, real time systems and computing systems architecture. He is a Senior Member of IEEE.

**Adriana Iamnitchi** is Assistant Professor in the Computer Science and Engineering Department of University of South Florida. Adriana received her Ph.D. in Computer Science from the University of Chicago in 2003. Her research interests are in distributed systems with a focus on self-organization and decentralized control in large-scale grid and peer-to-peer systems.