Dynamic Query Processing in DIOM *

Ling Liu
Department of Computing Science
University of Alberta
lingliu@cs.ualberta.ca

Calton Pu
Dept. of Computer Science and Engineering
Oregon Graduate Institute
calton@cse.ogi.edu

Abstract

In an open environment such as the Internet, query responsiveness involves both the capability of responding to a query within a reasonable time frame and the capability of dynamically incorporating the new information sources and the up to date information of existing data sources into the answer of a query. In this paper we present the dynamic distributed query processing framework, developed in the DIOM project, to demonstrate the feasibility and the benefit of improving responsiveness of querying across heterogeneous information sources, without relying on an integrated global view schema pre-defined over all the participating information sources. Then we discuss several issues with respect to improving query responsiveness in the context of DIOM, including how the information consumers' queries are dynamically processed and linked to the relevant information sources, and how the query scheduling framework scales up in an environment where information sources accessible to users are constantly and rapidly changing, in their content, numbers, identity, and connectivity.

1 Introduction

The continuous advancement of wide-area network technology has resulted in a rapid increase in both the number of data sources available on-line and the demand for information access from a diversity of clients over the Internet or intranets. The availability of various Internet information browsing tools further promotes information sharing across departmental, organizational, and national boundaries. One of the most distinct features of such open and loosely-coupled distributed environments is that information sources accessible to a user are not pre-defined *a priori*, contrary to the traditional database assumption of a closed universe in centralized or tightly-coupled distributed environments.

A serious problem that arises in open environments is that of *query responsiveness*: a potentially very expensive query evaluation process due to the large and evolving number of information sources and joins, and the unpredictability in network configuration. There are several reasons that query responsiveness poses new technical challenges: First, it is well known that accessing multiple remote data sources over networks is expensive, and of the large collection of available information sources,

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*The first author is partially supported by NSERC grant OGP-0172859 and NSERC grant STR-0181014. The second author is partially supported by NSF grant IRI-9510112, and grants from Intel and Hewlett-Packard.

only a small subset may actually contribute to the answer of a specific query. Second, data sources in open environments by their nature are heterogeneous and dynamically evolving. As a consequence, the query optimization and execution processes have to deal with the dynamic changes in content, numbers, identity, and connectivity of individual data sources. Third, the number of data sources that are available to a specific query varies from time to time not only due to the dynamic arrival of new information sources but also due to the vulnerability of data servers and communication links to the congestion on networks and the contention at sources [1].

We have identified three key steps in the optimization of query performance and responsiveness in open distributed data delivery systems: (1) to identify relevant information sources that can contribute to a specific query [4] based on both compile-time analysis and run-time information, thus minimizing the number of information sources accessed to answer a query, (2) to provide adequate support for establishing a dynamic interconnection between information consumers and information producers [6], thus allowing seamless incorporation of changes in content, numbers, identity, and connectivity of information sources into the planning and processing of queries, and (3) to consolidate run-time query plan modification techniques in the presence of delay and slow delivery of data or unexpected unavailability of information sources (see [1, 3, 4] or the papers appearing in this special issue).

Our work in DIOM [6, 4] has primarily targeted on the issues of source identification and dynamic interconnection. First, we have developed a three-level progressive pruning model and a set of dynamic query routing algorithms to identify relevant information sources for a query, thus reducing the overhead of contacting the information sources that do not contribute to the answer of the query. Efficient query routing not only reduces the query response time and the overall processing cost, but also eliminates a lot of unnecessary communication overhead over the global networks and over the individual information sources. Second, we have combined two independent but complementary strategies in the development of dynamic distributed query services for achieving higher system scalability and better query responsiveness:

- An incremental approach to construction and organization of information access through a network of domain-specific application mediators.
- A collection of facilities to allow information consumers to pose their queries on the fly and to dynamically link a user query to the relevant information sources.

The first strategy supports seamless incorporation of new information sources into the DIOM system. The second strategy allows the distributed query services to be developed as source-independent middleware services which establish the interconnection between consumers and a variety of information producers' sources at the query processing stage. As a result, the addition of any new sources into the system only requires each new source to have a DIOM wrapper installed. The DIOM services can dynamically capture the newly available information sources and seamlessnessly incorporate them into the distributed query scheduling process.

The main features that distinguish DIOM from other mediator-based approaches is the systematic development of a dynamic query mediation framework and the collection of dynamic query processing techniques for scalable data delivery [6, 5]. In what follows, we first give a brief overview of the system architecture and the dynamic query processing framework currently being implemented in the DIOM project. Then we describe the two main components of DIOM dynamic query processing: query routing and query planning, and their roles in improving query responsiveness. We conclude the paper with a summary and a brief report on our ongoing research and development.

2 Dynamic Query Processing: System Architecture

The DIOM system architecture [5] is a two-tier architecture offering services at both the mediator level and the wrapper level as shown in Figure 1.

A *mediator* is a software component which provides services to facilitate the integration and access of heterogeneous information. DIOM mediators are application-specific, consisting of a consumer's query profile and many information producer's source profiles, described in terms of the DIOM interface definition language (DIOM IDL) [5]. The consumer's query profile captures the querying interests of the consumer and the preferred query result representation. The producer's source profiles describe the content and query capabilities of individual information sources. A *wrapper* is a software module providing

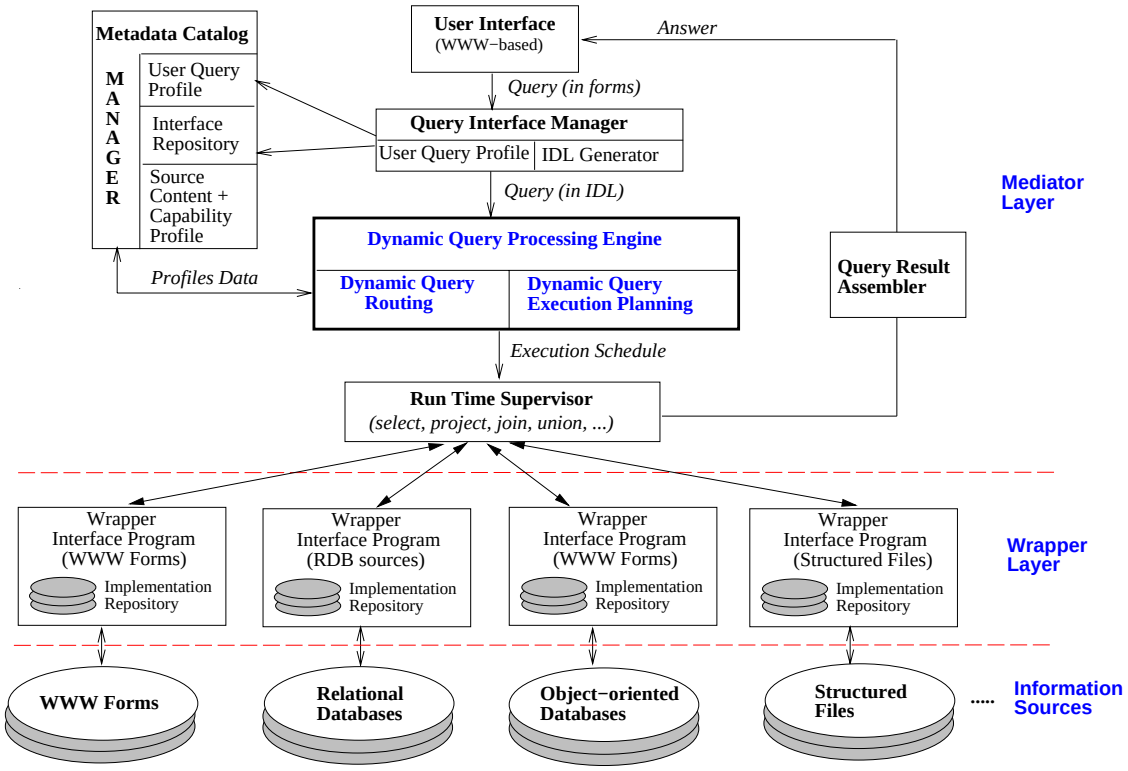

Figure 1: Dynamic Query Scheduling Framework

an appropriate interface for a component data repository. The main task of a wrapper is to control and facilitate external access to the wrapped information repositories by using the local metadata maintained in the implementation repository through wrapper interface functions. *Information sources* at the bottom of the diagram fall into three categories: *well structured* (e.g., RDBMS, OODBMS), *semi-structured* (e.g., HTML files, text based records), or *unstructured* (e.g., technical reports). Each information source is treated as an autonomous unit. Information sources may make changes without requiring consent from any mediators. Currently DIOM also provides access to *unstructured* data (e.g., technical reports) through external WWW keyword based browsers such as Altavista, Infoseek, Yahoo. A DIOM mediator may access multiple and remote information sources through communication with their corresponding DIOM-to-source wrappers [5].

The DIOM dynamic query scheduling manager coordinates the communication and distribution of

the processing of information consumer's query requests using the user query profiles and the description of the content and query capabilities of information sources. In addition, run-time information such as the intermediate results, the status of the networks and connections to information sources and the replication relationships among available information sources, is used for further refinements of the query routing result and the query execution plan. Figure 1 presents a sketch of the system architecture currently being implemented in the DIOM project. The main components of the DIOM distributed query scheduling manager are query interface manager, query routing, and query execution planning.

A user may submit his/her query on-the-fly, using WWW fill-out forms. The query interface manager in turn generates a user query profile which consists of (1) a form query in terms of the DIOM interface query language (DIOM IQL) [5], (2) virtual interface classes, described in terms of the DIOM interface definition language (DIOM IDL) [5], which are used as template holders for receiving and representing the resulting objects of the query; and (3) user query annotation (optional), which uses an interactive interface program to allow the user to annotate the scope and context of what are to be expected in a specific query, including the input and output parameters of the query. Then the query is passed to the dynamic query processing engine for query routing and query planning.

- **Dynamic query routing:** The main task of dynamic query routing is to select relevant information sources from a large collection of available information sources which can contribute to answering a query. This is performed by dynamically relating the user query profile with the content and query capability descriptions of the sources.

- **Dynamic query execution planner:** The dynamic query execution planner consists of two steps: *query decomposition*, in charge of rewriting an IQL query, posed by the information consumer on the fly, into a group of subqueries, each targeted at a single data source, and *query scheduler*, which is responsible for generating a query execution schedule that is optimal in the sense that it utilizes the potential parallelism and the useful execution dependencies between subqueries to restrain the search space, minimize the overall response time, and reduce the total query processing cost.

Suppose we are interested in purchasing a book. Consider a query Q: "*Find title, authors, price, publisher, and reviews of science fiction that were published after 1996*". By using the IDL generator, we have all the parameters of interest generated, i.e., the category (science fiction, health, and so on), title, authors, price, publish-year, and publisher for `Book` objects and the book reviews for `Review` objects. By using the interactive interface program for user query profile generation, we have the query scope {`Book, Review`}, the list of input parameters {`category, publish-year`}, and the list of output parameters {`title, authors, price, publisher, reviews`}. We can further annotate, for example, whether we are interested in purchasing books from bookstores, bookclubs or publishers, what is our preferred currency unit for the price information, and so forth. An information source profile contains the content description (such as names of the relations and classes accessible at the source) and the query capability description (such as the list of mandatory or optional input parameters and the list of allowed output parameters). The user query profile and the producers' data source profiles play an important role in the query routing module to restrain the search space of the query by isolating the query within a subset of information sources that are relevant to the query.

Once the relevant information sources are selected, the query execution planner first decomposes the query into a group of subqueries, each targeted at a selected information source, and then finds a relatively optimal schedule that makes use of the parallel processing potential and the useful dependencies between subqueries. The ultimate goal of query planning is to reduce the overall response time and the total query processing cost. The run-time supervisor executes the subqueries according to the execution schedule produced by the query planner. It communicates with wrappers which control and

facilitate external access to the information sources. Each wrapper translates the subquery request received from the run-time supervisor into an executable query program (or a function call) that can run at the local source. The result assembly process involves two phases for resolving the semantic variations among the subquery results: packaging each individual subquery result into a DIOM object (done at wrapper level) and assembling results of the subqueries in terms of the consumers' original query statement (done at mediator level). The semantic attachment operations and the consumers' query profiles are the main techniques that we use for resolving semantic heterogeneity implied in the query results [5].

3 Dynamic Query Routing

Query routing is the first step that constrains the search space for a query in open environments, in preparation for query planning. One of the main goals of query routing is to identify relevant information sources for a query as early as possible, thus reducing the overhead of contacting the information sources that do not contribute to the answer of the query. Query routing is particularly critical in open environments that contain large and growing collections of heterogeneous information sources.

Given a user query Q, a user query profile of Q, and a set of source content and capability descriptions, we design the query routing service as a three-step process, which determines the relevance of the sources in answering a query by incorporating compile-time analysis with run-time information.

Step 1: Level-one relevance pruning prunes the information sources whose content descriptions are obviously irrelevant to the scope of the query Q (e.g., in terms of substring matching or ontology mapping used in the DIOM implementation). For level-one relevance pruning we use the user query scope description of Q and the content and category description of the sources. At level-one routing stage, the redundancy or replication of the sources will be detected and removed, based on the source replication description. Other factors such as unavailability of the sources or affordability of the sources should be considered at this step too. We refer to the set of sources selected by this step as *target* information sources of *level-one relevance*.

Step 2: Level-two relevance pruning prunes the information sources that have level-one relevance but do not offer enough query capability to contribute to the answer of Q, either due to the restriction on the scope of query parameters of Q, or the restriction on the list of input or output arguments of the sources, or due to the conflict of query interest with the access constraints associated with the sources. The user query capacity description of Q and the source capability profiles are used in the level-two relevance pruning. We call the set of sources selected by Step 2 as *target* information sources of *level two relevance*.

Step 3: Level-three relevance pruning explores run-time information to further prune irrelevant information sources to answer Q. For the atoms in Q that are of form $a\theta v$ where a is an attribute and v is a constant, if there are sources which take a or its alternative as one of the input arguments, then it is in general worthwhile to ask some subqueries to these information sources and use the run-time information returned (intermediate results) to continue to prune the number of information sources selected. This step is especially helpful when the subset of information sources resulting from level-two relevance pruning is not significantly small comparing with the large collection of available information sources. We call the set of sources selected by this step as *target* information sources of *level three relevance*. The level-three relevance pruning is built as a plug-and-play component and can be tuned to play any time when the need arises.

4 Dynamic Query Planning

Dynamic query planning component generates a parallel access plan for a group of subqueries, which is a relatively optimal schedule that makes use of the parallel processing potentials and the useful execution dependencies between subqueries. Two-phase reduction approach is used to reduce the solution search space based on built-in heuristics and cost estimation. A sketch of the two-phase query planning architecture is shown in Figure 2. The first processing phase is based on heuristic query optimization

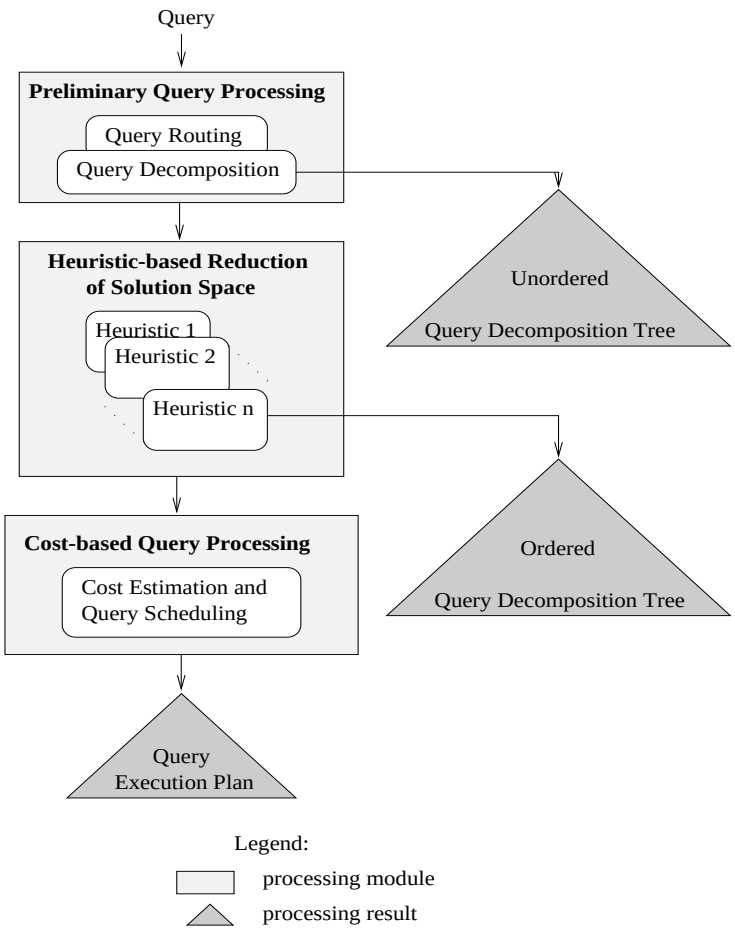

Figure 2: Two-phase query planning architecture

techniques. The second processing phase is the cost estimation and access plan scheduling.

The purpose of applying the heuristics first is to restrict the solution search space well enough so that the cost-based search is feasible and beneficial in the smaller solution space. The common heuristics used in the initial implementation of DIOM include (1) performing selections first; (2) performing joins that produce the smallest result earliest; and (3) performing *Cartesian product* last. The other heuristics observed in DIOM include (4) both union and Cartesian product should always be performed at the site where the result is expected because the communication cost of transferring the result is greater than the communication cost of transferring any one of the operands; (5) if the expected size of join result is smaller than any of its inputs, then it is beneficial to rewrite $\bowtie (\cup(Q_{11}, \ldots Q_{1n}), \cup(Q_{21}, \ldots Q_{2m}))$ into $\cup(\bowtie (Q_{11}, Q_{21}), \ldots, \bowtie (Q_{1n}, Q_{2m}))$; and (6) if the distribution of join over union does not lead to an

increase in the number of inter-site joins, then such interchange is performed, otherwise, the order of operations remains unchanged. For detailed algorithms in heuristic-based optimization or cost-dirven optimization refer to [7].

To cater to the needs of various database users, in DIOM we provide a flexible framework for dynamic query planning, which allows the plug-in of query optimization components on demand, thereby providing user-driven and customizable query optimization. For example, in general, the cost of query execution consists of three independent factors: communication cost, local processing cost, and total response time cost. They may be combined additively into a generic goal formula shown in Equation 1.

$$Cost = a_{cc} \cdot C + a_{lqp} \cdot L + a_{rt} \cdot R = A^T \cdot \begin{pmatrix} C \\ L \\ R \end{pmatrix}, \text{ where} \tag{1}$$

C is the total amount of communications over the network spanning the distributed database expressed in time units;
L is the total amount of local query processing, also expressed in time units;
R is the total response time of the query.

For a given query Q, a tree will be generated after the query routing process (see Figure 2). Let us denote the left subtree Q_{left}, the right subtree Q_{right}, and the binary operator that takes Q_{left} and Q_{right} as inputs Q_{this}. The communication cost $C(Q)$, the local processing cost $L(Q)$, and the total response time $R(Q)$ are computed recursively as follows:

$$\begin{aligned} C(Q) &= C(Q_{left}) + C(Q_{right}) + comm\_cost(Q_{this}); \\ L(Q) &= L(Q_{left}) + L(Q_{right}) + loc\_cost(Q_{this}); \\ R(Q) &= \max[R(Q_{left}), R(Q_{right})] + comm\_cost(Q_{this}) + loc\_cost(Q_{this}). \end{aligned} \tag{2}$$

where $comm\_cost(Q_{this})$ is the cost of shipping data to the site where the operator Q_{this} will be executed, and $loc\_cost(Q_{this})$ is the local processing cost of executing the operator Q_{this} at the chosen site. The coefficients associated with each of these components are the indicators of the desired optimization profile. They can be controlled by the user of the database by setting the profile via the components of vector A^T. The general cost estimation formula 1 serves as the goal function of the optimization process. For example, if the user's primary concern in finding an optimal query execution plan is the response time, then A^T is set to $(0 \quad 0 \quad 1)$. Vector $A^T = (0.3 \quad 0 \quad 0.7)$ would be specified by a user who would like to tune the dynamic query scheduler with respect to his particular requirement by allocating 30% of the total cost to the communication cost and 70% to the response time.

The first prototype implementation of DIOM dynamic query processing system (DQS) was developed and tested on Solaris platform using SunJDK version 1.1. The byte-code has been tested on the following platforms: Windows NT v.3.5.1 using Netscape Navigator v.2.01, and Sun OS v.4.1.4 using Netscape Navigator v.3.0. In the current prototype, the local processing cost estimation is based on the statistic information provided in the source query profiles. The communication cost estimation is based on the communication unit costs of connections, given by the mediator adminstrator. For the sources that have no statistic information available, or the network connections that have no communication unit cost, the system-defined default values will be used. Three main features that distinguish the DIOM dynamic query scheduling framework from conventional distributed query processing work: First, the multi-level progressive query routing process is a new query processing step that is not addressed in conventional distributed query processing paradigm. Second, in DIOM we not only provide the support for commonly used query optimization tactics such as the first three heuristics, but also introduce several semantic-based heuristics to enhance the quality of distributed query optimization

solutions generated in DIOM. Third, we provide user-driven trace functions (capabilities) to allow the performance of distributed query processing to be tuned as necessary, such as the coefficients associated with the Equation 1, the decision of communication unit cost and local processing unit cost, etc.

5 Summary

We discussed the DIOM dynamic query processing framework and the strategies used for improving query responsiveness. The most interesting features that distinguish the dynamic query processing in DIOM from other approaches, such as Carnot, Garlic [2], TSIMMIS [8], are summarized as follows: First, we allow users to pose queries on the fly, without relying on a pre-defined view that integrates all available information sources. Second, we identify the importance of query routing step in building efficient query scheduling framework for distributed open environments. Third, we develop a three-tier approach (i.e., query routing, heuristic-based optimization, and cost-based planning) to eliminate the worst schedules as early as possible. Last but not least, we delay the hard semantic heterogeneity problems to the result assembly stage through semantic attachments rather than semantic enforcement.

Our ongoing research towards improving query responsiveness continues in several directions. We are currently evaluating query routing algorithms using more than a hundred of data sources, and designing algorithms for query result packaging and assembly. We are working on the generation of optimal parallel access plan that takes into account the efficient processing of aggregate functions such as SUM, COUNT, MAX and MIN. We are also interested in exploring possibilities to adapt and incorporate the state of art research results in improving query responsiveness with the DIOM system, such as the query scrambling approach for dynamic query plan modification [1] and the online aggregation for fast delivery of aggregate queries by continuous sampling [3].

For additional information, including access to the demo and the more detailed technical reports and references about DIOM cited in this paper, the interested readers are invited to browse our WWW page: `http://www.cs.ualberta.ca/ diom`.

References

[1] L. Amsaleg, M. J. Franklin, A. Tomasic, and T.Urhan. Scrambling query plans to cope with unexpected delays. In *Proceedings of the International Conference on Parallel and Distributed Information Systems*, Miami Beach, Florida, December 1996.

[2] L. Haas, D. Kossmann, E. Wimmers, and J. Yan. Optimizing queries across diverse data sources. In *The International Conference on Very Large Data Bases*, 1997.

[3] J. Hellerstein, P. Haas, and H. Wang. Online aggregation. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Tucson, Arizona, May 1997.

[4] L. Liu. Query routing in structured open environments. Technical report, TR97-10, Department of Computing Science, University of Alberta, Edmonton, Alberta, Feb. 1997.

[5] L. Liu and C. Pu. An adaptive object-oriented approach to integration and access of heterogeneous information sources. *DISTRIBUTED AND PARALLEL DATABASES: An International Journal*, 5(2), 1997.

[6] L. Liu and C. Pu. A dynamic query scheduling framework for distributed and evolving information systems. In *IEEE Proceedings of the 16th International Conference on Distributed Computing Systems*, Baltimore, May 27-30 1997.

[7] K. Richine. Distributed query scheduling in the context of diom: An experiment. MSc. Thesis, TR97-03, Department of Computer Science, University of Alberta.

[8] Y. Papakonstantinou, S. Abiteboul, and H. Garcia-Molina. Object Fusion in Mediator Systems. In *VLDB 96*, Bombay, India, Sept. 1996.