Query Routing and Processing in Schema-Based P2P Systems

Marcel Karnstedt Katja Hose Kai-Uwe Sattler
Department of Computer Science and Automation, TU Ilmenau
P.O. Box 100565, D-98684 Ilmenau, Germany

Abstract

Recently, the peer-to-peer (P2P) paradigm has emerged, mainly by file sharing systems such as Napster and Gnutella and in terms of scalable distributed data structures. Due to the decentralization, P2P systems promise an improved robustness and scalability and therefore open also a new view on data integration solutions. However, several design and technical challenges arise in building scalable P2P-based integration systems. In this paper, we address one of them: the problem of distributed query processing. We discuss strategies of query decomposition and routing based on different kinds of routing indexes and present results of an experimental evaluation.

1. Introduction

Peer-to-peer (P2P) systems open new views on a wide range of problems, e.g., for data integration purposes, as we have shown in [7]. 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.

An important issue in the context of schema-based P2P systems is efficient query processing. Because of the lack of global knowledge, e.g., about data distribution, query planning is much more difficult than in centralized systems. In the worst case queries can only be answered by flooding the network with a query resulting in the same problems as already discussed for file sharing systems (e.g., high overhead / bandwidth requirements, fragmentation of the network because of the necessity to limit the number of hops etc. [10]).

In this paper, we address this problem by investigating strategies for routing and distributed processing of queries. 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. The routing approach is introduced in Section 4. For the implemented 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 5. After a discussion of related work in Section 6 we conclude the paper and point out to future work.

2. Data & Query Model

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. Besides schema definitions we have to express correspondences between schemas. For this purpose, several possible approaches exist. The most powerful way would be to use an XQuery-based view mechanism or to invent a dedicated mapping language. We use three main operations (*equivalence, child-of/part-of, transformation*) to express the correspondences. The model is shortly illustrated in the example in Section 4. For a more detailed description we refer to [7].

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 [11].

An important issue is the performance of query evaluation. Using a naive flooding in combination with a time-to-live parameter of query messages (in order to avoid overloading the network) 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

The two basic approaches when processing queries in distributed systems are *data shipping* and *query shipping* ([8]). If all data necessary to compute the result is shipped to the initiating peer and all operators are applied at that side this is called data shipping. In the query shipping approach the operators are applied at the peers where the data resides. Only data that may not be processed further is shipped to the requesting peer. Former works outlined that the query shipping approach outperforms data shipping in terms of message number and volume in large distributed database systems if no caching is present (e.g., [4]). As we for our part examine large scale P2P systems where the peers, for now, do not utilize caches, for the rest of this paper we focus on the query shipping approach.

In our implementation query execution plans are represented by *plan operators (POP)* using a graph-based model. We query other peers by sending query plans that are cloned and modified by the peers providing the according data. This approach is similar to the one presented in [9]. Our algorithm for the query shipping strategy is not completeley listed here, but the central part of it, a procedure called *process-POP()*. This procedure is called by a peer in order to start the actual query processing. An abridged version is shown in Figure 1.

Input:
POP q
Output:
q filled with data

```
1    /* fill with local data */
2    forall neighbor peers P do
3        if data-known-to-peer(q, P) do
4            /* send a message to P in order to process q*/
5        od
6    od
7    if found-one = false do
8        /* query all neighbours */
9    od
10   /* apply joins etc. */
11   /* send plan back to initiator */
```

Figure 1. Procedure *process-POP*

While applying the procedure locally several messages are sent to the neighboring peers. These messages are used to ask the connected peers to further process the query in parallel. After receiving such a message a peer starts its local *process-POP()* with the provided parameters, while the initiating peer continues to run his procedure. In line 3 we take *routing indexes* into our considerations. We will describe the functionality of the procedure *data-known-to-peer()* together with main aspects and problems we are faced when using routing indexes in Section 4. At this point it is important to realize that we only query peers we are expecting to provide data. In the case of flooding, e.g., if we have no schema or distribution information available at all, we simply assume each peer is able to provide data. The call of the procedure returns the according boolean values. In the case we have information available but are not able to determine any supporting peer, we get back to flood the network.

4. Query Routing

When a peer is processing a query it could need information about the quality of the data other peers can return. If no information is available a peer can only flood the network, which means to query each connected peer. This results in a very high amount of messages and a huge data volume sent through the network. As a consequence we need methods to route a query, despite the limited information available at each peer. The problem of routing is to decide which of the known peers is most suitable for answering a query. We use *routing indexes* to do this. 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. Each peer builds its own indexes, assigning to each established connection the data retrievable using that connection. As a consequence of the characteristics of P2P systems, these indexes must somehow be limited in their horizon. If the horizon is not limited they degenerate to a data structure representing global knowledge. In this case maintenance tasks will not be performable.

In our implementation we use a kind of *compound routing indexes* [3]. In contrast to this work we are indexing elements not only on instance level, but also on schema level. Indexing on schema level means the data is identified by the XPath expressions describing the objects. On instance level we refer to the physical objects actually stored, using according filter predicates. An entry of the index is formed by a path expression defining the element, called *category*, an id of the peer the connection is assigned to and two counters. The *cardinality* counter denotes how many data objects are provided, *compounded* over all peers reachable when querying the peer identified by the id. The second counter, called *#peers*, denotes the number of peers providing data according to the category.

The horizon a routing index should be limited to is implemented using a *hop count*. The hop count dedicates the distance in number of hops that peers may reside away from the local peer. Raising the value to a maximum we will get back to use global knowledge. Implementing a hop count

category (schema level)	predicate (instance level)	cardinality	# peers
painting	-	520	4
painting/title	-	520	3
painting/artist	artist='Monet'	100	1
painting/artist	artist≠'Monet'	420	2
painting/person	-	210	2
p./person/name	-	210	2
p./person/country	-	210	1
p./person/birth	[@date<'1800']	132	1
p./person/birth	[@date≥'1800']	78	1
...	...	...	...

Table 1. Example of a compound routing index at peer P_2

of 1 will limit the indexes to reflect only the locally defined correspondences. The resulting horizon comprises only the directly connected neighbors.

Table 1 illustrates a part of the routing index built at peer P_2 using a hop count of 2 (here it is the part according to the connection to P_1). To improve readability 'painting' is abbreviated to 'p.' in the longest paths. It is indicated how the correspondences are used when building the indexes.

The procedure *data-known-to-peer()*, in which we refer to our indexes, is omitted here. If no index is defined we assume that we want to flood the network. In this case the procedure always returns *true*. The rest of the procedure can be summarized in three sentences: If any unnest operator is found the procedure returns *true* if the queried path is indexed. If any selection operator is found in the operator tree then *true* is returned if the queried path is indexed and the comparison of according predicates returns *true*, too. If none of these is found, *false* is returned.

Open questions mainly involve building and maintaining the indexes. These are aspects of our current research. At the moment also coherency as well as changes in the indexed data are not supported.

Example

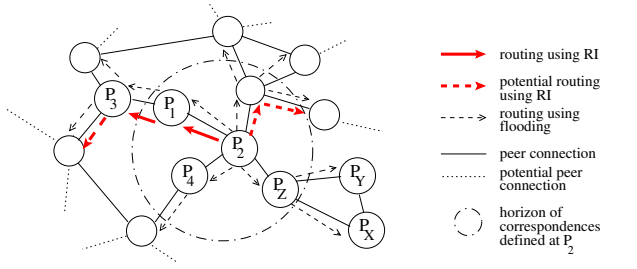


Figure 2. Subset of the example network

For illustration purposes we consider a simple scenario where the autonomous nodes $P_1 \ldots P_4$ form an information system integrating information about work of arts. 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*

In the following we present a small example in order to illustrate the procedure of query processing and routing. Figure 2 shows a subset of a P2P network which includes the mentioned peers and correspondences as well as some more peers not described in detail here. The circle surrounding peer P_2 symbolizes the horizon of knowledge deduced from the locally defined correspondences, complying to routing indexes using a hop count of 1. All peers located outside that circle are the peers we may only efficiently route our query to if we use the defined indexes. Imagine a query Q initiated at peer P_2:

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

When P_2 applies the locally defined correspondences it recognizes that all neighbor peers but P_1 cannot provide any according data. Using a routing index with a hop count greater than 1 it will also recognize that the part of the schema unknown to P_2 itself, which is *person/country*, is somehow supported by a peer reachable using the connection established to P_1. P_2 transforms Q into Q':

$$Q'\text{: } \beta_{painting,object}(\sigma_{[painting]person/country='Netherlands'}(\beta_{object,painting}(\mu_{[\perp]//object}(P2.xml)) \cup \mu_{[\perp]//painting}(P1.xml)))$$

Finally the data provided by P_2 is inserted into the execution plan of Q' and it is sent to P_1 for further processing. Without having the index defined there arise two possible approaches:

1. flood the network in order to retrieve data according to the unknown schema parts
2. only contact P_1, because it is the only peer providing data which we know about, and hope it will know about the unknown schema

If the correspondence to P_1 would also not exist we would have to decide whether to flood or to stop processing at all. In the picture the thick arrows indicate the directions we would route the query if we have defined indexes with a hop count high enough. The dashed ones correspond to possible mappings we have not listed here. The thin dashed arrows symbolize the way the query takes if we decide to flood the network.

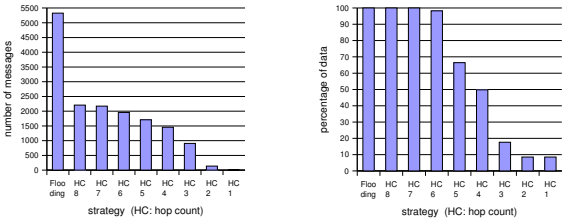
At P_1 the known correspondences are applied again. Recognizing the data provided by P_3 the transformed query that is sent to P_3 could now look like:

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

At each peer other possibilities to transform the query are conceivable, an imaginable modification in our example is the placement of the selection operator. Here general aspects of dynamic query optimization get indicated.

Investigating the known correspondences P_3 cannot find any further mapping possible to apply. P_3 adds the local data, executes final operations and sends back the query plan holding the result data. Again the thick dashed arrow indicates how the query gets routed if another corresponding mapping would exist.

5. Evaluation

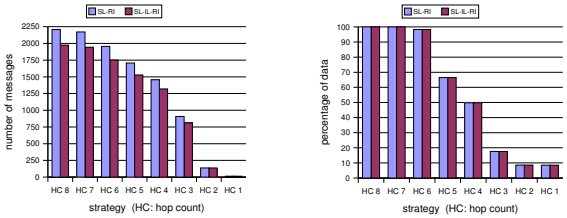


(a) Number of messages (b) Percentage of data

Figure 3. Comparison of routing indexes using different hop counts

In order to preliminarily evaluate the benefits of the different query processing and routing strategies we have implemented a testing environment. The data distributed over the relatively small network of peers is based on plays of Shakespeare ([1]). The network is formed by 40 peers establishing randomly chosen bidirectional connections between them. The bidirectional correspondences between the peer-schemas were defined manually. We are also investigating possibilities of generating these mappings using a (semi-)automatic rule-based algorithm. Using a querymix of 17 queries initiated at 4 different peers in the network we have tested a total of 68 queries. Measured values are the total number of messages generated and sent through the network as well as the percentage of data actually retrieved (we are able to retrieve the complete data if we flood the network). The time needed to answer a query is not expressive in our simulation and therefore not captured. The simulation environment was implemented using Java. The behavior of the peers is simulated using Threads. We have to omit a detailed analysis of the achieved results, because we are short in space. Thus we limit to a short conclusion at the end of this section.

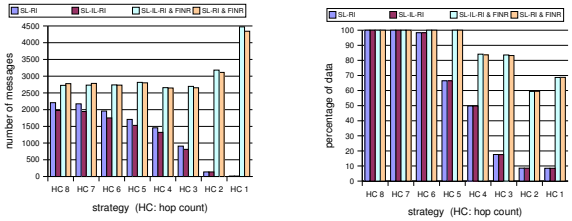
In our first tests we wanted to evaluate the impact of routing indexes on schema level using different hop counts. The results are shown in Figure 3. *HC* x indicates the strategy using a routing index on schema level with a hop count of x. The results we achieved by flooding the network are captured additionally.



(a) Number of messages (b) Percentage of data

Figure 4. Comparison of routing indexes with and without instance level

In the next tests we wanted to check the benefit of instance level indexes. If information about distribution of data is complete, as we ensured in our environment, we expect a high benefit and possible missing of relevant data to be minimal. The results we achieved are shown in Figure 4. *SL-RI* is short for *Schema-Level-Routing-Index* and denotes indexes defined only on schema level, *SL-IL-RI* stands for *Schema-Level-Instance-Level-Routing-Index* respectively and denotes the strategy using indexes on schema and instance level.



(a) Number of messages (b) Percentage of data

Figure 5. Comparison of routing indexes with use of FINR

In Section 4 we mentioned the problem that we en-

counter if the used index indicates that none of the neighbors may provide according data. If we stop processing at this point we could miss some important data. A simple approach to solve the problem is to query all neighbors, which corresponds to flooding again. We call this approach "Flooding If No Route", *FINR* for short. In order to evaluate a possible benefit from the instance level indexes we distinguished between indexes on only schema level (*SL-RI*) and on schema and instance level (*SL-IL-RI*) again. The results of this test are shown in Figure 5.

The preliminarily results presented in this section permit a simple statement: Routing indexes are very powerful in order to process queries more efficient. The indexes allow to take routing decisions independently from knowledge of the complete network by providing knowledge about the data of peers up to a certain horizon. In this way they provide peers with *partial* knowledge, improving routing by far without having to collect *global* knowledge, thus they are predestinated to be used in P2P systems. The most influencing factor is the used hop count, interacting with the achievable performance as well as the amount of retrievable data and the effort of maintaining the indexes. Defining indexes on instance level additionally to indexes on schema level can improve performance even more, but strongly depending on the amount and character of information available apriori. Methods to evaluate the quality of results using a certain hop count are essential as well as strategies to improve these results if they are not satisfying. One possible approach to this is to use FINR, but strategies that perform better are desirable.

6. Related Work

The state of the art in distributed query processing is presented in the survey [8]. In this work there is no special concern about P2P systems. Special concern on schema-based P2P systems is spend in [2]. 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.

Routing indexes especially for P2P systems are described in detail by [3]. There are many alternative ways of doing this sort of job, e.g., distributed hash table approaches (DHT), as in [5].

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.

7. 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 the essential problem of query routing in P2P scenarios under the assumption of limited local knowledge about schema and data placement. We have outlined the possible impact o routing indexes on schema level as well as on instance level in such a scenario. Open questions we have encountered are including index maintenance and the choice of an optimal hop count, reflecting the horizon of available knowledge. We ran some tests to preliminarily evaluate our approach. 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 [6] which we plan to address in our future work, too.

References

[1] J. Bosak. Shakespeare 2.00, 1999. The plays of Shakespeare, marked up by Jon Bosak
available at metalab.unc.edu/bosak/xml/eg/shaks200.zip.

[2] 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*, Sep 2003.

[3] A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *Proc. of the 28 tn Conference on Distributed Computing Systems*, July 2002.

[4] M. J. Franklin, B. T. Jónsson, and D. Kossmann. Performance tradeoffs for client-server query processing. In *Proceedings of the SIGMOD Conference*, pages 149–160, 1996.

[5] L. Galanis, Y. Wang, S. R. Jeffery, and D. J. DeWitt. Locating data sources in large distributed systems. In *Proceedings of VLDB 2003*, 2003.

[6] A. Gounaris, N. W. Paton, A. A. A. Fernandes, and R. Sakellariou. Adaptive Query Processing: A Survey. In *BNCOD 2002*, pages 11–25, 2002.

[7] M. Karnstedt, K. Hose, and K.-U. Sattler. Distributed Query Processing in Schema-Based P2P Systems with incomplete schema information. In *Proc. of the int. Conf. CAiSE (Workshop DIWeb'04), Riga, Latvia, To appear*, 2004.

[8] D. Kossmann. The State of the Art in Distributed Query Processing. *ACM Computing Surveys*, 32(4):422–469, 2000.

[9] V. Papadimos and D. Maier. Mutant Query Plans. *Information and Software Technology*, 44(4):197–206, April 2002.

[10] J. Ritter. Why Gnutella Can't Scale. No, Really, 2001. www.tch.org/gnutella.html.

[11] S. Viglas, L. Galanis, D. DeWitt, J. Naughton, and D. Maier. Putting XML Query Algebras into Context. submitted for publication, 2002.