Available online at www.sciencedirect.com

ScienceDirect

Parallel Computing 33 (2007) 339–358

PARALLEL COMPUTING

www.elsevier.com/locate/parco

Peer-to-peer systems for discovering resources in a dynamic grid ☆

Moreno Marzolla [b], Matteo Mordacchini [a,b,*], Salvatore Orlando [a,c]

[a] *Dip. di Informatica, Univ. Ca' Foscari di Venezia, via Torino 155, 30172 Mestre, Italy*
[b] *INFN Sezione di Padova, via Marzolo 8, 35131 Padova, Italy*
[c] *ISTI Area della Ricerca CNR, via G. Moruzzi 1, 56124 Pisa, Italy*

Available online 20 February 2007

Abstract

The convergence of the Grid and Peer-to-Peer (P2P) worlds has led to many solutions that try to efficiently solve the problem of resource discovery on Grids. Some of these solutions are extensions of P2P DHT-based networks. We believe that these systems are not flexible enough when the indexed data are very dynamic, i.e., the values of the resource attributes change very frequently over time. This is a common case for Grid metadata, like CPU loads, queue occupation, etc. Moreover, since common requests for Grid resources may be expressed as multi-attribute range queries, we think that the DHT-based P2P solutions are poorly flexible and efficient in handling them.

In this paper we present two P2P systems. Both are based on Routing Indexes, which are used to efficiently route queries and update messages in the presence of highly variable data. The first system uses a tree-shaped overlay network. The second one is an evolution of the first, and is based on a two-level hierarchical network topology, where tree topologies must only be maintained at the lower level of the hierarchy, i.e., within the various node groups making up the network. The main goal of the second organization is to achieve a simpler maintenance of the overall P2P graph topology, by preserving the good properties of the tree-shaped topology.

We discuss the results of extensive simulation studies aimed at assessing the performance and scalability of the proposed approaches. We also analyze how the network topologies affect the propagation of query and update messages.

Keywords: Peer-to-Peer; Multi-attribute range queries; Routing Indexes; Overlay networks; Resource discovery; Grid computing

1. Introduction

P2P networks have emerged as one of the most successful ways to share resources (e.g., data, storage, computational power) in a distributed, scalable, and fault tolerant way. Since large scale resource sharing is also

☆ This work has been partially supported by the European Research Network on Foundations, Software Infrastructures and Applications for large scale distributed, GRID and Peer-to-Peer Technologies (CoreGRID).

* Corresponding author. Address: Dip. di Informatica, Univ. Ca' Foscari di Venezia, via Torino 155, 30172 Mestre, Italy. Tel.: +39 050 315 2890; fax: +39 050 315 3464.

E-mail addresses: marzolla@pd.infn.it (M. Marzolla), mordacchini@dsi.unive.it (M. Mordacchini), orlando@dsi.unive.it (S. Orlando).

0167-8191/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.parco.2007.02.006

the goal of modern Grid systems, P2P systems and Grid worlds are slowly converging [8,20], leading to application of P2P techniques to Grid systems.

One of the core functionalities of Grid systems is the location of resources satisfying given constraints. This occurs when a Grid user submits a job, and specifies its requirements, like memory, disk space, Operating System version, etc. Locating data that match a given search criteria is one of the most studied problems in P2P systems. However, the Grid resource location problem is more complex, as resource attributes may be *dynamic*. For example, the available disk space available at a given storage element varies over time as users add and remove data. Location of dynamic data on distributed P2P systems is considerably more difficult than location of static data. Moreover, Grids users are interested in finding resources that match *multi-attribute range queries*, i.e. queries that identify all (or a subset of) the resources characterized by a set of attributes whose values fall into given intervals. Note that this is different from typical file sharing P2P systems, which usually index and manage (*key*, *values*) pairs, and naturally support exact queries for a value (e.g., File Location) given a search key.

A request for locating and acquiring resources in order to execute a job is a typical case of multi-attribute range query. Consider the following example:

$$Q = \{R \in \{R_1, \ldots R_N\} | \mathit{CpuSpeed}[R] \geqslant 2.0\ \text{GHz}\ \mathbf{and}\ \mathit{RamSize}[R] \geqslant 512\ \text{MB}\ \mathbf{and}\ \mathit{Utilization}10[R] \\ \leqslant 0.3\ \mathbf{and}\ 100\ \text{MB} \leqslant \mathit{Free\_Space}[R] \leqslant 300\ \text{MB}\}$$

which is a query that looks for computational resources *R* with CPU speed at least 2.0 GHz, at least 512 MB of RAM, with utilization over the last 10 min of at most 0.3, and with available disk space in the range [100, 300] MB. Note that the *Utilization* parameter is a typical dynamic attribute, which varies over time.

In the following we assume that a generic query predicate is a boolean composition of range conditions on attribute values, and discuss the proposal for novel P2P systems able to locate such resources. It is worth noting that, for expressing range queries, it is necessary to define a total ordering over the domain of resource attribute values. Such total ordering is implied for many attribute types such as numeric and string.

The first unstructured P2P systems that tried to solve the data location problem flood the entire network until all the desired data are collected or a stop condition is reached. This behavior generates a high network traffic which seriously limits the scalability of the system. In order to address this problem more efficiently, many data indexing systems and more structured P2P networks have been proposed, such as the DHT-based ones [2,19,16,17]. These systems only allow exact queries to be expressed, while one of the common way to retrieve data on Grids is through *multi-attribute range queries*. Some authors have thus proposed extensions to the cited algorithms in order to adapt these Distributed Hash Table (DHT) systems to the Grid needs [5,1,4].

Despite the good results achieved by DHT networks in several fields, they may not represent the best solution in presence of dynamic data, i.e., data whose value change frequently and unpredictably over time. The main problem with DHT-based networks is that each change in the attribute values requires to re-index the items whose content have changed. Moreover, when queries range intervals are very large or open, the cost of lookup for DHTs can degenerate from O(log*n*) (the cost for exact queries) to O(*n*). For these and other reasons, DHT-based solutions that have been proposed so far to support multi-attribute range queries can suffer from poor flexibility and efficiency.

We believe that the P2P networks that are less structured than the DHT ones are more suitable to deal with dynamic data and such kind of queries, and thus in this paper we propose a couple of P2P systems based on Routing Index (RI) [7]. To this end, we assume that each P2P node of our network manages and indexes information associated with a disjoint subset of all the Grid resources. Each node also maintains, for each attribute that characterizes the managed resources, a bitmap index, i.e., a condensed description of the local presence/absence of resources. We use bitmap indices not only to represent local resources, but also as a condensed description of the resources present in every neighbors with respect to the overlay network. The last bitmap indexes are thus used as a sort of RI [7] to route queries towards the location of resources possibly satisfying the query. Thanks to their simplicity, such the indices can easily be updated if some attribute value changes. The updates are propagated from the node responsible for a given resources by gossiping. Such updates are flooded by using the same overlay network exploited for routing the queries.

The first proposal we discuss deals with *Tree Vector*, a P2P discovery system based on a tree-structured overlay network. This allows the bitwise routing indices to represent the complete knowledge about the resources that are reachable by following a link that is connected to a given subtree of nodes. The aim of *Tree Vector* is to build a P2P system with a reasonable tradeoff between the need of efficiently route range queries, and the ability to reduce the overhead needed to modify the indices when data changes over time.

Our second proposal is an evolution of *Tree Vector*, and is based on a new topology organization of the overlay network. This new organization, called *Forest of Trees*, aims to simplify the maintenance of the network, while preserving the good features of the tree-shaped P2P overlay network. In order to avoid the cost and difficulty in maintaining a very large tree topology, we propose a two-level hierarchical network topology, where tree topologies must only be maintained at the lower level of the hierarchy. Since the upper level is instead completely unstructured, this may introduce some imprecision in the RIs. The idea is thus to tradeoff a simpler maintenance of the overall P2P network topology, with the introduction of some imprecision in the routing indices built at the upper level of the network hierarchy.

For each of the two proposed solutions, whose preliminary results were presented in [13,12], we perform extensive simulation experiments to assess the performance and scalability of the proposed approach. In particular, we study how the network topology affects the propagation of query and update messages, and derive simple analytical expressions, like the precision and the recall of our search algorithm as a function of query selectivity, index size, and network topology.

This paper is organized as follows. In Section 2 we review some previous results in the area of resource discovery in P2P networks. In Section 3 we precisely state the problem we address. Section 4 introduces a first solution to the dynamic resource discovery in Grids, based on a tree overlay network built on the set of peers. In Section 5 we partly relax the requirement of having a tree overlay network, to explore a different peer organization based on a forest of trees (individual trees may be arbitrarily connected to other trees). Final remarks and open issues are discussed in Section 6.

2. Related works

The problem of routing queries in P2P systems is well known. In order to avoid flooding the network with query messages (as done by systems like Gnutella [10]), many data indexing methods have been proposed. The most promising ones are the so-called DHT-based systems. In these systems every data item is associated with a key obtained by hashing an attribute of the object (e.g. its name). Every node in the network is responsible for maintaining information about a set of keys and the associated items. They also maintain a list of adjacent or neighboring nodes. A query becomes the search of a key in the DHT. When a peer receives a query, if it does not have the requested items, it forwards the query to the neighbor having keys which are closer to the requested one. Data placement ensures that queries eventually reach a matching data item.

In order to further enhance the search performance, many DHT-based protocols organize the peers into an overlay structure. So, in Chord [19] nodes are organized into a virtual circle, while in CAN [16] the identifier space is seen as a d-dimensional Cartesian space. This space is partitioned into zones of equal size and every peer is responsible of one of these zones. Other relevant examples of this kind of systems are Pastry [17] and Tapestry [21]. Although these networks show good performances and scalability characteristics, they only support exact queries, i.e., requests for data items matching a given key. Moreover the hashing mechanism works well with static object identifiers like file names, but is not suitable for handling dynamic object contents.

The ability to perform multi-attribute range queries over mutable data stores is a key feature in many scenarios, like distributed database and Grid resource discovery. Range queries are queries that requests all items whose attribute value fall into a given interval. Some systems have been proposed to support range and multi-attribute queries in P2P networks. The P-Tree [6] uses a distributed version of the B^+-tree index structure. Other protocols use locality preserving hash functions, like the Hilbert space-filling curve, to allow DHT to support range queries. For example, in [1] the authors propose an extension of the Chord protocol to support range and multi-attribute queries, by using a uniform locality preserving hash function to map items in the Chord key space. In [9] two methods are proposed. The first one (called SCRAP) adopts space filling curves as hash functions. The second one (MURK) partitions the data space into rectangles (hyper-rectangles) of different size, such that the amount of data stored on peers is equally distributed. Another common solution adopted in literature to resolve multi-attribute queries is to maintain a separate DHT layer for each attribute type. This solution is adopted, among others, by [5,4,18]. In [5] the authors extend the CAN protocol using the Hilbert space-filling curve and load balancing mechanisms, while in [4] a Grid P2P extension of the Symphony DHT system is proposed. In both cases, the authors adopt a solution that maintain a separate DHT for each attribute of the resources present in the network, but the nodes of each DHT also store information concerning the other attributes. Then, query routing is performed only in the DHT of the attribute with the lowest selectivity, as it requires the lowest communication cost. Once the resources matching the sub-query of such an attribute are found, the values of the other attributes are checked in order to find the final set of matching resources. In [18] the authors use an extension of the Pastry DHT network. The system uses one Pastry ring for each attributes. The index portions store not only the values of the resources but also the root values of so-called *Aggregation Points*. These are prefix trees of the resources attribute values (which are the leaves of these trees) and they are used to perform range queries in XenoSearch. The matching results of each sub-query are finally intersected at the query originator node.

Another form of distributed indexes can be supported even on unstructured P2P networks by using the so-called RI [7]. RIs are based on the content of the data present on each node. Each peer in the network maintains both an index of its local resources and a table for every neighbor, which summarizes the data that is reachable trough all the path that start from that neighbor. When a peer receives a query, it checks if the requested items are present locally and then forwards the query to the neighboring node which has, accordingly to the RI, the most relevant data with respect to the query. The process is iterated until a stop condition is achieved (e.g. the desired number of results is reached).

One of the common limitations of many of the techniques proposed in the literature is their inefficiency in maintaining the indexes in the presence of dynamic data. The need for DHT systems to re-index the items whose values have changed may lead to a great amount of overhead when changes happen frequently. Moreover, the advantages of DHT networks on exact queries may be reduced when dealing with range queries, particularly when the requested ranges are sufficiently large or open. In these cases the lookup cost could become linear, thus loosing the traditional advantage of logarithmic lookup in DHTs. In addition to this, some solutions presented for the multi-attribute query case show some scalability problems with respect to the number of attributes, as they use one separate DHT for each attribute type. We try to address all of these limitations in this article, by proposing a solution to the problem of dynamic data location with multi-attribute, range queries. We use a form of RI in order to achieve a good tradeoff between query routing efficiency and the need to limit updates occurring when some data items change value.

3. Problem statement

We suppose that each peer in the system holds a (possibly empty) set of data items, also called *local repository*. Each data item is described by a set of attribute-value pairs. While the number of data items can be large, the number of different attribute names is usually limited. In a distributed relational database, a data item would be a database record, and the attribute-value pairs would be the names and corresponding value of the attributes of each table. In a Grid Information System, such data items would model Grid resources, each in turn characterized by a set of features (attribute-value pairs). We also suppose that data items are dynamic, i.e., the value of the attributes may change over time. Users of the P2P system want to locate data items satisfying given search criteria, which are expressed as partial range queries over the set of attributes.

In the following we consider a P2P system with a set $\mathscr{P} = \{P_1, P_2, \dots P_N\}$ of N peers. We denote with $Data(P_i)$ the local repository on peer P_i. Each data item $D \in Data(P_i)$ is labeled with a subset of attribute-value pairs. We suppose that there is a limited number M of possible attributes and corresponding data types, namely $\mathscr{A} = \{A_1 : T_1, \dots A_M : T_M\}$. Each T_i can be any arbitrary data type, subject to the constraint that must be possible to define a total ordering over T_i. Let $AttList(D) \subseteq \mathscr{A}$ be the subset of all attributes associated with $D \in Data(P_i)$, and $D.A$ be the value of attribute $A \in AttList(D)$.

The system has to provide a query facility for locating all data items matching a user-defined partial range query Q. We consider queries generated by the following grammar (we assume that the usual operator precedence rules apply):

$$Q := Q \quad \textbf{and} \quad Q|Q \quad \textbf{or} \quad Q|v_1 \leqslant A \leqslant v_2$$
$$A := A_1|\ldots|A_M$$

We consider partial range queries over subsets of the attributes, that is, boolean compositions of range predicates $v_1 \leqslant A \leqslant v_2$. Multiple conditions over different attributes are possible. Conditions such as $A \leqslant v_2$, $A \geqslant v_1$ and $A = v_1$ are special cases of $v_1 \leqslant A \leqslant v_2$ which can be expressed by setting $v_1 = -\infty$, $v_2 = +\infty$ and $v_1 = v_2$ respectively.

The operations that each peer has to support are the following:

insert(D,$\{A_1 : V_1, \ldots, A_r : V_r\}$): Insert a new data item D in the local repository; the data item has attributes $A_1, \ldots, A_r$ with values $V_1, \ldots, V_r$ respectively.
update(D,$A : V_{\text{new}}$): Change the value of attribute A for data item D on the local repository; the new value will be V_{new}.
delete(D): Remove data item D from the local repository.
lookup(Q): Search for data items matching query Q over the whole P2P system (including the current node). The operation actually returns the set of the addresses of all the peers containing data items matching Q. These peers must be directly contacted to eventually retrieve the information on stored data.

Additionally, peers may join and leave the system at any time. As usual in P2P systems, we want to rely as few as possible on any centralized information.

A trivial way of locating resources in an unstructured P2P network would be to flood the range queries to all nodes within a given radius from the originating peer. This is clearly undesirable, as (1) flooding generates a potentially high message load on all nodes, including those which do not hold resources satisfying the queries; and (2) setting a maximum hop count to stop the query from flooding the entire network does not guarantee that all matches are located.

In order to limit the flooding of queries, we build a specific overlay network over the set of peers, and associate routing information with individual links. In particular, in the following we discuss a couple of P2P systems that exploit different overlay networks and particular forms of RIs to route queries and updates of attribute values stored in each peer.

We want now to point out some considerations about multi-attribute query resolution. Let Q be a multi-attribute, range query over a attributes, such that $Q = q_1\ op\ q_2 \cdots op\ q_a$, where $op = \vee$ or $\wedge$. Let p be the probability for a peer to match query Q, p_i the probability to match the sub-query q_i. We can express p as a function of the various p_i's: if $op = \wedge$, we have that $p = \prod_{i=1}^{a} p_i$. On the other hand, If $op = \vee$, we have that $p = 1 - \prod_{i=1}^{a}(1 - p_i)$. Thus, the match probability p_{ij} of a query in the form $q_i \wedge q_j$ is $p_{ij} = p_i p_j$, where the match probability of a query in the form $q_i \vee q_j$ is $p_{ij} = 1 - (1 - p_i)(1 - p_j)$.

Note that, since the query routing strategy is thus based on a selective flooding based on RIs, and the number of nodes targeted by a query Q depends on p, we can expect similar performances of the system in the presence of the same value of p, although obtained by the combination of different numbers of attributes.

4. Tree vector: a tree-based P2P discovery system

In this section we discuss a P2P discovery system for Grid resources called *Tree Vector*. It maintains an overlay network $\mathcal{T}$, which is a spanning tree over the set of all the peers $\mathcal{P}$. The goal of $\mathcal{T}$ is to limit the number of hops of both query and update messages routed in the system. The performances of the system depends on the topological characteristics of $\mathcal{T}$. In order to avoid degenerate cases and improve performance, $\mathcal{T}$ should have a low diameter, and such property should be maintained as nodes join and leave the system. For this purpose, it is possible to use the algorithm described in [14] to maintain a spanning tree with bounded degree and logarithmic diameter.

Given a spanning tree $\mathcal{T}$ over $\mathcal{P}$, let $Nb(P_i)$ be the set of neighbors of each peer P_i, i.e., the set of all peers directly connected to P_i on the overlay network. Moreover, let $\mathcal{T}(P_i \rightarrow P_j)$ be the subtree of $\mathcal{T}$ that contains P_j, and does not contain $P_i \in Nb(P_j)$. That is, $\mathcal{T}(P_i \rightarrow P_j)$ is the subtree containing node P_j which has been obtained after removing the link $P_i \rightarrow P_j$ from $\mathcal{T}$ (see Fig. 1).

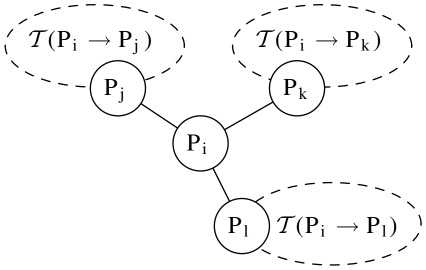
$\mathcal{T}(P_i \to P_j)$
$\mathcal{T}(P_i \to P_k)$
P_j
P_k
P_i
P_l
$\mathcal{T}(P_i \to P_l)$

Fig. 1. A portion of the tree-shaped overlay network.

Each peer P_i maintains a summary of all the information of its local resources as follows. If the domain of attribute A is the interval $[a,b]$, we select $k+1$ division points $a = a_0 < a_1 < \cdots < a_k = b$ such that $[a,b]$ is partitioned into k disjoint intervals $[a_i, a_{i+1})$, $i = 0, 1, \ldots, k-1$. Given an attribute A we encode the value $D.A$ with a k bit binary vector $DataIdx(D.A) = (b_0, b_1, \ldots, b_{k-1})$. If $D.A \in [a_i, a_{i+1})$ then $b_i = 1$ and $\forall j \neq i$, $b_j = 0$. Both the parameter k (number of bits of the bit vector) and the division points $a_0, a_1, \ldots, a_k$ may be different for each attribute. The final local bitmap index for an attribute A on peer P is obtained by a bitwise OR operation between the indices of all data items D of P:

$$NodeIdx(P_i, A) \equiv \bigvee_{D \in Data(P_i)} DataIdx(D.A) \tag{1}$$

Let us consider all the neighbors P_j, $P_j \in Nb(P_i)$, of a generic peer P_i. For each P_j, P_i keeps information on the data items which can be found by following the link $P_i \to P_j$ on the overlay network. To this end, P_i maintains an index called $LinkBitIdx(P_i \to P_j, A)$ for each attribute A of each data item D in $\mathcal{T}(P_i \to P_j)$. Since P_i can receive information only from its neighborhood, the index is calculated recursively in the following way:

$$LinkBitIdx(P_i \to P_j, A) \equiv NodeIdx(P_j, A) \bigvee \left(\bigvee_{P \in Nb(P_j) - P_i} LinkBitIdx(P_j \to P, A) \right) \tag{2}$$

The final result is that the index contains the bitwise union of all the bitmap indices $NodeIdx(P, A)$ associated with every peer P in $\mathcal{T}(P_i \to P_j)$. Note that $LinkBitIdx(P \to P', A)$ is a binary string of the same size of $NodeIdx(P, A)$, with possibly more than one bit set to 1.

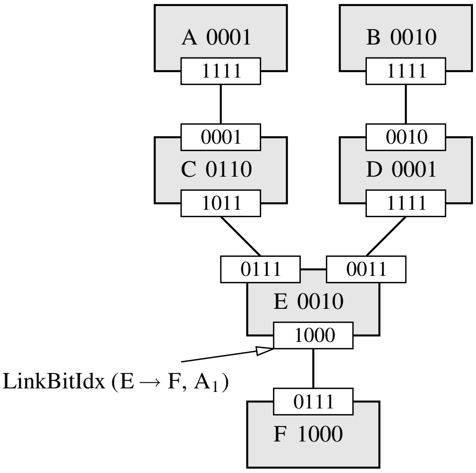

Fig. 2. Example of P2P network with bit vector indices.

Fig. 2 shows a P2P network with a single attribute A_1, whose values are encoded with a 4-bit vector index. The binary strings in the shaded boxes represent the bit vector indices for the local repository; binary strings in the small white boxes represent the values of $LinkBitIdx(P \rightarrow P', A_1)$. For example, node E has a local repository where $NodeIdx(E, A_1) = 0010$; the value of $LinkBitIdx(E \rightarrow F, A_1)$ is 1000.

Observe that $LinkBitIdx(A \rightarrow C, A_1)$ is, according to Eq. (2), the logical OR of the bit vector representation of values of $NodeIdx(E, A_1)$ for all nodes $P \in \{B, C, D, E, F\}$.

4.1. Handling queries

We now illustrate how queries are processed. Assume that a query Q originates from any node P in the system. As in Gnutella [10], Q is propagated from node P to its neighbors using a Breadth First Search (BFS) algorithm; however, unlike Gnutella, queries are not necessarily routed to all neighbors: our system performs a Directed BFS (DBFS) over the tree overlay network. The DBFS is driven by the vector indices associated with individual peers connections.

Recall from the previous section that node P knows the bit vector $LinkBitIdx(P \rightarrow P', A)$, for each $P' \in Nb(P)$, where $LinkBitIdx(\cdot)$ is defined according to Eq. (2). Suppose that node P receives from one of its neighbors P_{in} a multi-attribute query Q, where $Q_A := v_1 \leqslant A \leqslant v_2$ is one of its sub-queries. P thus receives the bit vector representation of each sub-query Q_A, i.e., $QueryIdx(Q, A) = (b_0, \ldots, ,b_{k-1})$, where $b_i = 1$ iff $\exists v$ in the corresponding domain interval $[a_i, a_{i+1})$ such that $v_1 \leqslant v \leqslant v_2$. The query is propagated along the connection from P to $P_{\mathrm{out}} \in Nb(P) - P_{\mathrm{in}}$ if a match is likely to be present in $\mathscr{T}(P \rightarrow P_{\mathrm{out}})$. A necessary condition for the existence of a match is that the logical AND between $LinkBitIdx(P \rightarrow P_{\mathrm{out}}, A)$ and $QueryIdx(Q, A)$ is nonzero.

Algorithm 1 illustrates the pseudocode executed by P to process a query message. Upon receiving a query from neighbor P_{in}, the query is forwarded to the remaining neighbors which have a potential match. Results are fanned back to P_{in}, until they eventually reach the originator. Note that this approach only works if the overlay network is guaranteed to be acyclic (i.e., is a tree), as we are assuming. The result of a query is the set of all peers with local data items matching the search criteria. For sake of readability we only present the case in which the query comes from another node in the network. In case of it originated directly from a user, P exactly performs the same operations, using an undefined value for P_{in} and returning the results back to the user. The pseudocode of function $Match(Q, P_i \rightarrow P_j)$, which is used to test for a potential match of query Q on the subtree $\mathscr{T}(P_i \rightarrow P_j)$, is shown in Algorithm 2.

Algorithm 1. lookup(Q) executed by peer P

```
loop
  Wait for query $Q$ from some $P_{in} \in Nb(P)$
  Let $R := \emptyset$                    {Query result}
  for all $P_{out} \in Nb(P) - P_{in}$ do
    if $Match(Q, P \rightarrow P_{out})$ then
      Relay $Q$ to $P_{out}$
      Let $R'$ be the reply reported by $P_{out}$
      Let $R := R \cup R'$
  if There are local matches to $Q$ then
    Let $R := R \cup$ the set of local resources matching $Q$
  Report $R$ to $P_{in}$
```

Algorithm 2. $Match(Q,P_i \rightarrow P_j)$

```
if $Q = Q_1$ and $Q_2$ then
  Return $Match(Q_1,P_i \rightarrow P_j) \wedge Match(Q_2, P_i \rightarrow P_j)$
else if $Q = Q_1$ or $Q_2$ then
  Return $Match(Q_1,P_i \rightarrow P_j) \vee Match(Q_2, P_i \rightarrow P_j)$
else if $Q$ is a single attribute query then
  Return $(LinkBitIdx(P_i \rightarrow P_j, A) \wedge QueryIdx(Q, A) \neq 0)$
```

4.2. Handling updates, deletions, and insertions

We now describe how updates can be processed. Suppose that an update affects $D.A$, where $D \in Data(P)$ and $A \in AttList(D)$. Let v_{new} and v_{old} be the new and old values of $D.A$, respectively. The first action taken by P is to compute the bitmap representation of v_{new}, $DataIdx(v_{\text{new}})$. If it is equal to $DataIdx(v_{\text{old}})$, no other actions are needed. Otherwise, P computes the new $NodeIdx(P, A)$. Again, it may happen that the new index is the same as the old one, and then no further actions are required. In case $NodeIdx(P, A)$ differs from the previous one, an update message is propagated. Update messages consists of the name of the attribute whose value is changed, and its up-to-date bit vector representation. The updated bit vector representation for attribute A to be associated with the link $P_{\text{out}} \to P$ can be re-computed by P using Eq. (2).

Algorithm 3 describes the actions taken by peer P when it notices a change in the local data store. Each peer executes Algorithm 4 to process update messages coming from incoming connections. It is very similar to Algorithm 3: updated bit vector indices are computed according to Eq. (2) and sent to neighbors.

Moreover, also the insertion/deletion of new data items into the P2P system can be done with the same algorithms just described for updates. When a new data item D is registered at (removed from) peer P, then for each $A \in AttList(D)$, P executes the procedure **initiate**_*update*(A,$D.A$) (outgoing messages can be batched together for efficiency).

Algorithm 3. initiate_update(A, v_{new}) executed by peer P

```
Let $v_{\text{old}} := D.A$
if $DataIdx(v_{\text{new}}) \neq DataIdx(v_{\text{old}})$ and $NodeIdx(P, A_{\text{new}}) \neq NodeIdx(P, A_{\text{old}})$ then
  for all $P_{\text{out}} \in Nb(P)$ do
    Let $B := NodeIdx(P, A)$
    for all $P' \in Nb(P) - P_{\text{out}}$ do
      Let $B := B \vee LinkBitIdx(P \to P', A)$
    Send bit vector $B$ for $A$ to $P_{\text{out}}$
```

Algorithm 4. process_update( ) executed by peer P

```
loop
  Wait for bit vector $B$ for $A$ from $P_{\text{in}}$
  if $B \neq LinkBitIdx(P \to P_{\text{in}}, A)$ then
    Let $LinkBitIdx(P \to P_{\text{in}}, A) := B$
    if $NodeIdx(P, A) \vee B \neq B$ then
      for all $P_{\text{out}} \in Nb(P) - P_{\text{in}}$ do
        Let $B' := NodeIdx(P, A)$
        for all $P' \in Nb(P) - P_{\text{out}}$ do
          Let $B' := B' \vee LinkBitIdx(P \to P', A)$
        Send $B'$ to $P_{\text{out}}$
```

4.3. Experimental results

We conducted several simulation experiments in order to evaluate the performances of *Tree Vector*. The experimental settings are the following. We consider an N node P2P system with single attribute data items. Attribute values are uniformly and randomly distributed in the $[0, 1]$ interval. We consider the following overlay tree network topologies: random, balanced with degree 5, and balanced with degree 10. We developed a C++ process-oriented simulation model in which the behavior of each peer is represented by a simulation process. Simulation results were computed as intervals with 90% confidence level and each measurement was repeated multiple times in order to get confidence intervals having width of less than 5% of the central value. The simulation model has been implemented using the C++ library described in [11].

We first analyze the maximum number of routing hops (*query radius*) needed to locate a data item as a function of the network size. Fig. 3 shows the results for three different overlay network topologies. In Fig. 3 we plot the total number of queried nodes (*query span*) as a function of the network size, for different topologies. Both the query radius and query span are Lower is Better (LB) metric. The data points were calculated by performing 100 random range queries on the network, each one originating from a uniformly chosen node.

As we can see, the query radius grows as O(log(N)), while the query span grows as O(N), N being the size of the network. Note also in Fig. 3 that the number of matches is linear with the size of the network. As the query mechanism is guaranteed to locate every existing match, the number of matches is a lower bound for the query span. Thus the query span is optimal considering the number of matches.

We can also define the precision of the query routing strategy. The *query precision* is defined as the ratio between the number of data items matching the query and the number of items matching the bit vector representation of the query (*Number of real matches/Number of potential matches*).

In our tests, we considered a network of $N = 1000$ nodes, and performed 100 range queries given a match probability p, which represents the probability that a node matches the query. The number of data items, distributed over the N nodes of the network, was exactly equal to N. The data items were characterized by a single attribute A, with values uniformly distributed in [0, 1]. In addition, the [0, 1] interval of the attribute values was partitioned into k equally sized bins.

The single-attribute range queries were of the form of $v \leqslant A \leqslant v + p$, for v uniformly chosen in [0, $1 - p$]. Basically, we thus performed different queries each given *interval width p*. However, since the attribute values were uniformly distributed in [0, 1], this corresponds to a match probability exactly equal to p, as stated above in our testing hypotheses.

In Fig. 4 we show the precision of our algorithm as a function of match probability p, where the simulation setting was the one discussed above.

From the figure we see that the precision turns out to be higher as the number k of bits in the vector indices increases. Also, the precision increases for large values of the match probability p. We try to explain this behavior by deriving an analytic formulation of the query precision. Note that the expected number of data items matching a range query of match probability p is Np. Moreover, due to the uniform distribution of the values of attribute A in the interval [0, 1], for $0 < p \leqslant 1 - 1/k$, the expected number of *false positives* (i.e., data items whose bit vector indices match the query, but their exact attribute values do not) is N/k. The reason is that, on average, each side of the $[v, v + p]$ interval will cover half a bin. The precision is thus $Np/(Np + N/k) = kp/(kp + 1)$. Finally, if $p > 1 - 1/k$, the expected number of false positives is $N(1 - p)$, which is the expected number of data items falling *outside* the interval $[v, v + p]$. In this case the precision is simply p. Let *Prec*(k,p) be . We can thus define *Prec*(k,p), i.e., the precision of queries with match probability p on a system with k bit indices, as follows:

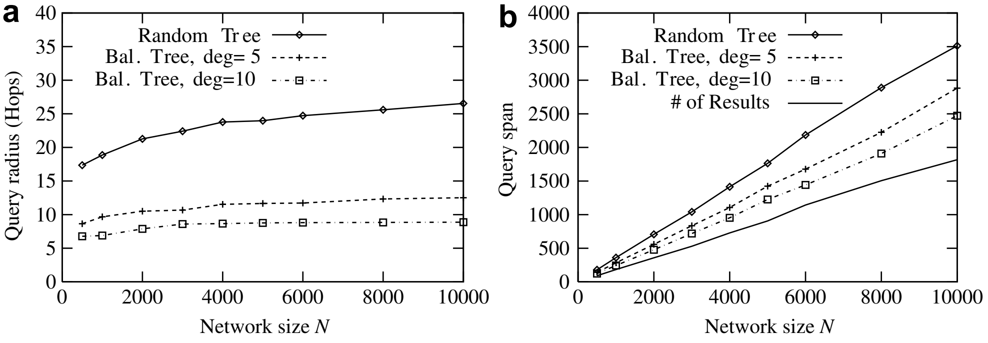

Fig. 3. (a) Query radius and (b) query span as a function of the network size ($k = 32$, lower is better).

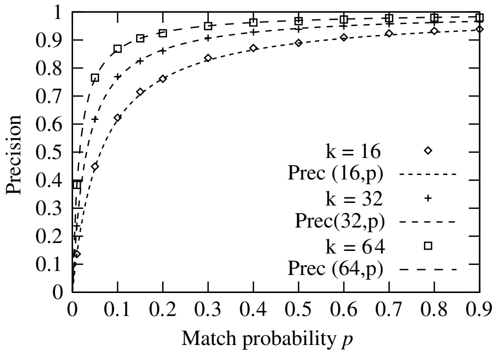

Fig. 4. Precision as a function of match probability ($N = 1000$, random tree, higher is better). Function $Prec(k,p)$ is defined in (3).

$$Prec(k,p) = \begin{cases} \frac{kp}{kp+1} & \text{if } 0 < p \leqslant 1 - 1/k \\ p & \text{if } 1 - 1/k < p \leqslant 1 \end{cases} \tag{3}$$

Fig. 4 confirms that this analytic formulation of precision is highly accurate.

We finally analyzed the behavior of the update mechanism. In Fig. 5a we plot the mean number of hops traversed by an update message (update radius) as a function of the network size; In Fig. 5b we plot the number of nodes reached by an update message (update span) as a function of the network size. From the figures we can observe that both the update radius and update span are independent of the network size. On the other hand, they are influenced by the degree of peers on the overlay network: a balanced tree of degree 10 produces larger update radius and spans than the balanced tree of degree 5, with the random overlay network topology standing between them.

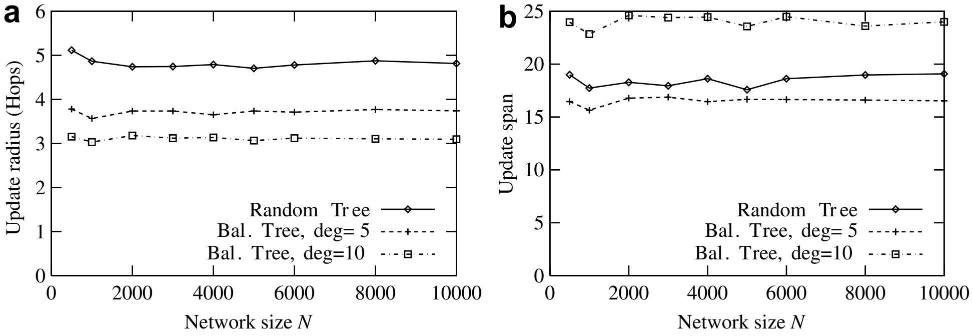

Fig. 5. (a) Update radius and (b) update span as a function of network size ($k = 16$, $p = 0.5$, lower is better).

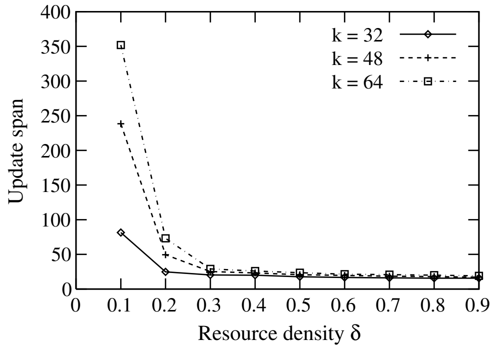

Fig. 6. Number of nodes updated as a function of δ ($N = 1000$, random topology, lower is better).

Fig. 6 plots the update span as a function of the resource density δ. The resource density δ is the probability that a node contains a resource. As usual, resource attribute values are uniformly distributed in the $[0,1]$ interval, hence the total number of resources in an N nodes network is $N\delta$. As expected, the update span decreases for larger values of δ: high data density implies that the vector indices associated with the links have a higher density of bits set to 1, thus updates are more likely not to propagate.

5. Forest of trees: a P2P system exploiting a hierarchical network

Despite its good performances, the structure described in the previous section has two main drawbacks. First, the topology may become hard to maintain when nodes join and leave the network, especially in the case we want the tree to remain (almost) balanced. Second, the nodes close to the tree root may become overloaded of routing requests. For these reasons, we also propose to partition the network into a set of trees, i.e. a forest. The new overlay network is thus called Forest of Trees. Each peer P belongs to a single tree $\mathcal{T}$, which we call *group* of P. The nodes that have a link with P are of two types: (1) nodes of same group, i.e. *local neighbors*, denoted with $LNb(P)$; (2) nodes that belong to other groups, i.e. *external neighbors*, denoted with $ENb(P)$. The set of all the peers connected to P is called the *neighborhood* of P and is denoted with $Nb(P)$, i.e. $Nb(P) = LNb(P) \cup ENb(P)$. For a given node P, $ENb(P)$ may be an empty set, but in order to have a connected network we require that, for each group of nodes, at least a node of the group must have an external neighbor, i.e. $\forall$ group G, $\exists P \in G | ENb(P) \neq \emptyset$. An example of a network of this type is shown in Fig. 7. For the sake of simplicity, the figure shows a *vertex clustering* of the network, i.e. only connections between groups are drawn.

We now describe how routing information and update schemes change with the new network structure, by first discussing the new bitwise RIs.

5.1. Bitmap indices

The information associated with of the local resources of a peer is computed in the same way as for the previous network. On the other hand, since we now have internal and external neighbors, we have to change the definition of the information associated with each outgoing link from a node. More precisely, given a peer P we want to make a distinction between the information related to resources present in the same group of P, and that related to the resources that P can reach through the external neighbors of either P or all the other nodes belonging to the group of P.

Let us change the definition of Eq. (2) in the following way:

$$LinkBitIdx(P_i \to P_j, A) \equiv NodeIdx(P_j, A) \bigvee \left(\bigvee_{P \in LNb(P_j) - P_i} LinkBitIdx(P_j \to P, A) \right) \tag{4}$$

As can be seen, $LinkBitIdx(\cdot)$ is now computed over only the local neighbors of a node. Thus, each peer P_i computes the quantity $LinkBitIdx(P_i \to P_j, A)$, for all $P_j \in LNb(P_i)$. This is used as an *internal routing index*, since it describes only the resources available inside a given group.

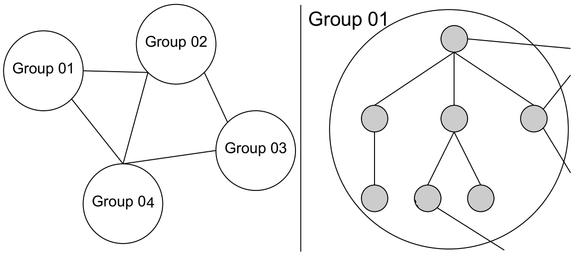

Fig. 7. An example of the Forest of Trees network organization.

As said in the previous section, each group has at least one external neighbor, i.e., there exists at least one of its member nodes that has an external connection with a node in another group. We want to exploit this fact and spread external information within the group, in order to allow each peer to reach resources that are located in groups different from its own. Let G be a group. We want each peer of G to know not only the resources it can find inside G, but also the resources that are reachable through external connections of the nodes of G. Let P_i be a node of G. P_i can directly reach resources that are located in an external group G_{ext}, connected to G, if P_i has at least one node of G_{ext} among its external neighbors. Otherwise, it can reach them by routing a query to another internal node of G that has a connection with G_{ext}. To this end, P_i maintains an *external bitmap index* that is computed in the following way:

$$ExtBitIdx(P_i \to P_j, A) = \begin{cases} LinkBitIdx(P_i \to P_j, A) & \text{if } P_j \in ENb(P_i) \\ \bigvee_{P \in Nb(P_j) - P_i} ExtBitIdx(P_j \to P, A) & \text{if } P_j \in LNb(P_i) \end{cases} \tag{5}$$

where $Nb(P_j) = ENb(P_j) \cup LNb(P_j)$.

In the above equation, $LinkBitIdx(P_i \to P_j, A)$ is computed as for Eq. (4). Note that in this case, since $P_i \notin LNb(P_j)$, $LinkBitIdx(P_i \to P_j, A)$ gives to P_i a complete summary of all the resources that can be found in the nodes of group G_{ext}, where $P_j \in G_{\text{ext}}$.

More generally, if $P_j \in ENb(P_i)$, then the new index $ExtBitIdx(P_i \to P_j, A)$ gives to P_i a description – according to the value of attribute A – of the resources of an external group G_{ext}, where $P_j \in G_{\text{ext}}$. Conversely, if $P_j \in LNb(P_i)$, then $ExtBitIdx(P_i \to P_j, A)$ gives to P_i a description of the external resources (belonging to neighboring groups) that are available through external connections from the local neighbor P_j. Fig. 8 shows an example of a system with two groups G_1 and G_2; only external indices of nodes A and C are shown. Internal indices are omitted, as they are computed in exactly the same way as for Tree Vectors.

5.2. Query routing

The routing algorithms of Forest of Trees also needs to be modified. As for the previous case, when a peer P receives a query $Q := v_1 \leqslant A \leqslant v_2$ from a neighbor P_{in}, it forwards it to neighbor $P_{\text{out}} \in Nb(P) - P_{\text{in}}$ only if a match is likely to be present in either $\mathscr{T}(P \to P_{\text{out}})$ or in one of the external groups connected to the nodes in $\mathscr{T}_G(P \to P_{\text{out}})$ where $\mathscr{T}_G(P \to P_{\text{out}})$ is the representation of group G as a tree rooted in $P_{\text{out}} \in G$. To this end, the result of the logical AND between the bitmap representation of the query range, and at least one of both $LinkBitIdx(P \to P_{\text{out}}, A)$ and $ExtBitIdx(P \to P_{\text{out}}, A)$ must be a nonzero vector.

As a consequence of the creation of two types of indices, we also have two possible types of matches. A query is forwarded to a node if the at least one between the internal and external indices shows a match.

The routing scheme described so far may not be enough for an efficient routing of queries. Note that the inter-group graph is unstructured. Therefore, it may also include loops among the groups. Let us consider the

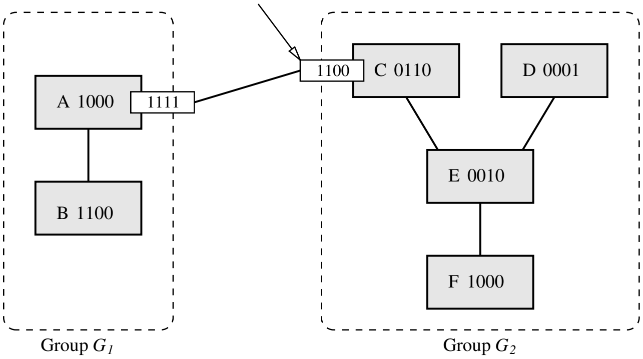

Fig. 8. Example of group indices.

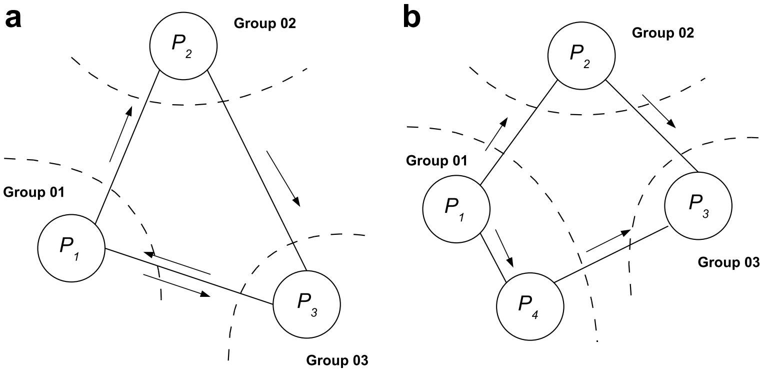

Fig. 9. Examples of loops (a) and multiple resolutions of the same query (b).

situations depicted in Fig. 9. In the first case, P_1, P_2 and P_3 have at least a resource matching query Q. Let us suppose that P_1 receives Q. After P_1 has detected the local match, it forwards the query to its neighbors. Node P_2 forwards the query to P_3. P_3 will then send the query back to P_1, that, in turn, will forward it again to P_2. The final result is an infinite loop.

In order to avoid loops, we need to improve the routing scheme. We add a further information to that associated with a query message: the list of all the IDs that are associated with the node groups that were already traversed by the query. Let *Q.Groups* be such a list. This is useful to prevent forwarding multiple times the same query Q from a group to another, when more than a single links connect nodes of the groups. When a peer P forwards Q to P', where P and P' belong to distinct groups, P adds the group ID of P' to *Q.Groups*. In this way, the nodes belonging to the same group of P forward Q to an external neighbor P' only if the group ID of P' is not listed in *Q.Groups*. Nothing changes for the internal neighbors. This solution eliminates cycles, but does not completely avoid the problem illustrated in Fig. 9. A query Q is forwarded by node P_1 to all nodes P_2, P_3 and P_4 that match it. As can be seen, node P_3 receives the same query from node P_2 and node P_4, which belong to distinct groups. The previously described mechanism does not prevent P_3 from processing the same query twice. For this reason we need to associate with each peer P a *query cache*, i.e. a cache that contains the latest m query IDs received by P. If P receives a query whose ID is already listed in its own query cache, it simply discards the query, avoiding processing it twice.

The new query routing method is described in Algorithm 5. As for the corresponding algorithm for the single tree, in Algorithm 5 we only show the case in which the query comes from another node in the network. The only difference between the two cases, i.e. queries coming from a neighbor or originated from the user, is the following: In the latter case P adds its own group ID to the groups traversed by query Q, poses an undefined value for P_{in} and return the results back to the user. Function $Match(Q, P \to P_{\text{out}})$, used in Algorithm 5, is exactly the same as the one illustrated in Algorithm 2, except for the last row of the pseudocode that is replaced by:

$$\texttt{Return}((LinkBitIdx(P_i \to P_j, A) \wedge QueryIdx(Q, A) \neq 0)\ \textbf{or}\ (ExtBitIdx(P_i \to P_j, A) \wedge QueryIdx(Q, A) \neq 0)).$$

Algorithm 5. lookup(Q) executed by peer P

```
loop
  Wait for query $Q$ from some $P_{\text{in}} \in Nb(P)$
  if $Q \notin P.QueryCache$ then
    Add $Q$ to $P.QueryCache$
    Let $R := \emptyset$                {Query result}
    for all $P_{\text{out}} \in Nb(P) - P_{\text{in}}$ do
      if $P_{\text{out}} \in ENb(P)$ then
        if $P_{\text{out}}.Group \notin Q.Groups$ then
```

```
            Add $P_{\mathrm{out}}.Group$ to $Q.Groups$
        else if $P_{\mathrm{out}}.Group \notin Q.Groups$ then
            Continue
    if $Match(Q,P \to P_{\mathrm{out}})$ then
        Relay $Q$ to $P_{\mathrm{out}}$
        Let $R'$ be the reply reported by $P_{\mathrm{out}}$
        Let $R := R \cup R'$
if There are local matches to $Q$ then
    Let $R := R \cup \{P\}$
Report $R$ to $P_{\mathrm{in}}$
```

5.3. Handling updates, deletions and insertions

The new network topology involves changes for the update scheme too. In particular, when a local update happens, the new internal bit vector is computed according to Eq. (4). Similarly, if there is a change in an external node, the *ExtBitIdx*(·) index associated with the inter-group link is updated. Then the change is propagated inside the group to update *ExtBitIdx*(·) indices only, according to Eq. (5).

Note that, as for the single tree case, the propagation of an update is stopped when it reaches a node where the new and old indices do not differ. Keeping this remark in mind, note that if a change happens on the local resources of a node P_i of a group G and this changes the RI for G, it has to be communicated to the external neighbors of G, in order to respect Eq. (5), thus transforming a local event into an external communication. An update can then be transmitted both internally, as an update of *LinkBitIdx* (·), and externally, as an update of *ExtBitIdx*(·). When it reaches an external group, it is forwarded inside that group only as an update of the *ExtBitIdx*(·) index. Note that update messages directed to an external group are not further propagated outside that group.

As for the Tree Vector case (see Section 4.2), object insertions and deletions are treated as updates and the same algorithm used for updates is executed in both cases.

5.4. Experimental results

In this section we report the results of simulation experiments performed to study how different measures – like query radius (hops), query span, precision, recall, etc. – change as a function of various parameters and network configuration.

Note that in a P2P system based on Forest of Trees, a query can be stopped during its propagation, thus not reaching all the nodes matching a given query. So, it becomes important to also study the *recall* of a query, i.e., the ratio between the number of (real) matches returned by a query, and the total number of matches existing in the whole network.

We start this discussion by an analytic evaluation of query radius and update propagation.

5.4.1. Analytical evaluation of query radius

In this section, we compute an analytical expression for the mean number of groups visited by a query. This quantity is directly related to the *query radius*, i.e. the maximum number of hops a query traverses. Let us consider a chain of $L+1$ groups $G_1, G_2, \ldots, G_{L+1}$, where for each i, $1 < i < L+1$, group G_i is connected with groups G_{i-1} and G_{i+1}. We assume that a query originates in a node of group G_1. We denote with p the match probability (the probability that a match can be found in a single node), and with N_G the mean number of nodes in each group. The probability P^G_{nomatch} that no match is found in a group can be expressed as:

$$P^G_{\mathrm{nomatch}} = (1-p)^{N_G}$$

Thus, $P_{\mathrm{Gmatch}} = 1 - P^G_{\mathrm{nomatch}}$ is the probability that at least one value matching the query can be found in the group. According to the query routing algorithm, inter-group query propagation stops when the query reaches a group whose neighbors (apart the one from which the query originated) do not have matches. We denote with X_L the discrete random variable representing the *number of inter-group hops* of a query over a chain of $L+1$ groups (thus, $X_L \in \{1,2,\ldots,L\}$). We know that $X_L = 0$ when G_2 contains no matches. This happens with probability P^G_{Gnomatch}. In general, for every integer $i \in \{0,1,\ldots,L\}$, the probability $\Pr(X_L = i)$ that the query traverses i hops can be expressed as:

$$\Pr(X_L = i) = \begin{cases} P^G_{\text{nomatch}} P^i_{\text{Gmatch}} & \text{if } 0 \leqslant i < L \\ P^L_{\text{Gmatch}} & \text{if } i = L \\ 0 & \text{otherwise} \end{cases}$$

The mean value of X_L is:

$$E[X_L] = \sum_{i=0}^{L} i \Pr(X_L = i) = \frac{P_{\text{Gmatch}}(P^L_{\text{Gmatch}} - 1)}{P_{\text{Gmatch}} - 1}$$

We plot in Fig. 10 the value of $E[X_L]$ as a function of L, for different values of the match probability p and mean group population N_G. The plots show the horizontal limit $P_{\text{Gmatch}}/(1 - P_{\text{Gmatch}})$: this means that as the chain length increases, the expected number of visited groups is bounded by $P_{\text{Gmatch}}/(1 - P_{\text{Gmatch}})$. In terms of our algorithm, this means that long chains of groups tend to degrade the *recall*, as the query routing algorithm is likely to stop forwarding the query early, even if matches could be found later on the chain. It is therefore very important to build the inter-group links by trying to shorten the group chain lengths, , i.e., to build networks where the average shortest paths, computed over the inter-group links, are kept small.

5.4.2. Analytical evaluation of update propagation

We now analyze the propagation of updates in the network. In Section 5.1 we described the two types of indices used in the system. *LinkBitIdx*(·) is maintained by each peer P as an *internal* RI. Since the internal topology of a group is a tree, the updates concerning *LinkBitIdx*(·) are handled in the same way as the updates for the single tree network. Thus, the update span and radius within a group will be the same as Tree Vector with a number of nodes equal to the group size.

The main difference between the two models is then determined by the presence of the *ExtBitIdx*(·) index. If a node P updates one of its local attribute index, this may result in a change in the external bitmap representation for the group P belongs to. In this case, the change has to communicate to P external neighbors. This fact can be derived also by the formulation of the *ExtBitIdx*(·) index given by Eq. (5).

The most expensive case in terms of communications occurs when the internal diffusion of an update generated by P also leads to a diffusion of an *ExtBitIdx*(·) update to all the external neighbors of P's group. However, when this update reaches an external group, it is no longer forwarded outside that group.

As previously stated, this worst-case scenario happens only if the update of attribute A on node P implies a change in the group's bitmap index for A . Suppose that attribute A is indexed by a k-bit bitmap index, and that the attribute value change implies a change in the ith position of the index. Let G_P be the group of P. The G_P representation for attribute A will not change if and only if there exists at least one node $P_j \neq P$ in G_P whose value for A is indexed exactly in the ith position of the index.

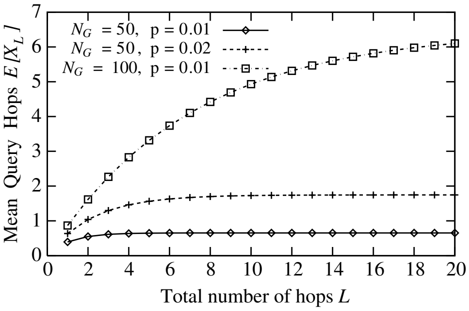

Fig. 10. Average number of inter-group hops $E[X_L]$ for a query message as a function of the total number of hops L.

Let p_i be the probability that a node has a resource whose index for A is set in the ith position. If the attribute values are uniformly distributed over their domains, $p_i = 1/k$. Then, $(1 - p_i)$ represents the probability for a node to not have such a setting in the index for attribute A. Let $N_G = |G_P|$. If an update on node P changes the *i-th* position of the index, the probability that the global G_P index is also changed is: $(1 - p_i)^{N_G - 1}$.

Thus, the larger a group size, the lower the chances that a local change must be forwarded to external neighbors. When groups are large enough, updates will mainly involve internal nodes.

5.4.3. Simulation results

The performances of the system based on a forest of trees has been evaluated by simulation, using the same approach described in Section 4.3. Again, simulation results are computed as confidence intervals with 90% confidence levels and width of less than 5% of the central value.

We have previously stated that, while the groups inner structure is a tree, the inter-group connections need not to be structured. In our simulation settings we considered two different inter-group network organizations. The first one is a scale-free network [3], while the second one is a network where each neighbor of each group is uniformly chosen between all the other groups, and the degrees of groups are uniformly distributed with a small variance. For both network organizations, we used the same average *group degree*. Moreover, also the number of edges connecting single nodes is almost the same in both networks.

We analyze the behavior of the system with respect to the group organization, as a function of the network size. We set the mean group size (number of nodes per group) as a fraction of the total number of peers in the system. This implies that the number of groups remains the same, while the number of nodes inside each group grows linearly with the network size. In our tests, if N_G is the mean group size, the actual number of nodes per groups varies uniformly in the range $\left[N_G - \frac{2}{3}N_G, N_G + \frac{2}{3}N_G\right]$.

We now present the simulation results for both network topology with respect to multi-attribute query resolution, propagation and update diffusion. The following results were obtained by using three different average values N_G of the group size: $0.0075N$ and $0.03N$, where N is the total number of nodes in the network. Moreover, we considered that the settings of the above simulations led to networks with only tens of groups. The largest number of groups is obtained in the first case ($N_G = 0.0075N$), corresponding to less than 150 groups. To study the network behavior with a larger number of groups, we used a lower value for N_G. In particular, in the simulations shown in Figs. 15 and 16, we used $N_G = 0.002N$. This means that the network has more than 500 groups. Another implication of this choice is that we had to test the system with larger sizes of the network, in order to have a reasonable number of nodes per group. Thus, the network size ranges from 5000 to 100,000 nodes.

In order to evaluate the query resolution performance, we executed 100 different random queries for each network size, each one originating from a uniformly chosen node. As results, we compute the average recall and the query radius (i.e., the maximum number of hops the query performed during diffusion) of these queries. See Figs. 11 through 16, which show these measures for both group topologies and different group sizes. We considered three different values for p, the probability that a node matches a query. More precisely, we considered $p \in \{0.2, 0.5, 0.8\}$.

Looking at Figs. 11, 13 and 15, we observe two facts about recall: (1) the recall is generally better for the scale-free network; (2) the recall is better for larger network sizes. The first finding is due to the fact that the diameter and the average shortest path of scale-free networks are lower than that of the uniform networks. Thus, in the latter case, the query has to span more groups in order to reach all the nodes that have potential matches. Let G_Q be the group where the query originated and let G_i be another group in the network. The longer the path between G_Q and G_i, the higher the probability that a group with no matches lies in this path, as explained in Section 5.4.1. This implies that the forwarding along that path will likely be stopped, as it will not possible to route a query through a group with no matches (see Section 5.2). Obviously, the probability that a group has a match decreases when either its size or the node match probability p decreases. This explains the behavior pointed out in the second observation above.

The radius of queries are shown in Figs. 12, 14 and 16. The radius is lower when the queries do not reach all the nodes. In these cases, the value of p becomes relevant, as it also determines the matching probability of each group. As previously stated, the fact that a group does not contain any matching node may break the routing chain from the originator node to the other peers. The query radius is smaller in the scale-free net-

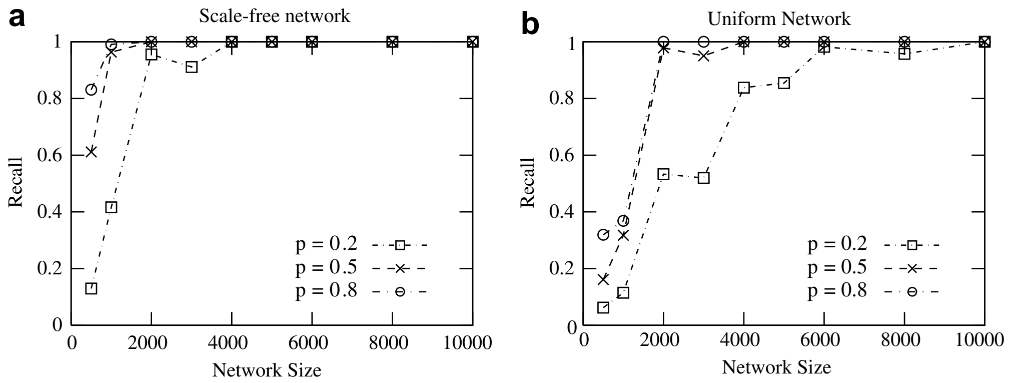

Fig. 11. Recall for the scale-free (a) and the uniform distributed (b) networks, for $N_G = 0.0075N$.

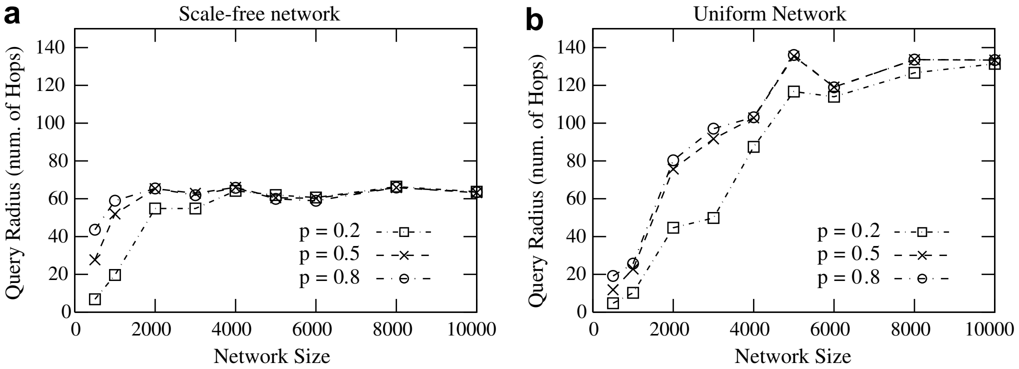

Fig. 12. Query radius for the scale-free (a) and the uniform distributed (b) networks, for $N_G = 0.0075N$.

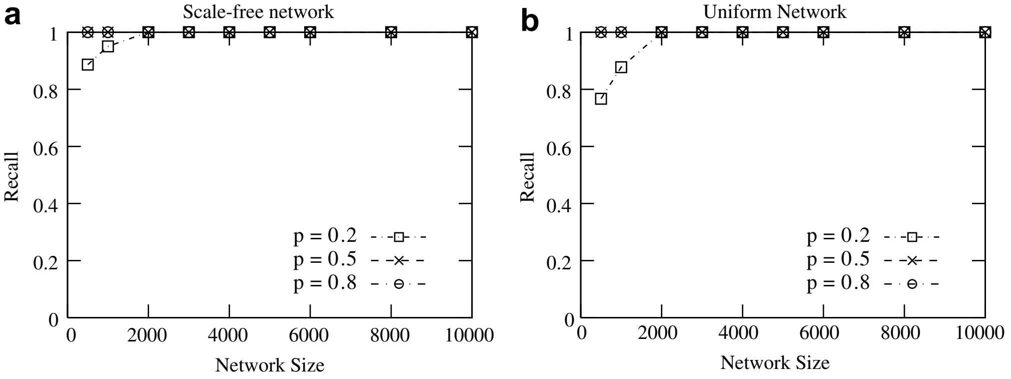

Fig. 13. Recall for the scale-free (a) and the uniform distributed (b) networks, for $N_G = 0.03N$.

work, since its diameter is also smaller. Moreover, when p is high enough to allow each matching node to receive the query, the query radius seems to stabilize in the scale-free case. In the uniform network it grows as the size of the system increases. The diameter of a scale-free network is not only lower than that of the other type of network, but it is also less influenced by the growth of the network.

The results regarding the propagation of updates, i.e. update radius and span, are very similar to those of the single tree case. As we already pointed out, the internal structure of each group is a tree, and the updates

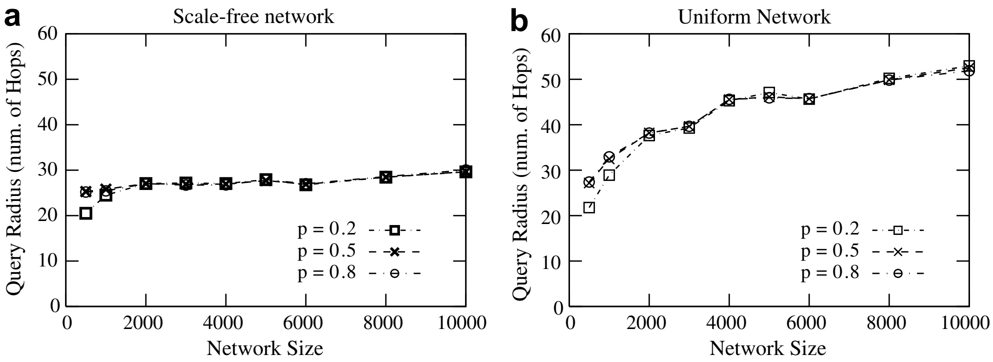

Fig. 14. Query radius for the scale-free (a) and the uniform distributed (b) networks, for $N_G = 0.03N$.

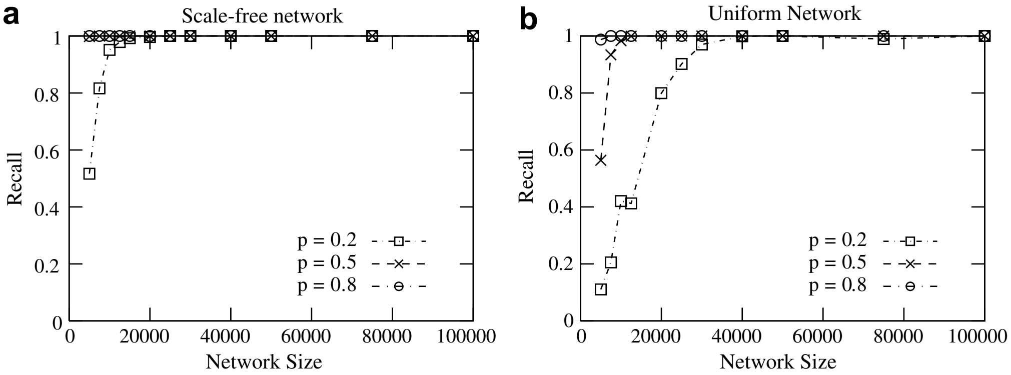

Fig. 15. Recall for the scale-free (a) and the uniform distributed (b) networks, for $N_G = 0.002N$.

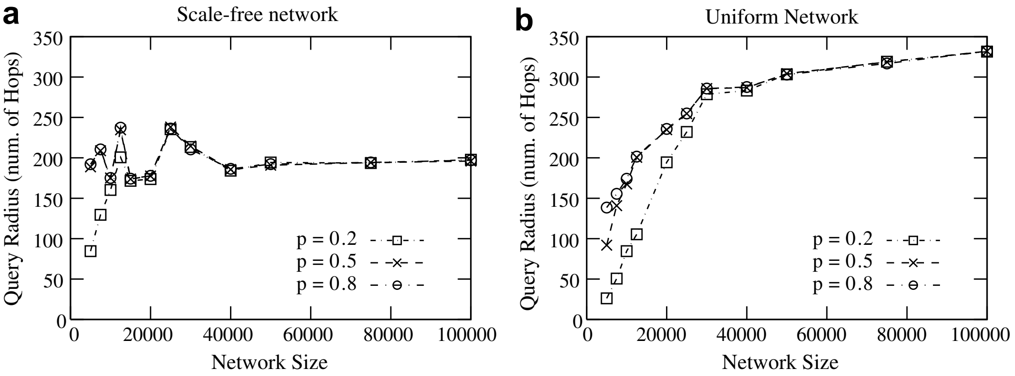

Fig. 16. Query radius for the scale-free (a) and the uniform distributed (b) networks, for $N_G = 0.002N$.

are propagated within each group according to Algorithm 4, thus obtaining results that are similar to those concerning tree-shaped networks of limited sizes. The difference is that, in the Forest of Trees setting, updates may need to be propagated to neighboring groups. Once a neighbor is contacted, the update is propagated to its internal nodes, but is not sent to other groups. Thus, the number of nodes affected by an update message depends on the group connectivity. However, if groups are made up of many nodes, an update is unlikely to change the bit vectors associated to the group external links, and in this case the update will *not* be propagated outside the group (see Section 5.4.2).

6. Conclusions

In this paper we studied the problem of resource discovery in dynamic Grids by means of P2P techniques. We considered systems where peers hold a set of local resources, each resource being described by a set of associated attributes. Attribute values vary over time, and this makes most of the existing P2P approaches not adequate. Users can locate resources by performing range queries over the set of all attributes. We described a routing strategy based on bit-vector RI, which can be used to route queries towards nodes of the system where matches are likely to be found. Moreover, the bit-vector indices can effectively be updated when attribute values change. We showed that the update and query propagation algorithms are efficient since they do not propagate messages over the whole network.

We applied bit-vector indices to two different peer topologies. The first one is a simple topology based on a single tree, where peers are connected via a tree-shaped overlay network. We then proposed a more relaxed network topology based on a forest of trees: in this network we thus have multiple groups of nodes, internally connected as a tree, while the inter-group connections are arbitrary.

One of the main feature of the solutions presented in this article is that they simplify the resolution of multi-attribute, range queries. Such queries are generally difficult to resolve when using DHTs. Many solutions presented in the literature resolve multi-attribute queries by means of a separate DHT for each attribute type. In these cases, the final result is computed by intersecting the lists of partial results obtained by each sub-query, one for each attribute (i.e., lists of matching resources). Moreover, even if the average complexity of locating a single item by using a DHT is usually logarithmic in the size of the network, the complexity may become linear in the case of queries asking for very large attribute ranges. This is because many DHT nodes, each responsible for a small range of attribute values, must be contacted sequentially.

Unlike the DHT-based networks, our RI approach is based on a natural partitioning/distribution of the global index to the various peers of the network, simply entailed by the resources that have been assigned to each peer node. For example, we can have one or more peer nodes for each Grid VO. Each node can thus compute its local index on the basis of its own resources, and all the attributes associated with them. Finally, every node is capable of locally resolving all the sub-queries on every type of attribute.

This is one of the main results we achieved, by using RI over a tree-shaped overlay network. This network supports frequent updates of the attribute values of resources, without the need to broadcast changes to all the nodes. Note that this is due to the intrinsic properties of both the proposed protocol and the bitwise RIs.

We proposed a second solution based on a forest of trees. This solution not only preserves these nice features of the simpler, tree-based one, but also introduces a hierarchical P2P network that is easier to maintain and manage. RI information can be maintained easily also in the forest-based approach, with the drawback that queries might not be able to retrieve all matching resources, thus obtaining a recall that is less that 100%. However, we showed that, under reasonable conditions, the system can still achieve a good recall, while the network topology ensures a limited radius of query propagation. This happens if the inter-group connections of this hierarchical network topology are modeled as a scale-free network – which is a very common case in the Web and in P2P networks – and the average group size is large enough.

We used simulation results supported by analytical evaluations in order to assess the performance of the query and update routing algorithms for the single tree and forest scenarios. As performance measures we considered the number of hops of messages, precision and recall of queries, and number of nodes receiving a message. Experimental results show that both our solutions, the bitwise RIs are very effective in limiting the message span and number of hops.

As future work, we are currently extending the proposed algorithms using histogram indices instead of simple bit vectors, following an approach similar to [15]. This allows us to store also information on the approximate number of matches, which can be very useful for certain applications.

Another open problem which will be investigated is related to limiting the number of links a message is allowed to traverse before being destroyed. In this case users may be unable to get the full list of resources satisfying a query, as potentially useful resources may be beyond the horizon of messages. Clearly, a tradeoff between the value of the time-to-live counter and ability to recall a significant fraction of resources needs to be identified.

References

[1] A. Andrzejak, Z. Xu, Scalable, efficient range queries for grid information services, in: P2P '02: Proc. of the Second Int. Conf. on Peer-to-Peer Computing, IEEE Computer Society, Washington, DC, USA, 2002.

[2] H. Balakrishnan, M.F. Kaashoek, D. Karger, R. Morris, I. Stoica, Looking up data in p2p systems, Comm. ACM 46 (2) (2003) 43–48.

[3] A.-L. Barabási, A. Reka, Emergence of scaling in random networks, Science 286 (5439).

[4] A.R. Bharambe, M. Agrawal, S. Seshan, Mercury: Supporting scalable multi-attribute range queries, in: Proc. ACM SIGCOMM 2004 Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communication, ACM Press, 2004.

[5] M. Cai, M. Frank, J. Chen, P. Szekely, Maan: a multi-attribute addressable network for grid information services, in: GRID '03: Proc. of the 4th Int. Workshop on Grid Computing, IEEE Computer Society, Washington, DC, USA, 2003.

[6] A. Crainiceanu, P. Linga, J. Gehrke, J. Shanmugasundaram, P-tree: a p2p index for resource discovery applications, in: WWW Alt. '04: Proc. of the 13th Int. World Wide Web Conference on Alternate Track Papers & Posters, ACM Press, New York, NY, USA, 2004.

[7] A. Crespo, H. Garcia-Molina, Routing indices for peer-to-peer systems, in: ICDCS '02: Proc. of the 22nd Int. Conf. on Distributed Computing Systems (ICDCS'02), IEEE Computer Society, Washington, DC, USA, 2002.

[8] I.T. Foster, A. Iamnitchi, On death, taxes, and the convergence of peer-to-peer and grid computing, in: M.F. Kaashoek, I. Stoica (Eds.), IPTPS, Lecture Notes in Computer Science, vol. 2735, Springer, 2003.

[9] P. Ganesan, B. Yang, H. Garcia-Molina, One torus to rule them all: multi-dimensional queries in p2p systems, in: WebDB '04: Proceedings of the 7th International Workshop on the Web and Databases, ACM Press, New York, NY, USA, 2004.

[10] Gnutella protocol development. http://rfc-gnutella.sourceforge.net/ (2006).

[11] M. Marzolla, `libcppsim`: a Simula-like, portable process-oriented simulation library in C++, in: G. Horton (Ed.), Proc. of ESM'04, the 18th European Simulation Multiconference, SCS–European Publishing House, Magdeburg, DE, 2004.

[12] M. Marzolla, M. Mordacchini, S. Orlando, A p2p resource discovery system based on a forest of trees, in: DEXA Workshops, IEEE Computer Society, Krakow, Poland, 2006.

[13] M. Marzolla, M. Mordacchini, S. Orlando, Tree vector indexes: Efficient range queries for dynamic content on peer-to-peer networks, in: Proc. of the 14th Euromicro Conference on Parallel, Distributed and Network-based Processing (PDP 2006), IEEE Computer Society, 2006.

[14] G. Pandurangan, P. Raghavan, E. Upfal, Building low-diameter peer-to-peer networks, IEEE J. Select. Areas Commun. 21 (6) (2003) 995–1002.

[15] Y. Petrakis, G. Koloniari, E. Pitoura, On using histograms as routing indexes in peer-to-peer systems, in: W.S. Ng, B.C. Ooi, A.M. Ouksel, C. Sartori (Eds.), DBISP2P, LNCS, vol. 3367, Springer, Toronto, Canada, 2004.

[16] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Schenker, A scalable content-addressable network, in: Proc. SIGCOMM '01, ACM Press, New York, NY, USA, 2001.

[17] A.I.T. Rowstron, P. Druschel, Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems, in: Middleware 2001: Proc. of the IFIP/ACM Int. Conf. on Distributed Systems Platforms, Springer-Verlag, Heidelberg, London, UK, 2001.

[18] D. Spence, T. Harris, Xenosearch: Distributed resource discovery in the xenoserver open platform, in: HPDC-12: Proc. Twelfth IEEE Int. Symposium on High Performance Distributed Computing, IEEE Computer Society, 2003.

[19] I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. Kaashoek, F. Dabek, H. Balakrishnan, Chord: a scalable peer-to-peer lookup protocol for internet applications, IEEE/ACM Trans. Netw. 11 (1) (2003) 17–32.

[20] D. Talia, P. Trunfio, Toward a synergy between p2p and grids, IEEE Internet Comput. 7 (4) (2003) 94–96.

[21] B. Zhao, J. Kubiatowicz, A.D. Joseph, Tapestry: An infrastructure for fault-tolerant wide-area location and routing, Tech. Rep. UCB Technical Report UCB/CSD-01-1141, Univ. of California Berkeley, Electrical Engineering and Computer Science Department (April 2001).