Data Mediation in a Grid Environment

Paul L. Bergstein

Dept. of Computer and Information Science
University of Massachusetts Dartmouth
Dartmouth, MA
pbergstein@umassd.edu

Abstract *– Many modern data applications need to process large amounts of data stored in heterogeneous databases which are distributed across multiple grid nodes. One of the key issues in developing such a grid DBMS is transparency, i.e. users and applications should not need to be aware of the data heterogeneity or distribution details. Another important issue is query optimization, especially for queries involving distributed joins.*

We have previously described the development of a data mediation service which provides the transparency that enables users and applications to pull data from a foreign data source without any knowledge of its actual structure or semantics. The mediator translates a query written against a well known (local) data source into a query against the foreign data source, executes the query, and then translates the data into the local format. In this scenario, all of the data retrieved is obtained from a single source.

In this paper, we describe how our mediator is easily extended to provide transparency and distributed query processing in a grid environment where a single query may require combining data from multiple sources.

Keywords: Data mediation, data integration, grid-dbms, distributed join processing.

1. Introduction

A serious problem facing many organizations today is the need to process large amounts of distributed data. The problems are particularly difficult when the nodes have been developed separately, in which case the heterogeneity of the data creates major obstacles to effective processing. Conflicts may exist in both the structure and the semantics of the data involved. Furthermore, the structure and semantics of a data source may change over time.

The data mediation approach to data interoperability relies on a common ontology that can be used to describe the structure and semantics of each of the systems that wish to participate in the information sharing. A data mediator uses these descriptions to resolve structural and semantic inconsistencies between nodes exchanging information. In a variation of this approach, a shared view is created, and the mediator translates queries written against the shared view. In either case, mediation has the advantages that there is no need to agree on standard formats, the metadata is made explicit (so it may be reused), and translations only occur where the structure or semantics between two systems differ.

Our mediation service uses a layered architecture as shown in Figure 1. In the sections 2 and 3 we will briefly describe the basic operation of the mediator. Then we will focus on our recent work on mediation in a grid environment.

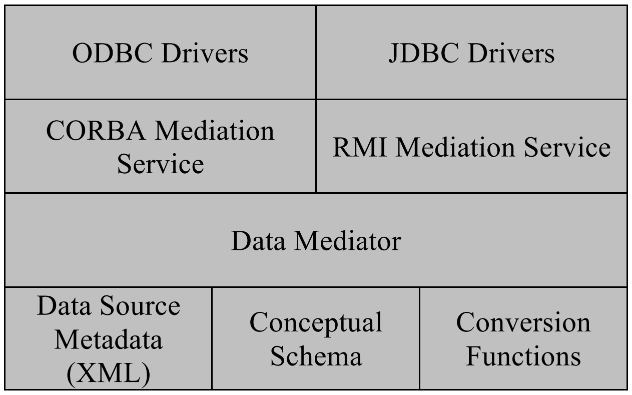

Figure 1

2. Background

Our data mediator was originally based on the following scenario: Suppose a user who knows the schema of only their local database, System A, wishes to retrieve information from a foreign database, System B. They write a query against the schema of System A, but indicate that they would like to use System B as the data source. The mediator translates the query against

System A into one or more queries against System B, executes the queries, and translates the results into the local format of System A.

In our current work we also consider a slightly different scenario: Suppose an application using ODBC or JDBC has been written to use a particular data source, System A, but we now want to use a different source, System B, with a different structure or semantics. We accomplish the change simply by plugging in our Mediator ODBC or JDBC driver in place of the System A driver. The application can now use the new data source without rewriting any code or queries. Notice that plugging in the System B driver will only work if Systems A and B have identical structure and semantics, otherwise mediation is required.

2.1 Conceptual Schema

In our implementation, the common ontology is expressed as a shared conceptual schema, which includes both ordinary classes (e.g. University, Student) and domain classes (e.g. Money, Date). The attributes of ordinary classes have domain classes as their types. For example, Student might have an attribute called graduation-date with type Date. For each domain class, we specify subclasses (sub-domains) for the known representations. When a new data source is registered with the mediator, it will typically be necessary to add sub-domains for data representations that are unique to that data source.

The conceptual schema is never populated in our system. It is used only as a reference for defining the structure and semantics of the actual data sources. In particular, the conceptual schema is *not* used as an intermediate data representation when transferring data from one source to another. Instead, the mediator synthesizes a plan for direct conversion between the data sources based on their structures and semantics as defined by their individual mappings to the conceptual schema.

2.2 Conversion Functions

The mediator uses a repository of functions for converting between representations within a domain. In our (java) implementation all conversion functions have the same interface. They take a java Properties object as parameter, and return a Properties object as the result, so they naturally support many-to-many mappings. For example, a position might be specified using a Properties object with latitude and longitude attributes, or (using Universal Transverse Mercators) with zone, easting, and northing attributes. In this case we have a two-to-three mapping. The repository is implemented as a java class with methods for each conversion function. The repository uses java introspection to search for suitable conversion functions.

2.3 Metadata

In order to register a data source with the mediator, a description of the data source (its metadata) must be supplied in XML format. For each data source there is a separate XML file prepared by someone familiar with that data source. Currently, the XML files are prepared manually, but we plan to develop tools to help generate these files. The metadata includes information required to connect to the data source as well as mappings to the conceptual schema that define the structure and semantics of the data source. We use XML[1] to map data elements of real databases onto attributes of ordinary classes in the conceptual schema. Each mapping to an attribute of an ordinary class includes the subdomain of the data element.

In the simplest case each data element of System A corresponds one-to-one with an element of the conceptual schema, which in turn corresponds one-to-one with an element of System B. If the conceptual schema contains an ordinary class called Employee with a salary attribute of type Salary, and System A has a Worker relation with a pay-rate attribute, then the XML file for System A would map Worker/pay-rate to Employee/salary and it would also map pay-rate to one of the subdomains of Salary such as Annual/USDollars or Monthly/Euros[2]. Similar mappings from System B provide the mediator with the information needed for translation.

Mappings between the conceptual schema and an actual database are not always one-to-one. Suppose that in the conceptual schema Professor's have a phone-number attribute of type PhoneNumber, but in the actual database Instructor's have area-code, exchange, and extension attributes. For the mediator to work, the PhoneNumber domain class must have a subdomain, say ACEE, for the area-code/exchange/extension representation of phone numbers, with attributes corresponding to the three parts of a phone number. Each of the area-code, exchange, and extension attributes is mapped to the Professor/phone-number attribute (and also to the

[1] For brevity, in this paper we mostly describe mappings without showing the XML syntax since the XML is trivial but verbose.

[2] In theory, the issues of currency units and frequency of payment should be separate, but we combine them for the sake of simplicity in our implementation, in order to focus on more interesting concerns.

appropriate attribute of the ACEE subdomain). The mapping (from actual to conceptual) is many-to-one.

If another database uses the same representation of phone numbers, so we have mappings like:

A: (code, exchg, ext) → phone-number
B: (area, exg, extension) → phone-number

then the translation will not use conversion functions (even if the data elements have different names). In other cases, such as:

A: (latitude, longitude) → position
B: (zone, easting, northing) → position

a conversion function is required.

Sometimes data source isn't a very good match for the conceptual schema. This is likely to happen, for example, when a new data source is added after the conceptual schema has been completed. Consider, for example, a conceptual schema that has entity classes for full-time students and part-time students, and a data source with graduate students and undergraduate students. In this case we map attributes, e.g. gpa, from both graduate and undergraduate students to attributes of both full-time and part-time students. Additionally, we supply conditions that determine, for example, which graduate students are part-time and which are full-time. These conditional mappings [5] are specified in both directions (to and from the conceptual schema).

2.4 Data Mediator

The data mediator manages the conversion function repository, the conceptual schema, and the data source metadata. It is responsible for synthesizing query and translation plans. When a query against the schema of System A is executed using System B as the data source, the mediator translates the query against System A into one or more queries against System B, executes the queries, and translates the results into the local format of System A. The details of our algorithm are beyond the scope of this paper, but will be reported elsewhere.

The mediator is implemented entirely in Java and uses JDBC to access the desired data source. Therefore, the mediator can be used to exchange data between any data sources that have JDBC drivers available, including most relational databases, all ODBC data sources (via a JDBC/ODBC bridge driver), and XML data (using an available XML JDBC driver).

3. Query Mediation

In this section we describe the mediator's processing of simple queries written against the schema of a well known (local) data source when the actual data resides in a different (foreign) data source. We start by considering simple queries consisting of only *select* and *from* clauses. In the next section we will consider the more complex issues of processing the *where* clause. For our examples we will use the local and foreign schemas for airplane data in Figure 2. For simplicity, we have not shown the shared conceptual schema.

Local schema:
Airplanes (aid, latitude, longitude, fuel_capacity, range, wingspan)

Foreign schema:
Aircraft (craftId, zone, easting, northing, fuel_tank_size, cruising_range, wingspan)

Figure 2

3.1 Select Clause Translation

The select clause is translated by replacing the name of each data element in the list with the data element(s) from the foreign data source that map to the same attribute(s) in the shared conceptual schema.

In the simplest case, the local element maps to a single attribute in the shared schema which in turn maps to a single element of the foreign data source, and the replacement mapping between the local and foreign data elements inferred by the mediator is one-to-one. In our example, the mediator would infer a one-to-one replacement of fuel_capacity with fuel_tank_size wherever fuel_capacity occurs in the select clause of the original query.

However, in general, the inferred replacements are one-to-many both because the local element may map to many attributes in the conceptual schema, and because each attribute of the conceptual schema may map to many elements of the foreign data source. For example, since the local attribute *latitude* (along with longitude) maps to the concept of position, and the foreign attributes *zone*, *easting*, and *northing* also map to the concept of position, *latitude* would be replaced with *zone*, *easting*, and *northing* when the query is translated. This one-to-three replacement is correct since the mediator needs all three UTM attributes to calculate a latitude. Note that if the select clause of the original query contained both latitude and longitude, the mediator would infer a one-to-three mapping for each of them. In a subsequent step, the mediator eliminates requests for duplicate columns.

3.2 From Clause Translation

The table names of the from clause are translated in a manner similar to the columns in the select clause. A single table in the where clause of the original query may map to multiple tables in the conceptual schema and each of those may map to multiple tables in the foreign schema.

In this case where the inferred replacement is one-to-many, there will be a separate query generated for each replacement. For example, if the foreign data source had its aircraft data split into two tables, say Jets and Propeller Aircraft, a single query on the Airplanes table of the local data source would result in two separate queries in the foreign data source – one selecting from Jets and one from Propeller Aircraft. The mediator would execute both queries and combine the results.

The other complication is that table mappings may be conditional [5]. This would come into play if we switched the local and foreign data sources for our example. In this case the mediator would replace Jets with Airplanes in the from clause, but not all airplanes are jets.

When mappings between a data source and the conceptual schema are conditional, the conditions are specified as part of the mapping. The mediator adds the appropriate mapping conditions to the where clause of the original query. The *to* conditions of the foreign schema mapping (specifying which foreign entities map *to* a conceptual class) and the *from* conditions of the local schema (specifying which conceptual entities map *from* the conceptual class) are added to the where clause of the query.

The *to* conditions are already written in terms of the foreign data source and don't require translation. The *from* conditions of the local schema, however, must be translated before the query can be executed in the foreign data source. The where clause processing is discussed in section 4.

3.3 Data Translation

After the translated queries have been executed in the foreign data source, the results must be translated into the format expected in the local data source. If there was a one-to-one replacement of an attribute in the select clause with a corresponding attribute from the foreign data source in the same format, no conversion is necessary. Otherwise, a conversion function from the mediator's repository is used. The values retrieved from the foreign data source are packaged as a java Properties object, passed to the appropriate conversion function, and the desired value is then extracted from the returned Properties object. For example, if latitude in the original query was replaced by zone, easting, and northing in the translated query, these three values from each row would be packaged as a Properties object and the latitude value would be extracted from the new Properties object with values for latitude and longitude returned from the conversion function.

3.4 Where Clause Processing

The central problem in where clause processing is to translate conditions involving data elements of the local schema into conditions that can be specified against the foreign schema. The simplest situation is where the local and foreign data elements are in the same format and correspond one-to-one. For example, if the where clause contains the condition *range > 1000,* and Airplanes range and Aircraft cruising_range are in the same format (units, scale, etc.), the mediator can simply replace *range* with *cruising_range*.

The next simplest situation is where, for example, range and cruising_range correspond one-to-one but are in different formats. If range is in kilometers and cruising_range is in miles, the mediator can apply a conversion function to the constant to generate the condition *cruising_range > 621.37.* The mediator can also modify conditions by applying operators to attributes, e.g. replacing expression *range* with *cruising_range * 0.62137*, although there are few cases in practice where this is useful.

Unfortunately, conditions involving attributes that do not map one-to-one are much more difficult to translate. Consider, for example, translating the condition *latitude > 40* into terms of zone, easting, and northing. While a human with adequate understanding of the two positioning systems could produce a translation, our mediator cannot.

In our early implementations we attempted to translate all where clause conditions and the mediator would throw an exception when presented with queries it could not handle. Once we realized that some where clause conditions could never be translated efficiently, we tried a radically different approach. In this new approach we eliminated the where clause altogether before executing the query in the foreign data source. After the data was returned the mediator applied the where clause to each data tuple as it was translated to the format of the local schema. By applying the where clause conditions to the translated data, it was not necessary to translate the conditions.

While this approach worked, it has a major drawback. Since the where clause is evaluated in the mediator, rather than the foreign data source, potentially large quantities of data that are not part of the final result must be brought across the network into the mediator.

Another suggestion was to implement conversion functions in the actual data sources. In this case the condition *latitude > 40* would be translated to *conv(zone, easting, northing) > 40* where *conv* is a conversion function defined in the foreign data source. However, this approach also has major drawbacks. First, not all data sources support this kind of function. More importantly, the approach would not scale. One of the important features of the mediation approach is that each data source is mapped only to the conceptual schema. Supplying each data source with conversion functions for every data element of every other data source is not realistic.

Our current approach is a compromise between the extremes of translating all conditions or eliminating the where clause entirely. In the most recent approach the mediator starts by rewriting the where clause in conjunctive normal form (CNF). The conjuncts can then be applied independently in sequential fashion. The conjuncts are partitioned into translatable and untranslatable groups. As many conjuncts as possible are translated and added to the where clause of the translated query for execution in the foreign data source, thereby minimizing the network traffic. The untranslatable conjuncts are applied in the mediator as the data returned from the foreign data source is translated into the format of the local data source.

Consider the query:

select aid from airplanes
where (latitude > 40 AND wingspan > 20)
OR (range > 2000 AND fuel_capacity > 500)

The mediator will start by rewriting the where clause conditions in CNF as:

(latitude > 40 OR range > 2000) AND
(latitude > 40 OR fuel_capacity > 500) AND
(wingspan > 20 OR range > 2000) AND
(wingspan > 20 OR fuel_capacity > 500)

The first two conjuncts contain the condition on latitude which cannot be translated so they will be applied in the mediator. The last two, however, are easily translated by replacing wingspan, range, and fuel capacity with the corresponding attribute names from the foreign data source, and converting the constant values into to the appropriate units. The last two conditions are translated and applied in the foreign data source to eliminate unnecessary network traffic.

4. Grid DBMS

Aloisio et. al. [15] have identified seven basic requirements that a Grid-DBMS must provide: security, transparency, easiness, robustness, efficiency, dynamicity, and intelligence. They further identify five forms of transparency which must be addressed in a Grid-DBMS: physical data location, network, data replication, data fragmentation, and DBMS heterogeneity. While they do not suggest a specific grid middleware, our mediator is well suited for adaptation to meet the identified requirements. In this section, we describe our initial efforts to adapt the mediator to a grid environment, focusing primarily on transparency and efficiency issues.

The most fundamental change in the mediator is almost trivial – nodes are treated as a single distributed data source rather than a collection of alternative sources of the same information. The nodes are mapped onto a conceptual schema exactly as before. Users can customize their view of the Grid-DBMS by mapping an "actual" schema onto the conceptual schema. This is exactly as before, except that the actual schema is not populated. Clients write queries in terms of their "view" and the mediator performs the necessary translations.

4.1 Transparency

The mediator service was designed from the beginning to hide details of physical data locations, network issues, and database heterogeneity from clients. These aspects of the mediator are unmodified when used as Grid-DBMS middleware.

The mediator was also designed to handle translation between data sources that use different partitioning schemes. It handles both horizontal partitioning (e.g. Aircraft into Jets and Propeller Planes) and vertical partitioning (e.g. Projects into ProjectFinancials and ProjectSchedules). Once again, the mediator functionality can be used unmodified in a Grid system. Although vertical partitioning of data will improve the efficiency of some queries, it may necessitate additional distributed join operations for others. We take a unique approach to distributed joins, which is discussed in the next section.

Data replication is one aspect of Grid-DBMS that the mediator was not designed to handle. In the original mediator implementation the various data sources are assumed to have been developed independently, whereas in a Grid-DBMS data is often replicated to improve performance and reliability. Extending the mediator to handle data replication in a transparent manner is not difficult. The solution is to add metadata to the mediator's repository to specify where data is replicated. The mediator will choose a particular source of replicated data according to its policy without involving the user. Currently, the mediator simply chooses a data source at random in the case of data replication. In the future, we would like to provide a mechanism for the system administrator to specify a policy in a flexible way through a configuration file.

4.2 Efficiency

Our efforts in the area of efficiency are currently focused on processing distributed joins. Our approach is based on the same basic idea as semi-joins. We try to reduce the network traffic as much as possible at the expense of increased local processing.

Ordinarily, we think of join conditions involving both of the relations to be joined. However, the conditions in the *on* clause of an SQL theta join may contain arbitrary conditions. Furthermore, joins are frequently followed by selections (i.e. *where* clause). We start by combining the join conditions with conditions from the selection (if there is one) according to the following rewriting rule:

$$\sigma_{\Theta_2}(r \bowtie_{\Theta_1} s) \Rightarrow r \bowtie_{\Theta_1 \wedge \Theta_2} s$$

Next, we rewrite the join condition ($\Theta_1 \wedge \Theta_2$) in conjunctive normal form (CNF), and partition the resulting terms into three groups: conditions that involve only r (Θ_r), conditions that involve only s (Θ_s), and conditions that involve both r and s (Θ_{rs}):

$$r \bowtie_{\Theta_1 \wedge \Theta_2} s \Rightarrow r \bowtie_{\Theta_r \wedge \Theta_s \wedge \Theta_{rs}} s$$

Now we perform selections locally in the data sources of r and s, and only the tuples from r that satisfy Θ_r and the tuples of s that satisfy Θ_s are brought into the mediator to compute the join:

$$\sigma_{\Theta_r}(r) \bowtie_{\Theta_{rs}} \sigma_{\Theta_s}(s)$$

This approach can be combined with semi-joins to further reduce the network overhead. Lu and Carey [17] demonstrated that the additional computational overhead of semi-joins can be higher than the savings in communication costs in certain circumstances. Similarly, there is a potentially high cost involved with transforming join conditions to CNF. In the worst case, the number of terms can increase exponentially. While we have not experienced this problem in practice, we intend to empirically investigate this issue in more detail.

5. Related Work

There are numerous other researchers [1-4, 7-12] who have investigated mediation as a way of resolving structural and semantic conflicts between data sources. However, as far as we can determine, there are no previous reports of adapting a mediation service to a Grid-DBMS environment.

6. Future Work

In the immediate future we will continue to focus on improving data replication features and efficiency. In particular we intend to develop a flexible and easy to use interface for configuring the mediator's policy for selecting among duplicate data sources. We will also attempt to determine the conditions where the savings in network overhead justify CNF transformations and/or semi-joins.

Another area where the mediator requires additional work is in security. In its current form, the mediator does not support encrypted connections to the grid nodes and relies on the individual nodes to perform authentication and authorization of requests. Future work will address both of these shortcomings.

7. References

[1] E. Sciore, M. Siegel, and A. Rosenthal, "Using Semantic Values to Facilitate Interoperability Among Heterogeneous Information Systems", *ACM Transactions on Database Systems*, vol. 19(2), June 1994, pp. 254-290.

[2] G. Wiederhold, "Mediators in the Architecture of Future Information Systems", Readings in Agents, Eds. M. N. Huhns and M. P. Singh, San Francisco, CA, USA: Morgan Kaufmann, 1997, pp. 185-196.

[3] P. B. Lowry, "XML data mediation and collaboration: A proposed comprehensive architecture and query requirements for using XML to mediate heterogeneous data sources and targets," *34th Annual Hawaii International Conference On System Sciences (HICSS)*, Maui, Hawaii, January 3-6, 2001, pp. 2535-2543.

[4] C. H. Goh, S. Bressan, S. Madnick, and M. Siegel, "Context interchange: new features and formalisms for the intelligent integration of information", *ACM Transactions on Information Systems*, vol. 17(3), July 1999, pp. 270.

[5] P. Bergstein and V. Shah, "Conditional Mapping in Data Mediation", *Proceedings of the*

International Conference on Information and Knowledge Engineering (IKE 2004). June 21-24, 2004, Las Vegas, Nevada, USA. CSREA Press 2004, ISBN 1-932415-27-0.

[6] P. Bergstein and A. Sikder, "A JDBC Data Mediation Service", *Proceedings of the International Conference on Information and Knowledge Engineering (IKE 2005),* pages 45-50, June 20-23, 2005, Las Vegas, Nevada. CSREA Press, ISBN 1-932415-81-5.

[7] L. S. Seligman and A. Rosenthal, "XML's Impact on Databases and Data Sharing", IEEE Computer, vol. 34(6), 2001, pp. 59-67.

[8] G. Neugebauer, "GLUE – Using Heterogeneous Sources of Information in a Logic Programming System", *Proceedings of the KI'97 Workshop on Intelligent Information Integration*, Freiburg, 1997.

[9] L. Serafini and F. Giunchiglia and F. Mylopoulos and P. Bernstein, "The Local Relational Model: A Logical Formalization of Database Coordination", *Proceedings of CONTEX'03*, 2003.

[10] H. Wache and H. Stuckenschmidt, "Practical Context Transformation for Information System Interoperability", *Lecture Notes in Computer Science*, vol. 2116, 2001, p. 367.

[11] B. Ludäscher, A. Gupta, and M. Martone, "Model-Based Mediation with Domain Maps", *17th International Conference on Data Engineering (ICDE '01)*, Washington-Brussels-Tokyo, April 2001.

[12] C. Baru, A. Gupta, B. Ludäscher, R. Marciano, Y. Papakonstantinou, P. Velikhov, and V. Chu, "XML-based Information Mediation with MIX", Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data: SIGMOD '99, Philadelphia, PA, June 1-3, 1999, SIGMOD Record, vol. 28(2), 1999, pp. 597-599.

[13] P. Bergstein, "An ODBC CORBA-Based Data Mediation Service", *Proceedings of the International Conference on Information and Knowledge Engineering (IKE 2006),* pages 196-202, June 26-29, 2006, Las Vegas, Nevada. CSREA Press, ISBN 1-60132-003-5.

[14] P. Bergstein, "Query Translation and Where Clause Processing in Data Mediation", *Proceedings of the International Conference on Information and Knowledge Engineering (IKE 2007),* pages 61-66, June 25-28, 2007, Las Vegas, Nevada. CSREA Press, ISBN 1-60132-050-7.

[15] G. Aloisio, M. Cafaro, S. Fiore, and M. Mirto, "The Grid-DBMS: Towards Dynamic Data Management in Grid Environments", *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'05)*, volume 2, pages 199-204, 2005, IEEE Computer Society, ISBN 0-7695-2315-3.

[16] D. Kossmann, "The State of the Art in Distributed Query Processing", *ACM Computing Surveys*, Vol. 32, No. 4, December 2000, pages 422-469.

[17] H. Lu and M. Carey, "Some Experimental Results on Distributed Join Algorithms in a Local Network", *Proceedings of the 11th International Conference on Very Large Data Bases (VLDB'85)*, August 1985, Stockholm, Sweden, pages 292-304.

[18] P. Mishra and M. Eich, "Join Processing in Relational Databases", ACM Computing Surveys, Vol. 24, No. 1 March 1992, pages 63-113.