Efficient Query Processing in P2P Networks of Taxonomy-based Sources

Jean-Marc Saglio[1], Michel Scholl[2], Tuan-Anh Ta[1]

[1] Département Informatique & Réseaux, ENST, 46 rue Barrault, 75013 Paris
{saglio, ta}@enst.fr
[2] Cedric/CNAM, 292 rue St Martin, 75141Paris Cedex03
scholl@cnam.fr

Abstract. In this paper we focus on RDF knowledge bases distributed in a peer-to-peer (P2P) network. We rely on a taxonomy-based model for indexing documents of a given peer. Metadata is modeled as an isA hierarchy of terms. Between hierarchies of two "neighbor" peers, related terms are connected through an isA semantics as well. Our contribution is threefold: we show i) that querying such RDF knowledge bases can be done with the help of RQL; ii) how to optimize the fixpoint evaluation of network queries by eliminating redundant calls to neighbor peers and show that by using labeling schemes network queries can efficiently be expressed in SQL; iii) we report on a preliminary evaluation of this optimization performed with a simulation of the P2P network by an Oracle database implementation on real metadata from the ODP hierarchies.

1 Introduction

Building community web portals is one among many web applications that have used the Resource Description Framework (RDF) to define metadata for indexing documents. A central repository is commonly used to manage RDF metadata and support querying and browsing documents in the portal as well [4]. However, there exist applications such as the Web of people [16, 18] that require a distribution of metadata among several sites. In this application each person relates to an RDF knowledge base in order to index the documents he/she has collected and wants to be shared with other persons of the network. It has been advocated that peer-to-peer (P2P) networks [1, 2, 5, 6, 10, 15, 17] are scalable and appropriate solutions for such distributed applications.

In this paper we focus on RDF knowledge bases distributed in a P2P network. We rely on a simple model [19, 20] for indexing documents of a given peer. Metadata is modeled as a hierarchy of terms with an *isA* semantics. Related terms of hierarchies of two "neighbor" peers are connected through an *isA* semantics as well. We follow the model of [20] which shows that network queries, i.e. queries that recursively query neighbor peers, can be expressed by a simple fixpoint. Our contribution is threefold: we show i) that querying such RDF knowledge bases can be done with the help of query languages such as RQL [12]; ii) how to optimize the fixpoint evaluation of such recursive network queries by eliminating redundant calls to neighbor peers and show that by using some labeling schemes for coding the terms of a hierarchy, such network queries can efficiently be expressed in SQL; iii) we report on a preliminary evaluation of this optimization performed with a simulation of the P2P network by an Oracle database implementation on real metadata from the ODP hierarchies (dmoz.org). The remaining of the paper is organized as follows: in section 2 we choose as an example a knowledge base, drawn from categories of the ODP portal (dmoz.org) and expressed in RDF. Queries against such a knowledge base are expressed in RQL. Section 3 specifies the model and the evaluation of network queries. The query evaluation can be optimized by the elimination of redundant evaluations. Section 4 gives a translation in SQL for the RQL queries of section 2, choosing a Dewey labeling scheme [8] for coding the hierarchies in order to efficiently traverse hierarchies of terms. Section 5 describes our experiment. In section 6 we survey some related work.

2 An example of P2P architecture of knowledge bases

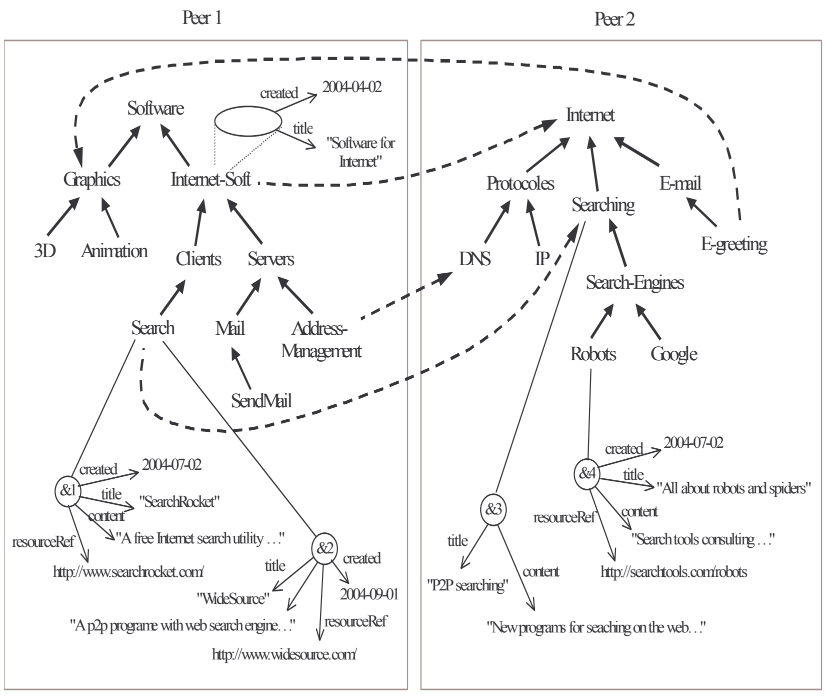


Fig. 1: A distributed hierarchy of computer topics

Fig. 1 displays a forest of terms distributed among two peers. Each peer manages an autonomous local knowledge base on a given subject (e.g. Computer software, Internet) independently of the other peers. The knowledge base is a hierarchy of terms, each term representing a topic of interest. The interpretation of term *t* is the set of user documents (called *posts* in the web of people [18]) indexed by *t*. Fig. 1 features a post *&1* on Search software with four attributes: a date of creation, a title, a resource reference and a full text description. Terms are connected through an *isA* link. As an example, Clients and Servers are sub categories of Internet-Soft. Terms come with attributes as well (in this example terms have two attributes: date of creation and title). Last, each peer can take advantage of the existence of other peers on related topics. A simple protocol allows two peers *p* and *p'* to agree that one term *t* of peer *p* is in relation *isA* with a term *t'* of peer *p'*. As an example, Peer 2 declares that term Internet-Soft in Peer 1 is in relation *isA* with local term Internet (dotted arrow in Fig. 1).

Fig. 2 displays an RDF schema for the example of Fig. 1. The multi-instanciation of a post (dashed arrows) is noteworthy. Terms are implemented as classes. The *isA* semantics is expressed by the subclass relation between classes whether the superclass and the subclass belong to the same peer or not. A post instance is an instance (link *rdf:type*) of the class *Post* and of a topic class. Two peers are modeled as two instances of class *Peer*. A post belongs to a peer according to the peer of the topic class (attribute *hasPeer*) it is an instance of. Note that topic classes are instances of a metaclass *Topic* with two attributes *title* and *created*.

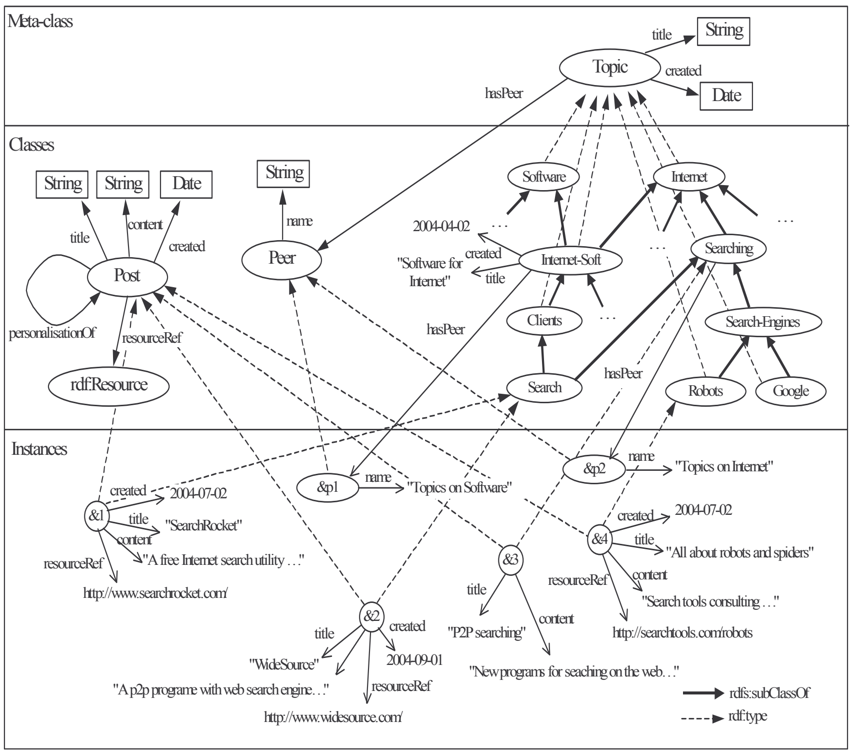


Fig. 2: An RDF schema for the distributed knowledge base

We now give several queries on this knowledge base. Among the various proposals for querying RDF data (see [14]) we chose RQL [12], because it brings the declarative power of database query languages to a kernel of the RDF language and because its syntax allows for a simple formulation of transitive closures of the relation *subclassOf*. Although tailored for the Web of people application the following queries illustrate the power of RQL to express queries in a number of other applications relying on metadata indexing.

Q1. Posts under Searching? (we look for the posts instances of topic Searching as well as instances of its subtopics in the whole network)
select x **from** Searching{x}

Searching{x} is a path expression starting from class Searching and ending at an instance *x*. The interpretation of such a path is that *x* is an instance of Searching or of any of its subclasses.

Q2. Local posts under Searching? (we look for the posts instances of Searching and of all the terms below Searching in the hierarchy of peer &p2)
select x **from** Searching{x}, ^$t{x}, Topic{t}.hasPeer{p} **where** $t=t **and** p=&p2

The from clause includes three path expressions where (i) in the second path expression ^$t{x} denotes there exists a class *$t*, *x* is a direct instance of ("^"stands for direct); (ii) the third path expresses *t* being a topic (an instance of metaclass *Topic*) and having *p* as its peer.

Q3. Posts under Searching created since 2004?
select x **from** Searching{x}.created{d} **where** d>=2004-01-01

The path {x}.created{d} gives the creation date of post *x*.

Q4. Local sons (one hop descendent) of topic Searching? (we look only for topics in peer &p2)
select t **from** Topic{t}.hasPeer{p}
where p=&p2 **and** t **in subclassOf**(Searching)

3 Network query model and evaluation optimization

For network queries that require a recursive traversal of several peers, we assume a decentralized evaluation process on hierarchies of terms distributed over peers. The model follows that of [19, 20]. A network query sent to a peer *p* is rewritten into the union of its local answer with the result of network queries sent to other peers for evaluation. When evaluating a network query one or several redundant paths may be followed. In other words a term t' in peer p' may be found as a descendent of a term *t* in peer *p* several times, by following several paths. We present an optimization of network queries evaluation avoiding redundant computations.

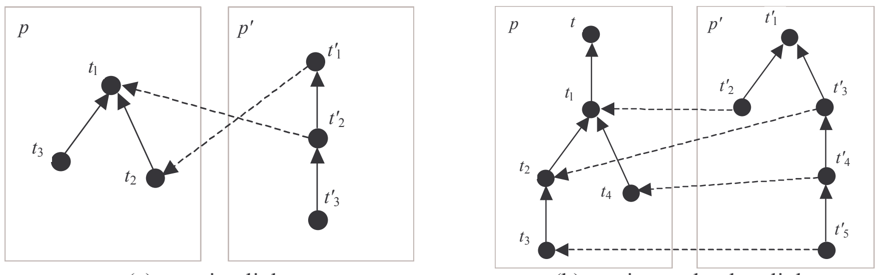
Definition 1 [19,20,21]: Each peer p owns a term hierarchy $(T_p, \prec)$ where T_p is a finite, non empty set of terms; $\prec$ is a partial order modeling *isA* relations between terms. Let I be the set of posts declared as instances of a term t or of any of its successors t' $(t \prec t')$ in the hierarchy; so if a post $po \in I(t')$ then $po \in I(t)$. In the following, we call $I(t)$, for short, the interpretation of t.

Definition 2: Let $t \in T_p$ and $t' \in T_{p'}$ be two terms. With the authorization of peer p', peer p can declare t' as "linked" to t. Let $Link_{p,p'}$ denote the finite set of such links $[t, t']$ between peer p and peer p'.

As in [19, 20], we model a link $[t, t']$ as an *isA* relation: $t \prec t'$, with the usual semantics that is if a post published in peer p' is in the interpretation of t' then it is in the interpretation of t.

Proposition 1 (redundant crossing links): Given two peers p and p', if there exists $[t_1, t'_2]$, $[t_2, t'_1] \in Link_{p,p'}$, and $t'_1, t'_2 \in T_{p'}$, $t'_1 \preceq t'_2$ on the one hand, and $t_1, t_2 \in T_p$, $t_1 \preceq t_2$ on the other hand, then link $[t_1, t'_2]$ is redundant.

As illustrated in Fig. 3(a), t'_2 is in the interpretation of t_1 by transitivity, since t'_2 is a successor of t'_1 which is linked to t_2, a successor of t_1. Crossing links can definitely be deleted (see section 4). Note that such links can as well be detected upon insertion.



(a) crossing links (b) runtime redundant links

Fig. 3: Example of redundant links

Definition 3: Let $L(p, t, p') = \{t' \mid t' \in T_{p'} \wedge \exists\, t_1 \in T_p\ (t \preceq t_1 \wedge [t_1, t'] \in Link_{p,p'})\}$ denote the set of terms in a peer p' that can be reached from a term t or any of its successors in peer p by following a link. We call *neighbor peer* of peer p a peer p' for which there exists at least one link between a term t' in p' and a term t in p. Let $N(p)$ denote the set of neighbor peers of p. Let $I_p(t)$ denote the interpretation of t without taking into account the links with neighbors. $I_p(t)$ is the set of posts declared in p as instances of t or of any of its successors. Let $I(t)$ denote the interpretation of t, including the posts declared in other peers under successors of t through terms linked to terms of p. Then we have:

$$I(t) = I_p(t) \cup \bigcup_{t' \in L(p,t,p_i),\, p_i \in N(p)} I(t') \qquad (1)$$

Definition 4: Let $Q(p, t)$ denote the query issued at peer p asking for all posts under a term t. Then the answer to t is precisely $I(t)$. $Q(p, t)$ is called a network query since it is translated into a local query $Q_p(t)$ which computes $I_p(t)$ and a set of queries sent to neighbors. From equation (1), we have:

$$Q(p,t) = Q_p(t) \cup \bigcup_{t' \in L(p,t,p_i),\, p_i \in N(p)} Q(p_i, t') \tag{2}$$

As an example, consider the network query $Q(p, t)$ in Fig. 3(b). It implies the evaluation of $Q(p', t'_2)$, $Q(p', t'_3)$, $Q(p', t'_4)$ and $Q(p', t'_5)$ which correspond to the four links between successors of t in p and terms of p'. Observe that for answering this query only links $[t_1, t'_2]$ and $[t_2, t'_3]$ are of interest. Links $[t_4, t'_4]$ and $[t_3, t'_5]$ are not necessary (and therefore redundant) for this query; indeed, t'_4 and t'_5 and their successors are anyway reached by following the link $[t_2, t'_3]$. Similarly, for query $Q(p, t_2)$, link $[t_3, t'_5]$ is redundant.

Proposition 2: Let $L_{\min}(p, t, p') = \{t' \mid t' \in L(p, t, p') \wedge \overline{\exists}\, t'_1 \in L(p, t, p') : t'_1 \prec t' \}$, we have:

$$Q(p,t) = Q_p(t) \cup \bigcup_{t' \in L_{\min}(p,t,p_i),\, p_i \in N(p)} Q(p_i, t') \tag{3}$$

The proof is straightforward from the aforementioned observation on Fig. 3(b). Links in $L(p,t,p')$ and not in $L_{\min}(p,t,p')$ are redundant. $L_{\min}(p,t,p')$ must be calculated at run time, i.e. upon answering the query. Algorithm A computes $L_{\min}(p, t, p')$ (for short $L_{\min}$) from $L(p, t, p')$ (for short L). In section 4 we give an efficient algorithm for computing $L_{\min}$.

Algorithm A – naive $L_{\min}$ computation

```
Input: $L$
Output: $L_{\min}$
(1)    $L_{\min} := \varnothing$
(2)    for each $t_i$ in $L$
(3)        if not exists $t_j$ in $L$ such that $t_j \prec t_i$ then $L_{\min} := L_{\min} \cup \{t_i\}$
(4)    return $L_{\min}$
```

Once $L_{\min}$ has been calculated there exist a number of strategies [20] to evaluate a network query (equation (3)). The knowledge at a given peer of the data existing in another peer is central to the choice of a strategy for evaluating network queries. In this paper we assume a totally distributed peer to peer architecture in which each peer acts independently of the others and has no knowledge of the term hierarchies of the other peers except for the *Link*(t, t') table. However for inserting a link between peer p and peer p', peer p' must provide to peer p in a read mode its term hierarchy $T_{p'}$. We further assume that no cycle exists when following a link from peer p to peer p'. In other words, there exists no *isA* path starting from t in p coming back to t' in p such that t' is an ancestor of t. We assume cycles are detected upon a link insertion.

The two following strategies differ in the way answers to a network query are forwarded to the user.

Strategy 1. Upon receiving a query $Q(p, t)$, peer p sends queries $Q(p', t')$ to its neighbor peers. Peer p evaluates its local query $Q_p(t)$ and waits for all answers of neighbor peers. Upon receiving an answer to query $Q(p', t')$ peer p makes the union with the current set of solutions. Once all peers have answered, peer p either sends back its answer to the user or to the peer that issued the query.

Algorithm 1 - *Q*(*p*, *t*)

```
Input: a peer $p$, a term $t$,
Output: $Q$ a set of posts
(1)   $Q := Q_p(t)$
(2)   for each $p' \in N(p)$
(3)       for each $t' \in L_{\min}(p, t, p')$
(4)           $Q := Q \cup Q(p', t')$
(5)   return $Q$
```

$N(p)$ denotes the set of neighbour peers of p. For evaluating the set of local posts $Q_p(t)$ all successors of t in p have to be scanned and for each of them the set of instances (posts) have to be collected. We give in the following section an efficient algorithm for this evaluation.

Strategy 2. Peers send queries to their neighbors, but do not wait for the answers. The latter are directly sent to peer p_0, which the initial query was issued to. The rationale behind this strategy is that it might take less time to send the answer back directly to p_0 instead of transferring the answer through a long chain of intermediary peers. The evaluation and comparison of these two strategies imply a network cost model/evaluation and is beyond the scope of this paper. Strategy 2 needs two procedures: (i) $Q(p, t, p_0)$, a query for the posts under term t sent to peer p and whose answer should be sent back to peer p_0; (ii) *Send*(R, p_0) which sends the result set of posts in R to peer p_0.

Algorithm 2

***Q*(*p*, *t*, *p*₀)**

```
Input: $p$ the peer to which the query is issued, $t$ a term, $p_0$ the peer to which the answer is forwarded
Output: updating the set of posts $Q$ if $p=p_0$
(1)   $R := Q_p(t)$
(2)   if $p=p_0$ then $Q := Q \cup R$
(3)   else $Send(R, p_0)$
(4)   for each $p' \in N(p)$
(5)       for each $t' \in L_{\min}(p, t, p')$
(6)           $Q(p', t', p_0)$
```

***Send*(*R*, *p*₀)**

Input: R a set of posts, p_0 peer to which R is sent back

```
Output: updating the set of posts $Q$
(1)   $Q := Q \cup R$
```

The query evaluation may be further optimized as follows. For a given query, a peer can receive several queries for a same term, coming from different peers. For example, peer p'' in Fig. 4 receives two queries on terms t''_1 and t''_2, coming from p and p' respectively. Clearly, the evaluation of t''_2 is redundant if t''_1 is evaluated before. In order to avoid such a redundancy, we keep track of the queries processed so far. The optimization is illustrated in the following for strategy 1 (Algorithm 1bis).

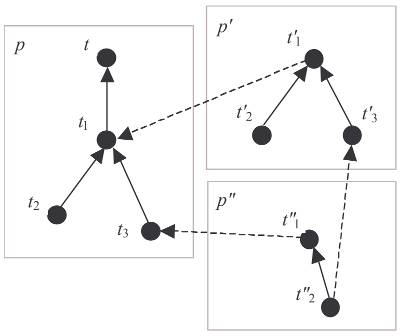


Fig. 4: Other type of redundant links at run time

Algorithm 1bis - *Q(p, t, q)*

```
Input: $p$ the peer to which the query is issued, $t$ a term, $q$ the query id of the initial query
Output: $Q$ a set of posts
(1)   $Q := \emptyset$
(2)   if not exists $[q_i, t_i]$ in $Trace$ such that $q_i=q$ and $t_i \preceq t$ then
(3)       $Trace = Trace \cup \{[q, t]\}$
(4)       $Q := Q_p(t)$
(5)       for each $p' \in N(p)$
(6)           for each $t' \in L_{\min}(p, t, p')$
(7)               $Q := Q \cup Q(p', t')$
(8)   end if
(9)   return $Q$
```

4 Efficient implementation of RQL queries in SQL

In this section we exhibit a relational implementation of the P2P knowledge base and of network queries. One outcome of this section is that the RQL queries of section 2 can be translated into SQL and efficiently answered, provided that each peer be equipped with any SQL engine.

We assume that each peer stores metadata satisfying a relational schema given in Fig. 5(a). Table Topic stores the term hierarchy local to a peer. turi is the address identifying the term, title and created are the topic title and creation date. The Link table stores links between term addresses of the peer and term addresses of other peers. The posts are stored in the Post table. puri identifies the post; title, created respectively denote the title and creation date of the post. In order to simplify the expression of RQL queries into SQL, the following schema is only a reduced translation of the general RDF schema in Fig. 2. Without loss of generality, we assume in the local term hierarchy that a term has at most one parent (represented by attribute parent in Topic). A post is indexed by only one term as well.

4.1 Coding terms with a labeling scheme

Another relational schema for peer metadata is proposed in Fig. 5(b) in which terms (topics) are coded by a labeling scheme. Numerous labeling schemes have been proposed in the literature for coding hierarchies of terms and XML documents (see for example [3, 9, 11, 13]). We show in the following how labeling schemes allow for an efficient implementation of the queries of section 2 and more specifically of network queries. More generally, labeling schemes allow for a significant speed up of ancestor/descendent queries: a labeling function l is such that if t is an ancestor of t' ($t \prec t'$) then $l(t) < l(t')$. Then a time consuming ancestor/successor query can be translated into an efficient SQL interval query. This allows for an efficient implementation of queries on metadata involving transitive closures of the *isA* relation.

Topic (turi, parent, title, created)	Topic (turi, label, title, created)
Link (turi, targetpeer, targetturi)	Link (label, targetpeer, targetlabel)
Post (puri, turi, title, created, resourceRef)	Post (puri, label, title, created, resourceRef)
(a) no labeling scheme	(b) using a labeling scheme

Fig. 5: Relational schema representation of peer metadata

To illustrate the mechanism, we chose a Dewey code [8]. In the schema of Fig. 5(b) attribute parent has been replaced by the label (Dewey code) of the term. Attribute turi (targetturi) has been replaced by the term label (targetlabel) in tables Link and Post. We are now ready to express the RQL queries in SQL with the schema given in Fig. 5(b).

4.2 Query evaluation

We give an execution plan for the network query **Q1** (posts under Searching) successively for strategy 1 and strategy 2. The term Searching is identified by a

Dewey code *l* in the execution plan. Previously, in any peer *p* crossing links can be eliminated (Proposition 1) by the following SQL command:

```
delete from Link r1 where
exists (select * from Link r2 where r1.peer = r2.peer and
        not (r1.label = r2.label and r1.targetlabel = r2.targetlabel) and
        r2.label >= r1.label and r2.label < r1.label || 'xFF' and
        r1.targetlabel>=r2.targetlabel and r1.targetlabel<r2.targetlabel||'xFF')
```

where || denotes string concatenation and label || 'xFF' is the greatest Dewey code > label (see [7] for a discussion on SQL implementation with labeling schemes).

Execution Plan Q1(*p*, *l*), **strategy 1**

```
(1)    $Q$ := select puri, title, created, resourceRef from Post
         where label >= $l$ and label < $l$ || 'xFF'
(2)    $N$ := select distinct targetpeer from Link
         where label >= $l$ and label < $l$ || 'xFF'
(3)    for each $tp$ in $N$
(4)        $L$ := select distinct targetlabel from Link
             where label >= $l$ and label < $l$ || 'xFF' and targetpeer = $tp$
(5)        $L_{\min}$ := $Min(L)$
(6)        for each $tl$ in $L_{\min}$
(7)            $Q := Q \cup \mathbf{Q1}(tp, tl)$
(8)    end for
(9)    return $Q$
```

Min(*L*) is a function calculating $L_{\min}$ from *L* based on one of the following algorithms 1bis and 1ter which take advantage of the order on labeling schemes: *t* is an ancestor of *t'* if *l*(*t*) is a prefix of *l*(*t'*); then in lexicographic order *l*(*t*)<*l*(*t'*).

Algorithm A1

```
Input: $L$
Output: $L_{\min}$
(1)    $L_{\min} := \varnothing$
(2)    $min$ := 'xFF'
(3)    for $i$ := 1..$|L|$ do
(4)        if $\neg\, prefix(min, L[i])$ then
(5)            $min := L[i]$
(6)            $L_{\min} := L_{\min} \cup \{min\}$
(7)        end if
(8)    return $L_{\min}$
```

Algorithm A1 is linear in the cardinality of *L*. *prefix*(*l*, *l'*) is a boolean function that returns true if *l* is a prefix of *l'*. Link is assumed to be initially sorted on the target label and the SQL computation of L is assumed to keep the target labels sorted. Algorithm A2 which does not assume any sorting on labels in L is quadratic in $|L|$.

Algorithm A2

```
Input: $L$
Output: $L_{min}$
 (1)    $L_{min} := \varnothing$
 (2)    for $i := 1..|L|$ do
 (3)        $min := L[i]$
 (4)        for $j := 1..|L|$ do
 (5)            if $prefix(L[j], min)$ then $min := L[j]$
 (6)        if $min = L[i]$ then $L_{min} := L_{min} \cup \{min\}$
 (7)    end for
 (8)    return $L_{min}$
```

Note that L_{min} can be computed by the following SQL query, but the execution of this SQL query is less efficient than the direct implementation of Algorithm A2.

```
Lmin = select targetlabel from Link r1
      where r1.label >= l and r1.label < l || 'xFF' and r.targetpeer = tp and
      not exists (select * from Link r2
          where r2.targetpeer = tp and r2.label >= l and r2.label < l || 'xFF' and
          r2.targetlabel > r1.targetlabel and r2targetlabel < r1.targetlabel||'xFF')
```

Execution plan Q1(p, l, p_0), **strategy 2**

```
 (1)  $R$ := select puri, title, created, resourceRef from Post
             where label >= $l$ and label < $l$ || 'xFF'
 (2)  if $p=p_0$ then $Q := Q \cup R$
 (3)  else $Send(R, p_0)$
 (4)  $N$ := select distinct targetpeer from Link
             where label >= $l$ and label < $l$ || 'xFF'
 (5)  for each $tp$ in $N$
 (6)      $L$ := select distinct targetlabel from Link
              where label >= $l$ and label < $l$ || 'xFF' and targetpeer = $tp$
 (7)      $L_{min} := Min(L)$
 (8)      for each $tl$ in $L_{min}$
 (9)          Q1($tp$, $tl$, $p_0$)
 (10) end for
```

One can design an execution plan for other network queries. As an example, **Q3**'s plan resembles **Q1**'s plan, but the local answer is replaced by the following command

```
Q := select puri, title, created, resourceRef from Post
     where label >= l and label < l || 'xFF' and created >= 2004-01-01
```

Queries evaluated on a single peer are straightforward as well. For example:

Q2. Local posts under Searching ? (*l* is the label of Searching)

```
select puri, title, created, resourceRef from Post
where label >= l and label < l || 'xFF'
```

Q4. Local sons (one hop descendent) topics of Searching?

```
select turi, title, created from Topic
where label >= l and label < l || 'xFF' and length(label) = length(l) + 1
```

5 Implementation and preliminary experimentation

In section 4 we showed that the fixpoint query evaluation could be optimized by eliminating redundant retrievals of the same posts when calling neighbor peers. The objective of this section is to assess this through an experiment, by evaluating the saving in number of (redundant) calls to neighbors as well as the saving in number of (redundant) posts returned to the user, through an experiment on real life metadata, namely 5 ODP (dmoz.org) hierarchies for which there exist links between terms of separate hierarchies. Table 1 gathers statistics on the ODP hierarchies. For instance, the "Software" hierarchy includes 2316 terms. The average depth of a leaf term is 4.79 and the maximal depth is 9. The average (maximal) number of subtopics of a given topic is 4.44 (62). The average number of posts attached to a topic is 17.9 (41472 posts in the whole hierarchy). The number of links to neighbor peers is 124: 5.3 percent of topics in the hierarchy have a link.

Categories	Topics			Posts		Links	
	#	Depth avg. (max)	Fan-out avg. (max)	#	Post/topic avg. (max)	#	Link/topic in avg.
Internet	1087	4.41 (8)	5.10 (65)	18307	16.8 (254)	73	0.067
Software	2316	4.79 (9)	4.44 (62)	41472	17.9 (469)	124	0.053
Hardware	226	3.71 (6)	4.26 (17)	6372	28.2 (255)	39	0.172
Systems	535	4.68 (8)	5.24 (51)	4642	8.7 (92)	38	0.071
Computer Science	190	4.18 (5)	6.78 (51)	1906	10 (121)	8	0.042
Total	4354	4.35 (9)	5.16 (65)	72699	16.7 (469)	282	0.065

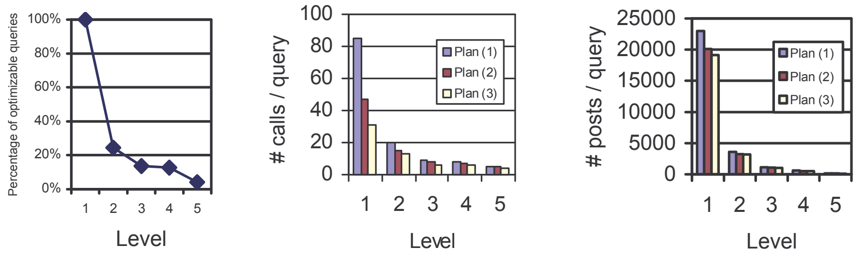
Table 1: Statistics on 5 ODP hierarchies

For this preliminary experiment, we made the following implementation choices:

1) Without loss of generality we choose query plan **Q1** as a query the performance of which is evaluated.
2) Provided that terms are coded according to some scheme (see section 4), interval SQL queries efficiently implement ancestor/descendent queries,

central to metadata querying in our context. It was shown in [7] that such term coding provides a very significant gain.

3) 5 peers (one per ODP hierarchy) were implemented by a single Oracle database whose schema obeys that of figure 6b, in which an attribute with name *peer* is added to each table. The recursive network query **Q1**(*p, l*) is implemented by a PL/SQL query. The same result, the same calls and same number of calls are obtained with this implementation as with that in which each of the 5 peers would have been implemented in a separate database.
4) As far as strategy is concerned, depending on the network traffic and topology, one among the two strategies 1 and 2 might be more efficient. As aforementioned, evaluating the impact of traffic and topology on the strategy choice is left as future work. Without loss of generality, strategy 1 was chosen.
5) We compare three implementations of the query plan. In variant (1), all links are followed. No redundant links are eliminated. In other words each link in the set called *L* is followed ($L_{\min}$ is not calculated: see section 4.2, execution of **Q1**, strategy 1). In the second implementation (variant (2)), the computation of $L_{\min}$ is done according to Algorithm A2 (the Oracle implementation does not allow *L* to be sorted and thus to use algorithm A1). Last, variant (3) implements Algorithm 1bis and thus saves other redundant calls as illustrated in Fig. 4.
6) We run queries with topics at different depths (levels). The queries were issued in all hierarchies and the number of queries per level is equal to the number of terms in the hierarchy at this level. Queries that do not require to follow a link (local queries) are discarded from the evaluation. For each networked query, we counted the number of calls to neighbor peers and the number of posts returned in each of the three variants. A query is *optimizable* whenever at least a redundant link would be eliminated by the evaluation in variant (3). 24 redundant links (among 282) were eliminated prior to running the queries (Proposition 1).



(a) (b) (c)

Fig. 6: Experimental results on the query evaluation

Fig. 6 displays for each level: (a) the percentage of optimizable queries, (b) the average number of calls and (c) the average number of posts returned by an optimizable query in each the three variants. From Fig. 6, the optimization of section 4 is worth only for queries with topics at small depth (level 1): most or all links are followed and then more redundancies occur. This is confirmed by Table 2 that displays for each level,

a) the total number of queries as well as those that required following at least a link to another peer and those for which at least a redundant link exists
b) the number of calls to neighbor peers in each of the three variants
c) the number of posts returned to the user in each of the variants.

Lev.	Net. Queries		Plan (1)		Plan (2)		Plan (3)	
	#	# opt.	#calls	#posts	#calls	#posts	#calls	#posts
1	5	5 (100%)	85	22987	47	20132	31	19121
2	57	14 (24.5%)	20	3620	15	3260	13	3196
3	88	12 (13.6%)	9	1144	8	1061	6	1018
4	55	7 (12.7%)	8	615	7	537	6	501
5	24	1 (4.1%)	5	177	5	177	4	93

Table 2: Experimental results on the query evaluation

6 Related work and conclusion

Managing RDF-based knowledge in P2P networks is a challenge toward combining the Semantic Web and P2P technologies for novel applications. Several projects follow different approaches to develop such P2P networks. Edutella [15] relies on a hybrid architecture where the peers are connected to a super-peer. Super-peers help routing a given query to the peers able to answer. SWAP [5] relies on an RDF metadata model to define knowledge in peers and to relate them in a social metaphor as a basis for semantic query routing. In contrast to the pure P2P approach, in the InfoQuilt system [17] a centralized ontology allows for the mediation of distributed queries over multiple peers. Last but not least, RDFPeers [6] is a scalable and distributed RDF repository for storing, indexing and querying individual RDF triples. It relies on a structured P2P network that stores metadata in self-organized nodes by applying globally known hash functions.

Another approach has been recently exploited to build semantic interoperability among data sources in a bottom-up manner. It relies on an architecture where semantic mappings are locally defined and used in different peers. In other words, data are organized and annotated according to local schemas without any global agreement. Due to the existence of such local mappings between different (peer) schemas, i.e., established in a P2P manner, queries could be translated and distributed over peers connected through a semantic glue. Such a system significantly differs from the aforementioned P2P networks where peers are only topologically connected. Piazza [10] is known as a first project that aims at building semantic interoperability among XML/RDF sources for semantic web applications.In the SomeWhere network [2], simple ontologies modeled in description logic are used to describe data sources. More generally, Chatty Web [1] allows for assessing the quality of semantic mappings in a network (i.e., information-loss incurred through a transformation, semantic similarity achieved through a cycle).

We have provided a P2P network that relies on a simple taxonomy model for managing distributed RDF metadata. As proposed in [19, 20], articulations among data sources in such a network are modeled with an isA semantics. In this paper, we have illustrated the power of RQL for querying both data and metadata in the network. Especially, we have exhibited two fully distributed and scalable strategies of query evaluation. An optimization has been considered for the fixpoint evaluation. Furthermore, we have applied in the network implementation an appropriate coding of topics in order to speed up query evaluation [7]. Since no real life data was available from applications such as the Web of people, we have assessed the performance gain on ODP hierarchies. As a future work we intend to perform a more thorough performance evaluation with larger real life or synthetic data sets, stored in a large number of peers, including a network and traffic cost model in order to compare the two evaluation strategies.

Acknowledgements

Thanks go to Dimitris Plexousakis for helpful discussions on the evaluation strategies and to the referees for their constructive comments

References

1. Aberer, K., Cudré-Mauroux, P., Hauswirth, M.: The chatty web: emergent semantics through gossiping. In Proc. of the 12th International World Wide Web Conference: 197-206, 2003.
2. Adjiman, P., Chatalic, P., Goasdoué, F., Rousset, M.C., Simon, L.: Somewhere in the semantic web. LRI Technical Report, 2004.
3. Agrawal, R., Borgida, A., Jagadish, H.V.: Efficient management of transitive relationships in large data and knowledge bases. In Proc. of the SIGMOD Inter. Conf. On Manag. Of Data, pages 253--262, 1989.
4. Alexaki, S., Christophides, V., Karvounarakis, G., Plexousakis, D.: On Storing Voluminous RDF Descriptions: The case of Web Portal Catalogs. In 4th International Workshop on the Web and Databases, ACM SIGMOD/PODS, CA, 2001.
5. Broekstra, J., Ehrig, M., Haase, P., van Harmelen, F., Kampman, A., Sabou, M., Siebes, R., Staab, S., Stuckenschmidt, H., Tempich, C.: A metadata model for semantics-based peer-to-peer systems. In Workshop on Semantics in Peer-to-Peer and Grid Computing, WWW'03 Budapest, 2003.
6. Cai, M., Frank, M.: RDFPeers: A Scalable Distributed RDF Repository based on a structured Peer-to-Peer network. In Proc. of the 13th International World Wide Web Conference, WWW’04 New York, 2004.
7. Christophides, V., Plexousakis, D., Scholl, M., Tourtounis, S.: On Labeling Schemes for the Semantic Web. In Proc. of the 12th International World Wide Web Conference, WWW'03 Budapest, 2003.
8. Dewey, M.: Dewey Decimal Classification and Relative Index. In Forest Press, 20 edition, 1989.
9. Dietz, P. F.: Maintaining order in a linked list. In Proc. of the Fourteenth Annual ACM Symposium on Theory of Computing (STOC'82), pages 122--127, 1982.

10. Halevy, A.Y., Ives, Z.G., Mork, P., Tatarinov, I.: Piazza: data management infrastructure for semantic web applications. In Proc. of the Twelfth International World Wide Web Conference, Budapest, 2003.
11. Kaplan, H., Milo, T., Shabo, R.: A comparison of labeling schemes for ancestor queries. In Proc. of the Thirteen Annual Symposium on Discrete Algorithms (SODA'02), 2002.
12. Karvounarakis, G., Alexaki, S., Christophides, V., Plexousakis, D., Scholl, M.: RQL: A Declarative Query Language for RDF. In Proc. of the 11th International World Wide Web Conference, WWW'02 Hawaii, 2002.
13. Li, Q., Moon, B.: Indexing and querying XML data for regular path expressions. In Proc. of the 27th Inter. Conf. on Very Large Data Bases (VLDB'02), 2001.
14. Magkanaraki, A., Karvounarakis, G., Christophides, V., Plexousakis, D., Ta. T.: Ontology Storage and Querying. Technical Report No 308, ICS-FORTH, April 2002.
15. Nejdl, W., Wolf, B., Qu, C., Decker, S., Sintek, M., Naeve, A., Nilsson, M., Palmr, M., Risch, T.: Edutella: a p2p networking infrastructure based on rdf. In Proc. of the eleventh international conference on World Wide Web, pages 604-615, ACM Press, 2002.
16. Plu, M., Agosto, L., Bellec, P., Van De Velde, W.: The Web of People: a dual view on the WWW. In Proc. of the 12th International World Wide Web Conference, WWW'03 Budapest, 2003.
17. Sheth, A., Thacker, S., Patel, S.: Complex relationships and knowledge discovery support in the InfoQuilt system. In The VLDB Journal, Volume 12, Issue 1, May 2003.
18. Ta, T.A., Saglio, J.M., Plu, M.: An architecture based on semantic weblogs for exploring the Web of People. In Workshop Application of Semantic Web Technologies to Web Communities, ECAI'04 Valencia, 2004.
19. Tzitzikas, Y., Meghini, C., Spyratos N.: Taxonomy-based Conceptual Modeling for Peer-to-Peer Networks. In Proc. of the 22th International Conference on Conceptual Modeling, ER'2003, Chicago, 2003.
20. Tzitzikas, Y., Meghini, C.: Query evaluation in peer-to-peer networks of taxonomy-based sources. In Proc. of the 10th International Conference on Cooperative Information Systems, CoopIS'03. Sicily, 2003.
21. Tzitzikas, Y., Spyratos, N., Constantopoulos, P.: Mediators over taxonomy-based information sources. In The VLDB Journal, Vol 15, No 1, 2005.