Processing and Optimization of Complex Queries in Schema-Based P2P-Networks

Hadhami Dhraief[2], Alfons Kemper[3], Wolfgang Nejdl[1,2], and Christian Wiesner[3]

Abstract. Peer-to-Peer infrastructures are emerging as one of the important data management infrastructures in the World Wide Web. So far, however, most work has focused on simple P2P networks which tackle efficient query distribution to a large set of peers but assume that each query can be answered completely at each peer. For queries which need data from more than one peer to be executed this is clearly insufficient. Unfortunately, though quite a few database techniques can be re-used in the P2P context, a P2P data management infrastructure poses additional challenges caused by the dynamic nature of these networks. In 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 have no complete schema and distribution instance available but rather work with distributed schema information which can only direct query processing tasks from one node to one or more neighboring nodes.
In this paper we first describe briefly the super-peer based topology and the "schema-aware distributed routing indices extended with statistics and describe how this statistics are extracted and updated. Then we show how these indices facilitate the distribution and dynamic expansion of query plans. Finally we propose transformation rules to optimize query plans and discuss different optimization strategies. Our techniques enable distributed query processing in a schema-based P2P network.

1 Introduction and Motivation

P2P computing provides a very efficient way of storing and accessing distributed resources, as shown by the success of music file sharing networks such as Gnutella, where we can use simple attributes to describe the resources. A lot of effort has been put into refining topologies and query routing functionalities of these networks. A new breed of P2P applications inspired from earlier systems like Napster and Gnutella have more efficient infrastructures such as the ones based on distributed hash tables. Less effort has been put into extending the representation and query functionalities offered by such networks. Projects exploring more expressive P2P infrastructures [22, 2, 1, 13] have only slowly started the move toward schema-based P2P networks.

In the Edutella project [10, 22] we have been exploring some issues arising in that context, in order to design and implement a schema-based P2P infrastructure for the

[1] L3S Research Center, University of Hannover, Germany, nejdl@l3s.de
[2] Information Systems Institute, University of Hannover, Germany, {hdhraief,nejdl}@kbs.uni-hannover.de
[3] Computer Science Department, University of Passau, Germany, {wiesner,kemper}@db.fmi.uni-passau.de

Semantic Web. Edutella relies on the W3C metadata standards RDF and RDF Schema (RDFS) to describe distributed resources, and uses basic P2P primitives provided as part of the JXTA framework [12]. In the ObjectGlobe project [5, 19] we have designed and implemented a distributed data network consisting of three kinds of suppliers: *data-providers*, which supply data, *function-providers*, that offer query operators to process data, and *cycle-providers*, which 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. Thus, both systems, Edutella and ObjectGlobe, deal with complex queries in a highly dynamic, distributed, and open environment. We rise actually the challenge to optimize the distributed query processing in schema-based super-peer networks.

Although distributed query optimization and execution are well known issues investigated in database research, distributed query processing in schema-based P2P networks 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 [21] central mapping information of all participating, can distributed data sources be queried. [25] introduces so called mutant query plans, which encapsulate partially evaluated query plans and data. Loss of pipelining during execution limits the general applicability for distributed query processing, and no user-defined operators are supported. AmbientDB [3] executes SQL queries over a P2P network. That approach is based on distributed hash tables and does not take into account user-defined operators.

A recent work of Stuckenschmidt [31] exploits schema paths for optimizing queries on distributed RDF repositories. That approach constructs the overall query plan in a mediator-like manner and uses replicated schema paths (which serve as global allocation schema of the data) to determine which portions of the query plan can be pushed to the data sources. The paper does not handle the case that individual portions of the pushed query plan can be further distributed. In a highly distributed environment like a P2P network it seems not realistic to assume global knowledge of the allocation schema. Especially, the update behavior of the join indices could be a problem, e.g., assuming new data sources with new RDF properties joining the network leads to an enormous grow of all join indices and huge transfer costs. The paper takes into account neither load balancing strategies during query plan generation, nor mechanisms for the dynamic placement of operators. The whole query processing facilities are limited to joins and selections. User-defined operators are not considered as the possibility that multiple resources contribute data to the same property, i.e., a union operator has to be placed in the query plan. This leads to an enormous explosion of the search space.

To enable dynamic, extensible, and distributed query processing in schema-based P2P networks, where we need to evaluate complex queries requiring data from several peers and where both standard query operators and user-defined code can be executed nearby the data, we have to distribute query processing to the (super-)peers. Since each peer in a P2P network usually has varying resources available, e.g., regarding bandwidth or processing power, exploiting the different capabilities in a P2P network can lead to an efficient network architecture, where a small subset of peers, called super-peers [34], takes over specific responsibilities for peer aggregation, query routing, and mediation.

Therefore, super-peers have to provide on the one hand query processing capabilities, and on the other hand functionality for the management of index structures and for query optimization. Super-peer based P2P infrastructures are usually based on a two-phase routing architecture, which first routes queries in the super-peer backbone, and then distributes them to the peers connected to the super-peers. Our routing mechanism is based on two distributed routing indices storing information to route within the super-peer backbone and between super-peers and their respective peers [24]. The query processors at the super-peers can be dynamically extended by special-purpose query operators that are shipped to the query processor as part of the query plan. In this way, query evaluation plans (QEPs for short) 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 dynamically generating good query plans from the queries they receive. We utilize these distributed query processing capabilities at the super-peers and distribute the user's query to the corresponding super-peers. This distribution process is guided by the (dynamic) distributed routing indices, which correspond to the (static) data allocation schema in traditional distributed DBMSs. However, as the index is dynamic and itself distributed over the super-peers, static query optimization as used in distributed DBMSs is not possible. Query optimization must be therefore be dynamic and based on the data allocation schema known at each super-peer.

This paper is based on the framework presented in [8] and reveals details of query optimization in P2P networks. First, we describe briefly the super-peer based topology and the "schema-aware distributed routing indices enriched with statistics. Second, we describe how these statistics are extracted and updated. In section 3, we describe how these indices facilitate the distribution and dynamic expansion of query plans. Then, we propose transformation rules to optimize query plans and discuss different optimization strategies. Finally, we conclude with a short overview of the implemented systems and our future work.

2 Distributed Routing Indices

Efficient query routing is one of the corner stones of advanced P2P systems. We rely on a super-peer topology with "schema-aware" routing indices.

The HyperCuP Topology Super-peers are arranged in the HyperCuP topology. The HyperCuP algorithm as described in [27] is capable of organizing super-peers of a P2P network into a recursive graph structure called a hypercube that stems from the family of Cayley graphs. Super-peers join the HyperCuP 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. The basic HyperCuP topology 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. Peers connect to the super-peers in a star-like fashion. Figure 1 shows an example super-peer based P2P network.

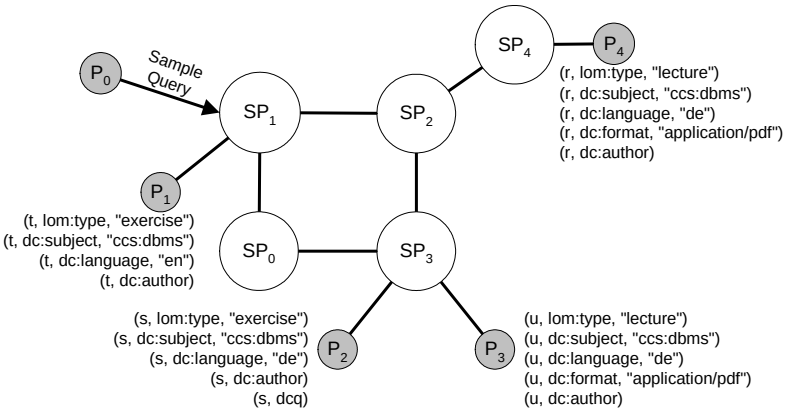


Fig. 1. Routing Example Network

Routing Indices Our super-peers [24] 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. The indices are local in the sense that all index entries only refer to direct neighbors (peers and super-peers). Network connections among the super-peers form the super-peer backbone that is responsible for message routing and integration/mediation of metadata.

Our super-peer network implements a routing mechanism based on two indices storing information to route within the P2P backbone and between super-peers and their respective peers. The super-peer/peer routing indices (SP/P indices) contain information about each peer connected to the super-peer, including schema and attribute information from the peers. On registration the peer provides this information to its super-peer. In contrast to other approaches (Gnutella, CAN [26]), our indices do not refer to individual content elements but to peers (as in CHORD [30]). The indices can contain information about peers at different granularities: schemas, schema properties, property value ranges and individual property values. Details are described in [24]. Using indices with different granularities enables us to state queries at different levels of accuracy. In order to avoid backbone we use 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 local SP/P indices maintained in the super-peers. 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. Queries are forwarded to super-peer neighbors based on the SP/SP indices, and sent to connected peers based on the SP/P indices.

Statistics in the Routing Indices The routing indices as described so far enable the efficient routing of the queries. Nevertheless, additional information (statistics, physical parameters of the network, etc.) both in the SP/P and the SP/SP routing indices are necessary to enhance the optimization process and enable the choice of the best query execution plan. As mentioned in the introduction we aim at using approved techniques and methods in databases particularly for distributed database systems. The most important parameters for query optimization within this context are number and size of the

stored documents at the different peers. This information is provided by the peers during the registration process. The following piece of the RDF-Schema *P*eerDescription shows the definition of the property elementCount, used for the documents count at a given peer at the property-value level.

```
(...)

<rdf:Property rdf:ID="elementCount">
<rdfs:isDefinedBy rdf:resource="http://www.learninglab.de/~brunkhor/rdf
/PeerDescription#"/>
<rdfs:label>elementCount</rdfs:label>
<rdfs:comment>An integer that specifies how often an element has occured.
Used in conjunction with hasPropertyValues.</rdfs:comment>
<rdfs:range rdf:resource="http://www.w3c.org/2000/01/rdf_schema#Literal"/>
<rdfs:domain rdf:resource="#Peer"/>
</rdf:Property/>

(...)
```

If we register documents only at the property value level, we can derive the information for the property level by accumulating the number and size of documents for each property. Multi-valued properties like dc:author complicate this aggregation. Histograms [15] can help to obtain more precise estimates. For this paper, we assume that the registration occurs at property level, property value level, and property value range level. The schema level can be considered as meta-level, which can be used to answer general queries (e.g. ”Which standards are used to annotate documents at Peer x?”). Thus, the information about the number and size of documents are not relevant at this level. Table 1 states the SP/P routing index of super-peer SP_1 including statistics at different granularities. In the following we will restrict the discussion on the size (si) and the number (n) of the documents. However, it is absolutely conceivable and feasible to add any useful further statistics such as the minimum, maximum, and average values and the total number of documents at each peer. If a peer P_y (re-)registers or leaves a given super-peer SP_x with a schema element set including document statistics $S_y(s_1(n_1, si_1), \ldots, s_m(n_m, si_m))$, an update of the SP/P and the SP/SP indices is needed. The algorithm for building and updating the SP/P routing indices described before remains unmodified. The peers simply register including their statistics information in addition to the schema elements. The update information of the SP/SP indices propagated via messages however must be extended as follows:

1. SP_x derives the total number and size of the documents (potentially further statistics) registered by the peers for each schema element $s_i \in S_y$ and sends these statistics combined with s_i to its neighbors in its spanning tree.
2. Any other super-peer in the spanning tree of SP_x updates its SP/SP index and derives the total number and size of the documents in its SP/SP index at each $s_i \in S_y$ and forwards the data to its neighbors.

3 Plan Generation, Distribution and Optimization

Using the indices described in the previous section we can now formulate how query plan generation and distribution works in our P2P network.

Granularity	**SP/P Index of SP_1**		
Schema	dc		P_1, P_0
	lom		P_1, SP_3
Property	dc:subject		P_1 [13], P_0 [16]
	dc:language		P_1 [15]
	lom:type		P_1 [10]
Property Value	lom:type	"exercise"	P_1 [10]
	dc:language	"de"	P_0 [15]

Table 1. SP/P Index of SP_1 at Different Granularities

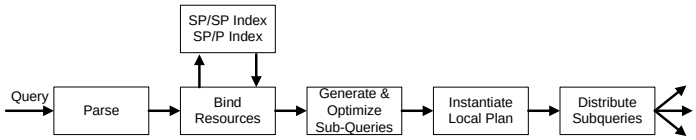


Fig. 2. Plan Generation at a Super-Peer

3.1 Distributed Plan Generation

In contrast to traditional distributed query optimization approaches, we cannot generate the query plan statically at one single host. Therefore we have to generate an abstract query plan at a super-peer which is partially executed locally and where we push other parts of the query plan to its neighbors. The plan generation at each super-peer therefore involves five major steps as depicted in Figure 2 and is described in details in [7].

At first the received query (e.g. in SQL) is parsed and transformed into an internal representation which is a decomposition of the query into its building blocks. Then, the local indices are consulted to de- termine the location of the required resources. For this purpose we have to distinguish between resource directions (RDs) and physical resources (PRs). Users specify the desired information by giving properties and property-values restricting LRs. These LRs are bound to RDs resp. PRs where all levels of granularity of the indices have to be considered. Thereby, multiple RDs and PRs can contribute data for the same LR. Based on the bindings, a local query plan is 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. Therefore, we determine which sub-plans are delegated to the neighboring (super-)peers. These remaining parts constitute the input to the local plan. To perform cost based optimization, the optimizer uses statistics of the input data, the network topology, and the hosts. Furthermore, the optimizer may collect and use response times, transfer rates, and even result sizes from previous query executions. Finally, the local query plan is instantiated at the super-peer, all user-defined code is loaded, the communication path to the super-peer which uses this part of the query plan as input is established, and the remaining sub-queries are distributed to the corresponding super-peers, where they are processed further.

3.2 Query Optimization

Let us now describe some of the details involved in the optimization process at a super-peer. We employ a transformation-based optimizer starting with an initial query plan. The optimizer applies equivalence transformations and determines the cost of the generated alternatives using a cost model. In contrast to bottom-up approaches employed in traditional dynamic programming based optimization we can stop at any time with a complete and valid query plan. In our implementation we use iterative improvement to enumerate plan alternatives. Superior techniques as shown in [28] are applicable.

In the following we present the set of the most important transformation rules, focusing on the ones relevant to processing joins and unions within the P2P context. Further rules can be added easily. Furthermore we extend conventional cost models taking the special requirements of P2P query processing into account. During the optimization process we employ heuristics that favor query plans with few sub-plans as this leads to more robust distributed query execution. A huge number of wide spread sub-plans accessing the same documents would be more error-prone and often inefficient to execute. Our decision also implies, that less messages are exchanged between the (super-)peers and less data is transferred.

The Initial Query Plan The initial (canonical) query plan accesses only logical resources and is constructed in the following way: Use all join predicates and join the logical resources. If logical resources could not be joined due to a lack of join predicates, the Cartesian product includes them into the query plan. Thereafter, all remaining selection predicates and user-defined filters are applied on top of the query plan. Finally, the result is submitted to the client.

The Transformation Rules The initial query plan is optimized top-down using a transformation-based optimizer. In such an approach we apply a set of transformation rules to the query plan and generate alternatives, which are then ranked using our cost model. The best (local) query plan is then executed. Transformation rules are represented as

$$\frac{\{\text{inputQEP}\} \quad [\text{condition/action}]}{\{\text{outputQEP}\}}$$

where one input query plan is transformed into an output query plan. The condition/action part may be omitted. We assume that the transformations are executed at host H_L. If H_L is a super-peer, we have access to the local routing indices *SPP* and *SPSP*.

Basic Transformation Rules We can express the *Bind Resources* step explained in the previous subsection as the following *Binding Transformation*:

$$\frac{\{LR\} \quad \left[PR_j@P_j \in \textit{match}(\textit{SPP}),\, RD_k@SP_k \in \textit{match}(\textit{SPSP})\right]}{\left\{\bigcup_j PR_j@P_j \cup \bigcup_k RD_k@SP_k\right\}}$$

The function *match* consults the local indices and determines the location of the match-

ing resources. The LRs are bound to RDs, if a corresponding data source is found in the SP/SP index. Using the SP/P index, LRs are bound to PRs, i.e., the URIs of registered resources. Multiple RDs and PRs can contribute data for the same LR. This is expressed by the union of RDs and PRs. $PR_j@P_j$ denotes that the j-th bound PR belongs to LR and references a resource at peer P_j. A similar argument applies for the RDs.

Applying the following two transformations to a query plan pushes selections and user-defined filters down towards the data sources. This enables us to reduce the amount of transferred data early.

$$\frac{\{\sigma(A\ op\ B)\}}{\{\sigma(A)\ op\ \sigma(B)\}} \qquad \frac{\{\sigma(op(A))\}}{\{op(\sigma(A))\}}$$

A and B are arbitrary sub-plans.

The next two rules apply the associative and commutative laws to unions, joins, and Cartesian products.

$$\frac{\{(A\ op\ B)\ op\ C\}\ \ [op \in \{\cup, \bowtie, \times\}]}{\{A\ op\ (B\ op\ C)\}} \qquad \frac{\{A\ op\ B\}\ \ [op \in \{\cup, \bowtie, \times\}]}{\{B\ op\ A\}}$$

where A, B, and C are arbitrary sub-plans.

Finally, each operator is annotated with the host where it is to be 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 first transformation rule. Then, an operator can be executed on host H_L, if all its inputs are computed at H_L, too.

$$\frac{\{A@H_1\ op\ B@H_2\}\ \ [H_1 \neq H_2]}{\{A@H_1\ op@H_L\ B@H_2\}} \quad \frac{\{A@H_1\ op\ B@H_2\}\ \ [H_1 = H_2]}{\{A@H_1\ op@H_1\ B@H_1\}} \quad \frac{\{op(A@H_1)\}}{\{op@H_1(A@H_1)\}}$$

A and B are sub-plans and $op@H_1$ indicates that the operator op is executed at host H_1. This rule enables us to execute mobile code at remote hosts, e.g., to push selective filter predicates, complex join predicates, or compression functions to the data sources.

The plans generated by the rules so far typically have one union operator for each logical resource. The degree of parallelism can be increased and distributed computing resources can be utilized better if operators are distributed over the P2P network.

Optimization Strategy: Union of Joins As shown above, several PRs and RDs can contribute data for the same LRs. The simplest way for incorporating the data for such an LR would be to union all the accessed physical resources before any other operation is considered for that LR. This would be done by the binding transformation. This naive strategy would produce good plans in some cases, but query optimization would be limited and possibly better plans might never be considered. Thus, several alternatives for the naive query plan must be considered by applying equivalence transformations. To increase the degree of distribution, the query plan can be transformed using the following transformation which turns the join of unions into a union of joins:

$$\frac{\{(A_1 \cup \ldots \cup A_n) \bowtie (B_1 \cup \ldots \cup B_m)\}}{\{(A_1 \bowtie (B_1 \cup \ldots \cup B_m)) \cup \ldots \cup (A_n \bowtie (B_1 \cup \ldots \cup B_m))\}}$$

If many RDs and PRs are bound to LRs and when this rule is applied recursively in combination with the associative and commutative laws the number of plans which have to be considered during query optimization is huge. [4] has derived a lower bound for the number of alternatives when joining two LRs, consisting of n_1 and n_2 bound resources:

configuration	number of plans
$UJ([2,2])$	8
$UJ([3,3])$	385
$UJ([4,4])$	144705
$UJ([5,5])$	913749304

Table 2. Explosion of the Search Space

$$\text{UJ}(n_1, n_2) = \sum_{j=1}^{n_1} \left(\left\{ \begin{matrix} n_1 \\ j \end{matrix} \right\} \text{bell}(n_2)^j \right) + \sum_{j=1}^{n_2} \left(\left\{ \begin{matrix} n_2 \\ j \end{matrix} \right\} \text{bell}(n_1)^j \right) - \text{bell}(n_1)\text{bell}(n_2)$$

In this definition $\left\{ \begin{matrix} m \\ k \end{matrix} \right\}$ denotes the Stirling number of the second kind which represents the number of ways a set with m elements can be partitioned into k disjoint, non-empty subsets. The term bell(m) denotes the Bell number which represents the number of ways a set with n elements can be partitioned into disjoint, non-empty subsets. The definition of *UJ* follows the construction of a query plan starting from its canonical form. First we have to select a LR constituting of different bindings. Each such binding has to be joined with an expression which is equivalent to the other LR. All these expressions are counted by the call to the function for the Bell numbers. At the end we have to consider duplicate QEPs which are generated when for every appearance of a LR in a QEP the same partitioning is selected. If the same partitionings are selected, the order in which the LRs are used in the construction of a QEP does not matter anymore. Therefore, the last term of the definition of *UJ* includes the number of QEPs with that property. Table 2 gives an impression of the search space explosion. generated plan may have a huge number of sub-queries.

Optimizing by Collecting Resources A very promising heuristics in a distributed environment is to collect as many bindings of one LR as possible at one host. To implement this strategy, the optimizer determines one "collecting host" to collect all data of one logical resource. The other hosts are informed to send all data to the collecting host (in the following this is done by the CollectSend Operator). In contrast to the canonical query plan this collecting host is determined dynamically and may change during query execution, i.e., we can place the resource-collecting union at an arbitrary (super-)peer. In particular, in well clustered networks it is useful to place the collecting union operator nearby the majority of the data and to ship only a few resources.

To include this strategy in our query optimization, we introduce Collect Resources (CRs) which can be used in the previous rules like bound resources. Additionally, we propose the following five transformation rules (shown in Figure 3):

– First, the collecting host H_C is selected from the set of all referenced neighbors (taken from the PRs and RDs) (Figure 3(a)). Then, we replace all bound resources, i.e., PRs and RDs, of the input plan with a collect resource which is executed at H_C and CollectSend operators are pushed to the other neighbors. These CollectSend operators ship all data of the LR to the collecting host H_C.

$$\frac{\left\{\bigcup_j PR_j@P_j \cup \bigcup_k RD_k@SP_k\right\} \quad \left[H_C \in \bigcup_j P_j \cup \bigcup_k SP_k\right]}{\left\{CR(LR)@H_C \cup \bigcup_j^{H_C \neq H_j} CollectSend(H_C, LR)@P_j \cup \bigcup_k^{H_C \neq H_k} CollectSend(H_C, LR)@SP_k\right\}}$$

(a) Collecting Host Selection

$$\frac{\left\{CollectSend(H_C, LR)\right\} \quad \begin{bmatrix} PR_j@P_j \in \textit{match}(\textit{SPP}), \\ RD_k@SP_k \in \textit{match}(\textit{SPSP}) \end{bmatrix}}{\left\{CollectSend(H_C, LR)@P_j, \ldots, CollectSend(H_C, LR)@SP_k\right\}}$$

(b) Propagate CollectSend

$$\frac{\left\{CollectSend(H_C, LR)\right\} \quad \begin{bmatrix} PR_j@P_j \in \textit{match}(\textit{SPP}), \\ RD_k@SP_k \in \textit{match}(\textit{SPSP}) \end{bmatrix}}{\left\{Send(H_C, \bigcup_j PR^j@P_j \cup \bigcup_k RD_k@SP_k)@H_L\right\}}$$

(c) Execute CollectSend

$$\frac{\left\{CR(LR)\right\} \quad \begin{bmatrix} PR_j@P_j \in \textit{match}(\textit{SPP}), \\ RD_k@SP_k \in \textit{match}(\textit{SPSP}) \end{bmatrix}}{\left\{Receive@H_L \cup \bigcup_j PR_j@P_j \cup \bigcup_k RD_k@SP_k\right\}}$$

(d) Execute Collect Resource At Host

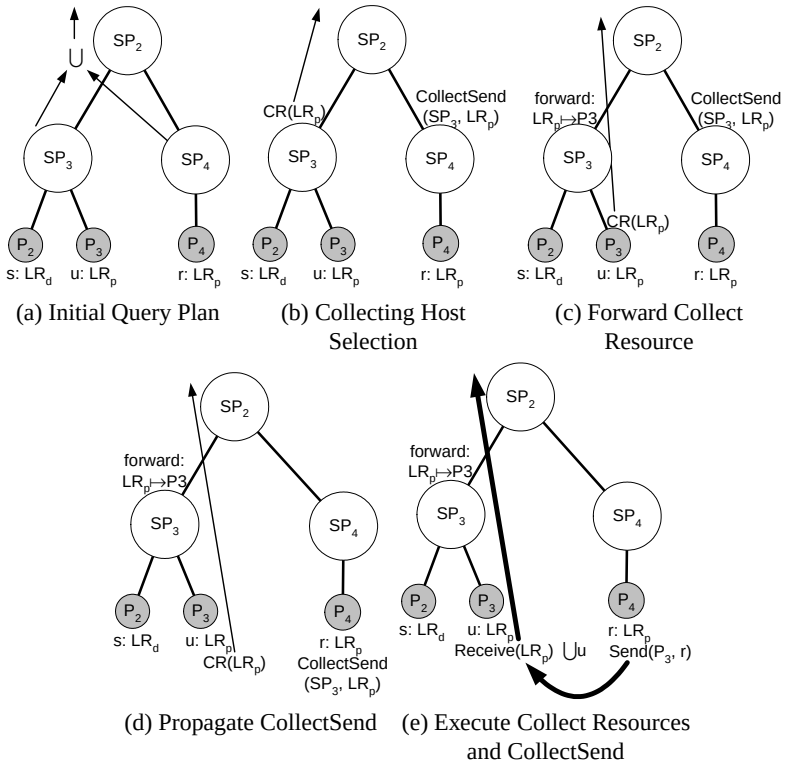
$$\frac{\left\{CR(LR)\right\} \quad \begin{bmatrix} PR_j@P_j \in \textit{match}(\textit{SPP}), RD_k@SP_k \in \textit{match}(\textit{SPSP}), \\ H_C \in \bigcup_j P_j \cup \bigcup_k SP_k, setForward(LR, H_C) \end{bmatrix}}{\left\{\begin{array}{l} CR(LR)@H_C \cup \bigcup_j^{H_C \neq H_j} CollectSend(H_C, LR)@P_j \\ \qquad \cup \bigcup_k^{H_C \neq H_k} CollectSend(H_C, LR)@SP_k \end{array}\right\}}$$

(e) Forward Collect Resource

Fig. 3. Transformation Rules for the "Collect Resources" Strategy

- When a CollectSend operator is received by a host, it can be propagated to all its matching neighbors (Figure 3(b)) which are determined from the local indices. The plan is split into multiple parts which are distributed broadcast-like to the neighbors.
- Hosts can also execute the CollectSend operator (Figure 3(c)). This is treated as a binding transformation where results are sent back to the collecting host.
- A collecting host can execute the CR operator by accepting resources belonging to the given LR (Figure 3(d)). The results are sent from sub-plans built by the latter two transformations. Additionally, resources are bound using the local indices.
- Finally, the CR operator can also be forwarded to a neighbor (Figure 3(e)). This means that first, we choose the new collecting host H_C from the neighbors and set an appropriate forward. Then, the CR is pushed to H_C and all matching neighbors are instructed to send their data for LR to H_C. During query instantiation a CollectSend operator follows the forwards and creates a proper Send operator with the actual collecting host as target. Thus, results are sent directly to the correct host.

Figure 4 illustrates the rules querying resources of LR_p, i.e., the documents r and u. Starting at SP_2 as the local host with the initial query plan (Figure 4(a)), SP_3 is selected as collecting host of LR_p (Figure 4(b)) and a CollectSend informs SP_4 to send all documents regarding LR_p to SP_3. SP_3 decides to forward the CR to P_3 where the results are sent directly back to the initial caller (bypassing SP_3 and SP_2) (Figure 4(c)).



(a) Initial Query Plan (b) Collecting Host Selection (c) Forward Collect Resource

(d) Propagate CollectSend (e) Execute Collect Resources and CollectSend

Fig. 4. Example Applications of "Collect Resources" Accessing LR_p
(Thin lines demonstrate the query plan during instantiation, bold lines show the flow of results.)

SP_4, on its part, propagates the CollectSend operator to P_4 (Figure 4(d)). Finally, P_4 finds out by considering SP_3 to send the local resource r to P_3 and P_3 executes the CR operator and returns u and the received document r (Figure 4(e)).

Splitting and Distributing the Query Plan Valid query plans must be completely annotated and all resources must be bound. The best query plan is split into a local plan and multiple remote query plans. The remote plans are shipped to the referenced hosts [1] where the optimization process continues on the smaller query plans. The local query plan is instantiated and combines the results of the remote query plans.

Algorithm 1 splits (in DFS manner) a QEP into the local plan Q_L and the remote plans. The remote plans are stored in the mapping Q_R from the host where to execute the remaining query parts onto the query plan itself. One remote host may execute multiple sub-plans. The recursive function is called with the top-level operator of the query plan. Then the child operators are examined. If a child is executed at the same host, i.e., the local host, the function is called recursively. Otherwise, this is the root of a remote sub-plan and a Send operator is put on top of the sub-plan including the child

[1] Note, that these are always neighboring hosts.

Algorithm 1 Splitting the Query Plan

```
1: Q_L = Q
2: Q_R = ∅
3: function splitPlan(op)
4:   for all childOp ∈ op.children do
5:     if childOp.host == op.host then
6:       splitPlan(childOp)
7:     else
8:       Q_R.put(childOp.host, Send(op.host, childOp))
9:       replace(Q_L, childOp, Receive(childOp.host))
10:    end if
11:  end for
12: end function
```

operator. The remote sub-plan is separated from the local plan and a Receive operator at the local host is responsible for the connection to the remote plan.

The Cost Model Some of the parameters used for our cost model are stored within the local SP/P and SP/SP indices as described in Section 2, others are determined periodically by the runtime environment. In our distributed query processing environment we are interested in the plan with the lowest response time. Such a response time cost model was devised in [11] and explicitly takes parallelism and pipelining into account.

The most important parameters of query optimization in traditional databases are number and size of intermediary results. The same applies to P2P query processing, where we utilize the number of documents and the overall/maximum/minimum size of the registered resources for estimating the costs of a query plan. Our cost model also considers physical properties of the network, e.g., the network bandwidth, the latency, and the number of hops to the neighbors. But it is also important to know CPU and memory load of the local (super-)peer and the neighbors, as especially the super-peers are in danger of being overloaded, when too many queries execute operators at the super-peers. This would slow down query execution, so the optimizer should be aware of the current load situation of the local super-peer and the neighboring (super-)peers and generate alternative query plans, e.g. by using the "Collect" strategy, which enables the query optimizer to place operators on low loaded hosts. For these reasons, we utilize load information as one important parameter for the optimizer's cost model. Load collectors are used to collect data for the optimizer's view of the load situation of all relevant resources at the neighboring hosts. We measure the average CPU and memory load on (super-)peers and send the current situation to the neighbors. The optimizer's view of the load situation is updated at intervals of several seconds to prevent overloading the network. Using this information the optimizer at each (super-)peer can decide whether a sub-plan can be pushed to a neighbor, or—in the case of an overload—an alternative query plan would produce faster results.

Additionally, adapting the techniques presented in [29], our cost model can be extended to take the response time of "similar" queries, i.e., queries accessing the same index entries, into account.

4 Implementation

The discussed techniques for processing and optimizing complex queries in the highly dynamic and open environment of schema-based super-peer networks are already implemented in Edutella. The Edutella System [10] which constitutes an RDF-based metadata infrastructure for JXTA [17] is an open source project written in java.

The organisation of the super-peer backbone in the HyperCup-topology occurs dynamically. The distributed routing indices are also built and updated dynamically based on the registration files of the peers/super-peers.

We distinguish between *m*etadata statistics such as document count and file size and *n*etwork statistic parameters. The *m*etadata statistics are automatically extracted from the registration files and stored in the SP/P and SP/SP routing indices. The *n*etwork statistic parameters can be extracted at a given super-peer in an active way (e.g. memory load) by asking the neighboring super-peers or in a passive way by storing for example the response time of a given super-peer or peer. The statistics are currently used during the plan generation. The complex query processing modules are included in the package *n*et.jxta.edutella.complexquery. We also implemented a subpackage *n*et.jxta.edutella.complexquery.graph for the visualization of the QEPs. The subpackage *n*et.jxta.edutella.complexquery.work includes all classes needed for the execution of the QEP's different steps.

The complex query processing techniques are also implemented in QueryFlow [18, 19] which is based on ObjectGlobe [5] and descends from distributed query processing. A demonstration of the QueryFlow-based implementation was given in [33].

5 Conclusion and Further Work

Peer-to-Peer data management infrastructures are emerging as one of the important infrastructures for data intensive networks on the World Wide Web. In this paper we have investigated query distribution and query optimization issues for schema-based peer-to-peer networks, which use complex and possibly heterogeneous schemas for describing the data managed by the participating peers. Specifically, we have addressed the shortcoming of current peer-to-peer networks that they are unable to handle queries which need data from several peers to compute answers.

Comparing P2P data management networks to conventional distributed and federated database systems, we have identified specific additional challenges which make it impossible to apply distributed database query planning and optimization techniques in a straightforward way. We have therefore specified an innovative query routing and planning architecture based on distributed routing indices managed by a suitably connected set of super-peers, which makes distributed query processing available also in P2P data management networks. Finally we have discussed how to use transformation-based techniques for incremental query optimization at each super-peer, and specified a set of transformation rules, relevant for processing joins and unions in such a network. These techniques allow us to place query operators next to data sources and utilize distributed computing resources more effectively.

Future work will concentrate on the further investigation of simulations and experiments to evaluate and extend our current transformation rules. We aim also at collecting and using statistics more intensively.

References

1. K. Aberer and M. Hauswirth. Semantic gossiping. In *Database and Information Systems Research for Semantic Web and Enterprises, Invitational Workshop*, University of Georgia, Amicalola Falls and State Park, Georgia, April 2002.
2. P. A. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu. Data management for peer-to-peer computing: A vision. In *Proceedings of the Fifth International Workshop on the Web and Databases*, Madison, Wisconsin, June 2002.
3. P. Boncz and C. Treijtel. AmbientDB: Relational Query Processing over P2P Network. In *International Workshop on Databases, Information Systems and Peer-to-Peer Computing*, Berlin, Germany, September 2003.
4. R. Braumandl. *Quality of Service and Query Processing in an Information Economy*. PhD thesis, Universität Passau, Fakultät für Mathematik und Informatik, D-94030 Passau, 2001. Universität Passau.
5. 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):48–71, August 2001.
6. D. Brickley and R. V. Guha. RDF vocabulary description language 1.0: RDF Schema, January 2003. http://www.w3.org/TR/rdf-schema/.
7. I. Brunkhorst, H. Dhraief, A. Kemper, W. Nejdl, and C. Wiesner. Distributed queries and query optimization in schema-baed p2p-systems. In *International Workshop On Databases, Information Systems and Peer-to-Peer Computing, VLDB 2003*, Berlin, Germany, September 2003.
8. I. Brunkhorst, H. Dhraief, A. Kemper, W. Nejdl, and C. Wiesner. Distributed Queries and Query Optimization in Schema-Based P2P-Systems. In *International Workshop on Databases, Information Systems and Peer-to-Peer Computing*, Berlin, Germany, September 2003. Springer.
9. A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *Proceedings International Conference on Distributed Computing Systems*, July 2002.
10. The Edutella Project. http://edutella.jxta.org/, 2002.
11. S. Ganguly, W. Hasan, and R. Krishnamurthy. Query optimization for parallel execution. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 9–18, San Diego, CA, USA, June 1992.
12. L. Gong. Project JXTA: A technology overview. Technical report, SUN Microsystems, April 2001. http://www.jxta.org/project/www/docs/TechOverview.pdf.
13. A. Y. Halevy, Z. G. Ives, P. Mork, and I. Tatarinov. Piazza: Data management infrastructure for semantic web applications. In *Proceedings of the Twelfth International World Wide Web Conference (WWW2003)*, Budapest, Hungary, May 2003.
14. R. Huebsch, J. M. Hellerstein, N. Lanham, B. T. Loo, S. Shenker, and I. Stoica. Querying the Internet with PIER. In *Proc. of the Conf. on Very Large Data Bases (VLDB)*, pages 321–332, 2003.
15. Y. E. Ioannidis. The History of Histograms. In *Proc. of the Conf. on Very Large Data Bases (VLDB)*, pages 19–30, 2003.
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*, Madison, USA, June 2002.
17. Project JXTA Homepage. http://www.jxta.org/.
18. A. Kemper and C. Wiesner. HyperQueries: Dynamic Distributed Query Processing on the Internet. In VLDB [32], pages 551–560.

19. A. Kemper, C. Wiesner, and P. Winklhofer. Building Dynamic Market Places using HyperQueries. In *Proc. of the Intl. Conf. on Extending Database Technology (EDBT)*, volume Lecture Notes in computer Science (LNCS), pages 749–752, Prague, Czech Republic, March 2002. Springer Verlag.
20. O. Lassila and R.R. Swick. W3C Resource Description Framework model and syntax specification, February 1999. http://www.w3.org/TR/REC-rdf-syntax/.
21. A. Y. Levy, D. Srivastava, and T. Kirk. Data Model and Query Evaluation in Global Information Systems. *Journal of Intelligent Information Systems (JIIS)*, 5(2):121–143, 1995.
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 *Proceedings of the 11th International World Wide Web Conference*, Hawaii, USA, May 2002. http://edutella.jxta.org/reports/edutella-whitepaper.pdf.
23. W. Nejdl, B. Wolf, S. Staab, and J. Tane. Edutella: Searching and annotating resources within an RDF-based P2P network. In *Proceedings of the Semantic Web Workshop, 11th International World Wide Web Conference*, Honolulu, Hawaii, USA, May 2002.
24. 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 *Proceedings of the International World Wide Web Conference*, Budapest, Hungary, May 2003. http://citeseer.nj.nec.com/nejdl02superpeerbased.html.
25. V. Papadimos and D. Maier. Distributed Query Processing and Catalogs for Peer-to-Peer Systems. Asilomar, CA, USA, January 2003.
26. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *Proceedings of the 2001 Conference on applications, technologies, architectures, and protocols for computer communications*. ACM Press New York, NY, USA, 2001.
27. M. Schlosser, M. Sintek, S. Decker, and W. Nejdl. HyperCuP—Hypercubes, Ontologies and Efficient Search on P2P Networks. In *International Workshop on Agents and Peer-to-Peer Computing*, Bologna, Italy, July 2002.
28. M. Steinbrunn, G. Moerkotte, and A. Kemper. Heuristic and randomized optimization for the join ordering problem. *The VLDB Journal*, 6(3):191–208, August 1997.
29. M. Stillger, G. M. Lohman, V. Markl, and M. Kandil. LEO - DB2's LEarning Optimizer. In VLDB [32].
30. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 Conference on applications, technologies, architectures, and protocols for computer communications*. ACM Press New York, NY, USA, 2001.
31. H. Stuckenschmidt, R. Vdovjak, G-J. Houben, and J. Broekstra. Index structures and algorithms for querying distributed rdf repositories. In *Proceedings of the 13th International World Wide Web Conference (WWW2004)*, New York, USA, May 2004.
32. *Proc. of the Conf. on Very Large Data Bases (VLDB)*, Rome, Italy, September 2001.
33. C. Wiesner, A. Kemper, and S. Brandl. Dynamic, Extendible Query Processing in Super-Peer Based P2P Systems (Demonstration). In *Proc. IEEE Conf. on Data Engineering*, Boston, USA, March 2004.
34. B. Yang and H. Garcia-Molina. Improving search in peer-to-peer systems. In *Proceedings of the 22nd International Conference on Distributed Computing Systems*, Viena, Austria, July 2002. http://dbpubs.stanford.edu:8090/pub/2001-47.