Query Processing in RDF/S-based P2P Database Systems

George Kokkinidis, Lefteris Sidirourgos and Vassilis Christophides

Institute of Computer Science - FORTH
Vassilika Vouton, PO Box 1385, GR 71110, Heraklion, Greece and
Department of Computer Science, University of Crete
GR 71409, Heraklion, Greece
{kokkinid, lsidir, christop}@ics.forth.gr

1 Introduction

Peer-to-peer (P2P) computing is currently attracting enormous attention, spurred by the popularity of file sharing systems such as Napster [31], Gnutella [15], Freenet [9], Morpheus [30] and Kazaa [25]. In P2P systems a very large number of autonomous computing nodes (the peers) pool together their resources and rely on each other for data and services. P2P computing introduces an interesting paradigm of decentralization going hand in hand with an increasing self-organization of highly autonomous peers. This new paradigm bears the potential to realize computing systems that scale to very large numbers of participating nodes while ensuring fault-tolerance.

However, existing P2P systems offer very limited data management facilities. In most of the cases, searching relies on simple selection conditions on attribute-value pairs or IR-style string pattern matching. These limitations are acceptable for file-sharing applications, but in order to support highly dynamic, ever-changing, autonomous social organizations (e.g., scientific or educational communities) we need richer facilities in exchanging, querying and integrating (semi-)structured data hosted by peers. To this end, we essentially need to adapt the P2P computing paradigm to a distributed data management setting. More precisely, we would like to support loosely coupled communities of peer bases, where each base can join and leave the network at free will, while groups of peers can collaboratively undertake the responsibility of query processing.

The importance of intensional (i.e., schema) information for integrating and querying peer bases has been highlighted by a number of recent projects [4, 34, 17, 1]. A natural candidate for representing descriptive schemata of information resources (ranging from simple structured vocabularies to complex reference models [40]) is the Resource Description Framework/Schema Language (RDF/S). In particular, RDF/S (a) enables a *mod-*

ular design of descriptive schemata based on the mechanism of *namespaces*; (b) allows easy *reuse* or *refinement* of existing schemata through *subsumption* of both class and property definitions; (c) supports partial descriptions since *properties* associated with a resource are by default *optional and repeated* and (d) permits *super-imposed descriptions* in the sense that a resource may be multiply classified under several classes from one or several schemata. These modelling primitives are crucial for P2P data management systems where monolithic RDF/S schemata and resource descriptions cannot be constructed in advance and peers may have only partial descriptions about the available resources.

In this chapter, we present the ongoing SQPeer middleware for routing and planning declarative queries in peer RDF/S bases by exploiting the schema of peers. More precisely, we make the following contributions:

- In Section 2.1 we illustrate how peers can formulate complex (conjunctive) queries against an RDF/S schema using *RQL query patterns* [23].
- In Section 2.2 we detail how peers can advertise their base at a fine-grained level. In particular, we are employing *RVL view patterns* [29] for declaring the parts of an RDF/S schema which are actually (or can be) populated in a peer base.
- In Section 2.3 we introduce a semantic routing algorithm that matches a given RQL query against a set of RVL peer views in order to localize relevant peer bases. More precisely, this algorithm relies on the query/view subsumption techniques introduced in [8] to produce *query patterns annotated with localization information*.
- In Section 2.4 we describe how SQPeer *query plans* are generated by taking into account the involved data distribution (e.g., vertical, horizontal) in peer bases. To this end, we employ an object algebra for RQL queries introduced in [24].
- In Section 2.5 we discuss several *compile* and *run-time optimization opportunities* for SQPeer query plans.
- In Section 3 we sketch how the SQPeer query routing and planning phases can be actually used by groups of peers in order to deploy *hybrid* (i.e., super-peer) and *structured* P2P database systems.

Finally, Section 4 discusses related work and Section 5 summarizes our contributions.

2 The SQPeer Middleware

In order to design an effective query routing and planning middleware for peer RDF/S bases, we need to address the following issues:

1. How peer nodes formulate queries?
2. How peer nodes advertise their bases?
3. How peer nodes route a query?
4. How peer nodes process a query?
5. How distributed query plans are optimized?

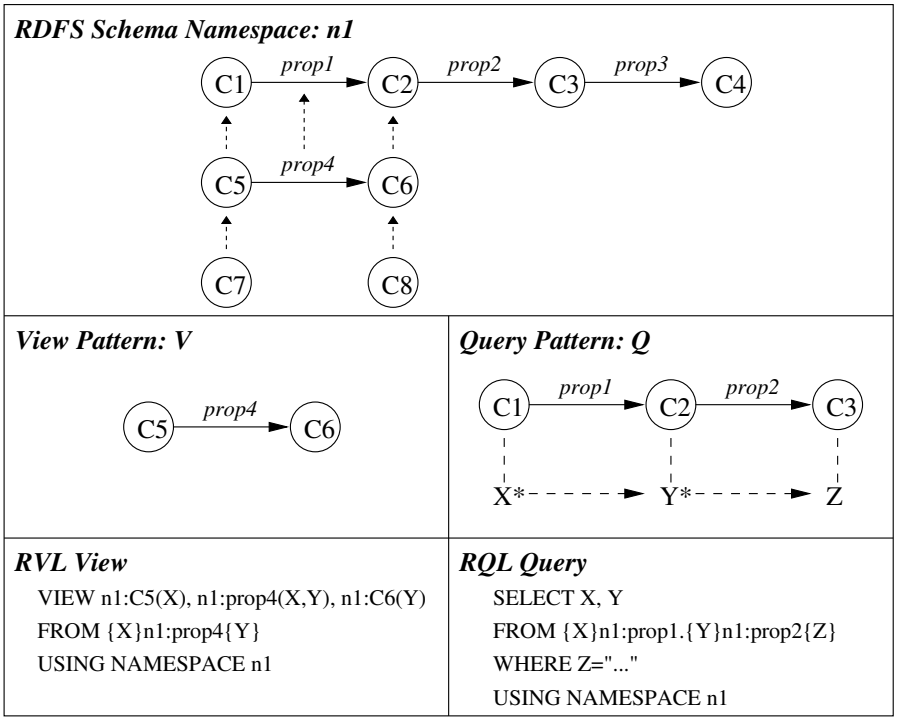


Fig. 1. An RDF/S schema, an RVL view and an RQL query pattern

In the following subsections, we will present the main design choices for SQPeer in response to the above issues.

2.1 RDF/S-based P2P databases and RQL Queries

In SQPeer we consider that each peer provides RDF/S descriptions about information resources available in the network that conform to a number of RDF/S schemata (e.g., for e-learning, e-science, etc.). Peers employing the same schema to create such descriptions in their local bases belong essentially to the same Semantic Overlay Network (SON) [10, 39]. In the upper part of Figure 1, we can see an example of an RDF/S schema defining such a SON, which comprises four classes, `C1`, `C2`, `C3` and `C4`, that are connected through three properties, `prop1`, `prop2` and `prop3`. There are also two subsumed classes, `C5` and `C6`, of `C1` and `C2` respectively, which are related with the subsumed property `prop4` of `prop1`. Finally, classes `C7` and `C8` are subsumed by `C5` and `C6` respectively.

Queries in SQPeer are formulated by peers in RQL, according to the RDF/S schema (e.g., defined in a namespace `n1`) of the SON they belong using an appropriate GUI [2]. RQL queries allow us to retrieve the contents of any peer base, namely resources classified under classes or associated to other

Path Patterns	**Interpretation**
Class Path Patterns	
$C	$\{c \mid c$ is a schema class$\}$
$C{X}	$\{[c, x] \mid c$ a schema class, x in the interpretation of class $c\}$
$C{X;$D}	$\{[c, x, d] \mid c,\ d$ are schema classes, d is a subclass of c, x is in the interpretation of class $d\}$
Property Path Patterns	
@P	$\{p \mid p$ is a schema property$\}$
{X} @P {Y}	$\{[x, p, y] \mid p$ is a schema property, $[x, y]$ in the interpretation of property $p\}$
{$C} @P {$D}	$\{\ [c, p, d] \mid p$ is a schema property, c, d are schema classes, c is a subclass of p's domain, d is a subclass of p's range$\}$
{X; $C} @P {Y; $D}	$\{[x, c, p, y, d] \mid p$ is a schema property, c, d are schema classes, c is a subclass of p's domain, d is a subclass of p's range, x is in the interpretation of c, y is in the interpretation of d, $[x, y]$ is in the interpretation of $p\}$

Table 1. RQL class and property query patterns

resources using properties defined in the RDF/S schema. It is worth noticing that RQL queries incur both intensional (i.e., schema) and extensional (i.e., data) filtering conditions. Table 1 summarizes the basic *class* and *property path patterns*, which can be employed in order to formulate complex RQL *query patterns*. These patterns are matched against the RDF/S schema or data graph of a peer base in order to bind graph nodes or edges to the variables introduced in the *from-clause*. The most commonly used RQL patterns essentially specify the fragment of the RDF/S schema graph (i.e., the intensional information), which is actually involved in the retrieval of resources hosted by a peer base.

For instance, in the bottom right part of Figure 1 we can see an RQL query **Q** returning in the *select-clause* all the resources binded by the variables `X` and `Y`. The *from-clause* employs two property patterns (i.e., {`X`}`n1:prop1`{`Y`} and {`Y`}`n1:prop2`{`Z`}), which imply a join on `Y` between the target resources of the property `prop1` and the origin resources of the property `prop2`. Note that no restrictions are considered for the domain and range classes of the two properties, so the end-point classes `C1`, `C2` and `C3` of `prop1` and `prop2` are obtained from their corresponding schema definitions in the namespace `n1`. The *where-clause*, as usual, filters the binded resources according to the provided boolean conditions (e.g., on variable `Z`). The right middle part of Figure 1 illustrates the pattern of query **Q**, where `X` and `Y` resource variables are marked with "*" to denote projections.

In the rest of this chapter, we are focusing on conjunctive queries formed only by RQL class and property patterns as well as projected variables (filter-

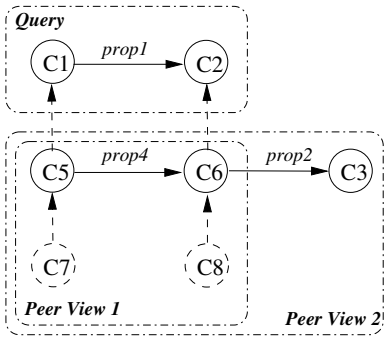


Fig. 2. Peer view advertisements and subsuming queries

ing conditions are ignored). We should also note that SQPeer's query routing and planning algorithms can be also applied to less expressive RDF/S query languages [16].

2.2 RVL Advertisements of Peer Bases

Each peer should be able to advertise the content of its local base to others. Using these advertisements a peer becomes aware of the bases hosted by others in the system. Advertisements may provide descriptive information about the actual data values (extensional) or the actual schema (intensional) of a peer base. In order to reason on the intension of both the query requests and peer base contents, SQPeer relies on materialized or virtual RDF/S schema-based advertisements. In the former case, a peer RDF/S base actually holds resource descriptions created according to the employed schema(s), while in the latter, schema(s) can be populated on demand with data residing in a relational or an XML peer base. In both cases, the RDF/S schema defining a SON may contain numerous classes and properties not necessarily populated in a peer base. Therefore, we need a fine-grained definition of schema-based advertisements. We employ *RVL views* to specify the fragment of an RDF/S schema for which all classes and properties are (in the materialized scenario) or can be (in the virtual scenario) populated in a peer base. These views may be broadcasted to (or requested by) other peers, thus informing the rest of the P2P system of the information actually available in the peer bases. As we will see in Section 3 peer view propagation depends strongly on the underlying P2P system architecture.

The bottom left part of Figure 1 illustrates the RVL statement employed to advertise a peer base according to the RDF/S schema identified by the namespace `n1`. This statement populates classes `C5` and `C6` and property `prop4` (in the *view-clause*) with appropriate resources from the peer's base according to the bindings introduced in the *from-clause*. Given the query pattern used in the *from-clause*, `C5` and `C6` are populated with resources that are direct instances of `C5` and `C6` or any of their subsumed classes, i.e., `C7` and `C8`. Actually,

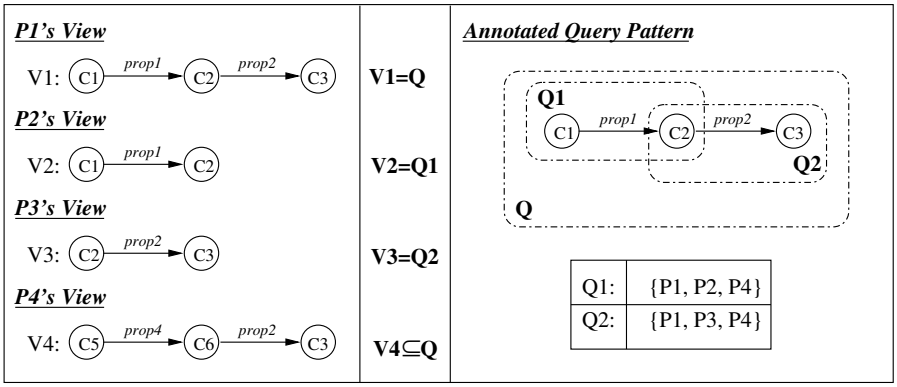


Fig. 3. An annotated RQL query pattern

a peer advertising its base using this view is capable to answer query patterns involving not only the classes `C5` and `C6` (and `prop4`), but also any of the classes (or properties) that subsume them. For example, Figure 2 illustrates a simple query involving classes `C1`, `C2` and property `prop1` subsuming the above peer view 1 (vertical subsumption). The second peer view illustrated in Figure 2 extends the previous view with resource instances of class `C3`, which are reachable through `prop2` with instances of `C6`. Peer view 2 can be employed to answer not only a query {`X;C5`}`prop4`{`Y;C6`}`prop2`{`Z;C3`} but also any of its fragments. As a matter of fact, the results of this query are contained in either {`X;C5`}`prop4`{`Y;C6`} or {`Y;C6`}`prop2`{`Z;C3`} (horizontal subsumption). So peer view 2 can also contribute to the query {`X;C1`}`prop1`{`Y;C2`}.

It is worth noticing that the class and property patterns appearing in the *from-clause* of an RVL statement are the same as those appearing in the corresponding clause of RQL, while the *view-clause* states explicitly the schema information related with the view results (see view pattern in the middle of Figure 1). A more complex example is illustrated in the left part of Figure 3, comprising the view patterns of four peers. Peer P1 contains resources related through properties `prop1` and `prop2`, while peer P4 contains resources related through properties `prop4` and `prop2`. Peer P2 contains resources related through `prop1`, while peer P3 contains resources related through `prop2`.

We can note the similarity in the intensional representation of peer base advertisements and query requests, respectively, as view or query patterns. This representation provides a uniform logical framework to route and plan queries through distributed peer bases using exclusively intensional information (i.e., schema/typing), while it exhibits significant performance advantages. First, the size of the indices, which can be constructed on the intensional peer base advertisements is considerably smaller than on the extensional ones. Second, by representing in the same way what is queried by a peer and what is contained in a peer base, we can reuse the RQL query/RVL view (sound and complete) subsumption algorithms, proposed in the Semantic Web Integration Middleware (SWIM [8]). Finally, compared to global schema-based ad-

Routing Algorithm:
Input: A query pattern QP.
Output: An annotated query pattern QP′.

```
1. QP′ := construct an empty annotated query pattern for QP
2. VP := lookup(QP)
3. for all view patterns VP_i ∈ VP, i=1 ... n do
       if isSubsumed(VP_i, QP) then
          annotate QP' with peer P responsible for VP_i
       end if
   end for
4. return QP′
```

Fig. 4. Query Routing Algorithm

vertisements [34], we expect that the load of queries processed by each peer is smaller, since a peer receives queries that exactly match its base. This also affects the amount of network bandwidth consumed by the P2P system.

2.3 Query Routing and Fragmentation

Query routing in SQPeer is responsible for finding the relevant to a query peer views by taking into account data distribution (vertical, horizontal and mixed) of peer bases committing to an RDF/S schema.

The routing algorithm (outlined in Figure 4) takes as input a query pattern and returns a query pattern annotated with information about the peers that can actually answer it. A lookup service (i.e., function *lookup*), which strongly depends on the underlying P2P topology, is employed to find peer views relevant to the input pattern. The query/view subsumption algorithms of [8] are employed to determine whether a query can be answered by a peer view. More precisely, function *isSubsumed* checks whether every class/property in the query is present or subsumes a class/property of the view (as previously illustrated in Figure 2).

Prior to the execution of the routing algorithm, a *fragmentor* is employed to break a complex query pattern given as input into more simple ones, according to the number of joins (input parameter *#joins*) between the resulting fragments, which are required to answer the original pattern. Recall that a query pattern is always a fragment graph of the underlying RDF/S schema graph. The input parameter *#joins* is determined by the optimization techniques considered by the query processor. In the simplest case (i.e., *#joins* equals to the maximum number of joins in the input query), both query and view patterns are decomposed into their basic class and property patterns (see Table 1). For each query fragment pattern, the routing algorithm is executed and all the available views are checked for identifying those that can answer it.

Algebraic Translation Algorithm:
Input: An annotated query pattern AQ' and current fragment pattern PP (initially the root).
Output: A query plan QP corresponding to the annotated query pattern AQ'.

1. QP := $\emptyset$
2. P := $\{P_1 \ldots P_n\}$, set of peers obtained by the annotation of PP in AQ
3. for all *peers* $P_x \in P$ do
   QP := QP$\bigcup$PP@P_x --*Horizontal Distribution*--
   end for
4. for all *fragment patterns* $PP_i \in$ *children(PP)*
   TP_i := *Algebraic Translation Algorithm*(PP_i, AQ')
   end for
   QP := $\bowtie_{Cp}$(QP, TP_1, ..., TP_m) --*Vertical Distribution*--
5. return QP

Fig. 5. Algebraic Translation Algorithm

Figure 3 illustrates an example of how SQPeer routing algorithm works given an RQL query **Q** composed by two property patterns, namely **Q1** and **Q2**, as well as the views of four peers. The middle part of the figure depicts how each pattern matches one of the four peer views. The variable *#joins* in this example is set to 1, so the two simple property patterns of query **Q** are checked. A more sophisticated fragmentation example will be presented in Section 3. P1's view consists of the property patterns **Q1** and **Q2**, so both patterns are annotated with P1. P2's view consists of pattern **Q1** and P3's view consists of **Q2**, so **Q1** and **Q2** are annotated with P2 and P3 respectively. Finally, P4's view is subsumed by patterns **Q1** and **Q2**, since `prop4` is a subproperty of `prop1`. Similarly to P1, **Q1** and **Q2** are annotated with P4. In the right part of Figure 3 we can see the annotated query pattern returned by the SQPeer routing algorithm, when applied to the RQL query and RVL views of our example.

It should be also stressed that SQPeer is capable to reformulate queries expressed against a SON RDF/S schema in terms of heterogeneous descriptive schemata employed by remote peers. This functionality is supported by powerful mappings to RDF/S of both structured relational and semistructured XML peer bases offered by SWIM [8].

2.4 Query Planning and Execution

Query planning in SQPeer is responsible for generating a distributed query plan according to the localization information returned by the routing algorithm. The first step towards this end, is to provide an algebraic translation of the RQL query patterns annotated with data localization information.

The algebraic translation algorithm (see Figure 5) relies on the object algebra of RQL [24]. Initially, the annotated query pattern (i.e., a schema

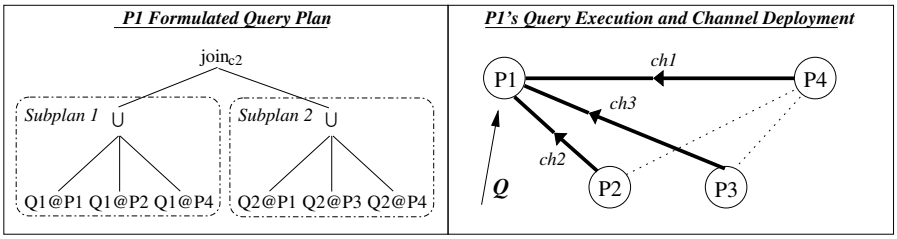


Fig. 6. Query plan generation and channel deployment in SQPeer

fragment) is traversed and for each subfragment considered by the fragmentation policy the annotations with relevant peers are extracted. If more than one peers can answer the same pattern, the results from each such peer base are "unioned" (horizontal distribution). As the query pattern is traversed, the results obtained for different patterns that are connected at a specific domain or range class are "joined" (vertical distribution). The final query plan is created when all fragment patterns are translated.

Figure 6 illustrates how the RQL query **Q** introduced in Figure 1 can be translated given the four peer views presented in Figure 3. In this example, we assume that P1 has already executed the routing algorithm in order to generate the annotated query pattern depicted in Figure 3. The algebraic translation algorithm, also running at P1, initially translates the root pattern, i.e., **Q1**, into the algebraic `Subplan 1` depicted in Figure 6 (i.e., P1, P2 and P4 can effectively answer the subquery). The partial results obtained by these peers should be "unioned" (horizontal distribution). By checking all the children patterns of the root, we recursively traverse the input annotated query pattern and translate its constituent fragment plans. For instance, when **Q2** is visited as the first (and only) child of **Q1** the algebraic `Subplan 2` is created (i.e., P1, P3 and P4 can effectively answer the subquery). Then, the returned query plan concerning **Q2** is "joined" (vertical distribution) with `Subplan 1`, thus producing the final plan illustrated in the left part of Figure 6 (i.e., no more fragments of the initial annotated query pattern **Q** need to be traversed). We can easily observe from our example that taking into account the vertical distribution ensures *correctness* of query results (i.e., produce a valid answer), while considering horizontal distribution in query plans favours *completeness* of query results (i.e., produce more and more valid answers).

In order to create the necessary foundation for executing distributed query (sub)plans among the involved peers, SQPeer relies on appropriate communication *channels*. Through channels, peers are able to route (sub)plans and exchange the intermediary results produced by their execution. It is worth noticing that channels allow each peer to further route and process autonomously the received (sub)plans, by contacting peers independently of the previous routing operations. Finally, channel deployment can be adapted during query execution in order to response to network failures or peer processing limitations. Each channel has a root and a destination node. The root node of a

channel is responsible for the management of the channel by using its local unique id. Data packets are sent through each channel from the destination to the root node. Beside query results, these packets can also contain information about network or peer failures for possible plan modification or even statistics for query optimization purposes. The channel construct and operations of ubQL [35] are employed to implement the above functionality in the SQPeer middleware.

Once a query plan is created and a peer is assigned to its execution (see Section 2.5), this peer becomes responsible for the deployment of the necessary channels in the system (see right part of Figure 6). A channel is created having as root the peer launching the execution of the plan and as destination one of the peers that need to be contacted each time according to the plan. Although each of these peers may contribute in the execution of the plan by answering to more than one fragment queries, only one channel is of course created. This is one of the objectives of the optimization techniques presented in the sequel.

2.5 Query Optimization

The query optimizer receives an algebraic query plan created and outputs an optimized execution plan. In SQPeer, we consider two possible optimization strategies of distributed query plans, namely *compile* and *run-time optimizations.*

Compile-time Optimization

Compile-time optimization relies on algebraic equivalences (e.g., distribution of joins and unions) and heuristics allowing us to push, as much as, possible query evaluation to the same peers. Additionally, cost-based optimization relies on statistics about the peer bases in order to reorder joins and choose between different execution policies (e.g., data versus query shipping).

As we have seen in Figure 6, the algebraic query plan produced contains unions only at the bottom of the plan tree. We can push unions to the top and consequently push joins closer to the leaves. This makes possible (a) to evaluate an entire join at a single peer (intra-peer processing) when its view is subsumed by the query fragment, and (b) to parallelize the execution of the union in several peers. The latter can be achieved by allowing for example each fragment plan (consisting of only joins) to be autonomously processed and executed by different peers. The former suggests applying the following algebraic equivalence as long as the number of inter-peer (i.e., between different peers) joins in the equivalent query plan is less than the intra-peer one. This heuristic comes in accordance to best effort query processing strategies for P2P systems introduced in [43]. Moreover, promoting intra-peer processing exploits the benefits of query shipping as discussed in [13].

Algebraic equivalence: Distribution of joins and unions

Given a subquery $\bowtie (\bigcup(Q_{11}, \ldots, Q_{1n}), \bigcup(Q_{21}, \ldots, Q_{2m}))$ *rewrite it into* $\bigcup(\bowtie (Q_{11}, Q_{21}), \bowtie (Q_{11}, Q_{22}), \ldots, \bowtie (Q_{1n}, Q_{2m}))$.

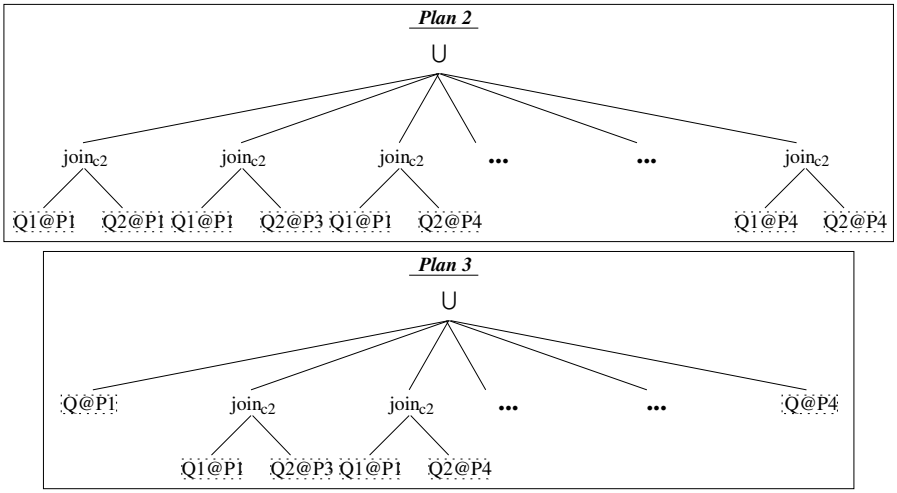


Fig. 7. Optimizing query plans by applying algebraic equivalences and heuristics

According to the above algebraic equivalence, the algebraic query plan of Figure 6 is transformed into the equivalent query execution `Plan 2` of Figure 7. One can easily observe that query `Plan 2` does not take into account the fact that one peer (e.g., P4) can answer more than one successive patterns, unless more sophisticated fragmentation is considered (see Section 2.4). To this end, we apply the following two heuristics for identifying those fragment plans that can be answered by the same peer.

Heuristic 1:

Given a subquery $\bowtie(Q_1@P_i, \ldots, Q_n@P_i)$ *rewrite it into* $Q@P_i$, *where* $Q = Q_1 \bowtie \ldots \bowtie Q_n$.

Heuristic 2:

Given a subquery $\bowtie(\bowtie(QP, Q_1@P_i), Q_2@P_i)$ *rewrite it into* $\bowtie(QP, Q@P_i)$, *where* $Q = Q_1 \bowtie Q_2$.

As we can see in Figure 7, the produced `Plan 3` enables to execute the entire query pattern **Q** to the relevant peers, i.e., joins on properties `prop1` and `prop2` will be executed by peers P1 and P4 respectively.

Furthermore, statistics about the communication cost between peers (e.g., measured by the speed of their connection) and the size of expected intermediary query results (given by a cost-model) can be used to decide which peer and in what order will undertake the execution of each query operator and thus the concrete channel deployment. To this end, the processing load of the peers should also be taken into account, since a peer that processes fewer queries, even if its connection is slow, may offer a better execution time. This processing load can be measured by the existence of slots in each peer, which show the amount of queries that can be handled simultaneously.

Having these statistics in hand, a peer (e.g., P1) can decide at compile-time between *data, query* or *hybrid shipping* execution policies. In the left part

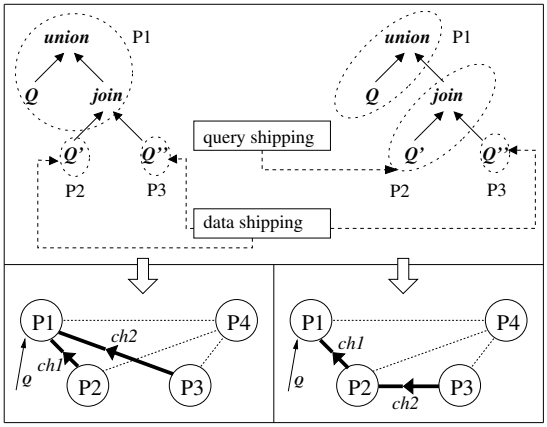


Fig. 8. Data and Query Shipping Example

of Figure 8 we can see the data shipping alternative, since P1 sends queries **Q'** and **Q"** to peers P2 and P3 and joins their results locally. In the right part of Figure 8 we can see the query shipping alternative, since P1 decides to forward the join operation down to P2, which in turn receives the results from P3 and executes the join locally before sending the full answer to P1 for further processing. At the bottom of the figure, we can see the deployment of the corresponding channels for each of these two alternative execution policies. In the case where the communication cost between peers P1 and P3 is greater than the cost between peers P2 and P3 or P2 intermediate results for query fragment **Q'** are large, query-shipping is preferable, since it exploits the fastest peer connection. In the case where peer P2 has a heavy processing load, data-shipping should be chosen, since P1 will execute both the union and the join operators of the plan. In a situation where we have to choose between two or more of the above optimizations, SQPeer favors the execution of the intra-site query operators.

Run-time Optimization

On the other hand, run-time adaptability of query plans is an essential characteristic of query processing when peer bases join and leave the system at free will or more generally when system resources are exhausted. For example, the optimizer may alter a running query plan by observing the throughput of a certain channel. This throughput can be measured by the number of incoming or outgoing tuples (i.e., resources related through one or several properties). Changing query plans may alter an already installed channel, as well as the query plans of the root and destination peer of the channel. These changes include deciding at execution time on altering the data or query shipping decision or discovering alternative peers for answering a certain fragment plan. The root peer of each channel is responsible for identifying possible problems

caused by environmental changes and for handling them accordingly. It should also inform all the involved peers that are affected by the alteration of the plan. Since the alteration is done on a fragment plan and not on the whole query plan, only the peers related to this fragment plan should be informed and possibly a few other peers that contain partial results from the execution of the failed plan. Finally, the root peer should create a new query plan by re-executing the routing and planning phases and not taking into consideration those peers that became obsolete.

We should keep in mind that switching to a different query plan in the middle of the query execution raises interesting problems. Previous results, which were already created by the execution of the query to possible multiple peers, have to be handled, since the new query plan will produce new results. Two are the possible solutions to this issue. The ubQL approach [35] proposes to discard previous intermediate results and all on-going computations are terminated. Alternatively [21] proposes a phased query execution, in which each time the query plan is changed, the system enters into a new phase. The final phase, which is called the cleanup phase, is responsible for combining the sub-results from the other phases in order to obtain a full answer. In SQPeer middleware, we have adopted the ubQL approach.

3 P2P Architectures and SQPeer

SQPeer can be used in different P2P architectural settings. Even though the specific P2P architecture affects peers' topology, the proposed algorithms can be applied to any particular architectural setting. Recall that the existence of SONs minimizes the broadcasting (flooding) activity in the P2P system, since a query is received and processed only by the relevant peers. In the sequel, we detail the possible roles that peers may play in each setting with respect to their corresponding computing capabilities.

On the one hand, we have *client-peers*, which may frequently join or leave the system. These peers have only the ability to pose RQL queries to the rest of the P2P system. Since these peers usually have limited computing capabilities and they are connected to the system for short period of time, they do not participate in the query routing and planning phases.

On the other hand, we may have *simple-peers* that also act autonomously by joining or leaving the system, maybe not so frequently as client-peers. Their corresponding bases can be shared by other peers during their connection to the P2P system. When they join the system, simple-peers can broadcast their views or alternatively request the RVL views of their known neighbors. Thus, a simple-peer identifies and connects physically with the SON(s) it belongs to and becomes aware of its new neighborhood. Simple-peers have also the ability to pose queries as client-peers, but with the extra functionality of executing these queries against their own local bases or coordinate the execution of fragment queries on remote peers.

Additionally, a small percentage of the peers may play the role of *super-peers*. Super-peers are usually highly-available nodes offering high computing capabilities and each one acts as a centralized server for a subset of simple-peers. Super-peers are mainly responsible for routing queries through the system and for managing the cluster of simple-peers that are responsible for. Furthermore, super-peers may play the role of a mediator in a scenario where a query expressed in terms of a globally known schema needs to be reformulated in terms of the schemata employed by the local bases of the simple-peers by using appropriate mapping rules.

In this context, we consider two architectural alternatives distinguished according to the topology of the peer network and the distribution of peer base advertisements. The first alternative corresponds to a *hybrid* P2P architecture based on the notion of *super-peers* while the second one is closer to a *structured* P2P architecture based on *Distributed Hash Tables* (DHTs). In the structured architecture, SONs are created in a *self-adaptive way*, while in the super-peer architecture SONs are created in a more *static way*, since each super-peer is responsible for the creation and further management of SONs. It should be stressed that while in the structured architecture, peers handle both the query routing and planning load, super-peers are primarily responsible for routing and simple-peers for query planning in two distinct phases. Additionally, super-peers are aware of all simple-peer views in a SON, while in the structured alternative this knowledge is distributed and becomes available through an adequate lookup service.

3.1 Hybrid P2P SONs

In a hybrid P2P system [44, 34] each peer is connected with at least one super-peer, who is responsible for collecting the views (materialized or virtual) of all its simple-peers. The peers, holding bases described according to the same RDF/S schema, are clustered under the same super-peer. Thus, each peer implicitly knows the views of all its semantic neighbors. In a more sophisticated scenario, super-peers are responsible only for a specific fragment of the RDF/S schema and thus a cluster of super-peers is responsible for the entire schema. Moreover, a hierarchical organization of super-peers can be adopted, where the classes and properties managed at each level are connected through semantic relationships (e.g., subsumption) with the class and properties of the upper and lower levels.

When a peer connects to a super-peer, it forwards its corresponding view. All super-peers are aware of each other, in order to be able to answer queries expressed in terms of different RDF/S schemata (or fragments), while a simple-peer should be connected to several super-peers when its base commits to more than one schemata. The exact topology of the P2P system depends on the clustering policy with respect to the number of available super-peers providing the bandwidth and connectivity guarantees of the system.

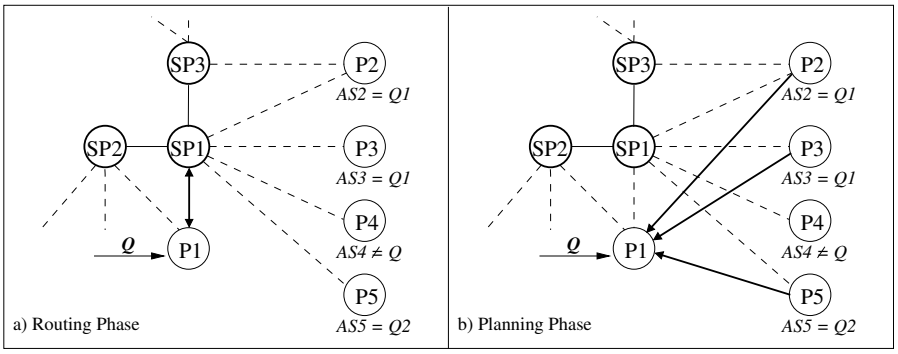


Fig. 9. SQPeer separated query routing and planning phases in a hybrid P2P system

A client-peer can connect to a simple-peer and issue a query request for further processing to the system. The simple-peer forwards the query to the appropriate super-peer according to the schema employed by the query (e.g., by examining the involved namespaces). If this schema is unknown to the simple-peer, it sends the query randomly to one of its known super-peers, which will consecutively discover the appropriate super-peer through the super-peer backbone. In this alternative, we distinguish two separate query evaluation phases: the first corresponds to *query routing* performed exclusively at the super-peers, while the second to *query planning and execution*, which is usually performed by the simple-peers.

For example, in Figure 9, we consider a super-peer backbone containing three super-peers, SP1, SP2 and SP3, and a set of client-peers, P1 to P5. All the simple-peers are connected with at least SP1, since their bases commit to the schema that SP1 is responsible for. When P1 receives a query **Q**, it initially contacts SP1, which is the super-peer responsible for the SON on which the query is addressed (Figure 9a). Since SP1 contains all related peer views, it can also decide on the appropriate fragmentation of the received query pattern according to the view patterns of its simple-peers. Then, SP1 creates an annotated query pattern containing the localization information that P2 and P3 can answer only the **Q1** pattern, while P5 can answer only the **Q2** pattern. SP1 sends this annotated pattern to P1 to generate the appropriate query plan. In our example, this plan implies the creation of three channels with P2, P3 and P5 for gathering the results (Figure 9b). P2, P3 and P5 send their results back to P1, who joins them locally in order to produce the final answer. We should point out that since a super-peer contains all the peer views related to a specific RDF/S schema, the annotated query pattern for **Q** will contain sufficient localization information for producing not only a correct but also a complete query plan and thus no further routing and planning phases for **Q** are required.

3.2 Structured P2P SONs

Alternatively, we can consider a structured P2P architecture [6, 7, 38]. Peers in the same SON are organized according to the topology imposed by the underline structured P2P architecture, e.g., based on Distributed Hash Tables (DHTs) [42, 20]. In DHT-based P2P systems, peers are logically placed in the network according to the value of a *hash function* applied to their IP, while a table of pointers to a predefined number of neighbor peers is maintained. Each information resource (e.g., a document or a tuple) is uniquely identified within the system by a key. In order to locate the peers hosting a specific resource, we need to match the hash value of a given key with the hash value of a peer and forward the lookup request to other peers by taking into account the hash table maintained by each contacted peer. In our context, unique keys are assigned to each view pattern and hence peers, whose hash values match those keys, are aware of the peer bases that are populated with data answering a specific schema fragment. An appropriate key assignment and hash function should be used in order neighbor peers to hold successive view patterns with respect to the class/property hierarchy defined in the employed RDF/S schema. This is necessary for optimizing query routing, since successive view patterns are likely to be subsumed by the same query pattern.

Unlike super-peers, in this alternative there is no peer with a global knowledge of all peer views in the SON. The localization information about remote peer views is acquired by the lookup service supported by the system. Specifically, we are interested in identifying peer views that can actually answer an entire (sub)query pattern given as input. This implies an *interleaved execution of query routing and planning phases* in several iteration rounds leading to the creation and execution of multiple query plans that when "unioned" offer completeness in the results. Note that the generated plans at each round can be actually executed (in contrast to bottom-up dynamic programming algorithms) by the involved peers in order to obtain the first parts of the final query answer. Starting with the initial query pattern, at each round, smaller fragments are considered in order to find the relevant peer bases (routing phase) that can actually answer them (planning phase). In this context, the interleaved query processing terminates when the initial query is decomposed into its basic class and property patterns. It should be also stressed that SQPeer interleaved query routing and planning favors intra-site joins, since each query fragment is looked up as a whole and only peers that can fully answer it are contacted.

For example, in Figure 10 we consider that peers P1 to P8 are connected in a structured P2P system. When P1 receives the query **Q**, it launches the interleaved query routing and planning. At round 1, P1 issues a lookup request for the entire query pattern **Q**, and annotates **Q** with peers P2 and P4. In this initial round, plan `Plan 1 =` $Q@P2 \bigcup Q@P4$ is created and executed. At round 2, the fragmentor is called with *#joins* equal to 1. The two possible fragmentations of query **Q** are depicted in Figures 10a and b.

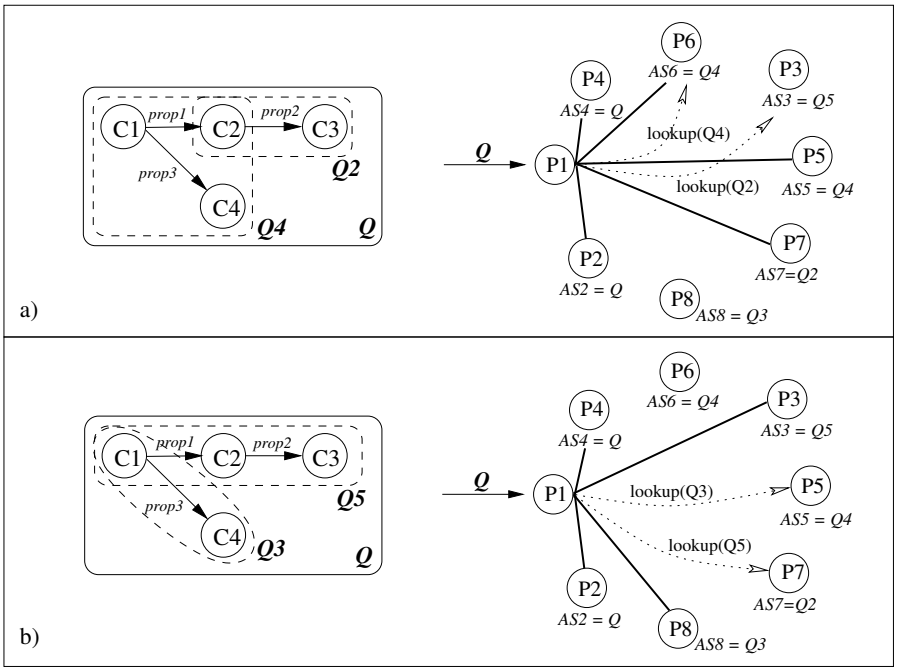


Fig. 10. SQPeer interleaved query routing and planning mechanism in a structured P2P system for a fragmentation round with *#joins*=1

First, peers P6 and P3 are contacted through the lookup service, since they contain the list of peer bases answering query fragment patterns **Q4** and **Q2** respectively (seen in the left part of Figure 10a). P6 returns the list of peers P2, P4, P5 and P6, while P3 returns peers P2, P3, P4 and P7. For this fragmentation, the query plan `Plan 2 =` $\bigcup(\bowtie(Q4@P2, Q2@P3), \bowtie(Q4@P2, Q2@P4), \ldots, \bowtie(Q4@P6, Q2@P4), \bowtie(Q4@P6, Q2@P7))$ is created and executed by deploying the necessary channels between the involved peers (see right part of Figure 10a). It is worth noticing that the generated plans at each round do not include redundant computations already considered in a previous round. For example `Plan 2` produced in round 2 excludes the query fragment plan $\bowtie(Q4@P2, Q2@P2)$ generated in round 1. Next, peers P5 and P7 are contacted through the lookup service, since they contain the list of peer bases answering query patterns **Q3** and **Q5** respectively (seen in the left part of Figure 10b). P5 returns the list of peers P2, P4, P5, P6 and P8, while P7 returns peers P2, P3 and P4 and the query plan `Plan 3 =` $\bigcup(\bowtie(Q3@P2, Q4@P3), \bowtie(Q3@P2, Q4@P4), \ldots, \bowtie(Q3@P8, Q4@P3), \bowtie(Q3@P8, Q4@P4))$ is created and executed (see right part of Figure 10b). Again, `Plan 3` is disjoint with the plans already generated. At the last round (*#joins* equals to 2), we consider all basic property and class patterns of query **Q** and run one more time the routing and planning algorithms to produce query plans returning the remaining parts of the final answer.

4 Related Work

Several projects address query planning issues in P2P database systems. Query Flow [26] is a system offering dynamic and distributed query processing using the notion of HyperQueries. HyperQueries are essentially fragment plans that exist in each peer and guide routing and processing of a query through the network. Furthermore, ubQL [35] provides a suite of process manipulation primitives that can be added on top of any declarative query language to support distributed query optimization. ubQL distinguishes the deployment from the execution phase of a query and supports adaptability of query plans during the execution phase. Compared to these projects, SQPeer does not require an a priori knowledge of the relevant to a query peers.

Mutant Query Plans (MQPs) [41] are logical query plans, where leaf nodes may consist of URN/URL references, or of materialized XML data. The references to resource locations (URLs) point to peers where the actual data reside, while the abstract resource names (URNs) can be seen as the thematic topics of the requested data in a SON. MQPs are themselves serialized as XML elements and are exchanged among the peers. When a peer N receives a MQP M, N can resolve the URN references and/or materialize the URL references, thus offering its local localization information. Furthermore, S can evaluate and re-optimize MQP fragment plans by adding XML fragments to the leafs. Finally, it can just route M to another peer. When a MQP is fully evaluated, i.e., reduced to a concreate XML document, the result is returned to the *target* peer, which has initiated the query. The efficient routing of MQPs is preserved by information derived from multi-hierarchic topic namespaces (e.g., for educational material on computer science or for geographical information) organized by assigning different roles to specific peers. This approach is similar to a super-peer architecture, with the difference that a distributed query routing phase is introduced involving more than one peers. Unlike SQPeer, MQP reduces the optimization opportunities by simply migrating possibly big XML fragments of query plans along with partial results of query fragments. In addition, it is not clear how subtopics can be exploited during query routing.

AmbientDB [6] addresses P2P data management issues in a digital environment, i.e., audio players exchanging music collections. AmbientDB provides full relational database functionality and assumes the existence of a common global schema, although peers may dispose their own schemata (mappings are used in this case). In AmbientDB, apart from the local tables stored at each peer, horizontal data distribution is considered, since fragments of a table, called distributed tables, may be stored at different peers. The query processing mechanism is based on a three-level translation of an "abstract global algebra" into stream based query plans, distributed over an ad-hoc and self-organizing P2P network. Initially, a query is translated into standard relational operators for selection, join, aggregation and sort over "abstract table types". Then, this abstract query plan becomes concrete by instantiat-

ing the abstract table types with concrete ones, i.e., the local or distributed tables that exist in the peer bases. Finally at the execution level, the concrete query plan is executed by selecting between different query execution strategies. The AmbientDB P2P protocol is responsible for query routing and relies on temporary (logical) routing trees, which are created on-the-fly as subgraphs of the Chord network. Chord is also used to implement clustered indices of distributed tables in AmbientDB. Each AmbientDB peer contains the index table partition that corresponds to it after hashing the key-values of all tuples in the distributed table. The user decides for the use of such DHTs, thus accelerating relevant lookup queries. Compared to AmbientDB, SQPeer provides a richer data framework, as well as exhibits a run-time adaptability of generated query plans. More importantly, DHT in SQPeer is based not on data values but on peer views, thus providing efficient intensional indexing and routing capabilities.

Other projects address mainly query routing issues in SONs. In [14] indices are used to identify peers that can handle containment queries (e.g., in XML). For each keyword in the query, a peer searches its indices and returns a set of peers that can answer it. According to the operators used to connect these keywords, the peer decides whether to union or intersect the sets of relevant peers. In this approach, queries are directly sent to the set of peers returned by the routing phase with no further details on how a set of semantically related peers can actually execute a complex query involving both vertical and horizontal data distribution.

RDFPeers [7] is a scalable distributed RDF/S repository based on an extension of Chord, namely MAAN (Multi-Attribute Addressable Network), which efficiently answers multi-attribute and range queries. Peers are organized into a Chord-like ring. In MAAN, each RDF triple is hashed and stored for each of its subject, predicate or object values in corresponding positions of the ring. Furthermore, for numerical attributes MAAN uses order preserving hash functions for placing close values to neighboring peers in the ring, thus optimizing the evaluation of range queries. Routing is performed as in Chord by searching for each value of the query and combining the results at the peer launching the initial query. This approach ignores RDF/S schema information during query routing, while distributed query planning and execution policies are not addressed.

In [36], a super-peer like P2P architecture is introduced, which relies on the extension of an existing RDF/S store. Authors propose an index structure for all the path patterns that can be extracted given an RDF/S schema. The paths in the index are organized hierarchically according to their length (simple properties appear as leaves of the tree). For each path in the tree, the index maintains information about the peers that can answer it, as well as the size of path instantiations. A query processing algorithm determines all possible combinations of the subpaths of a given query pattern, as well as, the peers that can answer it. The proposed index structure, which is considered to be controlled by a mediator, is difficult to be updated and handled in a

situation where peers frequently enter or leave the system. The localization information concerning different query fragments is held in a centralized way. Although schema information is used for indexing, RDF/S class and property subsumption is not considered as in SQPeer. Finally, optimization (based on a cost model) is focused only on join re-orderings, which is a subset of the optimizations considered in SQPeer.

The Edutella project [34] explores the design and implementation of a schema-based P2P infrastructure for the Semantic Web. In Edutella, peer content is described by different and extensible RDF/S schemata. Super-peers are responsible for message routing and integration/mediation of peer bases. The routing mechanism is based on appropriate indices to route a query initially within the super-peer backbone and then between super-peers and their respective simple peers. A query processing mechanism in such a schema-based P2P system is presented in [5]. Query evaluation plans (QEPs) containing selection predicates, aggregation functions, joins, etc., are pushed from clients to simple or super-peers where they are executed. Super-peers dispose an optimizer for generating plans determining which fragments of the original query will be sent to the next (super-)peers and which operators will be locally executed. This approach involves rather simple query/view rewriting techniques (i.e., exact matching of basic class and property patterns) which ignores subsumption. In addition, a query is fragmented in its simple class and property patterns, thus not allowing the handling of more complex fragment graphs of the employed RDF/S schema.

To conclude, although the use of indices and super-peer topologies facilitate query routing, the cost of maintaining (XML or RDF) extensional indices of entire peer bases is important compared to the cost of maintaining intensional peer views, as in the case of SQPeer. In addition, SQPeer's interleaved execution of the routing and planning phases enables to obtain quickly the first results of the query (and probably the most relevant ones) while planning is still running. This is an original feature of the SQPeer query processing, taking into account that the search space of plans required to obtain a complete result in P2P systems is exponential. Last but not least, SQPeer can be used to deploy both hybrid and structured P2P systems.

5 Summary

In this chapter, we have presented the design of the ICS-FORTH SQPeer middleware offering sophisticated query routing and planning services for P2P database systems. We presented how declarative RQL queries and RVL views expressed against an RDF/S schema can be represented as schema-based patterns. We sketched a semantic routing algorithm, which relies on query/view subsumption techniques to annotate query patterns with peer localization information. We also presented how SQPeer query plans are created and executed by taking into account the data distribution in peer bases. Finally, we have discussed several compile and run-time optimization opportunities for

SQPeer query plans, as well as possible architectural alternatives for static or self-adaptive RDF/S-based P2P database systems.

Several issues remain open with respect to the effective and efficient processing of distributed queries in SQPeer. The number of plans that need to be considered by our dynamic programming planner can be fairly large especially when we generate all fragmentation alternatives of a large query pattern given as input. To this end, we intend to investigate to what extend heuristic pruning techniques (e.g., iterative dynamic programming [28]) can be employed to prune fragment plans as soon as possible [11]. Furthermore, we plan to study the tradeoff between result completeness and response time of queries using appropriate information quality metrics (e.g., coverage of schema classes and properties [12, 33, 32, 18]) enabling to obtain quickly the Top-K answers [27, 37]. Finally, we plan to consider adaptive implementations of algebraic operators borrowing ideas from [3, 19, 22].

References

1. Aberer K, Cudre-Mauroux P, Hauswirth M (2003) The Chatty Web: Emergent Semantics Through Gossiping. In Proceedings of the 12th International World Wide Web Conference (WWW), Budapest, Hungary
2. Athanasis N, Christophides V, Kotzinos D (2004), Generating On the Fly Queries for the Semantic Web: The ICS-FORTH Graphical RQL Interface (GRQL). In Proceedings of the 3rd International Semantic Web Conference (ISWC'04), Hiroshima, Japan
3. Avnur R, Hellerstein JM (2000) Eddies: Continuously Adaptive Query Processing. ACM SIGMOD, pp.261–272, Dallas, TX
4. Bernstein PA, Giunchiglia F, Kementsietsidis A, Mylopoulos J, Serafini L, Zaihrayeu I (2002) Data management for peer-to-peer computing: A vision. In Proceedings of the 5th International Workshop on the Web and Databases (WebDB), Madison, Wisconsin
5. Brunkhorst I, Dhraief H, Kemper A, Nejdl W, Wiesner C (2003) Distributed Queries and Query Optimization in Schema-Based P2P-Systems. In Proceedings of the International Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P), Berlin, Germany
6. Boncz P, Treijtel C (2003) AmbientDB: relational query processing in a P2P network. In Proceedings of the International Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P), LNCS 2788, Springer Verlag
7. Cai M, Frank M (2004) RDFPeers: A Scalable Distributed RDF Repository based on A Structured Peer-to-Peer Network. In Proceedings of the 13th International World Wide Web Conference (WWW), New York
8. Christophides V, Karvounarakis G, Koffina I, Kokkinidis G, Magkanaraki A, Plexousakis D, Serfiotis G, Tannen V (2003) The ICS-FORTH SWIM: A Powerful Semantic Web Integration Middleware. In Proceedings of the 1st International Workshop on Semantic Web and Databases (SWDB), Co-located with VLDB 2003, Humboldt-Universitat, Berlin, Germany

9. Clarke I, Sandberg O, Wiley B, Hong TW (2001) Freenet: A Distributed Anonymous Information Storage and Retrieval System. In Proceedings of the International Workshop on Design Issues in Anonymity and Unobservability, Volume 2009 of LNCS, Springer-Verlag
10. Crespo A, Garcia-Molina H (2003) Semantic Overlay Networks for P2P Systems. Stanford Technical Report
11. Deshpande A, Hellerstein JM, (2002) Decoupled Query Optimization for Federated Database Systems. In Proceedings of the 18th International Conference on Data Engineering (ICDE'02), San Jose, California
12. Doan A, Halevy A (2002) Efficiently Ordering Query Plans for Data Integration. In Proceedings of the 18th IEEE Conference on Data Engineering (ICDE)
13. Franklin MJ, Jonsson BT, Kossmann D (1996) Performance Tradeoffs for Client-Server Query Processing. In Proceedings of the ACM SIGMOD Conference, pp.149–160, Montreal, Canada
14. Galanis L, Wang Y, Jeffery SR, DeWitt DJ (2003) Processing Queries in a Large P2P System. In Proceedings of the 15th International Conference on Advanced Information Systems Engineering (CAiSE)
15. The Gnutella file-sharing protocol. Available at : http://gnutella.wego.com
16. Haase P, Broekstra J, Eberhart A, Volz R (2004) A Comparison of RDF Query Languages. In Proceedings of the 3rd International Semantic Web Conference, Hiroshima, Japan
17. Halevy AY, Ives ZG, Mork P, Tatarinov I (2003) Piazza: Data Management Infrastructure for Semantic Web Applications. In Proceedings of the 12th International World Wide Web Conference (WWW)
18. Heese R, Herschel S, Naumann F, Roth A (2005) Self-Extending Peer Data Management. In GI-Fachtagung fur Datenbanksysteme in Business, Technologie und Web (BTW 2005), Karlsruhe, Germany
19. Huebsch R, Jeffery SR (2004) FREddies: DHT-based Adaptive Query Processing via FedeRated Eddies. Technical report, Computer Science Division, University of Berkeley
20. Stoica I, Morris R, Karger D, Kaashoek MF, Balakrishnan H (2001) Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications, ACM SIGCOMM 2001, pp.149–160, San Diego, CA
21. Ives ZG (2002) Efficient Query Processing for Data Integration. phD Thesis, University of Washington
22. Ives ZG, Levy AY, Weld DS, Florescu D, Friedman M (2000) Adaptive Query Processing for Internet Applications. IEEE Data Engineering Bulletin 23:19–26
23. Karvounarakis G, Alexaki S, Christophides V, Plexousakis D, Scholl M (2002) RQL: A Declarative Query Language for RDF. In Proceedings of the 11th International World Wide Web Conference (WWW), Honolulu, Hawaii, USA
24. Karvounarakis G, Christophides V, Plexousakis D, Alexaki S (2001) Querying RDF Descriptions for Community Web Portals. 17ie'mes Journees Bases de Donnees Avancees (BDA'01), Agadir, Maroc
25. The Kazaa file-sharing system. Available at : http://www.kazaa.com
26. Kemper A, Wiesner C (2001) HyperQueries: Dynamic Distributed Query Processing on the Internet. In Proceedings of the International Conference on Very Large Data Bases (VLDB), Rome, Italy
27. Kossmann D (2000) The State of the Art in Distributed Query Processing. ACM Computer Surveys 32:422–469

28. Kossmann D, Stocker K (2000) Iterative Dynamic Programming: A new class of query optimization algorithms, ACM Transactions on Database Systems, volume 25, number 1
29. Magkanaraki A, Tannen V, Christophides V, Plexousakis D (2003) Viewing the Semantic Web Through RVL Lenses. In Proceedings of the 2nd International Semantic Web Conference (ISWC)
30. The Morpheus file-sharing system. Available at: http://www.musiccity.com
31. The Napster file-sharing system. Available at : http://www.napster.com
32. Naumann F, Leser U, Freytag JC (1999) Quality-driven Integration of Heterogeneous Information Systems. In Proceeedings of the 25th International Conference on Very Large Data Bases (VLDB), Edinburgh, UK
33. Nie Z, Kambhampati S (2001) Joint Optimization of Cost and Coverage of Query Plans in Data Integration. In Proccedings of the 10th International Conference on Information and Knowledge Management, Atlanta, Georgia, USA
34. Nejdl W, Wolpers M, Siberski W, Schmitz C, Schlosser M, Brunkhorst I, Loser A (2003) Super-Peer-Based Routing and Clustering Strategies for RDF-Based P2P Networks. In Proceedings of the 12th International World Wide Web Conference (WWW), Budapest, Hungary
35. Sahuguet A (2002) ubQL: A Distributed Query Language to Program Distributed Query Systems. phD Thesis, University of Pennsylvania
36. Stuckenschmidt H, Vdovjak R, Houben G, Broekstra J (2004) Index Structures and Algorithms for Querying Distributed RDF Repositories. In Proceedings of the International World Wide Web Conference (WWW), New York, USA
37. Thaden U, Siberski W, Balke WT, Nedjl W (2004) Top-k Query Evaluation for Schema-Based Peer-to-Peer Networks. In Proceedings of the International Semantic Web Conference (ISWC2004), Hiroshima, Japan
38. Triantafillou P, Pitoura T (2003) Towards a Unifying Framework for Complex Query Processing over Structured Peer-to-Peer Data Networks. In Proceedings of the Workshop on Databases, Information Systems, and Peer-to-Peer Computing (DBISP2P), Collocated with VLDB '03
39. Triantafillou P, Xiruhaki C, Koubarakis M, Ntarmos N (2003) Towards High Performance Peer-to-Peer Content and Resource Sharing Systems. In Proceedings of the Conference on Innovative Data Systems Research (CIDR)
40. Magkanaraki A, Alexaki S, Christophides V, Plexousakis D (2002) Benchmarking RDF Schemas for the Semantic Web. In Proceedings of the 1st International Semantic Web Conference (ISWC'02)
41. Papadimos V, Maier D, Tufte K (2003) Distributed Query Processing and Catalogs for P2P Systems. In Proceedings of the 2003 CIDR Conference
42. Ratnasamy S, Francis P, Handley M, Karp R, Shenker S (2001) A Scalable Content-Addressable Network. ACM SIGCOMM 2001, San Diego, CA
43. Rosch P, Sattler K, Weth C, Buchmann E (2005) Best Effort Query Processing in DHT-based P2P Systems. ICDE Workshop NetDB 2005, Tokyo
44. Yang B, Garcia-Molina H (2003) Designing a Super-Peer Network. In Proceedings of the 19th International Conference Data Engineering (ICDE), IEEE Computer Society Press, Los Alamitos, CA