In *Proc. 8th International Conference on Database Systems for Advanced Applications (DASFAA 2003)*, Kyoto, Japan, March 2003.

Scalable View Expansion in a Peer Mediator System

Timour Katchaounov
Uppsala University
Timour.Katchaounov@it.uu.se

Vanja Josifovski
IBM Almaden Research Center
vanja@us.ibm.com

Tore Risch
Uppsala University
Tore.Risch@it.uu.se

Abstract

To integrate many data sources we use a peer mediator framework where views defined in the peers are logically composed in terms of each other. A common approach to execute queries over mediators is to treat views in data sources as 'black boxes'. The mediators locally decompose queries into query fragments and submit them to the data sources for processing. Another approach, used in distributed DBMSs, is to treat the views as 'transparent boxes' by importing and fully expanding all views and merge them with the query. The black box approach often leads to inefficient query plans. However, in a peer mediator framework full view expansion (VE) leads to prohibitively long query compilation times when many peers are involved. It also limits peer autonomy since peers must reveal their view definitions. We investigate in a peer mediator framework the tradeoffs between none, partial, and full VE in two different distributed view composition scenarios. We show that it is often favorable with respect to query execution and sometimes even with respect to query compilation time to expand those views having common hidden peer subviews. However, in other cases it is better to use the 'black box' approach, in particular when peer autonomy prohibits view importation. Based on this, a hybrid strategy for VE in peer mediators is proposed.

1. Introduction

There has been substantial interest in using the mediator/wrapper approach for integrating heterogeneous data [14, 25, 11, 22, 7]. Most mediator systems integrate data through a central mediator server accessing one or several data sources through a number of 'wrapper' interfaces that translate data to a *common data model* (CDM). However, one of the original goals for mediator architectures [27] was that mediators should be relatively simple autonomous distributed software modules that encode domain-specific knowledge about data and share abstractions of that data with higher layers of mediators or applications. Composite mediators would then be defined in terms of other mediators and data sources through a high-level declarative language.

Compositionality of mediators allows to reuse available distributed resources on the Internet and to create new value-added mediation services in terms of existing ones, while the autonomy of the sources and mediators is preserved. In the observable future it is most likely that data integration will be mostly a manual task. In order to scale integration to multiple autonomous sources, it is important that this task can be distributed among many parties with varying domain knowledge. We believe that a mediator architecture based on compositions of autonomous mediators is necessary to build large-scale data integration systems that are easy to tailor to existing infrastructure.

This paper investigates what are the implications of logical composition of distributed mediators on query compilation and execution performance and proposes a query processing technique suitable for the efficient execution of queries over composite mediators.

For our implementation we use the AMOS II peer mediator system [24]. To achieve modularity and distribution each mediator is an autonomous object-relational DBMS with its own query processor, storage, and catalog. AMOS II peers share many of the characteristics of peer-to-peer systems. AMOS II peers are autonomous because there is no global schema or global coordinator. Every mediator peer can act both as a client and a server to any number of other mediators. AMOS II peers communicate over the Internet via query compilation, query costing, view expansion and query execution requests in order to cooperatively process queries over composite mediators.

Mediator composition is based on a multidatabase query language that allows mediator peers to transparently access

views, tables, and functions from remote mediators or data sources [23]. Logical composition of mediators is achieved when *multidatabase views* are defined in terms of views, tables, and functions in other mediators or data sources. Multidatabase views make groups of mediator peers and data sources appear to the user as a single virtual database.

There are two traditional approaches to implement distributed information systems. The first is the *black box* approach where distributed modules communicate with each other through some protocol without revealing the implementation of the services they export. This is the approach used in CORBA based systems and web services based on SOAP [3] and WSDL [4]. In the AMOS II peer mediator architecture the black box approach is equivalent to not to expand external views at all. It is common knowledge that this may lead to suboptimal query execution plans *(QEPs)* because of missed optimization opportunities.

On the other end is the *full view expansion* (*transparent box*) approach in distributed DBMS, where all views are expanded and merged with the query [21], independent of the location of the base tables and views that are used in a view definition. This 'reveals' to the query compiler the information 'hidden' in the view definitions which allows for better QEPs. Full view expansion could also remove unnecessary access to mediator peers. However, in a large scale peer mediator system using a cost-based query optimizer, full view expansion leads to prohibitively high compilation cost. Furthermore, full view expansion can only be made when permitted by the peer, to respect its autonomy.

We generalize both approaches and treat external mediator views as *grey boxes*, that is, when multidatabase views are defined in terms of other multidatabase views some of the view definitions are revealed to remote clients that query the views. We do this through a new query compilation technique for peer mediators, *distributed selective view expansion (DSVE)*. In DSVE, for better overall performance, mediators control the level of transparency of the mediator peers by selectively expanding some multidatabase views.

To analyze the performance of DSVE we implemented two data integration scenarios scaled to up to 19 distributed AMOS II mediators with up to 12 commercial RDBMS data sources. As reference points we use the black box and the full view expansion approaches. We investigated the performance for both reference approaches under varying level of transparency and with respect to both query compilation and execution times. The analysis shows that DSVE can support the logical composition of mediators with little overhead and that this approach is superior to both black and transparent box approaches.

The rest of the paper is organized as follows. Section 2 investigates related work. Section 3 introduces the scenarios that are used throughout the paper. Section 4 describes the principles of DSVE and Section 5 investigates its performance followed by summary and future work in Section 6.

2. Related work

Distributed databases [1, 21] have complete global schemas describing on what sites different (fractions of) tables are located, while peer mediators do not have complete knowledge of meta-data from all mediators and data sources. Full expansion of all possible views in a distributed system with many nodes may be very costly. In [20] a restricted view expansion strategy for the System R* distributed database [5] is briefly mentioned but not evaluated.

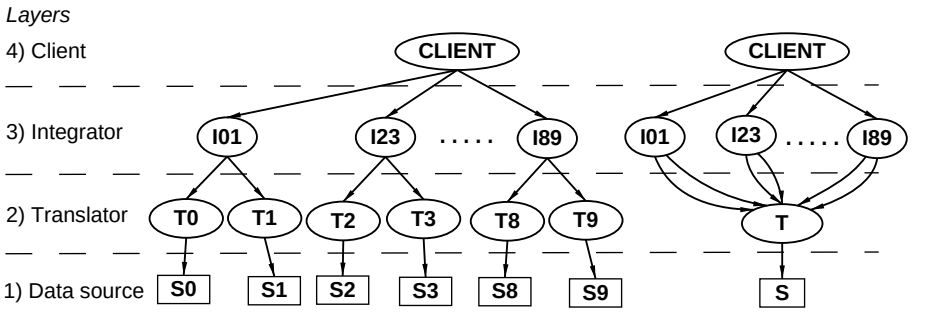
To the best of our knowledge, there is no other study of the effects of a varying degree of view expansion in a distributed mediator or database system. No other mediator system (e.g. [14, 25, 7, 8, 22, 19]) use distributed view expansion.

The peer *peer data management system* (PDMS) architecture in [13] differs from ours by having a centralized catalog and therefore it is closer to a DDBMS. That work concentrates on data placement for PDMS. In [2] a data model suitable for PDMS is presented. Neither of the PDMS works studies query processing performance. Based on the similarity of PDMS with our peer mediator architecture, our results are readily applicable to the PDMS architecture.

Peer-to-peer (P2P) systems and *web services* have addressed the creation of large-scale integrated systems on the Internet. P2P systems, e.g Gnutella [12] and Freenet [10], address the problem of large scale sharing and replication of simple information objects such as files. P2P systems provide simple keyword search capabilities and do not support high-level abstractions as views. Most of the work on large-scale composition of distributed systems on the Internet is performed in the context of web services [6]. Problems related to composition of services are usually investigated from the perspective of workflow composition [26]. Our focus is on data integration and not on workflow/process composition. Web services are based on the SOAP [3] and WSDL [4] standards which provide no means for view definition exchange. Thus current proposals for composed web services treat wrapped DBMS views as black boxes.

3. Mediator composition scenarios

Having a potentially unlimited number of ways to compose mediators, we implemented for our study two scenarios that are simple enough to analyze the performance implications of view expansion in a peer mediator system. Our choice of scenarios assumes that data integration is performed with no global control or knowledge. Users define peer mediator views in terms of views in other mediators without knowing how those remote views are defined.



(a) Tree topology (b) Common sub-mediator topology

Figure 1. Logical compositions of mediators

The integration scenarios are implemented using the AmosQL query language [24]. In this paper we define the scenarios in terms of equivalent SQL statements. Remote views defined in other mediators are referenced as $view@server$.

When participating in a logical composition, AMOS II peers can play several roles. *Translators* wrap different kinds of data sources and translate their data into the common data model (CDM) of AMOS II . *Integrators* reconcile conflicts and overlaps between similar real-world entities modeled differently in different sub-mediators [15, 16]. Users and applications can pose queries to any AMOS II peer, called the *client mediator* for the queries.

Scenarios. In the first scenario (Figure 1(a)) suppliers store information about parts in a RDBMS. Each supplier uses a translator that exports a view of the data. Several independent part resellers integrate information from the suppliers and present an integrated view hiding their information sources from their clients. A potential customer runs a mediator client that poses queries to the resellers' integrators.

In the second scenario (Figure 1(b)) the information about parts from all suppliers is stored in a single relational database. Each supplier has a single translator exporting the parts of that supplier. Each of the part resellers then exports an integrated view of the suppliers as in the first scenario. In a system with a global catalog such a scenario would look very artificial since the client mediator would discover in advance that all integrators access the same source of information. However in a peer system, this knowledge is not readily available. We assume that the integrators did not want to disclose their information source.

From the mediator client the two scenarios are equivalent and queries posed to the resellers' mediators would return exactly the same result. The differences are 'hidden' inside the view definitions of resellers' and suppliers' mediators.

Logical view integration graphs. To describe properties of mdiator compositions we define a *logical view integration graph (LVIG)* as a directed acyclic connected graph where vertices represent mediator peers or data sources and each directed arc represents the relationship *'is defined in terms of'* between a multidatabase view in one mediator and a view or table in another peer. Mediators are represented as ovals and data sources as rectangles. An LVIG represents a high-level view of the logical composition of mediators and data sources. Many distributed QEPs can be generated to compute the result of a query with the same LVIG.

The LVIGs of the two scenarios on Figure 1 differ in the topology of their LVIGs. Based on that we will name the first one as the *TREE* scenario and the second one as the Common Sub-Mediator (*CSM*) scenario.

3.1. Definitions of the mediators

The mediators and sources in the two scenarios are divided into four layers based on their roles:

The data source layer contains data stored in RDBMS. In the $TREE$ scenario the data for ten part suppliers is stored in different relational database tables, $PART$, each stored in its own DBMS S_i with the following schema:

```
CREATE TABLE part
(pnum integer not null,
 pname char(16) not null,
 quality integer,
 primary key(pnum))
```

In the CSM scenario all data about parts is stored in one relational database S in a single $PART$ table having one more column - a supplier id - and a composite key consisting of the part number and the supplier id. To simplify it is assumed that the same 'real' part has the same key $pnum$ in every relational source.

The translator layer consists of mediators providing views over the $PART$ tables. The translators T_i and T access the source data through an ODBC wrapper [9]. The

translators could be hosted by independent application service providers or data source owners. In the $TREE$ scenario there is one translator T_i per relational source S_i. In the CSM scenario the single relational source S is wrapped by the translator T. In addition, in T each part supplier has a view, $part_i$, that selects parts from that supplier.

The integrator layer defines reconciliation views over the $part$ views defined in the translator layer. All integrator views are defined through the template below, where $[i]$ and $[j]$ are replaced by the indexes of the integrated translators for the $TREE$ scenario, and the indexes of the $PART$ tables for the CSM scenario, respectively. Each scenario uses only one of the two $FROM$ clauses.

```
CREATE VIEW part@I[ij] as
SELECT p0.pnum, p0.pname
  combine_quality(p0.quality,
                  p1.quality)
  AS quality
/* TREE scenario: */
  FROM part@T[i] p0, part@T[j] p1
/* CSM scenario: */
  FROM part[i]@T p0, part[j]@T p1
WHERE p0.pnum = p1.pnum;
```

In the $TREE$ scenario each mediator I_{ij} integrates information about parts from two translators T_i and T_j in the first $FROM$ clause. The $pnum$ attribute of the view is defined as the $pnum$ property of one of the joined tables. The $quality$ property is defined by the user-defined $combine\_quality$ function that encapsulates the knowledge of how to combine part qualities from different sources.

The integrators I_{ij} in the CSM scenario combine views of parts from the same part suppliers as in the $TREE$ scenario. However all the views $part_i$ in the translator T are defined in terms of the same relational table $PART$ in S as reflected by the second $FROM$ clause of the template.

From the mediator client both scenarios are indistinguishable as they export exactly the same views. Nevertheless the sources of information of the integrators differ.

Finally, *the top layer* has one mediator $CLIENT$ through which users pose queries to the $part$ views defined in the integrators I_{ij}. Depending on the remote views referenced in a query the corresponding LVIG may look different. The LVIGs on Figure 1 correspond to queries that reference all five available integrators.

To investigate multidatabase view expansion with respect to the number of participating mediator peers we use a class of test queries over a varying number of $part@I_{ij}$ views. A sample query over the $part@I01$ and $part@I23$ views defined in the integrators $I01$ and $I23$ is shown in Figure 2. The $quality\_part$ query states *what are the high-quality parts known to the* $I01$ *and* $I23$ *integrators*, where the $quality$ property ranges from 1 to 10.

```
select p1.pname
from part@I01 p1, part@I23 p2
where p1.quality >= 7 and
      p2.quality >= 7 and
      p1.pnum = p2.pnum;
```

Figure 2. Query $quality\_parts$ over I01 and I23

The $quality\_parts$ query is scaled by adding more $part@I_{ij}$ views from other integrators through equi-joins on the $pnum$ attribute and inequality predicates on each $quality$ attribute.

4. Multidatabase view Expansion

First the black box approach to process queries over multidatabase views is described, followed by a discussion of its potential problems. To remedy the major deficiency of the black box approach, poor QEP quality, we describe how to extend the mediator query processor with a general mechanism for exchanging view definitions between the mediator peers. In its simplest form this mechanism is equivalent to full view expansion. After discussing the advantages and problems of full view expansion we describe what is needed to achieve the best of both worlds - a generalized approach to multidatabase view expansion that allows the query optimizer of each mediator peer to explore the full range of possibilities between no and full view expansion.

4.1. Processing multidatabase views as black boxes

Queries in AMOS II are parsed and rewritten [18, 9, 15, 16] into a typed predicate calculus representation, ObjectLog [18], extending Datalog with predicate type signatures. In this paper we use SQL notation. For local queries rewritten calculus expressions are transformed by a cost-based query optimizer into an optimized object algebra expression [18, 9] which is interpreted to produce the query result. For multidatabase queries, before the query optimization phase, the calculus representation of the query is decomposed into multidatabase subqueries. At each mediator peer its cost-based optimizer generates optimized QEPs for the each of the subqueries. The query decomposition is performed in two main stages [17]: heuristic-based *predicate grouping* and cost-based *subquery optimization*.

The predicate grouping groups the query predicates into one or more composite predicates (subqueries). The result is one or more subqueries per each remote peer. After the predicate grouping phase the query in Figure 2 is divided into two subqueries (views) $SQ@I01$ and $SQ@I23$ that consist of predicates from $I01$ and $I23$ (Fig. 3 and 4).

The *subquery optimization* phase decides on the execution order of the subqueries which determines the data flow

```
create view SQ@I01 as
select p0.pnum, p0.pname
from part@I01 p0
where p0.quality > 7;

create view SQ@I23 as
select p0.pnum
from part@I23 p0
where p0.quality > 7;
```

Figure 3. Subqueries after predicate grouping

```
select s0.pname
from SQ@I01 s0, SQ@I23 s1
where s0.pnum = s1.pnum;
```

Figure 4. $quality\_parts$ **after grouping**

between the mediators. The two subqueries $SQ@I01$ and $SQ@I23$ are sent for compilation and costing to the integrators $I01$ and $I23$ to determine variable bindings and execution order for the subqueries. Based on the binding and cost information an executable plan is produced for the query in the client mediator and the subqueries in their respective mediators. These optimized plans for given binding patterns are saved in the mediator databases. The same process is applied recursively for subqueires that are themselves multidatabase queries in their respective mediators. Notice that the client mediator does not 'know' (and does not have to know) that $part@I01$ and $part@I23$ are actually views.

Distributed data flow graphs. A useful tool to understand distributed QEPs is a graph that represents the flow of data during the execution of a multidatabase query. A *distributed query execution data flow graph (DDFG)* is a directed connected graph where the vertices in the graph represent mediator peers or data sources. There are two kinds of edges with respect to each vertex: *call* edges are out-edges that represent remote subquery execution requests (with optional parameters), *data* edges are the in-edges of a vertex representing the incoming flow of tuples that correspond to each request. All edges are numbered according to their execution order. DDFGs reflect only the distribution aspects of a query execution plan. Many DDFGs may correspond to a single multidatabase query.

For the $quality\_parts$ example query the black box approach to distributed query compilation described above generates DDFGs similar to those in Figure 5. All other DDFGs corresponding to the same query are different only in the order the nodes from the same layer are accessed. As one may expect the DDFGs on Figure 5 are very similar to the LVIGs for the same query in Figure 1. Thus the black box approach to query compilation produces QEPs that follow the logical view composition topology.

Advantages and disadvantages of the black box approach. Treating remote views as black boxes has some advantages. When remote views are not expanded a multidatabase view definition is often smaller and refers to fewer mediators than the expanded one. All the compilation effort spent to generate plans for the remote views can be reused because AMOS II stores precompiled parameterized views as functions that can be directly invoked. Therefore we can expect better compilation times when no views are expanded. Another advantage is that the integrators do not have to reveal their view definitions to the client mediator. This respects the autonomy of the mediators and the black box approach may be the only possible one if a peer mediator doesn't reveal view definitions to other mediators.

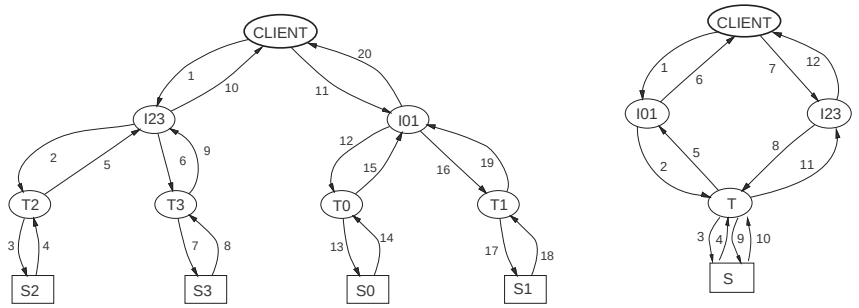
The main disadvantage of the black-box approach is that it can lead to suboptimal QEPs. In the context of a peer mediator system sub-optimality can be due to several reasons. A QEP may not be able to make use of hidden existing indexes in other mediators or sources. Similarly it is not possible to increase the selectivity of subqueries by merging predicates from remote views in different mediators. As in Figure 5 intermediate mediators are accessed despite that their view definitions do not access any local data. In deep mediator networks this may result in considerable network overhead and unnecessary load on mediators. In the case of queries with LVIGs having TREE topology the distributed subquery scheduler at each mediator peer has fewer options for distributed join ordering. For queries with LVIGs having CSM topology a client cannot detect that more than one of its sub-mediators access data from the same source as in the scenario on Figure 5(b).

4.2. Full expansion of multidatabase views

To solve the problems of the black box approach described in Section 4.1 a logical step is to follow the approach employed in modern DBMSs (distributed or not) - to fully expand all view definitions. For the $quality\_parts$ example query this implies that the definitions of the view $part$ in the integrators $I01$ and $I23$ should be revealed to the client mediator.

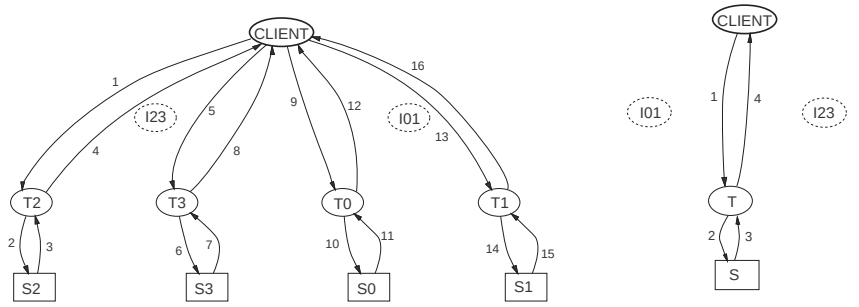
After collecting the expanded definitions of all the remote subqueries, the subqueries in the original query are replaced by their expanded definitions and all predicates are grouped into subqueries. The query processing continues with the cost-based subquery optimization phase in the same way as in the black box approach.

Figure 6 shows some possible DDFGs for the $quality\_parts$ query from Figure 2 after performing full view expansion. The $CLIENT$ mediator eliminates all redundant mediators (dotted circles). In the the CSM scenario in Figure 6(b) the view definitions at the two integrators are combined in a single query together with the query



a) Treee topology b) Common sub-mediator topology

Figure 5. DDFGs for query $quality\_parts$ generated by the black box approach



a) Treee topology b) Common sub-mediator topology

Figure 6. DDFGs for query $quality\_parts$ generated by the full view expansion approach

predicates and the translator T is accessed only once. When supported by the data sources the combined predicates can be pushed to the sources which may further improve performance.

While full view expansion is very promising in terms of potential benefits in execution time, the cost to compile queries over fully expanded views may be prohibitively high. An expanded remote view definition may reveal that it has been defined through many mediators thus resulting in an explosion of the number of peers the query optimizer must consider. For example if we scale our scenario to ten integrators, each of them having an integrated view over ten translators, full view expansion of a query over the ten integrators will lead to a distributed query involving a hundred peers. At the same time each of the subqueries of the distributed plan may contain many join predicates. As there is no global catalog in a peer mediator system the query optimizer must execute a remote cost estimate request for every query fragment that can be executed in a remote peer. This may result in a high cost of getting the cost. Finally due to incorrect cost estimates typical in a distributed mediator system the optimizer might still produce sub-optimal QEPs.

Finally, full view expansion does not respect mediator and source autonomy by forcing all mediators to reveal their view definitions. This makes full VE unsuitable for integration of data from independent information providers.

4.3. Selective expansion of multidatabase views

A natural idea is to generalize the processing of multidatabase views so that the query processor adapts itself to the query being compiled, the logical composition topology of the multidatabase views being queried, and the autonomy requirements of each mediator peer. Such a general approach should combine the good sides of both the black box and the full view expansion approaches: reasonable query compilation cost, good query execution performance, and respect of site autonomy.

We have implemented such a generalized mechanism in the AMOS II mediator system, named *distributed selective view expansion (DSVE)*. It allows to selectively expand only some of the multidatabase views. DSVE is generic in the sense that it allows various strategies to be used to select which of the subqueries in a multidatabase query should be view expanded. In particular, when no remote views are expanded DSVE is reduced to the black box approach, and when all subqueries are expanded DSVE is equivalent to the full view expansion approach. We use the term *partial view expansion (partial VE)* for all other DSVE strategies.

To achieve good performance DSVE's view selection

strategy should expand views if it leads to high QEP quality improvement without dramatically increasing the optimization time for the expanded query. The DSVE strategy should scale well over the number of remote views. To preserve the autonomy of the mediator peers the strategy used in DSVE should require as little information as possible to be imported from mediator peers.

To investigate the tradeoffs between compilation time and QEP quality with varying number of expansions, we start with a family of simple strategies where each strategy performs a fixed number, $NExp$, of expansion requests per query. When $NExp$ is equal or bigger than the total number of subqueries, DSVE is equivalent to full view expansion. If $NExp = 0$ DSVE reduces to the black box approach. Let us denote each of these strategies as $DSVE\_N$. Figure 7 shows the resulting DDFG after the compilation of the $quality\_parts$ test query when the $DSVE\_1$ strategy was used to expand the view in integrator $I01$. In the following section we perform a set of experiments where we vary the $DSVE\_N$ strategy for both the TREE and CSM mediator composition scenarios from Section 3.

5. Experimental evaluation

The experimental goals are: *i)* quantify tradeoffs between no, full and partial VE; *ii)* test hypothesis that DSVE may lead to best overall performance; *iii)* understand properties of a DSVE strategy with good overall performance.

In all experiments we execute scaled versions of the test query $quality\_parts$ in Figure 2 for both scenarios. This allows us to include the topology of the LVIG of the query as a parameter in the experiments. We investigate the scalability of view expansion by varying number of expanded views and the number of integrators joined by the test query.

5.1. Experimental setup

We used three 600 MHz Dell Optiplex GX1 computers with 512 MB RAM running Windows 2000 interconnected by a fast 100 Mbit LAN. Each of the mediator layers (client, integrator, translator) run on separate computers. The query compiler of AMOS II generated synchronous QEPs allowing us to run several mediators on the same computer without any interference. During the experiments it was ensured that each of the nodes preallocates enough RAM to complete the experiment without swapping. All translators accessed a DB2 RDBMS through an ODBC wrapper. The $PART$ tables in the DB2 databases were populated with synthetic data, all with the same number of rows and even distribution of all join columns. All join columns of the $PART$ tables were indexed.

5.2. Compilation tradeoffs

First measurements investigate how the compilation time for a multidatabase query over multidatabase views depends on the number of view expansions for varying number of integrator views. Figures 8 and 9 show this dependency for LVIGs with TREE and CSM topology. Each point in the graphs corresponds to one compilation experiment. There is one curve per fixed number of expansions. Points with the same x-axis (same number of integrators) correspond to the same query compiled with different number of view expansions. The curves in the graphs partially coincide when the number of expansions are equal to or more than the total number of integrators. While our experiments were performed for all possible numbers of expansions between none and full, for clarity we removed some the experimental curves that do not change our conclusions.

The compilation cost of a multidatabase query is distributed among the components of the query compiler: the local query compiler and the distributed query optimizer. Both optimizer components use dynamic programming (DP) to find the optimal executable order of subqueries and the predicates in subqueries. Therefore query compilation cost depends exponentially both on the number of remote sub-queries and the number of predicates per sub-query.

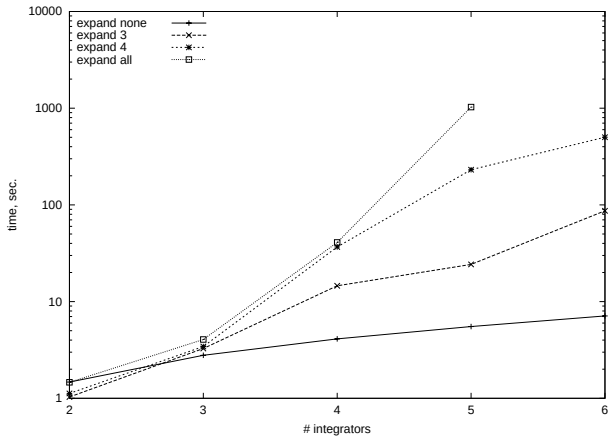
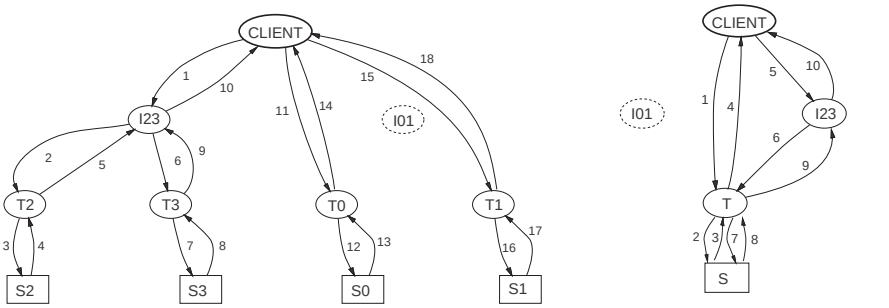


Figure 8. Query compilation times for different DSVE strategies, $TREE$ topology

Figure 8 shows experimental results for queries with TREE topology LVIGs. The y-axis of the graph is in logarithmic scale because of high value ranges. As expected, the more expansions are performed, the longer compilation time. Full VE expansion leads to exponentially growing compilation time and for 5 integrators it is 186 times more than with no VE. For 6 integrators and full VE (curve $expandall$) the experiment could not complete in 10000



a) Treee topology b) Common sub-mediator topology

Figure 7. DDFGs for query $quality\_parts$ generated by the $dsve_1$ strategy

seconds. Two factors contribute to the exponential behavior of full VE: *i)* DP is used to find optimal execution order of the remote subqueries; *ii)* in our scenario each expansion of a view on the integrator level reveals two more views from the translator level, thus increasing the distributed query optimizer search space. All other strategies result in compilation times between the two naive strategies: black box (curve $expandnone$) and full VE (curve $expandall$).

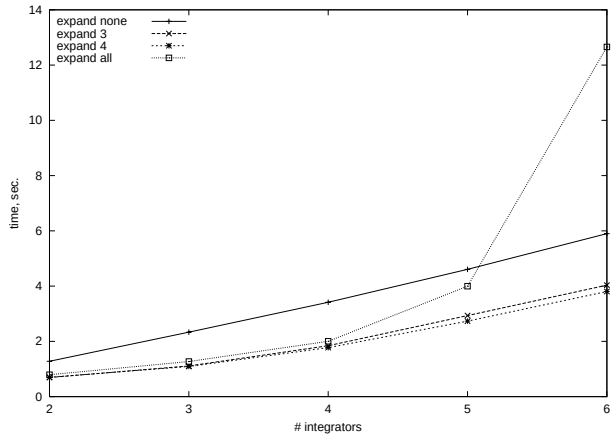


Figure 9. Query compilation times for different DSVE strategies, CSM topology

The experiments for queries with CSM LVIG topology (Figure 9) uncover completely different behavior than with TREE topology. The total time to compile the worst case of 5 integrators is 257 times less than with the TREE topology. Contrary to the common belief that the more views are expanded, the higher compilation cost, here we observe the opposite behavior up to 5 integrators: the more views are expanded, the less compilation time. This unexpected result is due to savings both in the local and the distributed query optimizer components. When expanded, the views on the integrator level reveal that they are defined in terms of the same mediator, the translator CSM on Figure 1(b). After all expanded views are merged and their predicates are grouped into a single subquery (executed at the translator T) it is simplified by query rewrites (Section 4.2). As a result the distributed sub-query optimizer at the client mediator has fewer predicate groups to optimize (just one) while the number of predicates for the local optimizer does not grow. After the number of integrators grows over 5, full VE leads to slower compilation time due to the large number of relational sources being accessed by the large subquery resulting from the view merge. This subquery is compiled in the translator T and increases the compilation time there.

We conclude that the more distinct sub-views are revealed by VE, the higher is compilation cost, and the dependency is exponential in the worst case. Furthermore if DP is used for query optimization full VE becomes too expensive when it results in more than 9 to 10 distinct subviews. Finally, expansion of views with a common submediator does not increase compilation time dramatically, and in some cases it may result in lower compilation time.

5.3. Execution plan quality

The next step in our evaluation of VE is to check two hypotheses made earlier: *i)* the more views are expanded the better the quality of the resulting QEP and *ii)* partial VE leads to sufficiently good plans with low compilation costs. Figure 10 represents the execution time of the test query $quality\_parts$ in Figure 2 scaled to 5 integrator views where all $PART$ tables contain 6000 tuples. The test query is precompiled for both LVIG topologies ($TREE$ and CSM) with varying number of view expansions resulting in different QEPs. The number of expansions varies between 0 (black box) and 5 (full VE). The quality of the QEPs is evaluated by measuring their actual running time.

For both topologies we observe improvement in the QEP quality in Figure 10 when the number of expansions grows. This confirms assumption *i)*. Notice that full VE improves the plan quality in the $TREE$ topology with only 24%

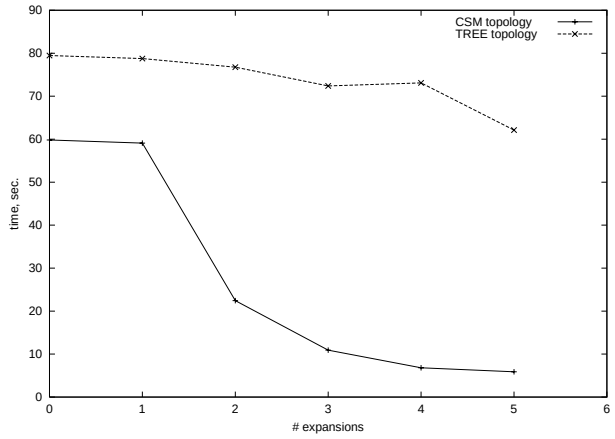


Figure 10. Plan quality for 5 integrators

number of expansions	0	1	2	3	4	5
TREE						
abs. comp. (sec.)	5.5	5.7	9.5	24.3	231.3	983.6
rel. comp.	1	1.04	1.7	4.4	41.8	177.9
improvement	1	1.01	1.04	1.1	1.1	1.3
rel. cost for improvement	1	1.03	1.66	3.99	38.38	139.01
CSM						
abs. comp. (sec.)	4.6	4.3	3.5	2.9	2.7	3.8
rel. comp.	1	0.9	0.8	0.6	0.6	0.8
improvement	1	1.01	2.7	5.5	8.8	10.2
rel. cost for improvement	1	0.91	0.28	0.12	0.07	0.08

Table 1. Compilation cost vs quality

while in the CSM topology the improvement is 10 times.

Table 1 compares the ratio between the relative time to compile a query with varying number of expansions and the corresponding relative quality improvement for the experiment on Figure 10. The table consists of two similar parts, one for the test query being compiled and run against a LVIG with $TREE$ topology, and one for the CSM toplogy. For each topology the first row [abs. comp.] shows the absolute times to compile the query, the next row [rel. comp.] shows the compilation times relative to the time for 0 view expansions. The row [improvement] shows the ratio of the execution time with no expanded views (as a worst case) to all execution times from Figure 10. Finally the row [rel. cost for improvement] shows the ratio between the [rel. comp.] cell and the [improvement] cell which is an estimate of how much did it cost in compilation time to achieve an improvement in the quality of the QEP.

For the $TREE$ topology the last row [rel. cost for improvement] shows that the more views we expand the more costly it is to improve the quality of the QEP while at the same time from row [improvement] we can see that even with full view expansion (5 expansions) we achieve only minor improvement of 1.28 times (22%) for which it took 177.9 times longer (983.6 seconds) to compile the query. In this case a good tradeoff is to perform partial expansion of 3 integrator views which takes only 4.4 times longer (24.3 sec.) to achieve 1.1 times (9%) improvement. Therefore in the case of a $TREE$ topology partial VE produces a better plan with relatively low cost, while full VE leads to prohibitively high cost for plan improvement which confirms assumption *ii)*. We can also notice that even with no VE at all the resulting QEP is pretty good.

The compilation and execution of the test queries in the CSM topology exposes radically different behavior. Partially expanding 3 integrator views improves the plan quality 5.5 times where the compilation time is 60% of the time for the non-expanded case. Therefore assumption *ii)* is true in the case of CSM topology as well. Full VE in this case leads to 10.2 times improvement in the quality of the QEPs which requires less time (only 80%) than with no VE.

The conclusions are that in the general case partial VE produces sufficiently good plans with relatively low compilation cost. If we know that we are compiling a query over views with a $TREE$ topology of the LVIG, the compilation cost can be radically reduced by not expanding any views at all without sacrificing the quality of the QEP. By contrast, when compiling queries against views with CSM topology, full VE can lead to radical improvements in the quality of the QEPs with very low compilation cost.

6. Conclusions and future Work

We proposed a new approach, *distributed selective view expansion (DSVE)*, to process compositions of multidatabase views in a peer mediator system. In DSVE, some of the views defined in remote mediators are selectively expanded to balance between query compilation time and QEP quality for best overall performance. To minimize the number of expansion requests and to allow optimizations of the expanded remote views DSVE uses predicate grouping to combine query predicates into subqueries. We present a performance study of DSVE with respect to its scalability over the number of remote views both for query compilation and query execution. As a reference we use two traditional approaches, the black box and the full VE approach which are special cases of DSVE.

The experiments show that neither of the two reference approaches (black box and full VE) is suitable for a peer mediator system, because none of them performs well in all cases. Contrary to the common belief that VE is always beneficial, our experiments show that it is not favorable to always perform full VE because in some cases it leads to very high compilation costs without radical improvements

in query execution time. In LVIGs with $TREE$ topology VE increases the number of views directly visible to a client node, and given that cost estimates are highly unreliable in a peer mediator system, this often results in suboptimal plans. Therefore VE for $TREE$-like LVIGs defeats its own purpose - to improve the quality of the QEPs. On the contrary, more view expansions for queries with CSM-like LVIGs result in compilation times orders of magnitude lower than in a $TREE$-like LVIG, while the quality of the plans improves up to 12 times. In the case of CSM-like LVIG topologies VE can drastically reduce the query execution time when information from several hidden sub-mediators can be combined. The topology of the LVIG of a multidatabase query plays a crucial role in the VE process. For $TREE$ topologies the best strategy is to expand only few of the remote views while for the CSM topology all (or almost all) views should be expanded.

The performance improvements of DSVE in processing queries over logically composed mediators are due to more selective queries, smaller data flows between the servers, fewer servers involved in the query execution, while spending relatively little effort in query compilation. Our performance study shows that DSVE allows for efficient query processing in logically composed mediators.

We are currently designing a view expansion strategy for DSVE that selects for expansion the views most likely to lead to an improved QEP with low compilation cost. Such a strategy should utilize the information hidden in the topology of the LVIG to leverage the common view definitions for better plans and lower compilation cost. A DSVE strategy should also evaluate the potential number of remote subqueries it will produce for the distributed optimizer and take into account the total number of predicates per subquery to reduce the distributed and local query compilation costs.

References

[1] P. M. G. Apers, A. R. Hevner, and S. B. Yao. Optimization algorithms for distributed queries. *IEEE Transactions on Software Engineering*, 9(1):57–68, January 1983.

[2] P. A. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu. Data management for peer-to-peer computing: A vision. *Proc. 5th Intl. Workshop on the Web and Databases, WebDB 2002*, Madison, Wisconsin, June 2002.

[3] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer. *Simple object access protocol (SOAP) 1.1*. W3C Note, http://www.w3.org/TR/SOAP/, May 2000.

[4] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. W3C Note, http://www.w3.org/TR/wsdl, March 2001.

[5] D. Daniels, P. G. Selinger, L. M. Haas, B. G. Lindsay, C. Mohan, A. Walker, and P. F. Wilms. An introduction to distributed query compilation in R*. *Proc. of the 2nd Intl. Symposium on Distributed Data Bases*, 291–309, Berlin, September 1982. North-Holland Publishing Company.

[6] *IEEE Data Engineering Bulletin*. Special issue on infrastructure for advanced E-services. 24(1), March 2001.

[7] W. Du and M. Shan. Query processing in Pegasus. In O. Bukhres, A. Elmagarmid (eds.):*Object-Oriented Multidatabase Systems: A Solution for Advanced Applications*. Pretince Hall, Englewood Cliffs, 1996.

[8] C. Evrendilek, A. Dogac, S. Nural, F. Ozcan: Multidatabase Query Optimization. *Distributed and Parallel Databases*, Kluwer, 5(1), 77-114, 1997.

[9] G. Fahl and T. Risch. Query processing over object views of relational data. *VLDB Journal*, 6(4):261–281, 1997.

[10] *Freenet.* http://freenet.sourceforge.com/.

[11] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. D. Ullman, V. Vassalos, and J. Widom. The TSIMMIS approach to mediation: Data models and languages. *Journal of Intelligent Information Systems*, 8(2):117–132, April 1997.

[12] *Gnutella.* http://www.gnutella.com/.

[13] S. Gribble, A. Halevy, Z. Ives, M. Rodrig, and D. Suciu. What can databases do for peer-to-peer? *Proc. 4th Intl. Workshop on the Web and Databases, WebDB 2001*, 31–36, June 2001.

[14] L. M. Haas, D. Kossmann, E. L. Wimmers, and J. Yang. Optimizing queries across diverse data sources. *Proc. 23rd Intl. VLDB Conf.*, 276–285, Athens, Greece, August 1997.

[15] V. Josifovski and T. Risch. Functional query optimization over object-oriented views for data integration. *Journal of Intelligent Information Systems*, 12(2-3):165–190, 1999.

[16] V. Josifovski and T. Risch. Integrating heterogenous overlapping databases through object-oriented transformations. *Proc. of 25th Intl. VLDB Conf.*, 435–446, Edinburgh, Scotland, UK, September 1999.

[17] V. Josifovski and T. Risch. Query decomposition for a distributed object-oriented mediator system. *Distributed and Parallel Databases*, 11(3):307–336, May 2002.

[18] W. Litwin and T. Risch. Main memory oriented optimization of oo queries using typed Datalog with foreign predicates. *IEEE Transactions on Knowledge and Data Engineering*, 4(6):517–528, 1992.

[19] L. Liu and C. Pu. An adaptive object-oriented approach to integration and access of heterogeneous information sources. *Distributed and Parallel Databases*, 5(2):167–205, April 1997.

[20] G.M.Lohman, C.Mohan, L.M.Haas, D.Daniels, B.G.Lindsay, P.G.Selinger, and P.F.Wilms. Query processing in R*. In W.Kim, D.S.Reiner, and D.S.Batory (eds.): *Query Processing in Database Systems*, 31–47. Springer Verlag, 1985.

[21] M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems*. Prentice Hall, 2nd edition, 1999.

[22] K. Richine. *Distributed query scheduling in DIOM*. Tech. report TR97-03, Comp. Sc. Dept., Univ. of Alberta, 1997.

[23] T. Risch and V. Josifovski. Distributed data integration by object-oriented mediator servers. *Concurrency and Computation: Practice and Experience*, 13(11):933–953, 2001.

[24] T. Risch, V. Josifovski, and T. Katchaounov. *Amos II concepts*. http://www.csd.uu.se/∼udbl/amos/doc/, 2000.
[25] A. Tomasic, L. Raschid, and P. Valduriez. Scaling access to heterogeneous data sources with DISCO. *IEEE Trans. on Knowledge and Data Engineering*, 10(5):808–823, 1998.
[26] *VLDB Journal*. Special issue on E-services. 10(1), 2001.
[27] G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, 1992.