

# Distributed Queries and Query Optimization in Schema-Based P2P-Systems

Ingo Brunkhorst<sup>1</sup>, Hadhami Dhraief<sup>2</sup>, Alfons Kemper<sup>3</sup>, Wolfgang Nejdl<sup>1,2</sup>, and Christian Wiesner<sup>3</sup>

**Abstract.** Databases have employed a schema-based approach to store and retrieve structured data for decades. For peer-to-peer (P2P) networks, similar approaches are just beginning to emerge, also motivated by the fact, that sending (atomic) queries to the appropriate peers clearly fails for queries which need data from more than one peer to be executed. While quite a few database techniques can be re-used in this new context, a P2P data management infrastructure poses additional challenges which have to be solved before schema-based P2P networks become as common as schema-based databases. Because of the dynamic nature of P2P networks, we can neither assume global knowledge about data distribution, nor are static topologies and static query plans suitable for these networks. Unlike in traditional distributed database systems, we cannot assume a complete schema instance but rather work with a distributed schema which directs query processing tasks from one node to one or more neighboring nodes.

In this paper, we will first discuss a suitable topology for schema-based P2P networks and how distributed knowledge about data distribution can be stored, accessed and updated based on that topology. Second we will describe how this knowledge can be used to distribute abstract query plans through the P2P network and expand them on the fly such that we can place query operators next to data sources and utilize distributed computing resources more effectively.

## 1 Introduction

P2P applications have been quite successful, e.g., for exchanging music files, where networks use simple attributes to describe these resources. A lot of effort has been put into refining topologies and query routing functionalities of these networks, and simple systems like Napster and Gnutella have inspired more efficient infrastructures such as the ones based on distributed hash tables (e.g., CAN and CHORD [26, 28]). Less effort has been put into extending the representation and query functionalities offered by such networks, and projects exploring more expressive P2P infrastructures [22, 2, 1, 13] have only slowly started the move toward schema-based P2P networks.

At the same time, database systems have evolved toward a higher degree of distribution. While it has been a long way from central databases to truly distributed databases, we currently see first explorations toward true peer-to-peer data management infrastructures which will have all characteristics of P2P systems, i.e., *local control of data, dynamic addition and removal of peers, only local knowledge of available data and schemas and self-organization and -optimization*. In this view, schema-based P2P systems are the point where these two directions of research meet [11] (see Figure 1).

In the Edutella project [7, 22, 24] we have been exploring some issues arising in that context, with the goal of designing and implementing a schema-based P2P infrastructure

---

<sup>1</sup> Learning Lab Lower Saxony, University of Hannover, Germany, {brunkhor, nejdl}@learninglab.de

<sup>2</sup> Information Systems Institute, University of Hannover, Germany, {hdhraief, nejdl}@kbs.uni-hannover.de

<sup>3</sup> Computer Science Department, University of Passau, Germany, {wiesner, kemper}@db.fmi.uni-passau.de

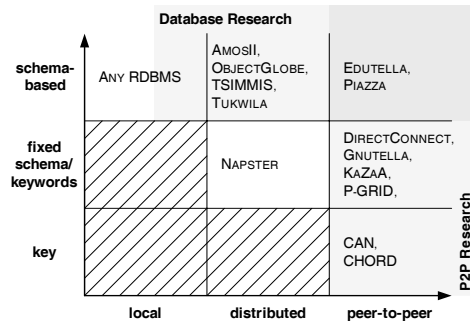


Fig. 1. Schema Capabilities and Distribution

for the Semantic Web. Edutella relies on the W3C metadata standards RDF and RDF Schema (RDFS) [19, 4] to describe distributed resources, and uses basic P2P primitives provided as part of the JXTA framework [9]. In the ObjectGlobe project [3, 17, 18] we have designed and implemented a distributed data network consisting of three kinds of suppliers: *data-providers* supply data, *function-providers* offer query operators to process data, and *cycle-providers* are contracted to execute query operators. ObjectGlobe enables applications to execute complex queries which involve the execution of operators from multiple function providers at different sites (cycle providers) and the retrieval of data and documents from multiple data sources.

In this paper, we discuss in Section 2, how a super-peer based topology and “schema-aware” routing indices allow us to efficiently route queries only to appropriate peers, and how these indices are built and updated, when new peers enter or leave the network. In Sections 3 and 4 we describe how these indices facilitate the distribution and dynamic expansion of query plans, and will explore different strategies for optimizing query plans in this environment. Section 5 gives an overview of two prototypes implementing the proposed techniques and Section 6 concludes with further ideas.

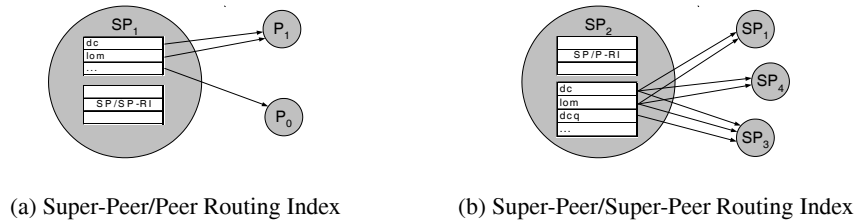
## 2 Routing in Schema-Based P2P-Systems

Efficient query routing is one of the corner stones of advanced P2P systems. By relying on a super-peer topology with “schema-aware” routing indices we show how to advance the efficiency of recent P2P systems. We will start from super-peer networks as a particularly appropriate topology for our schema-based P2P topologies, and discuss topology and routing indices in such a network.

### 2.1 Super-Peer Networks

Each peer in a P2P network usually has varying resources available, e.g., regarding bandwidth or processing power. As discussed in [29], exploiting the different capabilities in a P2P network can lead to an efficient network architecture, where a small subset of peers, called super-peers, takes over specific responsibilities for peer aggregation, query routing, and mediation. An example of a simple super-peer based architecture is the KaZaA network [15], more elaborate versions are described in [5] and [23].

Super-peer based P2P infrastructures are usually based on a two-phase routing architecture, which routes queries first in the super-peer backbone, and then distributes them to the peers connected to the super-peers. Caching of the peers data at the super-peers avoids the second query distribution step but requires considerable amount of storage space and



**Fig. 2.** SP/P and SP/SP Routing Indices

predictable peer behavior for data integrity reasons which is not necessarily guaranteed in P2P systems. Super-peer routing is usually based on different kinds of indexing and routing tables, as discussed in [5] and [23]. Here, we will discuss a routing mechanism based on two indices which store information to route within the P2P backbone and between super-peers and their respective peers.

## 2.2 Routing Indices

**The Edutella Super-Peer Topology** Edutella super-peers [23] employ routing indices which explicitly acknowledge the semantic heterogeneity of schema-based P2P networks, and therefore include schema information as well as other possible index information. Network connections among the super-peers form the super-peer backbone that is responsible for message routing and integration/mediation of metadata.

Super-peers in the Edutella network are arranged in the HyperCuP topology. The HyperCuP algorithm described in [27] is capable of organizing super-peers of a P2P network into a recursive graph structure called hypercube that stems from the family of Cayley graphs. Super-peers join the HyperCuP based super-peer topology by asking any of the already integrated super-peers which then carries out the super-peer integration protocol. No central maintenance is necessary for changing the HyperCuP structure.

HyperCuP enables efficient and non-redundant query broadcasts. For broadcasts, each node can be seen as the root of a specific spanning tree through the P2P network. The topology allows for  $\log_2 N$  path length and  $\log_2 N$  number of neighbors, where  $N$  is the total number of nodes in the network (i.e., the number of super-peers in our case). Peers connect to the super-peers in a star-like fashion, providing content and content metadata. Alternatives to this topology are possible provided that they guarantee the spanning tree characteristic of the super-peer backbone, which we exploit for maintaining our routing indices and distributed query plans.

**Super-Peer/Peer Routing Indices** The super-peer/peer routing indices (SP/P indices for short) contain information about metadata usage at each peer, i.e., which schema and attributes are used to describe the content stored at the peers. On registration the peer provides this information to its super-peer.

In contrast to other approaches (Gnutella [8], CAN [26]), our indices do not refer to individual content elements but to peers (as in CHORD [28]). The indices can contain information about peers at different granularities: schemas, schema properties, property value ranges and individual property values:

**Schema Index** We assume that different peers will support different schemas and that these schemas are uniquely identified (by a URI). The routing index contains the schema identifier as well as the peers supporting this schema.

Granularity	Index of $SP_2$		
Schema	dc		$SP_1, SP_3, SP_4$
	lom		$SP_1, SP_3, SP_4$
	dcq		$SP_3$
Property	dc:subject		$SP_1, SP_3, SP_4$
	lom:type		$SP_1, SP_3, SP_4$
	dc:format		$SP_3, SP_4$
Property Value Range	dc:subject	ccs:dbms	$SP_1, SP_2, SP_3$
Property Value	lom:type	“exercise”	$SP_3$
	dc:language	“de”	$SP_3, SP_4$

Table 1. SP/SP Index of  $SP_2$  at Different Granularities

**Property/Sets of Properties Index** Peers might choose to use only a selection of properties from (one or more) schemas to describe their content. While this is unusual in conventional database systems, it is more often used for data stores using semi-structured data, and very common for RDF-based [4] systems. In this kind of index, super-peers use the properties (uniquely identified by schema ID plus property name) or sets of properties to describe their peers.

**Property Value Range Index** For properties which contain values from a predefined hierarchical vocabulary we can use an index which specifies taxonomies or part of a taxonomy for properties.

**Property Value Index** For some properties it may also be advantageous to create value indices to reduce network traffic. This case is identical to a classical database index with the exception that the index entries do not refer to the resource, but the peer providing it. This index contains only properties that are used very often compared to the rest of the data stored at the peers.

Figure 2(a) shows an example of an SP/P index at the schema granularity. Peer  $P_1$  uses three schema standards for describing its content, the Dublin Core standard [6] (dc for short), the Learning Object Metadata standard [21] (lom for short), and the Qualified Dublin Core Element Set (dcq for short).

**Super-Peer/Super-Peer Routing Indices** In order to avoid query broadcasting (flooding) in the super-peer backbone we introduce super-peer/super-peer routing indices (SP/SP indices) to forward queries among the super-peers. These SP/SP indices are essentially extracts and summaries from all super-peer local SP/P indices. Similar to the SP/P indices they contain schema information at different granularities, but refer to the super-peers' neighbors in the super-peer backbone (as shown in Figure 2(b)). Queries are forwarded to super-peer neighbors based on the SP/SP indices, and sent to connected peers based on the SP/P indices. For instance, Table 1 states the SP/SP routing index of the super-peer  $SP_2$  at different granularities.

For constructing the SP/SP index a super-peer can be seen as the root of a spanning tree. The SP/SP index is build dynamically based on the SP/P indices of all the super-peers on this spanning tree, by backward propagation and aggregation of the SP/P information. The other super-peers update their SP/SP indices accordingly.

For example, the SP/SP routing index of  $SP_2$  states at the schema level that all neighbors ( $SP_1, SP_3, SP_4$ ) support the Dublin Core Schema *dc* and the Learning Object Metadata schema *lom*, but only  $SP_3$  contains information described by the Qualified Dublin

Core Element Set  $dcq$ ). Thus, if a query requires both  $dcq$  and  $lom$ , it will not be routed to  $SP_1$  and  $SP_4$  but to  $SP_3$ . The same routing mechanism applies for queries on the other levels of granularity. A special case is the *Property Value Range* level which gives specific properties in combination with classification hierarchies (like the ACM Computing Classification System, ACM CCS<sup>1</sup>). Making use of the topic hierarchy, the routing index can contain aggregate information in order to reduce the index size.

### 2.3 Peers Registering at Super-Peers

Peers connecting to a super-peer have to register their metadata information at this super-peer thus providing the necessary schema information for constructing the SP/P and SP/SP routing indices. For registration an XML registration message encapsulates a metadata-based description of the peer properties. A peer must register at least one schema (e.g., the DC or the LOM element set) with a set of properties (possibly with additional information), or with information about specific property values. A complete registration example in the RDF-syntax can be found at [14].

The behavior of (super-)peers is rather unpredictable in a P2P network. Thus, these registration messages are valid for a certain period only, and peers have to re-register periodically. By invalidating the peers' registrations periodically we chose a behavior similar to other protocols for dynamic settings (like DHCP) since peers may leave the network without any notice. If a super-peer fails, its formerly connected peers must re-register with another super-peer. We are currently investigating deterministic reconnection strategies using testaments which specify alternative super-peers, and clustering strategies, grouping similar peers (in terms of supported schema) together.

### 2.4 Update of Routing Indices

**Update of the SP/P Index** An update of the SP/P index of a given super-peer occurs, when a peer leaves the super-peer, a new peer registers, or the metadata information of a registered peer changes (e.g., new attributes are added or deleted).

If a peer leaves the super-peer all references to this peer have to be removed from the SP/P index of the respective super-peer. The same applies if a peer fails to re-register periodically. In the case of a peer joining the network or re-registering, its respective metadata/schema information are matched against the SP/P entries of the respective super-peer. If the SP/P routing index already contains the peers' metadata only a reference to the peer is stored in the index otherwise the respective metadata with references to the peer are added to the index. The following algorithm formalize this procedure:

We define  $S$  as a set of schema elements<sup>2</sup>:  $S = \{s_i || i = 1..n\}$ . The super-peer  $SP_x$  already stores a set  $S_x$  of schema elements in its SP/P index. The SP/P index of a super peer  $SP_x$  can be considered as a mapping  $s_i \mapsto \{P_j || j = 1..m\}$ . A new peer  $P_y$  registers to the super peer  $SP_x$  with a set  $S_y$  of schema elements.

1. If  $S_y \subseteq S_x$ , then add  $P_y$  to the list of peers at each  $s_i \in S_y$
2. Else if  $S_y \setminus S_x = \{s_n, \dots, s_m\} \neq \emptyset$ , then update the SP/P index by adding new rows  $s_n \mapsto P_y, \dots, s_m \mapsto P_y$ .

<sup>1</sup> Note that `ccs:networks` is a common super concept of `ccs:ethernet` and `ccs:clientserver` in the ACM CCS taxonomy

<sup>2</sup> A complete schema, e.g., `dc` is also considered as schema element

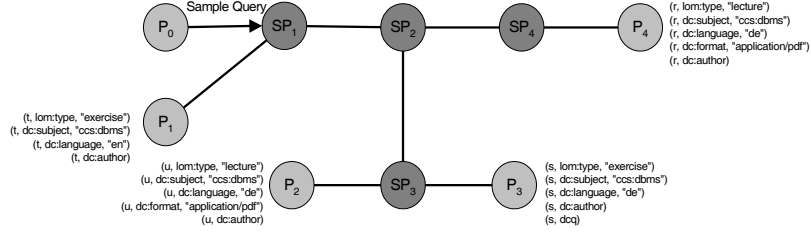


Fig. 3. Routing Example Network

**Update of the SP/SP Index** Let us first consider how to update the SP/SP indices in the backbone, when one of them has been modified as described before. We assume here, that each SP/P modification triggers the update process for SP/SP indices, though we can also collect the modifications for a given period and trigger the SP/SP update process then.

We further assume that the super-peers cluster peers according to their schema characteristics, so that peers connected to a super-peer usually have similar characteristics, and SP/P modifications trigger SP/SP index updates less frequently. If we take for example the network in Figure 3 and the example SP/SP index of  $SP_2$  shown in Table 1, a new peer  $P_x$  registering at super-peer  $SP_1$  with the property  $dc:language$  does not trigger the update process since this metadata information already exists in the SP/P index. If a new peer  $P_y$  registers at  $SP_1$  with the property  $dcq:created$ , the SP/SP update process starts, as this property was not included in the index before.

*SP/SP Update Process.* Remember that super-peers in the network are organized into a HyperCuP topology, which implicitly defines each super-peer as root of a spanning tree. Query routing takes place along the spanning trees (restricted by the SP/SP indices), so the update of SP/SP indices has to be done in the reverse direction. For these updates, again each super-peer acts as the root of a spanning tree (in the “backward direction”), as shown in Figure 4 for the super-peer G. In this example we have a simple (complete) cube, which has three dimensions (0,1,2), such that every node has 3 neighbors.

In order to update the SP/SP indices after an update of the SP/P index of the super-peer  $SP_x$  we build the spanning tree of the super-peer  $SP_x$  as follows:  $SP_x$  sends the update message to all its neighbors, tagging it with the edge label (dimension) on which the message was sent. Super-peers receiving the message update their SP/SP index accordingly and forward the update message, but only to those super-peers tagged with lower edge labels. Furthermore, whenever a message does not change the SP/SP index at a receiving super-peer  $SP_y$ , forwarding stops. The update is done as follows:

- For all  $s_i \in S_x \cap S_y$  add dimension of  $SP_x$  to the list of dimensions at row  $s_i$  if this dimension does not exist.
- For all  $s_i \in S_x \setminus S_y$  add a new row  $s_i \mapsto dimension(SP_x)$

*Adding new Super-Peers.* Adding a new super-peer is a bit more complicated. For a new super-peer, the HyperCuP protocol takes care of identifying new neighbors as discussed in [27]. In this process one of the super-peers is “responsible” for integrating the new super-peer. In most cases the new super-peer will fill a “vacant” position in the hypercube, which has temporarily been administered by the responsible super-peer. In this process, this super-peer, who has been holding an additional SP/SP and SP/P index for the vacant position, transfers these indices to the new super-peer. If the new super-peer opens a new

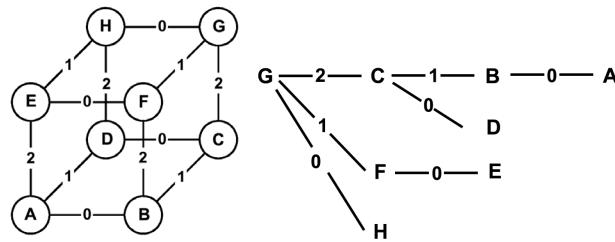


Fig. 4. HyperCup Topology and Spanning Tree Example

dimension, it has to take over some peers from the old super-peer, and the SP/SP index has to be split into two indices. The neighboring super-peers have to update their indices accordingly, by exchanging the responsible super-peer with the new super-peer on the appropriate dimension. Beyond the immediate neighbors, no further update is necessary.

*Removing Super-Peers.* The HyperCuP protocol also takes care of super-peers leaving the backbone. We usually assume that the leaving super-peer coordinates this operation, and specifically asks appropriate super-peer(s) (more than one if the leaving super-peer temporarily fills several positions) that will administer its position afterwards. In this process the administering super-peers take over the SP/SP and SP/P indices of the leaving super-peer, and the neighbors of the leaving super-peer as well as of the administering ones have to update their SP/SP indices. Again, no update is required beyond the immediate neighbors. Peers of the leaving super-peer reconnect to the super-peer which administers the vacant position.

In the case of unexpected link failure its neighbors determine the “closest” (regarding smallest hop distance) super-peer. This super-peer then coordinates the administration of the open position with the same procedure as described above. Peers of the failing super-peer have to reconnect at some other super-peer, possibly triggering further SP/SP update messages.

### 3 Query Processing in P2P Networks

As described before, P2P networks can be divided into the two classes: pure P2P networks and schema-based P2P networks. In this section we demonstrate at first the deficiencies of traditional query processing in both classes as they rely on data shipping. Then we propose our approach on dynamic, extensible, and distributed query processing in schema-based P2P networks. We illustrate query processing by Figures 5 and 6 where a client peer states a query to search for some information and uses its own filter predicates (the two stars) to select the relevant information.

Although distributed query optimization and execution are well known problems in databases, distributed query processing on distributed metadata is novel. Middleware systems, e.g., Garlic [16], have been used to overcome the heterogeneity faced when data is dispersed across different data sources. In [20] a central mapping information of all participating, distributed data sources is queried. [25] introduces so called mutant query plans which encapsulate partially evaluated query plans and unevaluated data. Their approach is not capable of supporting user-defined operators. Furthermore, loss of pipelining during execution limits the general applicability for distributed query processing.

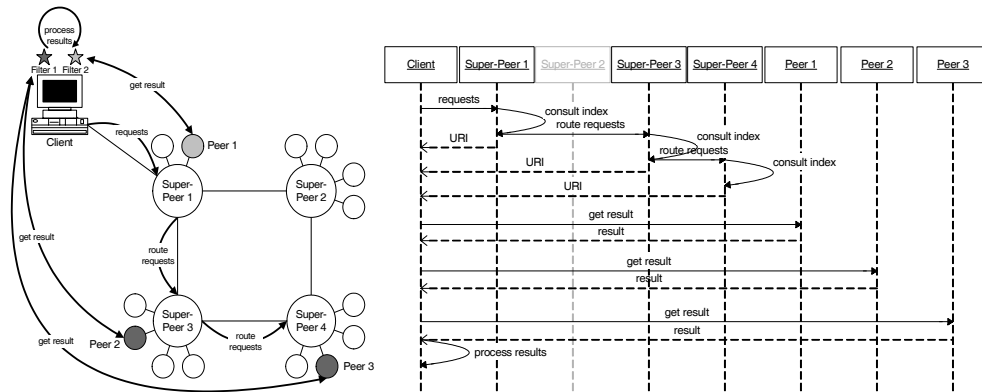


Fig. 5. Traditional Query Processing in Schema-Based P2P Networks (Routing Requests)

### 3.1 Pure P2P Query Processing: Flooding Requests

In pure P2P systems like Gnutella query processing takes place entirely at the client. Therefore, all required data has to be shipped to the client whereby the network is flooded with requests for resources. These requests are propagated to all neighbors through the network up to a particular horizon. Usually the majority of these peers host none of the desired information. On the other side, due to the horizon, some information is never discovered. The URIs of the results are returned to the client and another round-trip is necessary to obtain the data itself. The results of this initial data shipping phase are processed centrally at the client, i.e., only at the client the user-defined filtering (execution of special-purpose code) on the data can take place and possibly large volumes of data are shipped to the client.

### 3.2 Schema-Based P2P Query Processing: Routing Requests

In schema-based P2P networks a distributed index systematically guides the search to the appropriate (super-)peers. Whenever a new participant joins the P2P network the index is updated with the metadata of the resources by the new peers as described in the previous section. Figure 5 shows on the left hand side that the search of information usually involves only a small part (depending on the clustering) of the network. The right hand side of Figure 5 illustrates that the local indices are consulted to selectively propagate the search. The routing of requests constitutes an enormous improvement compared to flooding requests in pure P2P networks. All relevant information is found and must be shipped to the client to make the user-defined filters applicable. Search in schema-based P2P networks is much more efficient, as only the necessary peers are contacted and flooding the network with requests is avoided. Nevertheless, entire query processing still takes place only at the client and user-defined filters and complex operators can only be applied after the data has already been shipped to the client.

### 3.3 Extensible Distributed Query Processing: Pushing Code-Carrying QEPs

To enable dynamic, extensible, and distributed query processing in schema-based P2P networks, where both standard query operators and user-defined code can be executed

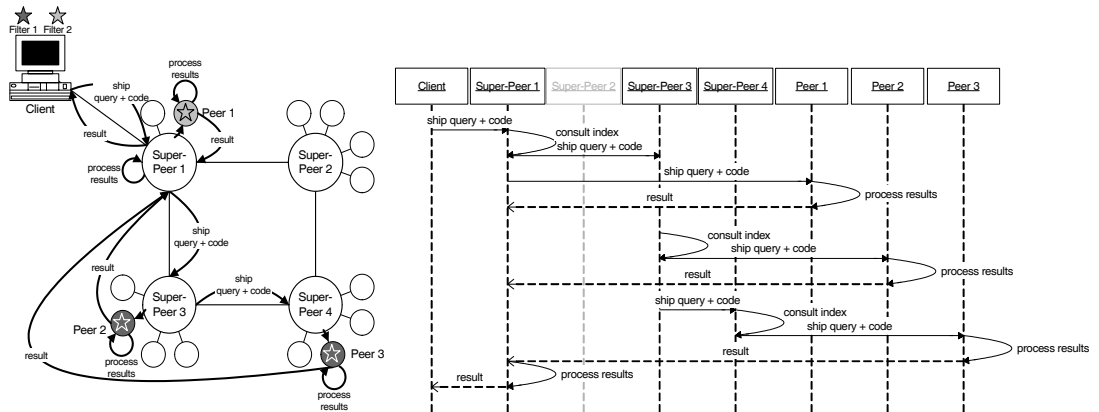


Fig. 6. Pushed Code-Carrying Query Evaluation Plans (QEPs)

nearby the data, we distribute query processing to the super-peers and peers. Therefore, super-peers provide functionality for the management of the index structures, query optimization, and query processing capabilities. Additionally, we expect that peers provide query processing capabilities to be a full member of the P2P network. These query processors can be dynamically extended by special-purpose query operators that are shipped to the query processor as part of the query plan. This way, query evaluation plans (QEPs) with user-defined code, e.g., selection predicates, compression functions, join predicates, etc., can be pushed from the client to the (super-)peers where they are executed. Furthermore, super-peers have to provide an optimizer for generating good query plans from the queries they receive. We utilize these distributed query processing capabilities at the super-peers and distribute the query stated by the user to the corresponding super-peers. This distribution process is guided by the index which is dynamic and corresponds to the data allocation schema in traditional distributed DBMSs. However, as the index is dynamic and dispersed, static query optimization is not possible. Thus, query optimization must also be dynamic and based on the allocation schema of the data known at the super-peer.

Figure 6 illustrates schema-based P2P networks with extensible distributed query processing capabilities. We assume for our example, that the (super-)peers install a fully functional optimizer and an extensible query processor as mentioned above. The left hand side of Figure 6 shows the architecture and the flow of messages in our approach where queries and code are pushed through the network. The client sends the query including user-defined operators to the first super-peer where the local indices are consulted and the query is split into two parts. The first part including Filter 1 is shipped to Peer 1, where the filter can be applied directly on the data before shipping the results to Super-Peer 1. The later part of the original query including Filter 2 is pushed to Super-Peer 3, where the same process is repeated. Again, the results are sent to Super-Peer 1, where they are processed further and finally returned to the client. The client just needs to display the results, and very limited resources suffice for clients. Less data shipping enables thin clients and even mobile devices, e.g., cellular telephones and PDAs, to query the P2P network. The right hand side of Figure 6 shows the sequence of index lookups, shipping of queries and code, and local query processing at the (super-)peers. The query plan is decentrally optimized, whereby each super-peer optimizes just the piece of the query it receives. The

```

select  $r_1$ .data,  $r_2$ .data,  $r_3$ .data
from Resources  $r_1$ , Resources  $r_2$ , Resources  $r_3$ 
where  $r_1$ .lom_type = "lecture" and  $r_1$ .dc_subject = "ccs:dbms" and  $r_1$ .dc_language = "de" and
 $r_1$ .dc_format = "application/pdf" and occur( $r_1$ .data, "transaction processing") >= 2 and
 $r_2$ .lom_type = "exercise" and  $r_2$ .dc_subject = "ccs:dbms" and  $r_2$ .dc_language = "de" and
 $r_3$ .lom_type = "exercise" and  $r_3$ .dc_subject = "ccs:dbms" and  $r_3$ .dc_language = "en" and
 $r_1$ .dc_author =  $r_2$ .dc_author and  $r_1$ .dc_author =  $r_3$ .dc_author

```

Fig. 7. SQL Formulation of the Example Query

remaining parts are pushed further. This way, user-defined code such as filter predicates on the data are pushed to the data sources.

### 3.4 Summary and Classification of P2P Networks

The following table classifies P2P networks summarizing the most important characteristics regarding query processing facilities and usage of index structures:

with extensible, distributed query processor	<b>Flooding Code-Carrying QEPs</b> <ul style="list-style-type: none"> <li>+ user-defined operators</li> <li>+ query processing at (super-)peers</li> <li>+ pushing QEPs &amp; code to the data</li> <li>+ low transfer volumes (only results)</li> <li>– inefficient search by flooding</li> <li>– many peers are queried</li> </ul>	<b>Pushing Code-Carrying QEPs</b> <ul style="list-style-type: none"> <li>+ user-defined operators</li> <li>+ query processing at (super-)peers</li> <li>+ pushing QEPs &amp; code to the data</li> <li>+ low transfer volumes (only results)</li> <li>+ efficient search by routing</li> <li>+ only necessary peers are queried</li> </ul>
	<b>Flooding Requests</b> <ul style="list-style-type: none"> <li>– fixed set of query operators</li> <li>– query processing at client</li> <li>– data shipping to client</li> <li>– huge transfer volumes</li> <li>– inefficient search by flooding</li> <li>– many peers are queried</li> </ul>	<b>Routing Requests</b> <ul style="list-style-type: none"> <li>– fixed set of query operators</li> <li>– query processing at client</li> <li>– data shipping to client</li> <li>– huge transfer volumes</li> <li>+ efficient search by routing</li> <li>+ only necessary peers are queried</li> </ul>
	without index	with index

## 4 Plan Generation and Distribution

In this section we describe the generation of a query evaluation plan at one super-peer using the allocation schema provided by the index structures. For illustration let us consider the following example query: “Retrieve the data of resources  $r_1$ ,  $r_2$ , and  $r_3$  where  $r_1$  is a lecture about *ccs:dbms*, and the language is *de*; furthermore, resource  $r_1$  should be a PDF file containing at least twice the phrase “transaction processing”;  $r_2$  is an exercise about *ccs:dbms*, the language is *de*;  $r_3$  is an exercise about *ccs:dbms*, the language is *en*; the resources should be written by the same *author*.” The corresponding SQL formulation of this query is shown in Figure 7. The query accesses “Resources”, which represents the collection of all resources registered in the P2P network. The attribute *data* represents all the data belonging to the registered resource, i.e., in our example the PDF file. The user-defined filter `occur( $r_1$ .data, 'transaction processing') >= 2` counts the number of appearances of the string in the PDF file. This operation has to be executed nearby the data sources to reduce the network traffic.

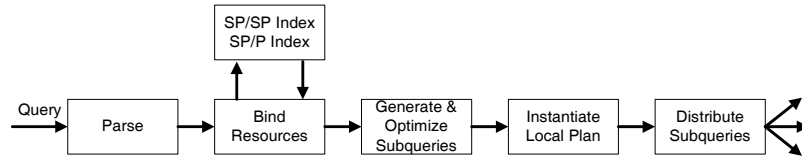


Fig. 8. Plan Generation at a Super-Peer

#### 4.1 Details of the Plan Generation and Distribution

In contrast to traditional distributed query optimization, the plan is not generated statically at one single host. In our approach, super-peers generate partial query plans which are executed locally and the remainders of the query are pushed to the neighbors. Thereby, plan generation involves five major steps as depicted in Figure 8:

*Parse* The received SQL query is parsed and transformed into an internal representation which is a decomposition of the query into its building blocks. The succeeding steps are prepared, i.e., properties, property-values, and user-defined operators are identified.

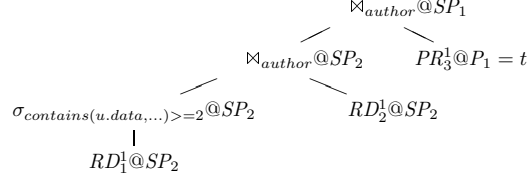
*Bind Resources* The local indices are consulted to determine the location of the required resources. For this purpose we introduce resource directions (*RD*), physical resources (*PR*), and logical resources (*LR*): Users specify the desired information by giving properties and property-values which restrict LRs. These LRs are bound to RDs, if a corresponding data source is found in the SP/P index. Using the SP/P index, LRs are bound to PRs, i.e., the URIs of registered resources. Binding the LRs to PRs and RDs, all levels of granularity of the indices have to be considered. In our example scenario we obtain the following bindings at  $SP_1$ :  $r_1 = RD_1^1 @ SP_2$ ,  $r_2 = RD_2^1 @ SP_2$ , and  $r_3 = PR_3^1 @ P_1 = t$ , where, e.g.,  $RD_2^1 @ SP_2$  denotes the first resource direction for the logical resource  $r_2$  and references super-peer  $SP_2$ . Note, that multiple RDs and PRs can contribute data for the same LR.

*Generate & Optimize Subqueries* Based on the bindings, a local query plan is generated. For the remaining parts subqueries are generated. As super-peers have a very limited view of the whole P2P network (only the neighbors are known), it is obvious that no comprehensive static plan in the traditional sense can be produced. Furthermore, we determine which subplans are executed at the neighboring (super-)peers.

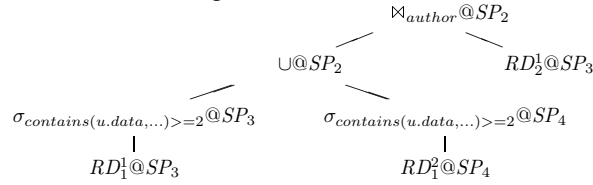
As only partial information is available to the query optimizer this is a non-standard query optimization problem where only a part of the query plan is generated. The remainder of the query which could not be executed locally is identified and grouped by host. These remaining parts constitute the inputs to the local plan. To perform cost based optimization, the optimizer uses statistics of the input data, the network topology, and the hosts. When LRs are bound at least the number of referenced resources should be provided by the index structures. This forms the basis of the cost estimation. Furthermore, the optimizer may learn response times, transfer rates, and even result sizes from previous query executions whereby the techniques presented by [12] can be adopted for P2P query processing to obtain fine-grained and up-to-date statistics.

During plan generation, each query operator is annotated with the host where it is executed. This is done bottom up from the leaves of the operator tree, which constitute

PRs and RDs. The annotations of the leaves are given by the binding phase. Now, an operator can be executed on a host  $H$ , if all its input operators are executed at  $H$ , too. For instance, the following query plan could be generated at  $SP_1$ :



The left-hand subtree can be executed at  $SP_2$ , a subquery is pushed to  $SP_2$  which will be optimized analogously (based on the bindings  $r_1 = RD_1^1@SP_3 \cup RD_1^2@SP_4$  and  $r_2 = RD_2^1@SP_3$ ). The following query plan could be generated at  $SP_2$  for the subquery, where again the individual subtrees can be pushed to  $SP_3$  and  $SP_4$ :



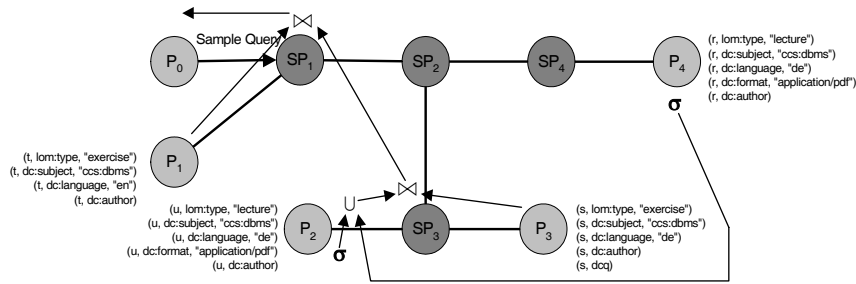
*Instantiate Local Plan* The local query plan is instantiated at the super-peer, all user-defined code is loaded and the communication path to the super-peer which uses this part of the query plan as input is established. The execution of the local query plan is not started until the distributed subqueries have established their communication paths. When the subqueries are instantiated the plan is executed following the iterator model [10].

*Distribute Subqueries* The remaining subqueries are distributed to the corresponding super-peers, where they are processed further.

## 4.2 Optimization Strategies

As shown above, several PRs and RDs can contribute data for the same LR. The simplest way for incorporating the data for such an LR would be to union all the affected physical resources before any other operation is considered for that LR. This naive strategy would limit the query optimization, however, and possibly better plans might not be considered. Thus, several alternatives for the naive query plan must be considered by applying equivalence transformations. Unfortunately, the number of plans which has to be considered during query optimization when all possible equivalence transformations should be taken into account, is rather large. The naive strategy is acceptable, if the bound resources are spread widely over multiple hosts. To increase the degree of distribution, this query plan can be transformed using an equivalence transformation which turns the join of unions into a union of joins, e.g.,  $(R_1 \cup R_2) \bowtie S = (R_1 \bowtie S) \cup (R_2 \bowtie S)$ . The joins may then be distributed to the neighboring super-peers. This plan may have a huge number of subqueries, however, which may not be efficient.

It is also possible to collect as many bindings of one LR as possible at one host. Thereby one host is informed to collect all data and the other hosts send all data to the collecting host. This designated collecting host may change during the plan generation. Figure 9 shows the mapping of the query plan onto the network, in which this strategy is used. In this plan all resources for  $r_1$  are collected at first at  $P_2$  (where the union is executed), then the join of resources  $r_1$  with  $r_2$  is done at  $SP_3$ .



**Fig. 9.** Query Plan Mapped onto the P2P Network  
(The Arrows Indicate the Flow of Results)

## 5 Implementation

### 5.1 The QueryFlow System as a Basis for P2P Query Processing

One of our platforms for implementing the ideas described above is the QueryFlow system ([17, 18]). As the QueryFlow system is based on ObjectGlobe, we first give an overview of ObjectGlobe ([3]). The idea of ObjectGlobe is to create an open market place for three kinds of suppliers: *data-providers* supply data, *function-providers* offer query operators to process data, and *cycle-providers* are contracted to execute query operators. A single site (even a single machine) may comprise all three services. ObjectGlobe enables applications to execute complex queries which involve the execution of operators from multiple function providers at different sites (cycle providers) and the retrieval of data and documents from multiple data sources.

The system is written in Java, as are user-defined query operators which are loaded on demand and executed at the cycle provider in their own Java sandbox. User-defined query operators, e.g., filters or joins using complex predicates, must implement the `open`, `next`, `close`, and `reopen` methods following the iterator model of [10].

The QueryFlow system extends the idea of dynamic query execution by introducing incomplete query plans. Hyperlinks reference single query plans (HyperQueries), which are embedded as virtual attributes into the database of one host. The HyperQueries reside on hosts in the Internet and are accessed dynamically. Whenever a virtual attribute is accessed during execution, the referenced HyperQuery is executed at the remote host and the result of the execution is returned to the caller. As the QueryFlow system is based on ObjectGlobe, arbitrary pieces of code can be executed as a HyperQuery. Furthermore, the execution can recursively continue such that more hosts are involved in the execution. This way, queries and data objects flow through the Internet. In our prototypical implementation, we assume that each super-peer is a fully functional cycle provider, i.e., HyperQueries and operations such as joins, selection, projections, and user-defined operators can be executed by them.

Code-carrying query plans do not really transmit the Java code of the operators but the query plan consisting of query operators is annotated with information which indicates the function-provider the user-defined operator is loaded from. A class loader loads the bytecode of the operator on demand into memory, whereby access to safety critical system resources is controlled by Java's security manager. The security manager is used to create a so-called *sandbox* in which untrusted code is safely executed.

## 5.2 The Edutella P2P Infrastructure

The Open Source Edutella project [7, 22, 24] has the goal to design and implement a schema-based P2P infrastructure for the Semantic Web. Edutella relies on the W3C meta-data standards RDF and RDF Schema (RDFS) [19, 4] to describe distributed resources. It connects heterogeneous RDF (and also XML) repositories describing the resources available in the network. Provider peers integrate heterogeneous data sources into the network, by allowing local translation (wrapping) from the common data and query model (ECDM) [22], to backend databases using either memory-based RDF models, database systems using SQL, or systems based on logic (e.g., Prolog). The ECDM and the corresponding query exchange language RDF-QEL is based on Datalog semantics and is represented in RDF. Edutella is written in Java, and uses the JXTA architecture [9] developed by Sun for basic P2P functionality, like initial peer discovery, network groups and pipe-based communication between peers.

The Edutella super-peers [23] include schema information as well as other possible index information in the corresponding routing indices. The current implementation uses the routing indices to select (super-)peers who can answer a given query using the index information and the query characteristics. We are currently implementing distributed query processing and query plans as discussed in Section 3.

## 6 Conclusions and Future Work

In this paper we have discussed additional challenges for P2P data management regarding query routing and query planning, based on the specific characteristics of schema-based P2P systems, which make straightforward adoptions of distributed database techniques impossible. We have discussed an innovative query routing and planning architecture based on distributed routing indices, which allows us to place query operators next to data sources and utilize distributed computing resources more effectively. In the future we try to merge both systems by processing code-carrying query plans in Edutella and supporting more schema-based features by the QueryFlow system.

For further optimization, we want to investigate the inclusion of additional statistical information in the indices, e.g., average response time, amount of registered data regarding schematas, properties, etc. Furthermore, top-down query optimization seems to be an interesting strategy especially as query optimization can be interrupted at any time and a query plan is provided. At last, studying other strategies for the plan generation and a more dynamic placement of operators would offer new possibilities.

## References

1. K. Aberer and M. Hauswirth. Semantic gossiping. In *Database and Information Systems Research for Semantic Web and Enterprises, Invitational Workshop*, 2002.
2. P. A. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zahirayeu. Data management for peer-to-peer computing: A vision. In *Proc. of the 5th Intl. Workshop on the Web and Databases*, 2002.
3. R. Braumandl, M. Keidl, A. Kemper, D. Kossmann, A. Kreutz, S. Seltzsam, and K. Stocker. ObjectGlobe: Ubiquitous query processing on the Internet. *The VLDB Journal: Special Issue on E-Services*, 10(3), 2001.
4. D. Brickley and R. V. Guha. RDF vocabulary description language 1.0: RDF Schema, 2003. <http://www.w3.org/TR/rdf-schema/>.

5. A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *Proc. Intl. Conf. on Distributed Computing Systems*, 2002.
6. Dublin core metadata initiative.
7. The Edutella Project. <http://edutella.jxta.org/>, 2002.
8. J. Frankel. Gnutella. [www.gnutella.com](http://www.gnutella.com), March 1999. Information portal with community, development information and downloads.
9. L. Gong. Project JXTA: A technology overview. Technical report, SUN Microsystems, 2001. <http://www.jxta.org/project/www/docs/TechOverview.pdf>.
10. G. Graefe. Query Evaluation Techniques for Large Databases. *ACM Computing Surveys*, 25(2), 1993.
11. S. Gribble, A. Y. Halevy, Z. G. Ives, M. Rodrig, and D. Suciu. What can databases do for peer-to-peer. In *Proc. of the 4th Intl. Workshop on the Web and Databases.*, 2001.
12. J. R. Gruser, L. Raschid, V. Zadorozhny, and T. Zhan. Learning response time for websources using query feedback and application in query optimization. *The VLDB Journal*, 9(1), 2000.
13. A. Y. Halevy, Z. G. Ives, P. Mork, and I. Tatarinov. Piazza: Data management infrastructure for semantic web applications. In *Proc. of the 12th Intl. World Wide Web Conf.*, 2003.
14. H.Dhraief. Peer Registration RDF Document. <http://www.kbs.uni-hannover.de/~hdhraief/edutella/>.
15. N. Hemming. KaZaA. [www.kazaa.com](http://www.kazaa.com).
16. V. Josifovski, P. Schwarz, L. Haas, and E. Lin. Garlic: A New Flavor of Federated Query Processing for DB2. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, 2002.
17. A. Kemper and C. Wiesner. HyperQueries: Dynamic Distributed Query Processing on the Internet. In *Proc. of the Conf. on Very Large Data Bases*, 2001.
18. A. Kemper, C. Wiesner, and P. Winklhofer. Building dynamic market places using hyperqueries. In *Proc. of the Intl. Conf. on Extending Database Technology*, 2002.
19. O. Lassila and R.R. Swick. W3C Resource Description Framework model and syntax specification, 1999. <http://www.w3.org/TR/REC-rdf-syntax/>.
20. A. Y. Levy, D. Srivastava, and T. Kirk. Data Model and Query Evaluation in Global Information Systems. *Journal of Intelligent Information Systems*, 5(2), 1995.
21. IEEE Learning Technology Standards Committee, IEEE P1484.12 Learning Object Metadata Working Group.
22. W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmr, and T. Risch. EDUTELLA: a P2P Networking Infrastructure based on RDF. In *Proc. of the 11th Intl. World Wide Web Conf.*, 2002.
23. W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M. Schlosser, I. Brunkhorst, and A. Loser. Super-peer-based routing and clustering strategies for RDF-based peer-to-peer networks. In *Proc. of the Intl. World Wide Web Conf.*, 2003.
24. W. Nejdl, B. Wolf, S. Staab, and J. Tane. Edutella: Searching and annotating resources within an RDF-based P2P network. In *Proc. of the Semantic Web Workshop, 11th Intl. World Wide Web Conf.*, 2002.
25. V. Papadimos and D. Maier. Distributed Query Processing and Catalogs for Peer-to-Peer Systems. 2003.
26. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *Proc. of the 2001 Conf. on applications, technologies, architectures, and protocols for computer communications*, 2001.
27. M. Schlosser, M. Sintek, S. Decker, and W. Nejdl. HyperCuP—Hypercubes, Ontologies and Efficient Search on P2P Networks. In *Intl. Workshop on Agents and P2P Computing*, 2002.
28. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of the 2001 Conf. on applications, technologies, architectures, and protocols for computer communications*, 2001.
29. B. Yang and H. Garcia-Molina. Improving search in peer-to-peer systems. In *Proc. of the 22nd Intl. Conf. on Distributed Computing Systems*, 2002.