CoreGRID
European Research Network on Foundations, Software Infrastructures and Applications
for large scale distributed, GRID and Peer-to-Peer Technologies
A Network of Excellence funded by the European Commission

Tree Vector Indexes: Efficient Range Queries for Dynamic Content on Peer-to-Peer Networks

Moreno Marzolla, Matteo Mordacchini
{moreno.marzolla|matteo.mordacchini}@pd.infn.it
INFN Padova, via Marzolo 8, 35100 Padova, Italy

Salvatore Orlando
orlando@dsi.unive.it
ISTI Area della Ricerca CNR, via G. Moruzzi 1, 56124 Pisa, Italy

CoreGRID Technical Report
Number TR-0017
October 20, 2005

Institute on Knowledge and Data Management
Institute on System Architecture

CoreGRID - Network of Excellence
URL: http://www.coregrid.net

CoreGRID is a Network of Excellence funded by the European Commission under the Sixth Framework Programme

Project no. FP6-004265

Tree Vector Indexes: Efficient Range Queries for Dynamic Content on Peer-to-Peer Networks

Moreno Marzolla, Matteo Mordacchini
{moreno.marzolla|matteo.mordacchini}@pd.infn.it
INFN Padova, via Marzolo 8, 35100 Padova, Italy

Salvatore Orlando
orlando@dsi.unive.it
ISTI Area della Ricerca CNR, via G. Moruzzi 1, 56124 Pisa, Italy

Abstract

Locating data on peer-to-peer networks is a complex issue addressed by many P2P protocols. Most of the research in this area only considers static content, that is, it is often assumed that data in P2P systems do not vary over time. In this paper we describe a data location strategy for dynamic content on P2P networks. Data location exploits a distributed index based on bit vectors: this index is used to route queries towards areas of the system where matches can be found. The bit vectors can be efficiently updated when data is modified. Simulation results show that the proposed algorithms for queries and updates propagation have good performances, also on large networks, even if content exhibits a high degree of variability.

1. Introduction

Peer-to-Peer (P2P) networks have emerged as one the most successful way to share resources (e.g. data, storage, computational power) in a distributed, fault tolerant way. Large scale resource sharing is also the goal of Grid systems; for this reason, the P2P and Grid worlds are slowly converging [7, 15], 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: the user submits a job specifying its requirements (i.e., memory, disk space, Operating System version). 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 characteristics may vary over time. For example, the available disk space at a storage element varies as users place and remove data on/from it. Location of dynamic data on distributed, P2P-like systems is considerably more difficult than location of static data.

We may consider a set of resources on a typical Grid system organized as peers on a P2P network. Each resource has some attributes whose values identify its characteristics: CPU speed, free disk space, available memory and so on. Users query the system to locate resources satisfying some criteria, such as:

$$\begin{aligned} &(\text{CPU\_Speed} \geq 500\text{MHz}) \\ \textbf{and} \quad &(\text{OS\_Type} = \text{“Linux”}) \\ \textbf{and} \quad &(100\text{MB} \leq \text{Free\_Space} \leq 300\text{MB}) \end{aligned}$$

to denote all computational resources with a CPU speed of at least 500 MHz, running the Linux operating system, and with available disk space in the range $[100, 300]$ MB. In this example, some attribute values (e.g., the amount of free disk space) may change over time.

In this paper we consider the problem of locating dynamic data on P2P systems. In particular, we propose a protocol that allows peers to locate data matching range queries, i.e. queries that search for all data items whose values fall into a given interval.

The first P2P systems that tried to solve the data location problem usually flood the entire network until all the desired data are collected or a stop condition is reached. This behavior implies a large network traffic overhead which seriously limits the scalability of the system. In order to address this problem more efficiently, many data indexing systems have been proposed, such as Distributed Hash Table (DHT) [3]. DHTs only support exact matches. Many extensions and variations of these systems are described in the literature for supporting range queries, but only few of them can be used on dynamic content. In general, many indexing methods face serious overhead problems due to the need of re-index the items whose content have changed.

In this paper we extend and refine previous results introduced in [9], showing how to build an overlay structure over the set of peers and how to associate routing information with links of the overlay network in order to route queries toward potential matches. The routing information can be efficiently updated when data change. Such routing information corresponds to a condensed representation of all data owned by peers reachable along a link of the overlay network. Our aim is to build a P2P system with a reasonable trade-off between the need of efficiently route range queries and the ability to reduce the overhead needed to modify the indexes when data changes over time.

The rest of this paper is organized as follows: in Section 2 we relate our approach with other P2P systems described in the literature. In Section 3 we describe our approach in detail. In Section 4 we conduct a simulation study of our system and discuss the results. Finally, conclusions and future work are reported in Section 5.

2. Related Works

The problem of routing queries in P2P systems is a well known topic. In order to avoid flooding the network with query messages (as done by systems like Gnutella [1]), 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 [14] nodes are organized into a virtual circle, while in CAN [12] 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 [13] and Tapestry [16]. 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 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 [5] 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 [2] the authors propose an extension of the Chord protocol to support range and multi-attribute queries. Range queries are implemented using a uniform locality preserving hash function to map items in the Chord key space. Multi-attribute queries are implemented in two ways: an iterative approach and a single attribute dominated routing. In [4] the authors extend the CAN protocol using the Hilbert space-filling curve and load balancing mechanisms. In [8] 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. Routing is performed using a list of neighbors, that are the nodes responsible of adjacent zones plus some other nodes (“skip points”) determined either randomly or using a space-filling curve mapping of the nodes. Another form of distributed indexes are the so-called Routing Index (RI) [6]. 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 variable data. This limitation is addressed in this paper: we present a solution to the problem of dynamic data location with 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.

In this paper we extend previous results [9] by performing 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 a simple analytical expression of the precision of our algorithm as a function of query selectivity and index size.

3. System Overview

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. For example, 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. We suppose that data items are dynamic, in the sense that 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.

More specifically, we consider a P2P system where each peer implements the following operations:

insert$(D, \{A_1 : V_1, \ldots A_r, V_r\})$ Insert a new data item D on the local repository; the data item will be associated with attributes $A_1, \ldots A_r$ with values $V_1, \ldots V_r$ respectively.

update$(D, A : V_{\textbf{new}})$ Change the value of attribute A for data item D on the local repository; the new value will be V_{new}.

lookup(Q) Search for any data item matching query Q over the whole P2P system (including the current node).

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.

3.1. Notation and Data Structures

In the following we consider a P2P system with a set $P = \{P_1, P_2, \ldots P_N\}$ of N peers. We denote with *Data* (P_i) the local repository on peer P_i. Each data item is labelled with a set of attribute-value pairs. We suppose that there is a limited number of different attribute names. We denote with $\{A_1 : T_1, \ldots A_M : T_M\}$ the set of all the M possible attribute names with their corresponding types. Data types T_i can be any arbitrary data types, subject to the constraint that there must be a total ordering defined over T_i. Each data item can be labelled with any nonempty subset of attributes of $\{A_1, \ldots A_M\}$. For each data item D, we denote with *AttList* (D) the set of all attribute names defined for D. Moreover, for each attribute $A \in$ *AttList* (D), $D[A]$ denotes the value of attribute A for data item D.

The system provides 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):

$$\begin{array}{rcl} Q & := & Q \textbf{ and } Q \mid Q \textbf{ or } Q \mid v_1 \leq A \leq v_2 \\ A & := & A_1 \mid \ldots \mid A_M \end{array}$$

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

Observe that the user is not required to specify conditions on all attributes of a data item. The result of a **lookup**(Q) operation is to return the set of all the locations (i.e., the set of peer IDs) of all data items D matching Q.

A trivial way of locating resources would be that of flooding 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 an overlay network over the set of peers, and associate routing information with individual links. In particular, we maintain an undirected spanning tree over the set P of peers. The overlay network is used only to route query and update messages, while individual peers can communicate directly with each other (e.g., using TCP connections over the underlying physical network). We denote with $\mathcal{T} = (P, E)$ a spanning tree over P, where $E \subseteq \{\{P_i, P_j\} \mid 1 \leq i, j \leq N\}$ is the set of links connecting pairs of nodes. In a system with N nodes, there are $N - 1$ links on the spanning tree. For each $P_i \in P$, we denote with $\mathit{Nb}\,(P_i)$ the set of neighbors of P_i, that is, the set of all peers directly connected to P_i on the overlay network.

Let $\mathcal{T}(P_i \to P_j)$ be the subtree of $\mathcal{T}$ which contains P_j and does not contain $P_i \in \mathit{Nb}\,(P_j)$. That is, $\mathcal{T}(P_i \to P_j)$ is the subtree containing node P_j which has been obtained after removing the link $\{P_i, P_j\}$ from $\mathcal{T}$ (see Fig. 1).

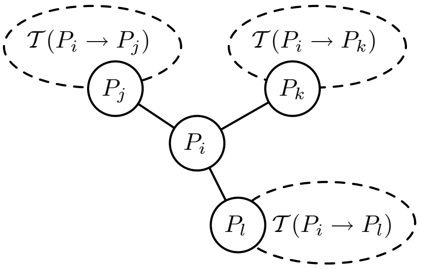

$\mathcal{T}(P_i \to P_j)$
$\mathcal{T}(P_i \to P_k)$
P_j
P_k
P_i
P_l
$\mathcal{T}(P_i \to P_l)$

Figure 1.

Each peer P_i maintains a summary of all the information which can be found by following its outgoing links, in the following way. If the domain of attribute A is the interval $[a, b]$, we select $k + 1$ division points $a = a_0 < a_1 < \ldots < a_k = b$ such that $[a, b]$ is partitioned into k disjoint intervals $[a_i, a_{i+1})$, $i = 0, 1, \ldots k - 1$. For each attribute type we may define a specific partitioning of its domain. Given an attribute A, for each data item D for which A is defined, we encode the value $D[A]$ with a k bit binary vector $\mathit{BitIdx}\,(D[A]) = (b_0, b_1, \ldots b_{k-1})$, such that:

$$b_i = \begin{cases} 1 & \text{if } D[A] \in [a_i, a_{i+1}) \\ 0 & \text{otherwise} \end{cases} \qquad i = 0, 1, \ldots k - 1$$

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 type $T_1, \ldots T_M$.

Let us consider a generic peer P_i. For each neighbor $P_j \in \mathit{Nb}\,(P_i)$, P_i keeps information on the data items which can be found by following the link $\{P_i, P_j\}$ on the overlay network. For each attribute A of each data item D in $\mathcal{T}(P_i \to P_j)$, P_i knows the following quantity:

$$\mathit{LinkBitIdx}\,(P_i \to P_j, A) \equiv \bigvee_{D \in \mathit{Data}(\mathcal{T}(P_i \to P_j))} \mathit{BitIdx}\,(D[A]) \tag{1}$$

which is the bitwise union of all the bitmap indexes $\mathit{BitIdx}\,(D[A])$ associated with every data item in $\mathcal{T}(P_i \to P_j)$. Note that $\mathit{LinkBitIdx}\,(P \to P', A)$ is a binary string of the same size of $\mathit{BitIdx}\,(D[A])$, with possibly more than one bit set to 1.

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 indexes for the local repository; binary strings in the small withe boxes represent the values of $\mathit{LinkBitIdx}\,(P \to P', A_1)$. For example, node E has a local data item D with $\mathit{BitIdx}\,(D[A_1]) = 0010$; the value of $\mathit{LinkBitIdx}\,(E \to F, A_1)$ is 1000.

Observe that $\mathit{LinkBitIdx}\,(A \to C, A_1)$, according to Eq. 1, is the logical "or" of the bit vector representation of values of $D[A_1]$ on nodes B, C, D, E, F.

We summarize in Table 1 the notation used in this paper. It will be used in the following sections to describe the algorithms to process queries and updates on the P2P system.

3.2. Handling Queries

We now illustrate how queries are processed. We assume that queries originate from any node P in the system. As in Gnutella [1], queries are 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 indexes associated with individual peers connections.

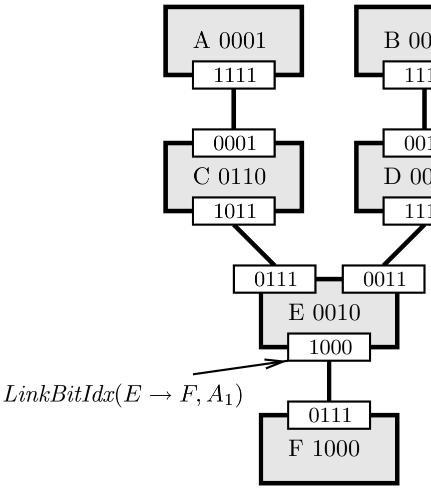


Figure 2. Example of P2P network with bit vector indexes.

Data (P_i) The set of data items present in peer P_i

Nb (P_i) The set of neighbors of node P_i on the overlay network

AttList (D) The set of attributes defined for data item D

$D[A]$ The value of attribute A for data item D

BitIdx $(D[A])$ Bit-vector representation of $D[A]$

$\mathcal{T}(P_i \to P_j)$ The subtree of $\mathcal{T} = (P, E - \{P_i, P_j\})$ which does not contain P_i

LinkBitIdx $(P_i \to P_j, A)$ The bit vector index for attribute A associated with the link from P_i to P_j

Table 1. Notation used in this paper

Recall from the previous section that node P knows the bit vector *LinkBitIdx* $(P \to P', A)$, for each $P' \in$ *Nb* (P), where *LinkBitIdx* $(\cdot)$ is defined according to Eq. 1. Suppose that node P receives query $Q := v_1 \leq A \leq v_2$ from one of its neighbors P_{in}. The query is propagated along the connection from P to $P_{out} \in$ *Nb* $(P) - P_{in}$ if a match is likely to be present in $\mathcal{T}(P \to P_{out})$. A necessary condition for the existence of a match is that the logical "and" between *LinkBitIdx* $(P \to P_{out}, A)$ and the bit vector representation of the interval $[v_1, v_2]$, 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.

We show in Algorithm 2 the function *Match* $(Q, P_i \to P_j)$, which is used to test for a potential match of query Q on the subtree $\mathcal{T}(P_i \to P_j)$. Query Q is decomposed according to the grammar described in the previous section. For each instance of the terminal production $Q := v_1 \leq A \leq v_2$, the function compares the bit vector representation of interval $[v_1, v_2]$ with *LinkBitIdx* $(P_i \to P_j, A)$. If the intersection is zero, then no match exists on $\mathcal{T}(P_i, P_j)$. If the intersection is nonzero, then there *may* be a match on $\mathcal{T}(P_i, P_j)$.

3.3. Handling Updates and Insertions

We now describe how updates can be processed. Let us assume that the value for $D[A]$ for a data item $D \in$ *Data* (P) changes from v_{old} to v_{new}. The peer P executes procedure **initiate_update** shown in Algorithm 3 to generate an update

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 $Match(P \to P_{out}, Q)$ 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 $R := R \cup \{P\}$
  Report $R$ to $P_{in}$
```

Algorithm 2 *Match* $(Q, P_i \to P_j)$

```
if $Q := Q_1$ and $Q_2$ then
  Return $Match(Q_1, P_i \to P_j) \wedge Match(Q_2, P_i \to P_j)$
else if $Q := Q_1$ or $Q_2$ then
  Return $Match(Q_1, P_i \to P_j) \vee Match(Q_2, P_i \to P_j)$
else if $Q := v_1 \leq A \leq v_2$ then
  Let $a_0, a_1, \ldots a_k$ be the subdivision points for $A$
  for all $i = 0 \ldots k-1$ do
    Let $b_i = \begin{cases} 1 & \text{if } [a_i, a_{i+1}) \cap [v_1, v_2] \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$
  Let $B := (b_0, b_1, \ldots b_{k-1})$
  Return $(LinkBitIdx(P_i \to P_j, A) \wedge B \neq 0)$
```

messages. First, the new value v_{new} is converted into the corresponding bit vector representation. If *BitIdx* (v_{new}) is equal to *BitIdx* (v_{old}), then the update is not propagated to neighbors; if the bit vector representations are different, then the update message is propagated in order to preserve the property defined by Eq. 1. 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 to the link $P_{out} \to P$ can be computed by P according to the following relation:

$$LinkBitIdx(P_{out} \to P, A) = BitIdx(D[A]) \vee \left(\bigvee_{P' \in Nb(P) - P_{out}} LinkBitIdx(P \to P', A) \right) \tag{2}$$

where *BitIdx* $(D[A])$ is the bit vector representation of $D[A]$ for data item D on node P.

Algorithm 3 describes the actions executed by peer P when it notices a change in the local data store. If the bit vector representation of the new and old values are the same, nothing is done. Otherwise, an update vector index is computed and sent to each of its neighbors.

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

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

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.

Algorithm 4 process_update$()$ executed by peer P

```
loop
  Wait for bit vector $B$ for $A$ from $P_{in}$
  if $B \neq \mathit{LinkBitIdx}\,(P \to P_{in}, A)$ then
    Let $\mathit{LinkBitIdx}\,(P \to P_{in}, A) := B$
    if $\mathit{BitIdx}\,(D[A]) \vee B \neq B$ then
      for all $P_{out} \in \mathit{Nb}\,(P) - P_{in}$ do
        Let $B' := (0, 0, \ldots 0)$
        for all $P' \in \mathit{Nb}\,(P) - P_{out}$ do
          Let $B' := B' \vee \mathit{LinkBitIdx}\,(P \to P', A)$
        Send $B'$ to $P_{out}$
```

Insertions 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 peer P, then for each $A \in \mathit{AttList}\,(D)$, P executes the procedure **initiate_update**$(A, D[A])$ (outgoing messages can be batched together for efficiency).

3.4. Nodes Joining and Leaving the system

In order to limit the number of hops of the messages processed in the system, it is necessary to build an appropriate overlay network on the top of the set of peers $\mathcal{P} = \{P_1, P_2, \ldots P_N\}$. The algorithms presented above rely on a tree-structured overlay network $\mathcal{T}$, which is a spanning tree over the set of nodes $\mathcal{P}$. Algorithms 1–4 are of course totally independent from the way the overlay network topology is maintained: every algorithm for maintaining a distributed spanning tree over the set of peers can be applied when nodes join or leave the network.

However, the performance of the system depends on the topological characteristics of the overlay network, as we will see in more details in the next section. In order to avoid degenerate cases, the overlay network should have 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 [10] to maintain the spanning tree $\mathcal{T}$ with bounded degree and logarithmic diameter.

4. Simulation results

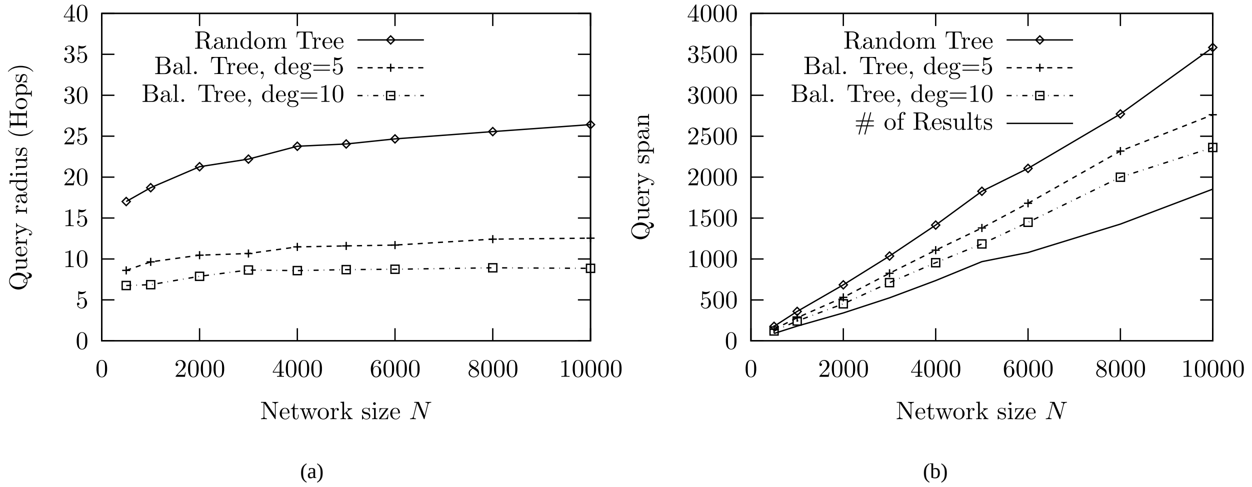
We conducted several simulation experiments in order to evaluate the performances of the proposed P2P system. In this section we discuss the results of the simulation study.

The experimental settings are as follow. We consider a N node P2P system with single attribute data items. Attribute values are uniformly randomly distributed in the $[0, 1]$ interval. Each peer has one data item with probability p (usually set to 0.5), and has no data items with probability $1-p$; thus, the expected number of data items in the network is Np. We consider the following overlay tree network topologies: random, balanced with degree 5, and balanced with degree 10. Simulation results are computed as confidence interval with 90% confidence level. Each measurement was repeated several times in order to get confidence intervals having width of less than 5% the central value (in the figures we only show the central value).

We first analyze the maximum number of routing hops (*query radius*) needed to locate a data item as a function of network size. Fig. 3(a) shows the results for three different overlay network topologies. In Fig. 3(b) 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(b) 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 define the precision of the query routing strategy; the 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*).



(a) (b)

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

We consider a network of $N = 1000$ nodes, and performed 100 range queries given a *selectivity parameter* s. The queries have the form $(v \leq A_1)$ **and** $(A_1 \leq v + s)$ for v uniformly chosen in $[0, 1 - s]$. In Fig. 4 we show the precision of our algorithm as a function of query selectivity for single attribute range queries.

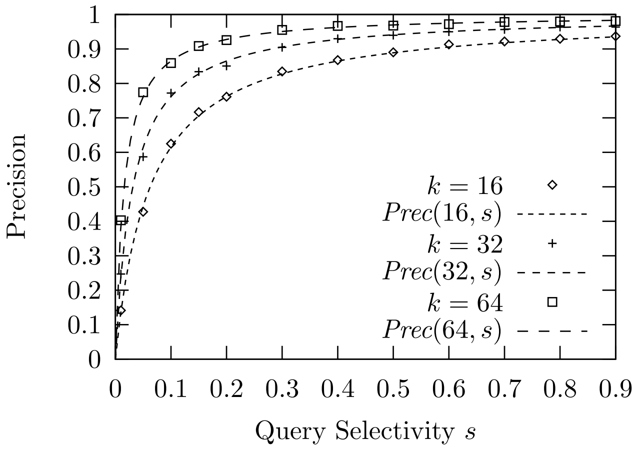


Figure 4. Precision as a function of selectivity ($N = 1000$, random tree, higher is better). Function *Prec*(k, s) is defined in 3.

From the figure we see that the precision is higher as the number k of bits in the vector indices increases. Also, the precision increases for large values of the selectivity parameter s. Remember that in our simulation experiments we have Np data items with a single attribute over the N-node network. Attribute values are uniformly distributed in $[0, 1]$, and we assume that the $[0, 1]$ interval is partitioned into k equally sized bins. The expected number of data items matching a range query with selectivity s is Nps. For $0 < s \leq 1 - 1/k$, the expected number of false positives (i.e., data items whose bit vector indexes match the query, but their exact attribute values do not) is Np/k. The precision in this case is $Nps/(Nps + Np/k) = ks/(ks + 1)$. If $s > 1 - 1/k$, the expected number of false positives is $Np(1 - s)$, and the precision is equal to s. Thus, we can given an analytical expression of the precision as:

$$\mathit{Prec}(k,s) = \begin{cases} \dfrac{ks}{ks+1} & \text{if } 0 < s \leq 1 - 1/k \\ s & \text{if } 1 - 1/k < s \leq 1 \end{cases} \tag{3}$$

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

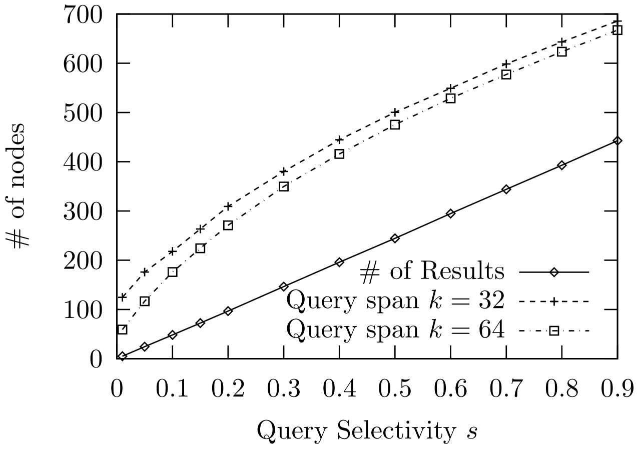


Figure 5. Query span as a function of selectivity (random tree, $k = 16$, lower is better)

In Fig. 5 we plot the query span as a function of the selectivity. Remember that the query span is defined as the number of peers who receive a query message (even if they don't have any matching data item).

We finally analyze the behavior of the update mechanism. In Fig. 6(a) we plot the mean number of hops traversed by an update message (update radius) as a function of the network size; in Fig. 6(b) we plot the number of nodes reached by an update message (update span) as a function of the network size. From the figures we observe that both the update radius and update span are independent from 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 laying in between.

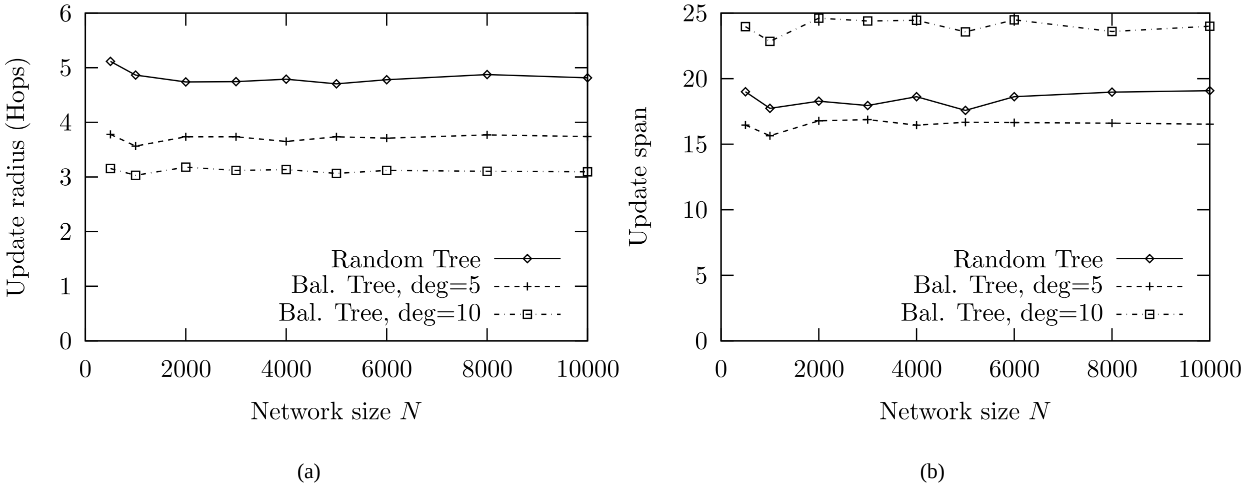


Figure 6. (a) Update radius and (b) update span as a function of network size ($k = 16, p = 0.5$, lower is better).

Fig. 7 plots the update span as a function of the data density p. As expected, the update span decreases for larger values of p: high data density implies that the vector indexes associated with the links have a higher density of bits set to 1, thus updates are more likely not to propagate.

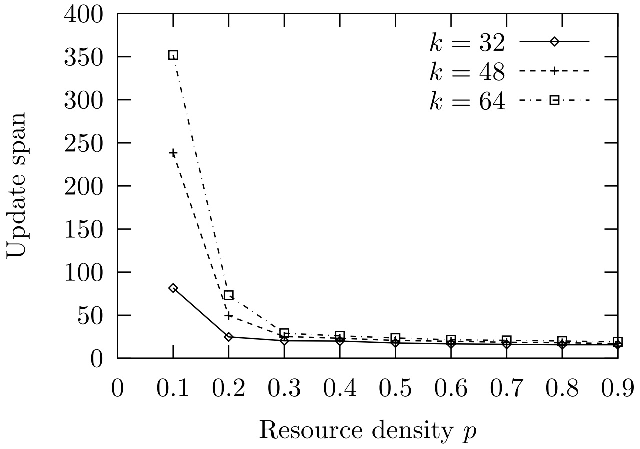


Figure 7. Number of nodes updated as a function of p ($N = 1000$, random topology, lower is better)

5. Conclusions

In this paper we described a P2P system which supports range queries over dynamic content. Data location is implemented using a distributed data structure based on bit vectors. Routing informations are used to drive queries away from regions of the network where matches cannot be found. The routing informations can be efficiently updated when data is modified. Simulation results show that the proposed update and query processing algorithms have good scalability properties.

We are currently extending the proposed algorithms using histogram indexes instead of bit vectors, following an approach similar to [11]. This allows us to store also informations on the approximate number of matches, which can be very useful for certain applications.

References

[1] Gnutella protocol development. http://rfc-gnutella.sourceforge.net/.

[2] A. Andrzejak and Z. Xu. Scalable, efficient range queries for grid information services. In *P2P '02: Proc. of the Second Int. Conf. on Peer-to-Peer Computing*, page 33, Linköping, Sweden, 2002. IEEE Computer Society.

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

[4] M. Cai, M. Frank, J. Chen, and P. Szekely. Maan: A multi-attribute addressable network for grid information services. In *GRID '03: Proc. of the 4th Int. Workshop on Grid Computing*, page 184, Washington, DC, USA, 2003. IEEE Computer Society.

[5] A. Crainiceanu, P. Linga, J. Gehrke, and 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*, pages 390–391, New York, NY, USA, 2004. ACM Press.

[6] A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *Proc. of the 22nd Int. Conf. on Distributed Computing Systems (ICDCS'02)*, pages 23–33, Washington, DC, USA, 2002. IEEE Computer Society.

[7] I. Foster and A. Iamnitchi. On death, taxes, and the convergence of peer-to-peer and grid computing. In *2nd Int. Workshop on Peer-to-Peer Systems (IPTPS'03)*, Berkeley, CA, Feb. 2003.

[8] P. Ganesan, B. Yang, and 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*, pages 19–24, New York, NY, USA, 2004. ACM Press.

[9] M. Marzolla, M. Mordacchini, and S. Orlando. Resource discovery in a dynamic grid environment. Tech. Report CS-2005-3, Dipartimento di Informatica, Università Ca' Foscari di Venezia, Italy, Mar. 2005. To appear in Proc. GLOBE'05.

[10] G. Pandurangan, P. Raghavan, and E. Upfal. Building low-diameter peer-to-peer networks. *IEEE J. on Selected Areas of Communications*, 21(6):995–1002, Aug. 2003.

[11] Y. Petrakis, G. Koloniari, and E. Pitoura. On using histograms as routing indexes in peer-to-peer systems. In W. S. Ng, B. C. Ooi, A. M. Ouksel, and C. Sartori, editors, *DBISP2P*, volume 3367 of *LNCS*, pages 16–30, Toronto, Canada, Aug. 29–30 2004. Springer.

[12] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proc. SIGCOMM '01*, pages 161–172, New York, NY, USA, 2001. ACM Press.

[13] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proc. of the IFIP/ACM Int. Conf. on Distributed Systems Platforms*, pages 329–350, London, UK, 2001. Springer-Verlag.

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

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

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