A CONSTRAINT-AWARE QUERY OPTIMIZER FOR WEB-BASED DATA INTEGRATION

Jing Lu and Bernhard Mitschang
IPVS, University of Stuttgart, Universitaetsstrasse 38, Stuttgart, Germany

Keywords: XML, XQuery, Data Integration, Constraints, Semantic Query Optimization.

Abstract: Web has brought forth opportunities to connect information sources across all types of boundaries. Data integration is to combine data residing at different sources and providing the user with a unified view of these data. Currently users are expecting more efficient services from such data integration systems. This paper describes a query optimizer, which uses constraints to semantically optimize the queries. The optimizer first translates constraints from data sources into constraints expressed at the global level and stores them in the constraint repository. Then the optimizer can use semantic query optimization technologies including detection of empty results, join elimination, and predicate elimination to generate a more efficient but semantically equivalent query for the user. The optmizer is published as a web service and can be invoked by many data integration systems. We carry out experiments and first results show that performance can be greatly improved.

1 INTRODUCTION

The rising of the web and its subsequent ubiquity make it possible to connect information sources across all types of boundaries (local, regional, organizational, etc). Integrating data scattered on the web has been a challenge faced by many enterprises. It is especially crucial in large enterprises where a large variety of web-based applications demand access and integration of up-to-date information from multiple distributed and heterogeneous information systems. Data integration system (DIS) provides the user with a virtual view, called global schema, which is independent from the model and the physical origin of the sources. Integration wrappers are responsible to translate the local schema into the global schema, a global query into a local query and the local query result back into the result in the global schema. Nowadays people are expecting more efficient queries from data integration systems where query optimization is greatly different from that in the traditional database context. First, since the sources are autonomous, the optimizer may have no statistics about the sources, or just unreliable ones. Hence, the optimizer cannot compare between different plans, because their costs cannot be sufficiently well estimated. Second, since the sources may have different processing capabilities, the query optimizer needs to consider the possibility of exploiting the query processing capabilities of a data source. Finally, in a traditional system, the optimizer can reliably estimate the time to transfer data from the disc to main memory. But in a data integration system, data is often transferred over a wide-area network, and hence delays may occur for a multitude of reasons. Therefore, even a plan that appears to be the best based on cost estimates may turn out to be inefficient if there are unexpected delays in transferring data from one of the sources accessed early on the plan.

The heterogeneity and web-orientation of modern applications have benefited from the flexibility of XML and therefore, there appear many XML-based data integration systems, among which are EXIP (Papakonstantinou and Vassalos, 2002), Xyleme (Abiteboul et al., 2001), Silkroute (Fernandez et al., 2000), XPERANTO (Carey et al., 2000), BEA AquaLogic (Carey and the BEA AquaLogic Team, 2006), etc. A major difficulty in optimizing queries here is that once a query is submitted, control over its execution becomes no longer possible (Ouzzani and Bouguettaya, 2004). Under this situation, using semantic rules to minimize the cost of communication becomes more attractive. Semantic query optimization (SQO) has been applied to relational and deductive databases (Chakravarthy et al., 1990). SQO uses the integrity constraints associated with a database to improve the efficiency of query evaluation. With SQO more efficiency can be achieved than by only using syntactic query optimization techniques, because syntactic optimizer does not understand semantic knowledge, and thus leads to a suboptimal execution for the given query. Certain queries, which can be answered without any table scan in the database, cannot be detected by a syntactic optimizer, thus resulting in unnecessary database access. Furthermore, syntactic optimizer cannot detect and eliminate semantically redundant restrictions or joins in the queries or introduce some useful restrictions and joins into the query to reduce the overall cost of query execution, either (Cheng et al., 1999). Here, we demonstrate that SQO can also be adapted to XML-based data integration system. Our contributions include:

- We provide the architecture of the optimizer and show how queries to different data integration systems are optimized by the optimizer.
- We explain how constraints are integrated into the constraint repository and managed by the optimizer. We describe the details of the query adapter, which is the kernel of the optimizer.
- We implement three SQO techniques of in the optimizer: detection of empty results, join elimination and predicate elimination. We carry out experiments and analyze the performance.

The structure of this paper is: Section 2 presents the architecture of the semantic query optimizer. Section 3 explains each component in the optimizer. Section 4 discusses three SQO techniques which are implemented in the semantic query optimizer. Experiments and performance analysis are in section 5. Related work is presented in section 6. Section 7 gives the conclusion.

2 ARCHITECTURE

The optimizer should be designed as not being specific to a particular data integration system, but being applicable to different ones. It should have ease of integration with and adaptation to a data integration system. To support this, we use Service-Oriented Architecture (SOA) (Huhns and Singh, 2005). The architecture of the semantic query optimizer is shown in Figure 1. There are altogether three parties which interact with the optimizer: the user of data integration system, the administrator of the data integration system and the administrator of the data sources. They can call the web services of our semantic query optimizer to register constraints, manage constraints, or optimize queries.

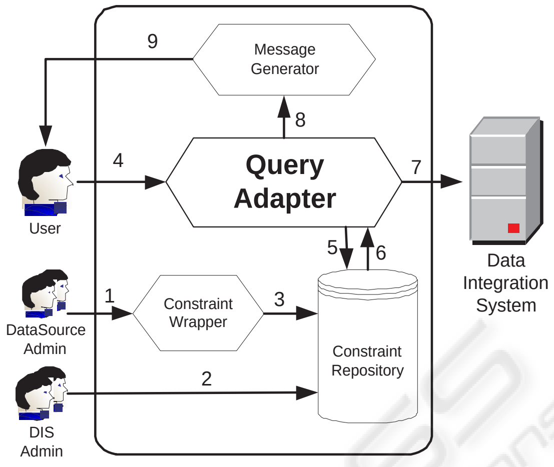

Figure 1: Architecture of the Semantic Query Optimizer.

The data sources can register their constraints to the optimizer through constraint wrappers (Step 1). The administrator of the data integration system can also add or delete global constraints (Step 2). The constraint wrappers will translate the local constraints into constraints in global schema with the help of the schema mapping information. The translated constraints will be added into the constraint repository (Step 3). The data sources can also deregister constraints and the constraint wrappers will delete the corresponding constraints from the constraint repository. The user can submit the query to the optimizer and the query adapter will accept the query (Step 4). Then the query adapter will consult the constraint repository (Step 5) and check whether there are constraints related to the query (Step 6). If the query adapter finds that there are some applicable constraints in the constraint repository, the constraints will be fetched out and compared with the query condition. When the query adapter finds that there is conflict between the query condition and the constraints so that the query will return empty result, a message will be generated (Step 8) and the user will be informed (Step 9). When there is no conflict, the query adapter will try to find whether there is possibility to optimize the query using the constraint information. If there exists the possibility, the query adapter will generate a new query and send it to the query processor of the data integration system (Step 7). More details about the query adapter will be given in section 3.3.

3 COMPONENTS

3.1 Constraint Wrapper

In order to make the semantic query optimizer understand, the constraints from typically heterogeneous data sources should be translated into constraints expressed at a global level in a global schema. Our constraint wrappers borrow the ideas from data integration wrappers. We have relational constraint wrapper for relational constraint mapping, XML wrapper for XML files, etc. We also provide a special way for those data sources which lack constraint mechanisms to submit constraints. These data sources include web services, HTML files, text files, etc. We permit them to submit predicate-like constraints. Constraint mapping depends on schema mapping. When translating constraints, the data integration system must provide schema mapping information. After the translation by the constraint wrapper, the constraints are expressed in the global schema. The details of the constraints in a global schema are given in (Lu and Mitschang, 2007).

3.2 Constraint Repository

Constraint repository is the general constraint storage, where the local constraints expressed in global schema (translated by the constraint wrapper), the global domain constraints and the global referential constraints are stored. The global domain constraints which spread multiple data sources and the global referential constraints between the data sources are defined, for example, by the data integration system administrator.

3.3 Query Adapter

Query adapter consists of four components: query decomposer, constraint fetcher, conflict detector, and query reformulator, as illustrated in Figure 2. Generally, the query adapter accepts an XML query from the user as input. Then it uses the semantic knowledge stored in the constraint repository to generate a semantically equivalent but more efficient query.

Query Decomposer. The query decomposer takes an XML Query from the user as input and decomposes the query into three parts: condition part, data source part and result return part. It forwards the query condition part and the data source part to constraint fetcher (Step 1), the query condition part to conflict detector (Step 2), and the whole query to query reformulator (Step 3).

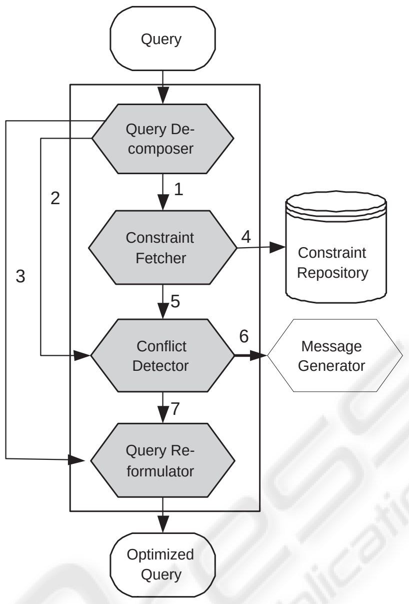

Figure 2: Components and Processing of Query Adapter.

Constraint Fetcher. The constraint fetcher searches the constraint repository to find the related constraints (Step 4) and forwards the constraints to conflict detector (Step 5).

Conflict Detector. The conflict detector takes query condition and the constraints as input. The constraints and the query condition can be arbitrary literals. Both the query condition part and the constraints are transformed into DNF (Disjunctive Normal Form). Conflict detector builds a constraint DNF tree. Then it uses the constraint DNF tree to evaluate the query condition. If there is a conflict, it will inform the message generator and an error message is sent to the user (Step 6). If there is no conflict, query reformulator will be invoked (Step 7).

Query Reformulator. If the conflict detector finds no conflict, the query reformulator will check whether there exist possibilities to eliminate joins or predicates. Query reformulation rules are discussed in section 4.

4 QUERY REFORMULATION RULES

There are five SQO techniques most often discussed in literature: predicate introduction, join introduction, predicate elimination, join elimination, and detection of empty results (Cheng et al., 1999). The first two are based on the index mechanism of RDBMS. Due to the fact that in an XML-based data integration system the data sources are often non-relational and most of them do not have an indexing mechanism, these two techniques are not considered. Join elimination, predicate elimination and detection of empty results are used as the premier query reformulation rules in our optimizer.

Detection of Empty Results. When the conflict detector detects that there is a conflict between constraints and the query so that the query will return an empty result set, the messager generator will inform the user about the detection result.

Predicate Elimination. The main idea of predicate elimination is that if a predicate is known by the constraints to be true, it can be eliminated from the query. We use the constraint DNF tree and the query condition DNF tree again to test whether there exists possibility to eliminate predicates. If the domain predicate of query condition is subsumed as true by each leaf in the constraint DNF tree, we can eliminate the domain predicate from the query condition.

Join Elimination. The main idea of join elimination is that when a query contains a join for which the result is known a priori, it does not need to be evaluated, for example, when the two attributes of the join are related by a referential integrity constraint. The query transformer takes the selected referential constraints and the query condition as input to see whether there exist redundant joins. The first step is to extract the attributes in the query condition part. Then the query reformulator will find all the data sources that have these attributes. The query reformulator will search the constraints which are filtered out according to the query condition and result part. When there are referential constraints, the query reformulator will build a reference chain, where the father source is the source the primary key belongs to, and the child source is the data source the foreign key belongs to. The query reformulator will search the reference chain and extract all the attributes from the query return part. If there is no attribute from the first data source in the query return part and there is no attribute from the first data source in the query condition part except the primary key in the join, the join between the primary key and the foreign key will be eliminated.

Table 1: Performance of Detection of Empty Results.

Q	DV	OrT(s)	OpT(s)	RB
Q1	1000	0.059	0	90.77%
	5000	1.112	0	95.77%
	15000	2.234	0	97.90 %
Q2	1000	0.875	0	94.06%
	5000	3.906	0	98.67%
	15000	12.469	0	99.58 %

5 EXPERIMENTS

We used BEA Weblogic as the application server, BEA LiquidData as the data integration system (BEA-Systems-Inc, 2003), Tamino XML Server (Software-AG, 2006) as constraint repository, Apache Tomcat (The-Apache-Software-Foundation, 2006) to run the web service. We used DB2 as relational data source and XML files as another non-relational data source to carry out our experiments. Five data sources are used to participate in the whole experiments:

- A DB2 database *CSCO*, with two tables: *EuropeCustomer* and *EuropeCustomerOrder*, referenced by *CustomerID*;
- A DB2 database *CSC*, with one table *EuropeCustomerR*, the replication of *EuropeCustomer* in *CSCO*;
- A DB2 database *CSO*, with one tabble *EuropeCustomerOrderR*, the replication of *EuropeCustomerOrder* in *CSCO*. There is a global referential constraint: two replication tables are referenced by *CustomerID*.
- An XML file *AmericanCustomer.xml*.
- An XML file *AmericanCustomerOrder.xml*. There is a global refential constraint: two XML files are referenced by *CustomerID*.

5.1 Performance

We use 1000, 5000, and 15000 records respectively in the data sources for testing. We use the following abbreviations in the performance tables: Q: Query; DV: Data Volume; OrT: Original Time(s); OpT: Optimized Time(s); RB: Relative Benefit(%).

We design two queries, *Q*1 and *Q*2, which are built to return empty result by our optimizer. *Q*1 is queried over only the relational database *CSCO*, while *Q*2 is queried over only the XML file *AmericanCustomerOrder.xml*. The performance results are shown in Table 1.

We design three queries: *Q*3, *Q*4 and *Q*5, whose joins are eliminated by our optimizer.

Table 2: Performance of Join Elimination.

Q	DV	OrT(s)	OpT(s)	RB
Q3	1000	0.922	0.314	50.65%
	5000	1.982	0823	51.36%
	15000	5.712	2.609	51.86 %
Q4	1000	1.687	0.272	75.52%
	5000	4.489	0.92	76.36%
	15000	11.953	2.588	77.17 %
Q5	1000	1.86	0.797	48.92%
	5000	7.672	3.714	49.60%
	15000	23.266	11.487	49.97 %

Table 3: Performance of Predicate Elimination.

Q	DV	OrT(s)	OpT(s)	RB
Q6	1000	0.731	0.603	11.35%
	5000	1.724	1.412	15.49%
	15000	2.812	2.241	18.71 %
Q7	1000	1.02	0.812	15.39%
	5000	4.945	3.701	24.13%
	15000	15.301	10.885	28.53 %

*Q*3 is queried only over the relational database *CSCO*, *Q*4 is queried over two relational databases, *CSC* and *CSO*, and *Q*5 is queried over two XML files, *AmericanCustomer.xml* and *AmericanCustomerOrder.xml*. The performance results are shown in Table 2.

We design two queries, *Q*6 and *Q*7, whose predicates are partially eliminated by our optimizer. *Q*6 is queried over the relational database *CSC* and *Q*7 is queried over the XML file *AmericanCustomer.xml*. The performance results are shown in Table 3.

From the performance tables we can see that the larger the data volume is, the more improvement we obtain.

5.2 Discussion

When the queried data sources are only RDBMSs, it is possible that after join elimination or predicate elimination the eliminated joins or predicates include attributes which are indexed in the RDBMSs. Exploiting the index typically enhances query performance. If that predicate or join is eliminated, this efficient access might not be chosen anymore. Our optimizer might generate a suboptimal query. Again in the experiments we find that when the query is related only to one RDBMS and we submit the same query for many times, the query execution becomes always quicker. This is because of the caching mechanism of RDBMS. So we conclude that when the underlying data source of the query is only one RDBMS, applying our optimizer might decrease the execution efficiency.

We conclude, that it is not worth applying our optimizer when the data volume is small and when the query execution cost is expected to be low. However, when the data volume is large or when the query execution cost is expected to be high, our optimizer becomes very useful. Normally in the data integration system, most of the data sources are not RDBMSs, so our optimizer works very well. The reason is that the non-relational data sources typically lack indexing mechanism and normally the query execution cost is very high.

6 RELATED WORK

In (Shenoy and Ozsoyoglu, 1989) a user specified query is optimized by describing a scheme to utilize semantic knowledge. The semantic is represented as function-free clauses in predicate logic. The scheme uses a graph theoretic approach to identify redundant joins and restrictions in a given query. An optimization algorithm is presented which eliminates redundant nonprofitable specifications. Relationships among entity schema, semantics and query form the basis of the algorithm. This work uses subset constraints and implication constraints, while we use integrity constraints. This work uses the index of a database while we do not rely on indexes.

An implementation of predicate introduction and join elimination in DB2 universal database is described in (Cheng et al., 1999). Only referential constraints and check constraints are used. This work is targeted to DB2 and ours is to XML-based DIS. In addition, it depends on the indexing mechanism of DB2.

A system focusing on semantic query optimization for query plans of heterogeneous multi database systems is presented in (Hsu and Knoblock, 2000). This approach reduces the cost of query plans generated by an information mediator by eliminating unnecessary joins in a conjunctive query of arbitrary join topology. The optimization here is targeted to query plan generated by the mediator while ours is targeted to the query generated by the users. This optimizer must be embedded in the query mediator while ours is built on top of the query mediator and thus provides better extendability and flexibility.

An intelligent system using tool-supported techniques to optimize mediated queries is proposed by (Beneventano et al., 2001). The techniques rely on the availability of integration knowledge including: local source schemata, a virtual mediated schema and its mapping descriptions while we rely on the integrity constraints.

An approach in which an ontology with a database is exploited to capture semantics for the query transformation process is proposed by (Necib and Freytag, 2005). A set of rewriting rules relying on semantic information extracted from the ontology associated with the database are developed. This approach allows to rewrite a query into another one which is not necessary equivalent but can provide more meaningful result satisfying the user's intention while our approach aims at providing a semantically equivalent query. This approach can only be applied in a single database while ours can be used in a data integration system.

7 CONCLUSIONS

In this paper we present a semantic query optimizer for XML-based data integration systems. Constraints from different data sources are translated into constraints expressed in global schema through constraint wrapper and stored in the constraint repository. Global constraints can be defined and stored too. Queries submitted by the users will be optimized semantically in the semantic query optimizer using constraints in the constraint repository. We implement three semantic query optimization techniques: detection of empty results, join elimination, and predicate elimination. We carry out experiments to test the performance. The results are very promising. We analyze the performance issues and conclude that the semantic query optimizer works best when the underlying data sources are mixed, i.e., not purely relational DBMSs, and when the data volume is large, as typically is the case in real world scenarios.

REFERENCES

Abiteboul, S., Segoufin, L., and Vianu, V. (2001). Representing and querying xml with incomplete information. In *Proceedings of the Twentieth ACM Symposium on Principles of Database Systems*. ACM Press.

BEA-Systems-Inc (2003). Bea liquiddata for weblogic, building queries and data views. In *http://edocs.bea.com/liquiddata/docs81/querybld/index.html*.

Beneventano, D., Bergamaschi, S., and Mandreoli, F. (2001). Extensional knowledge for semantic query optimization in a mediator based system. In *International Workshop on Foundations of Models for Information Integration*.

Carey, M., Florescu, D., Ives, Z., Lu, Y., Shanmugasundaram, J., Shekita, E., and Subramanian, S. (2000). Xperanto: Publishing object-relational data as xml. In *Proceedings of the Third International Workshop on the Web and Databases (WebDB)*.

Carey, M. and the BEA AquaLogic Team (2006). Data delivery in a service-oriented world: The bea aqualogic data services platform. In *Proc. of ACM SIGMOD Conf. on Management of Data*. ACM Press.

Chakravarthy, U., Grant, J., and Minker, J. (1990). Logic-based approach to semantic query optimization. In *ACM Transactions on Database Systems (TODS)*. ACM Press.

Cheng, Q., Gryz, J., Koo, F., Leung, T. Y. C., Liu, L., Qian, X., and Schiefer, K. B. (1999). Implementation of two semantic query optimization techniques in db2 universal database. In *Proceedings of the 25th International Conference on Very Large Data Bases*.

Fernandez, M., Tan, W., and Suciu, D. (2000). Silkroute: Trading between relations and xml. In *9th International World Wide Web Conference*.

Hsu, C. and Knoblock, C. A. (2000). Semantic query optimization for query plans of heterogeneous multi-database system. In *IEEE Transactions on Knowledge and Data Engineering, VOL. 12, NO. 6*. IEEE Press.

Huhns, M. and Singh, M. (2005). Service-oriented computing: Key concepts and principles. In *IEEE Internet Computing, 1(9)*. IEEE Press.

Lu, J. and Mitschang, B. (2007). Dis-cs: Improving enterprise data integration by constraint service. In *ISCA 20th International Conference on Computer Applications in Industry and Engineering (CAINE 07)*.

Necib, C. B. and Freytag, J. C. (2005). Semantic query transformation using ontologies. In *9th International Database Engineering and Application Symposium (IDEAS 2005)*. IEEE Press.

Ouzzani, M. and Bouguettaya, A. (2004). Query processing and optimization on the web. In *Distributed and Parallel Databases*. Kluwer Academic.

Papakonstantinou, Y. and Vassalos, V. (2002). Architecture and implementation of an xquery-based information integration platform. In *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, vol. (25)*. IEEE Press.

Shenoy, S. T. and Ozsoyoglu, Z. M. (1989). Design and implementation of a semantic query optimizer. In *IEEE Transactions on Knowledge and Data Engineering. Vol 1, No 3*. IEEE Press.

Software-AG (2006). Number one in xml management: Tamino xml server. In *Technical Factsheet*.

The-Apache-Software-Foundation (2006). Apache tomcat 6.0. In *http://tomcat.apache.org*.