

Distributed Query Processing in P2P Systems with incomplete schema information

Marcel Karnstedt Katja Hose Kai-Uwe Sattler

Department of Computer Science and Automation, TU Ilmenau
P.O. Box 100565, D-98684 Ilmenau, Germany

Abstract. The peer-to-peer (P2P) paradigm has emerged recently, mainly by file sharing systems like Napster or Gnutella and in terms of scalable distributed data structures. Because of the decentralization P2P systems promise an improved scalability and robustness, and they open a new view on data integration approaches, too. By exploiting already available mappings between pairs of peers a new peer joining the systems can immediately participate and access all the available data after establishing a correspondence mapping to at least one other peer. One of the technical challenges in building scalable P2P based integration systems is the efficient processing of queries which is complicated by the locally restricted knowledge about data placement and schema information. In this paper, we address this problem by investigating query processing strategies dealing with incomplete schemas and present results of our experimental evaluation.

1 Introduction

In the past years data integration problems were mainly solved by centralized solutions, either using a virtual approach where a mediator decomposes queries on a global schema and sends appropriate sub-queries to the sources for processing or by using a materialized approach where data from the sources is collected and integrated at a central place. Though this makes sense from a classical database point of view because of the availability of global knowledge (in terms of a global schema) and the possibility of central control there is a major drawback: scalability. In large settings it is often difficult to agree on a global schema even if integration takes place on a semantic level, e.g. by using an ontology. Furthermore, large-scale systems are often affected by frequent changes in terms of schemas or the participating sources. Finally, a central component for providing the knowledge of the schema and the sources as well as for initiating and coordinating queries represents a bottleneck and a single point of failure.

Thus, decentralization seems to be a natural way for solving such problems. Peer-to-Peer (P2P) systems are a consequent realization of this idea. In such systems there is no global knowledge: neither a global schema nor information of data distribution or indexes. The only information a participating peer has is information about its neighbors, i.e. which peers are reachable and which data they provide. The suitability of this approach was already demonstrated by the success of the well-known file sharing systems like Napster or Gnutella. Other promising variants are distributed data structures such as CAN, Chord or P-Grid [1]. Applying the P2P idea to the data integration problem means that each peer acts as a data source providing its own local schema and that

pairwise schema correspondences are defined between peers [2]. Furthermore, we cannot assume the existence of a global schema even not as the sum of the schemas of the neighbor peers because adding a new peer could trigger schema modifications for all other peers of the system.

There are several obvious advantages of such a schema-based P2P system. A main advantage is that adding a new source (peer) is simplified because it requires only to define correspondences to one peer already part of the system. Using this neighbor the new peer becomes accessible from all other peers. Of course, such advantages are not for free. Because of the lack of global knowledge, e.g. about data distribution, query planning is much more difficult than in centralized mediator systems. Another issue is the question of schema management. If we allow to define a “global” schema including information from non-neighbors we are violating the basic idea of P2P and giving up of some of the advantages. Restricting ourselves to only locally defined correspondences it is difficult to query data from peers for which the schemas (or parts of them) are unknown. In order to deal with such *incomplete schemas* we need a way to formulate and execute queries without knowing all schema elements in advance in order to avoid to overload the system due to flooding.

In this paper, we address this problem by investigating different strategies for processing queries on incomplete schemas. Our contribution is (1) dealing with the problem of incomplete schema information and (2) a detailed comparison of different query processing strategies in this context. The remainder of the paper is organized as follows. Based on the brief introduction of the underlying data and distribution model as well as the query model in Section 2 we classify possible processing strategies in Section 3. For these strategies we performed a comparison in terms of query execution cost as well as to determine the impact of the improvements. The results of this evaluation are presented in Section 4. After a discussion of related work in Section 5 we conclude the paper and point out to future work.

2 Data & Query Model

To some extent data integration requires a “canonical” data model into which the schemas of all participating sources are translated to and which is used to express correspondences (schema mappings), e.g. in the form of views (either Local-as-View or Global-as-View [3]). Usually, data model transformation is implemented using wrappers which encapsulate the internally used concepts and translate them to schema elements of the canonical data model. In recent years, semistructured data models and particularly XML-based models have been successfully used for data integration purposes. In the following we assume XML as the native data model for all peers, i.e. the schema of each peer is expressed in the form of a DTD or XML Schema.

Based on this assumption we have to deal with two issues. First, we have to express correspondences between two schemas and second, we have to formulate queries without complete schema information. For the first issue, several possible approaches exist. The most powerful form would be to use an XQuery-based view mechanism [4] or to invent a dedicated mapping language. However, because we do not consider all sorts of integration conflicts here, we reduce the problem to a core set of correspondence oper-

ations comprising equivalence and child/parent-of relationships. Let be D_1 and D_2 two XML documents and e_1 and e_2 paths in the document D_1 and D_2 resp. then

- *equivalence* denoted as $(D_1)//e_1 \equiv (D_2)//e_2$ means that element e_1 in D_1 is semantically the same object as described by element e_2 in D_2 . In fact, this represents a *horizontal* fragmentation. Please note, that this can be further refined by extensional relationships (overlap, disjoint, inclusion etc.).
- *child-of/part-of* denoted as $(D_1)//e_1 \prec_{id_1=id_2} (D_2)//e_2$ means that element e_2 in D_2 is a child of e_1 . The condition $id_1 = id_2$ is the join condition where id_i are itself paths in D_i . This corresponds to a *vertical* fragmentation.

In addition, a *transformation* τ is required that can be used as part of the above two correspondence operations. $\tau((D_1)//e_1)$ transforms the subtree of D_1 addressed by e_1 in a given way. This is not considered in the following and we simply assume that such transformations and their inverse operation can be expressed and used for query rewriting.

Using these relationships we are able to specify schema correspondences between two peers. For illustration purposes we consider a simple scenario (Fig. 1) where the autonomous nodes $P_1 \dots P_4$ form an information system integrating information about work of arts (paintings, sculptures, information about artists and details descriptions from an art catalog).

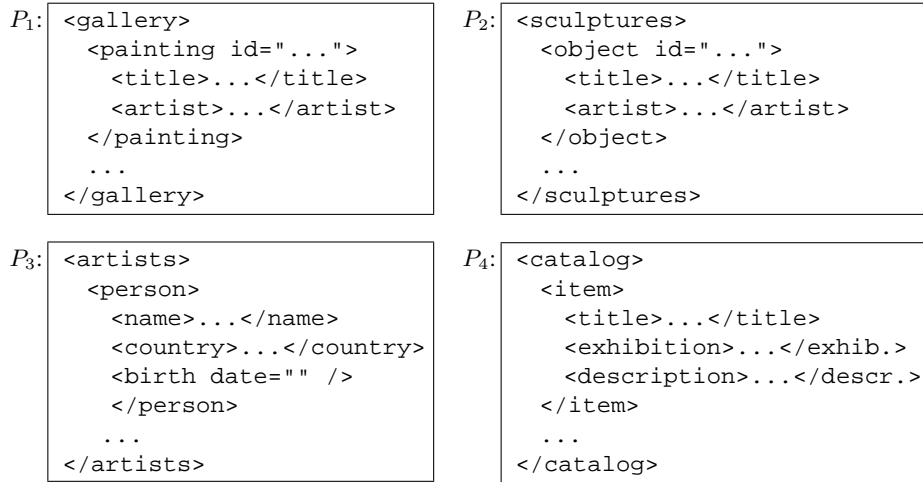


Fig. 1. Example Scenario

These four nodes are integrated in a P2P manner by defining the following bidirectional correspondences:

1. $(P_1)//painting \equiv (P_2)//object$
2. $(P_1)//painting \prec_{artist=name} (P_3)//person$
3. $(P_2)//object \prec_{title=name} (P_4)//item$

For query formulation we assume a subset of XQuery corresponding to XPath with joins. However, because in this paper we focus on query rewriting and evaluation strategies, we use a simple set of XML algebra operators for representing queries. Due to the

lack of a standard XML algebra we use our own notation which is inspired by the work of [5]. Here, path expressions consist of two distinct parts: a context path and a forward path denoted by $[context]forward$ where \perp as context represents the root of the document tree. This addresses elements reachable by the *forward* path whose relative location in the document tree is specified by *context*. Contexts are required to keep intermediate results generated through prior navigation. Based on this, we use the following operators:

- unnest $\mu_p(D)$ expands the element collection of the input D by the nodes which are reachable by the path expression p given as argument,
- rename $\beta_{o,n}(D)$ renames elements o from the input D by changing the tag to n ,
- select $\sigma_c(D)$ extracts a collection of the input D if it satisfies the selection condition c ,
- join $\bowtie_c(D_1, D_2)$ joins the collections of the input collections D_1 and D_2 if they satisfy the join condition c ,
- construct $C_{t,p_1,\dots,p_n}(D)$ constructs a new element with tag t and consisting of the child elements of D specified by the path expressions p_1, \dots, p_n ,
- set operations such as $\cup, \cap, -$.

Please note that further operators are required in order to form a complete algebra, but we omit them here for simplicity.

In our environment a user may query a peer in terms of its local schema, which than is routed to other peers providing corresponding data, or in terms of a schema (partially) unknown to the initiating peer. The real challenge are queries which go beyond the local schema, i.e. which reference schema elements that are not explicitly captured by correspondences. As an example consider a query Q initiated at peer P_2 , where the element “person” is not known to that peer. The query is represented using the above introduced algebra operators. In the long version of this paper we present an extended example [6].

$$Q: \sigma_{[object]person/country='Netherlands'}(\mu_{[\perp]//object}(P2.xml))$$

Using the above given correspondences this query could be rewritten in a straightforward manner if we assume global knowledge of all existing correspondences:

$$Q': \beta_{painting,object}(\begin{aligned} &C_{painting,[painting]title,[painting]artist,[\perp]person}(\ \\ &\bowtie_{artist=name}((\beta_{object,painting}(\mu_{[\perp]//object}(P2.xml)) \\ &\cup \mu_{[\perp]//painting}(P1.xml)), \sigma_{[person]country='Netherlands'}(\mu_{[\perp]//person}(P3.xml)))) \end{aligned})$$

Basically, there are three possible ways for resolving such unknown schema elements during query evaluation. One is to use a correspondence mapping $e_1 \equiv e_2$, which means e_1 can be translated to e_2 instead if used in a query. The second possibility is to use path expressions such as $e_1//e_2$, where all descendants of e_1 which are elements of type e_2 are selected. If e_2 is not known at the peer of e_1 it could be found by querying all peers with elements equivalent to or child of e_1 . At least, if no correspondences are defined for a certain element, one could try to find semantically equivalent elements through string (similarity) matching. However, in all these cases the query result can be more or less incomplete depending on the following factors:

- the reachability of the peers,
- the existence and quality of correspondence specifications,

- the matching of queries with the locally available schema.

A further important factor is the performance of query evaluation. Using a naive flooding in combination with a time-to-live parameter of query messages, i.e. each peer queries all known neighbors and so on up to a certain horizon, allows only to contact a limited number of peers and therefore may lead to incomplete results. Thus, it is important to restrict the number of “visited” peers to the relevant set. Ideally, one would use complete schema and distribution information. However, because this is contrary to the P2P paradigm we have to find a trade-off between required knowledge and performance loss. Thus, in the following sections we discuss appropriate strategies and their impact on query evaluation performance.

3 Query Processing Strategies

In distributed systems the general question is whether to execute the query at the initiator’s side or at the peers that store the relevant data. In the first case the data is moved to the initiator and all operations are executed there. This is called *data shipping* ([7]). The second approach is called *query shipping*, because in this case the query is moved to the data and only that part satisfying the query is shipped to the requestor for further processing. Applying this strategy the amount of data moved through the network is reduced, because only the necessary data a queried peer cannot process is sent to other peers. Query and data shipping are the two general approaches when processing queries distributed, but neither query shipping nor data shipping is the best policy for query processing in all situations. Other techniques, trying to combine the advantages of both approaches, have been developed. An example is called *hybrid shipping* ([8]).

In the query shipping approach the first intention is to decompose the query into subqueries according to the known peers and their querying capabilities. In this way each peer receives that part of the query it (or the peers connected to it) is expected to support. After decomposition the peer computes the corresponding result, or forwards the query to other peers if itself does not provide all queried data. Another technique evolved is called *Mutant Query Plans* ([9]): An execution plan constructed from the original query is sent as a whole to other peers. Each peer decides by itself if it can deliver any data. If yes, it writes the data into the plan replacing the corresponding part of the query. Using such mutating plans also provides the opportunity of optimizing (parts of) the plan decentralized.

In Section 4 we compare these two general query processing approaches. The implemented query shipping technique is a variant of mutating query plans. The query plan is shipped to the connected peers in parallel and each peer inserts the data it can provide. Beside the general approaches based on flooding additional approaches using global knowledge (all data at all peers is known to each peer) have been tested in order to outline the benefits of query shipping even more. The difference is that in the first approach there are more control messages generated than in this global knowledge approach. In our implementation global knowledge is gained from using routing indexes with a hop count correspondingly high enough. It should also be mentioned that we did not implement pure data shipping. Instead of shipping complete documents we select parts corresponding to XPath expressions in the leafs of our operator tree.

Query Processing Strategies in P2P Systems

In P2P systems a query can be initiated at any peer. As mentioned before the real challenge we will focus on is the processing of queries formulated in a foreign schema not completely captured by the local schema or the defined correspondences. The approaches mentioned in the previous section, suitable for distributed systems, are not suitable for real P2P systems without modifications. We cannot assume having all defined correspondences known to each peer. The general techniques for processing queries must be modified in order to accomplish the required tasks of query transformation and data collection step by step, having each peer responsible for querying local neighbors using only the locally defined mappings.

If the processed query is formulated in a schema unknown to the processing peer the simplest way of processing it is to query all neighbors, which we call flooding. This is an applicable strategy even if no correspondences are defined, because by sending the query to each peer in the network (up to a certain horizon) it will finally be shipped to those peers knowing the used schema. These peers can provide the queried data and send it back to the initiator. Naturally, this comes along with a huge impact of data sent through the network, because the number of messages created is very large.

As a consequence we need methods to route a query in the network, despite the restrictions we encounter in P2P systems. The problem of routing is to decide which of the known peers is most suitable for answering the query. A strategy adapted to our needs will have to use *partial* information available. A possible approach is to use the defined correspondences in terms of routing. The defined mappings provide for each peer information about what data is stored at the neighbors. In this way they provide us with knowledge in a horizon only including directly connected neighbors. If a matching peer is found, the processing peer may query it prior or instead of other peers.

Another approach for routing is to build a kind of routing tables, which assign (parts of) schemas (elements, paths, ...) to peers providing the according data. We implemented a variant of routing tables, called *compound routing indexes* ([10]), which we describe in the next section.

Routing Indexes

In order to reduce the message number using *routing indexes* can be quite useful. A routing index is a data structure that allows to route queries only to peers that may store the queried data. Therefore the data stored at each peer must somehow be associated with data identifiers. Routing indexes may also be used to generate a list of priorities for querying the peer's neighbors. In our approach data identifiers could be element names, element mappings (correspondences between two schemas) or path expressions.

We implemented compound routing indexes. Each peer stores the amount of elements it can retrieve using the connection to one of its neighbors. These values are related to the entries of the index. In order to control the trade-off between the effort for maintaining the indexes and the achieved benefits we limit the indexes by a hop count, also referred to as *hop count indexes*. In this variant the values are cumulated up to a certain horizon. Only elements provided by peers located at most the specified hop count

Table 1. Example of a Routing Index

known peer (id)	category on schema level	category on instance level
1	painting painting/title painting/artist	- painting/title='Landscape' painting/artist='van Gogh'
3	painting/person painting/person/name painting/person/country painting/person/birth	- painting/person/name≠'van Gogh' - painting/person/birth<1800
4	item item/title item/exhibition item/description	-

away are indexed. An entry of an index is called *category*. For a more detailed description of routing indexes in P2P systems we refer to [10]. In this work routing indexes which refer to keywords are implemented. We distinguish between indexes referring objects on instance level and indexes referring objects on schema level. On schema level the names of the schema elements form the categories of the index. On instance level, the categories refer to physically stored data objects. For our testing purposes we implemented two different variants of compound routing indexes. In our first variant, the names of the elements describing the data are used as category identifiers. In the second variant XPath expressions are used as identifiers instead. Correspondences can be understood as a specialization of routing indexes. They comply to routing indexes using a hop count of 1. The limit reflects that the horizon of knowledge provided by the correspondences only includes the directly connected neighbors. The difference is that an ordinary index normally does not define mappings, but includes them.

For illustrating purposes Table 1 shows a routing index storing path names, which is not compounding. In the figure identifiers for both, schema level and instance level categories, are depicted. This index may be built at peer P_2 from the introduced example. It is indicated how the mappings are integrated into the index, in this case when creating the categories containing a subpath 'person', which is in fact unknown to P_2 . In order to illustrate compound indexes and to indicate the differences between paths as category identifiers and elements instead, we present a second example in Table 2.

Table 2. Path-Based vs. Element-Based Routing Indexes

neighbor id	category is path	counter	category is element	counter
1	play/title	15	title	41
	play/fm/p	31	p	31
	play/personae/title	26	<i>already captured</i>	-
	play/scndescr	12	scndescr	12

5	play/title	20	title	34

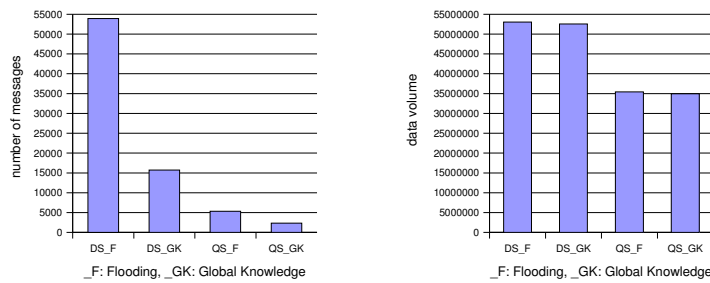
Maybe the most important factor using these indexes is the defined hop count. In a typical integration scenario a limited routing index may sophisticate delivered results, because they may be incomplete. In a file sharing scenario for instance, querying only a part of the peers storing similar or identical data may be suitable, because files may be replicated and retrieving one location may be as good as retrieving multiple possible locations of a file.

Open questions mainly involve problems of maintaining the routing indexes. This includes how to build them and how to ensure data coherency. These are some aspects for further research activities.

4 Evaluation

In this section we investigate different query processing strategies. In order to measure the benefits we achieve applying each strategy and to evaluate possible approaches we implemented a testing environment. The data distributed over the relatively small network of peers is based on plays of Shakespeare ([11]). The network is formed by 40 peers establishing randomly chosen bidirectional connections between them, so each peer is connected to only a subset of the others. The bidirectional correspondences between the peer-schemas were defined manually. We are also investigating a (semi-)automatic rule-based approach for defining the correspondences.

Using a querymix of 17 queries initiated at 4 different peers in the network we have tested a total of 68 queries. The achieved results present an orientation and should be interpreted accordingly. Measured values are the total number of messages generated and sent through the network as well as the total amount of data volume. The time needed to answer a query is not expressive in our one-machine implementation and therefore not captured. This is a main disadvantage of our simulation, but we are working at an extended environment supporting realistic time measurements. For implementation we used Java and Threads. The routing indexes used in our tests are utilizing paths as category identifiers. We did not evaluate the indexes on instance level mentioned before, but this is part of our ongoing work in the near future.



(a) Number of messages

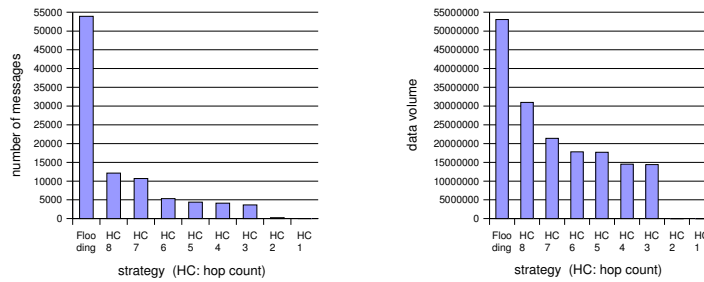
(b) Total data volume

Fig. 2. Comparison of data shipping (*DS*) and query shipping (*QS*)

In the first of our tests we investigated the differences between the two main strategies using our flooding approach. This approach is applicable having any knowledge

about the data distributed over the network available or not. The results are compared to a global knowledge approach in figure 2. 'DS' denotes data shipping, 'QS' the query shipping strategy.

As one would have expected query shipping outperforms data shipping. Data shipping does not make much sense if the queries are essentially select-project queries. Furthermore caching is not utilized. The methods using global knowledge, implemented for better comparison, come along with a significant lower number of messages as the flooding techniques. The shipped data volume almost equals to the flooding approaches, but using data shipping it is again by far higher as when using query shipping in conjunction with global knowledge. In the following we will present our results for each of the two strategies separately.



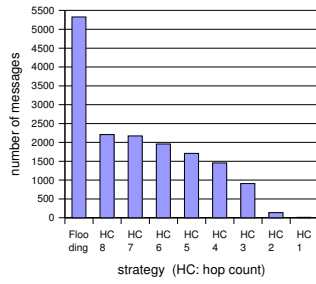
(a) Number of messages

(b) Total data volume

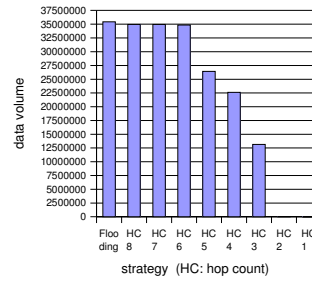
Fig. 3. Results using data shipping and limited routing indexes

The next test uses our indexes in combination with the data shipping strategy. The results are shown in Figure 3. $HC x$ indicates the strategy using a routing index with a hop count of x . If the hop count is 1 using the indexes corresponds to using only the locally defined mappings. A hop count of 8 symbolizes global knowledge for a bulk of the peers in the network. This is not given for peers where any other peer is located more than 8 hops away. The number of messages and the shipped data volume are lowered by far when using the indexes. It seems as if the gained benefits are really high using data shipping. But, as we will see when looking at the actually retrieved part of the whole result, this is actually not given. Before that we look at the benefits we gain from the indexes when using the query shipping strategy. These results are shown in Figure 4. Again, $HC x$ denotes a strategy using routing indexes with a hop count of x .

In both strategies the results indicate a great benefit even when using only the local correspondences. As expected the number of messages is still significant lower for all hop counts using query shipping as when using data shipping. At a first look the high amount of data volume generated is surprising, it is even higher as when using data shipping. The reason for this will get evident when looking at the retrieved percentage of result data. When we use data shipping we simply cannot utilize our indexes as much as with the query shipping strategy. Using query shipping the original query is sent and each receiving peer may have more information available in order to route the query than the one that sent it. This applies to each operator of the tree. In our data shipping variant only XPath subqueries from the leafs of the operator tree are sent through the



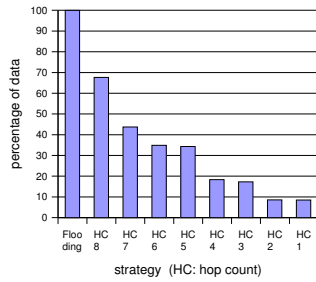
(a) Number of messages



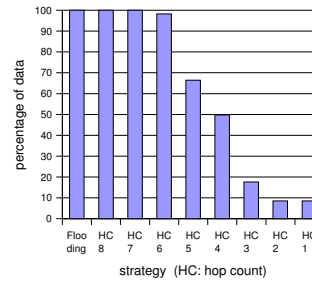
(b) Total data volume

Fig. 4. Results using query shipping and limited routing indexes

network, so significantly less information may be exploited. The percentage of result data actually retrieved, shown in Figure 5, outlines these problems.



(a) Data shipping



(b) Query shipping

Fig. 5. Percentage of data actually retrieved

The routing indexes are not very useful in combination with data shipping. Often a query may not be routed, because no corresponding information at all is available. In this case the processing stops and we miss data. In order to receive hundred percent of the result data in our example we would have to choose a hop count of 12 or higher. In the case of query shipping we narrow our horizon when choosing low hop counts and may miss data, too. Reaching over 90 percent of the achievable data the routing indexes limited by a hop count of 6 are fine, a hop count of 5 may still be acceptable. The small horizon of only locally defined correspondences up to a hop count horizon of 3 can speed up query processing by far, but the retrieved part of the data is unsatisfying small. In the examples we contemplate in this work such incomplete results are mostly not suitable for the user. Using a higher hop count more peers are included into the process of query evaluation and thus more data is collected. The strict need for methods compensating the miss of data if no needful information is found in the indexes gets evident. One possible approach is to flood all neighbors again.

We also compared the two mentioned index variants, one using paths as category identifiers, the other using elements. The version using paths generally performs slightly better, because data is indexed on a finer granularity. The differences are that small we omit the achieved results here. They can be found in the long version of this paper [6].

Our evaluation obviously shows that query shipping outperforms data shipping. Routing indexes can be very useful in terms of routing to speed up query processing even more, but are impractical to use with data shipping. One of our main conclusions is the importance of the defined hop count. Setting it to a low value we may miss too much data, e.g. when using only the locally defined correspondences. Rising it to a value nearly degenerating the index to global knowledge will maximize the effort required for maintenance. In our ongoing work we plan to analyze detailed relationships between these factors on a variety of P2P network topologies and schema mappings.

5 Related Work

The state of the art in distributed query processing is presented in the survey [7]. In this work there is no special concern about P2P systems. Special concern on schema-based P2P systems is spend in [12]. The Edutella system is based on a super-peer backbone and schema-aware routing indexes. The problems arising in distributed query processing are shifted to the super-peers. No mediation takes place.

The Piazza system ([4]) is based on schema mappings between the participating peers. The transitive closure of these mappings is built and used for query evaluation. Each peer has an own "view of the world", it's *peer-schema*. XML-based correspondences, the *peer-descriptions*, relate the different views. The data actually stored is described using *storage descriptions* relating a stored schema to a peer-schema. Centrally placed routing indexes are used for query processing. In this system mainly schema mapping and composition are reviewed, efficient query processing is less concerned.

Routing indexes especially for P2P systems are described in detail by [10]. There are many alternative ways of doing this sort of job, e.g. distributed hash table approaches (DHTs, e.g. [13]) or distributed routing indexes a la CHORD [14]. Another work addressing the problem of managing data in P2P networks is AmbientDB ([15]). AmbientDB is a current research project at the University of Twente in the Netherlands and was designed for relational query processing. Whereas AmbientDB tries to speed up query processing by introducing distributed hash tables, we apply routing indexes to attain the same goal. Using these clustered indexes in AmbientDB data is duplicated and distributed among the network's peers according to a defined hash function. In further research AmbientDB is planned to work with XML data.

In [9] *Mutant Query Plans* are presented. This is a technique very similar to our implemented query shipping technique. In this work neither decomposition or parallelism, nor data translation or integration takes place.

6 Conclusion

Efficient query processing is – beside others - one of the main challenges in P2P-based data integration systems. The decentralized nature of P2P systems makes it difficult to directly use well-known processing strategies from distributed database systems. In this paper, we have investigated this problem by discussion adaptations of such strategies in P2P scenarios under the assumption of limited local knowledge about schema and

data placement. We have argued that one has to deal with cases where schema information is incomplete and have shown that flooding can be avoided to some extent using schema-based routing indexes. However, this is only the first step towards a distributed P2P query engine. So far, we have ignored query costs and cost-based decisions about alternative query plans. Choosing an appropriate cost model and managing up-to-date cost information in a P2P system is part of our ongoing work. Furthermore, the highly dynamic and unpredictable nature of a P2P system requires adaptive techniques [16] which we plan to address in our future work, too.

References

1. Aberer, K., Hauswirth, M.: P2P Information Systems: Concepts and Models, State of the Art, and Future Systems. In: ICDE 2002. (2002) 187– Advanced Technology Seminar.
2. Gribble, S.D., Halevy, A.Y., Ives, Z.G., Rodrig, M., Suci, D.: What Can Database Do for Peer-to-Peer? In: WebDB 2001. (2001) 31–36
3. Lenzerini, M.: Data Integration: A Theoretical Perspective. In: Proc. ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS) 2002, Madison, Wisconsin. (2002) 233–246
4. Tatarinov, I., Ives, Z., Madhavan, J., Halevy, A., Suci, D., Dalvi, N., Dong, X.L., Kadiyska, Y., Miklau, G., Mork, P.: The Piazza peer data management project. SIGMOD Rec. **32** (2003) 47–52
5. Viglas, S., Galanis, L., DeWitt, D., Naughton, J., Maier, D.: Putting XML Query Algebras into Context. submitted for publication (2002)
6. Karnstedt, M., Hose, K., Sattler, K.U.: Distributed Query Processing in P2P Systems with incomplete schema information (extended version), available at <http://sundb1.prakinf.tu-ilmenau.de/~karn/papers/diweb04ext.pdf> (2003)
7. Kossmann, D.: The State of the Art in Distributed Query Processing. ACM Computing Surveys **32** (2000) 422–469
8. Franklin, M.J., Jónsson, B.T., Kossmann, D.: Performance tradeoffs for client-server query processing. In: Proceedings of the SIGMOD Conference. (1996) 149–160
9. Papadimos, V., Maier, D.: Mutant Query Plans. Information and Software Technology **44** (2002) 197–206
10. Crespo, A., Garcia-Molina, H.: Routing indices for peer-to-peer systems. In: Proc. of the 28 int. Conference on Distributed Computing Systems. (2002)
11. Bosak, J.: Shakespeare 2.00 (1999) The plays of Shakespeare, marked up by Jon Bosak available at metalab.unc.edu/bosak/xml/eg/shaks200.zip.
12. Brunkhorst, I., Dhraief, H., Kemper, A., Nejd, W., Wiesner, C.: Distributed Queries and Query Optimization in Schema-Based P2P Systems. In: International Workshop on Databases, Information Systems and Peer-to-Peer Computing, Berlin, Germany. (2003)
13. Galanis, L., Wang, Y., Jeffery, S.R., DeWitt, D.J.: Locating data sources in large distributed systems. In: Proceedings of VLDB 2003. (2003)
14. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: Proceedings of ACM SIGCOMM, ACM Press (2001) 149–160
15. Boncz, P., Treijtel, C.: AmbientDB: Relational Query Processing in a P2P Network. In: Proc. Workshop On Databases, Information Systems and P2P Computing 2003, Berlin (co-located with VLDB'03), LNCS 2788, Springer Verlag (2003)
16. Gounaris, A., Paton, N.W., Fernandes, A.A.A., Sakellariou, R.: Adaptive Query Processing: A Survey. In: BNCOD 2002. (2002) 11–25