Pushing Quality of Service Information and Requirements into Global Query Optimization*

Haiwei Ye
Université de Montréal
ye@iro.umontreal.ca

Brigitte Kerhervé
Université du Québec à Montréal
Kerherve.Brigitte@uqam.ca

Gregor v. Bochmann
University of Ottawa
bochmann@site.uottawa.ca

Vincent Oria
New Jersey Institute of Technology
oria@homer.njit.edu

Abstract

In recent years, a lot of research effort has been dedicated to the management of Quality of Service (QoS), mainly in the fields of telecommunication networks and multimedia systems. Emerging applications such as electronic commerce, health-care applications, digital publishing or data mining also have requirements regarding the quality of the service, the cost of the service, the quality of data to be delivered, the accuracy, and the precision of the retrieved data. These examples show the need to consider the concept of QoS from a broader perspective, requiring the collaboration of all the distributed system components involved. In this paper, we propose an approach to integrate user-defined QoS requirements, together with the dynamic properties of the system components involved, into a distributed query processing environment. We then propose a query optimization strategy in which multiple goals may be considered with separate cost models. Furthermore, we discuss some experiment results confirming the effectiveness of our approach.

1. Introduction

Quality of Service (QoS) management has attracted a lot of research in the last decade, mainly in the fields of telecommunication networks and multimedia systems. To support QoS activities, mechanisms have been provided mainly for individual components such as operating systems, transport systems, or multimedia storage servers and integrated into QoS architectures for end-to-end QoS provisions[1]. None of these proposals take database systems into consideration although database systems are an important component of present distributed systems.

Traditional database optimizers aim at minimizing the query response time and/or the number of disk I/O. Consideration of QoS within query processing means the inclusion of other dimensions such as the cost of the query, the data quality, or the throughput of the database systems. Single optimization goal strategies deployed in the traditional database optimizers cannot satisfy such QoS requirements. We argue that query optimization should take into account user-defined quality of service constraints[2]. In an electronic commerce application for example, a user could specify QoS requirements such as: "*I want the most up-to-date information even if it takes time. However, if the response time is longer than 3 minutes, I will accept less recent information, but only if it is less than 10 hours old*". Based on the specified QoS requirements and using the QoS metadata, in this example the query optimizer has to choose the most up-to-date information from the catalogs.

In our approach, the treatment of QoS requirements is reflected in the aspects of integrating multiple optimization goals and how to select a query access plan that is overall optimal. The related issues consist of identifying the possible optimization goal, the selection of cost models, the way to obtain the user's priority between different optimization goals, and how to obtain an overall optimal goal according to the user's preference.

In this paper, we propose an approach to integrate user-defined QoS requirements, in addition to the dynamic properties of the system components involved, into a distributed query processing environment. We then propose a query optimization strategy in which multiple goals may be considered with several cost models. Furthermore, we discuss some experimental results confirming the effectiveness of our approach.

The rest of the paper is organized as follows. The next section describes our QoS-based distributed query

*This work was supported by a grant from the Canadian Institute for Telecommunication Research (CITR), under the Network of Center for Excellence Program of the Canadian Government, a collaborative research and development grant from NSERC no CRD-226962-99, by a student fellowship from IBM and an individual research grant from NSERC no RGPIN138210.

processing. Section 3 presents the results of our experimentation. Section 4 discusses related work. Section 5 concludes and suggests future work.

2. QoS-based query processing

To support QoS in database systems, we propose to enrich query processing with some QoS features. We consider factors like user requirements, dynamic network performance, and dynamic server load in the procedure of global query processing. By global query processing, we mean that we position our work on top of existing database systems. The QoS features are plugged into the query processor that deals with inter-database operations. Therefore, our method does not require the modification of local database query processors. The main objective is to provide a flexible QoS model for multidatabase management systems and to offer differentiated services.

We base our work on classes of users, cost models for distributed query processing, and utility functions to describe system or user satisfaction for different optimization goals. Usually the utility function maps the value of one QoS dimension to a real number, which corresponds to a satisfaction level. For example, the following formulas give the utility functions for the response time and the service charge:

$$u_t(t) = 1 / t$$

$$u_\$(x) = 1 / x$$

where t is the response time for a query plan and x is the corresponding service charge for that plan. Utility functions are used in our cost model to achieve an overall optimization since it is used to compare the quality of the access plans. It also provides an important link between the quality of a query plan and the user satisfaction. A user class is a generalization of a number of users sharing common characteristics. Classification of the users may be based on different policies and criteria[3]. For example, different users may exhibit various patterns of navigation through an e-commerce site, therefore based on the user's *navigation behavior*, we may segregate users into two classes: *buyer* and *browser*. We propose a new approach to the problem of evaluating the cost of a query plan in a multidatabase system. Our cost models are adaptive in the sense that first, they combine multiple optimization criteria (for example response time and money cost, into a simple cost model) and second they can give a more precise response time estimation based on the information captured by QoS monitoring of the network and the server.

2.1. Query processing and optimization revisited

When designing the QoS-based query processor we are guided by two main goals: 1) recognition of individual user requirements, and 2) consideration of the dynamic nature of the underlying system. A logical architecture is proposed in Figure 1 to show the relationships between QoS management and query processing.

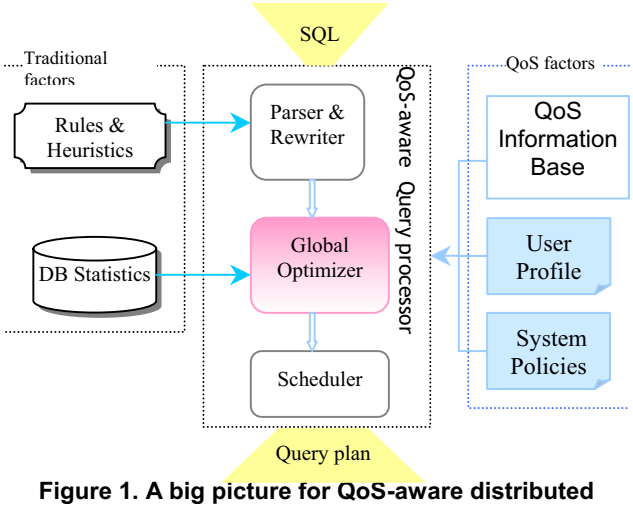


Figure 1. A big picture for QoS-aware distributed query processor

In this framework, we include the typical components introduced in [4]. The user's query is sent to the *Parser* to be syntactically analyzed and validated against the database schema. The output of the parser is transformed by a set of rewriting rules in the *rewriter*. These rules correspond to heuristics that transform the query into a semantically equivalent form that may be processed more efficiently.

The main tasks of the *Global Optimizer* are 1) choose an execution plan which satisfies the optimization objectives and 2) send it to the scheduler who coordinates the execution of the plan among the participating component DBMSs. We keep the traditional factors[4] considered in the query processor. These typical factors include table statistics, column statistics, and index statistics. In addition, we include the QoS factors, which are information from the *QoS Information Base (QoSIB)*, the *User Profile*, and the *System Policies*.

Adding QoS factors into a distributed query processing environment has several impacts and requires:

- to provide new optimization goals;
- to modify the corresponding cost models; and
- to propose a new algorithm for query optimization.

2.2. User profiles, QoS monitoring information and system policies

Pushing QoS into a distributed query processing environment requires the description of the information related to the user's requirements, the QoS level provided by the different system components and the objectives of the system in terms of resource allocation.

User profiles. A user profile is built to store the user's QoS expectation for a particular service. The QoS expectation is expressed according to different QoS dimensions[5]. For example, a good quality of service level may be expressed by the dimensions of response time and dollar cost. The user profile allows users to specify their QoS requirements by defining utility functions for each dimension. As mentioned previously, a utility function translates the values of an attribute into "utility" units. We consider decreasing utility functions since this type of utility function is practical in the case that the utility decreases with the increasing of one QoS dimension. Examples of such dimensions are response time and service charge.

The user profile is also useful to derive the trade-off between QoS dimensions, which is represented by the weight assigned to each dimension. In our approach, the Analytic Hierarchy Process (AHP) [6] is used to derive the weights from user's preference. This method only requires the user to provide his or her judgments about the relative importance of each criterion over another one (pairwise comparison of goals) and then specify a preference index. Based on these preference indexes, the output of the AHP is a prioritized ranking indicating the overall weights for each of the alternative decisions. In short, the utility functions and weights are then used to guide the optimizer for selecting a query access plan.

QoS information base. The QoS information base (QoSIB) stores some information about the service level offered by the different system components. Since we are working in the context of Internet-like networks, the performance of the TCP protocol is our key consideration when talking about network performance. Among all the performance factors, *TCP throughput* and *TCP delay* are two key parameters considered in our distributed query processor.

For the server performance category, the parameters of interest include availability and server load (CPU usage, memory usage, and the frequency of disk I/O). The availability is the fundamental measurement of a server. It includes the availability of hardware as well as the software. In our research, we refer to the availability of the database services. In our prototype, QoS information is stored as XML files.

System policies. We believe that many future applications, especially e-commerce systems, will be able to provide different levels of service to different classes of users[7]. In the simplest sense, the policy consists of one or more rules that describe the action(s) to occur when specific condition(s) exist[8]. In our study, the system policies determine the constraints under which the system resources can be used for providing services to the users. Usually, a policy is a formal set of statements that define the levels of services to be provided to particular classes of users. If written in a natural language, policy statements may take the following forms:

"Give the *VIP* users the best service"

"Give the *normal* users the resource-effective service"

Different policies may be enforced to different classes of users. Policy statements are stored in *System Policies*. The parameters that make up a system policy include the optimization goals defined (as presented in Table 1), user class information, and the weighting factors associated with each goal.

Table 1 Example of optimization goals

Optimization category	Optimization goal
Performance oriented	- Minimize response time - Maximize DB throughput
Money oriented	- Minimize the cost of a service - Maximize the benefit of the database system
Data quality	- Multimedia vs. Plain text - Recency of data
System oriented	- Minimize resource utilization

When various optimization goals exist along multiple QoS dimensions, we should find an *optimal* solution that satisfies all of them, optimal either from the user perspective or the system perspective, or both. One way of combining various optimization objectives is to use *weighted combination* (for example, a weighted sum) of different goals. A weighted combination can express the overall satisfaction of all the optimization goals. The user must be presented with enough options that his or her desires can be adequately expressed and they can then be mapped to weighting factors associated with the different objectives.

All this information is later integrated into the QoS-aware distributed query processing for access plan selection. Different optimization goals may lead to different cost models or query processing strategies. In the performance category, the cost factors comprise the measures of local processing time, the communication time as well as some overhead due to parallelism. There are two types of query parallelism: *inter*-query parallelism (which enables the parallel execution of multiple queries) and *intra*-query optimization (which makes the parallel execution of multiple operations possible within the same query). For the optimization goals related to the monetary, the cost measures include information on the resource usage and the pricing scheme.

2.3. Global Query Optimization

Global query optimization is generally implemented in three steps[4]. After parsing, a global query is first decomposed into query units (subqueries) such that the data needed by each subquery is available from a single

local database. Second, an optimized query plan is generated based on the decomposition results. Finally, each subquery of the query plan is dispatched to the related local database server to be executed and the result for each subquery is collected to compute the final answer.

In our study, we focus on the first two steps and map them to the problems of *global query decomposition*, *inter-site join ordering* and *join site selection*[10]. Before describing these three steps, we give an explanation about the evaluation of the cost of query plans.

2.3.1. Evaluating the cost of query plans. We propose a new approach to the problem of evaluating the cost of a query plan in a multidatabase system. Our approach relies on the information from QoS monitor user profiles. The novelty of our approach lies in the consideration of user requirements, user classes as well as the way to deal with dynamic network performance.

In our work, three levels of cost models are used. The first level is the global cost model, which is used to calculate the overall utility of a query access plan. The second level is used to calculate the cost for each node in a query access plan. The last level is the local cost model, which is used to estimate the cost of an operator locally.

Global cost model. As discussed earlier, multiple optimization goals over different QoS dimensions are considered in our query optimizer. Consequently, the global cost model should reflect them. For our cost model, we adopt the method proposed for multi-criteria optimization in Operations Research area. Accordingly, the general cost model for one user is

$$max\{\sum_{i=1}^{n} \omega_i \cdot u_i(C_i)\}$$

where u_i is the utility function for cost component C_i (based on one of the QoS dimensions i); ω_i is the weighting factor assigned to the cost component C_i. Note that we want to maximize the utility for a given user; therefore this model could also be called a utility model. The range of ω_i is [0,1] and $\Sigma\omega_i = 1$.

Plan cost model. A query access plan is represented by a binary tree. Each internal node is an inter-site join operation and each leaf node is the subquery executed at one database server. Since we consider several cost components, the cost of each node is also expressed according to multiple dimensions. For example, if we select the response time, the service charge and the availability as our cost components, then the cost information recorded in each node will include three parts: time, dollar, and availability. The cost information for leaf nodes is based on the local cost model and the QoS Information Base (e.g. availability). The cost information for the internal node is calculated as a combination of the cost information of its left and right child nodes. The cost formula for each QoS dimension is different. Table 2 lists the cost functions for time, dollar, and availability. The join time for each node is determined by the load of the server and the current TCP performance. The formula for each join is:

$$T_{join} = \text{local}(site, query) + \text{net}(site_i, site_j)$$

where local (*site*, *query*) represents the local execution time for the *query* at *site*, net ($site_i$, $site_j$) represents the data transfer time spent over the network.

Table 2 Cost functions for each cost component

Cost Component	Cost function	Brief Description
Response time	Join-time + max (left.respose_time, right.response_time)	The join time is the response time to perform the join between the left and the right child.
Service Charge	Join-charge + left.charge + right.charge	The join charge is the money cost to perform the join between the left and the right child.
Availability	Left.availability * right.availability	The probability that both servers are available.

Figure 2 shows an example query plan marked with cost information for each node. We use a vector (time, money, availability) to record the cost information for each node in the plan tree. By using this representation, the cost information for the root node of the tree is the plan cost. Each item in the vector associated with that node is computed using the formula given in Table 2.

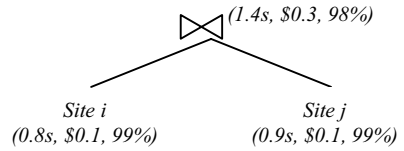


Figure 2. Cost calculation for a join node

Local cost model. As mentioned earlier, the local cost information relies on the estimation of the execution of a query at a local server, the pricing policy applied by the local server for a service charge and the server availability. Each local database server must report the price and the availability. However, the execution strategy, and therefore the execution time, of a query is hard to obtain since local database systems do not report the needed statistical information. To estimate the local database cost, we adopt the sampling method[9], where multiple regression models are used to guess the local cost structure (in terms of time). The idea of the query sampling method can be characterized by the following steps: 1) queries are classified according to a number of criteria; 2) sample queries from each class are selected and issued to run against the local database; and 3) the response time is then measured to derive the local cost

model by multiple regression analysis. Such a cost formula includes a set of variables that affect the costs of queries and a number of coefficients that reflect the performance behaviour of the underlying DBMS. Due to space limitation, we will not give detailed information here. A complete discussion can be found in[11].

2.3.2. Global query processing. In our work, global query processing is implemented by three steps: global query decomposition, join ordering, and join site selection.

Global query decomposition. The main task of the global query decomposition is to break down a global query into several subqueries so that the tables involved in each subquery target one location. This is an NP-Complete problem[12]. Therefore, this step is usually guided by heuristics. Two goals used in our algorithm are to simplify the optimization at the global level and to reduce the data transmission among different sites. Therefore, the heuristic used is to decompose a global query into the largest possible subqueries.

The cost model used for this step mainly depends on the local information, based on the optimization goal selected. For example, if the optimization goal is the response time, the cost model could be the response time for each subqueries under various server loads. We do not consider data transfer in this step; therefore communication cost is not involved. The QoS factor considered is mainly the system performance information from QoS information base.

Join ordering. The global query decomposition phase generates a set of subqueries with location information. In the following join ordering step, the optimizer tries to come up with a good ordering of how to combine these joins between subqueries. The join ordering can be represented as a binary tree, where leaf nodes are the sub-queries and internal nodes are inter-site join operations. Because we want to utilize the distributed nature of the multidatabase system, we try to make this tree as low as possible, which means we hope the join can be done in parallel as much as possible.

A typical way is to generate a linear tree first and then balance this linear tree to a bushy tree [10][12]. Following the same method, we first build a left-deep tree using dynamic programming. The next step in the join ordering is to transform the left deep join tree into a more balanced bushy join tree. A feasible approach is to apply a sequence of basic transformations that can be easily identified and performed[11].

The cost models used in this step consist of both global cost model and local cost model. In this step all the QoS factors introduced in Section 3.2 are included in the decision.

Join site selection. In case of data duplication, one subquery might have several potential locations, thus the optimizer should decide at which location this subquery will be executed. Like the join ordering problem, all the QoS metrics are taken into account.

The key issue in the site selection is to decide which site is the best (depending on how the user defines his or her optimization goal) for each binary operator. Traditionally, the possible site to perform the join or the union is chosen from one of the operand sites, i.e. the site where one of its operands is located. However, there may be circumstances when shipping the two operand tables to a third site is a better solution, in terms of response time. We call the join site to be a *third* site if the selected site is neither of the operand sites.

For a binary operator node such as join or union, the selection process becomes complicated when several *third* sites are capable of handling the operator node.

After we decide which candidate set to choose for the "third site", the procedure of join site selection can be regarded as deciding (based on the cost model) the site for each internal node (which is usually the inter-site join operation) in the query access tree. This process may be done in a bottom-up fashion. In our algorithm, we use post order tree traversal to visit the internal nodes of the tree[11].

2.4. Prototype implementation

In order to validate our approach, we implemented a prototype where we concentrated on those aspects that are representative for the QoS-based distributed query processing we propose. For simplicity, we only integrate two QoS dimensions in the prototype. However, the implementation is not limited to these two dimensions, the modules implementing other dimensions can be easily plugged into our prototype. Highlights of the implementations are given below.

1) User classes: In order to show the differentiated services in our prototype, we have adopted the priority-based user classification and considered two user classes, namely *VIP user* and *normal user*.
2) QoS consideration. The dynamic characteristics of the underlying systems for our QoS consideration are *network performance*, *server load*, and *availability*. For the network performance, TCP performance is our main concern. Accordingly the QoS dimensions we considered are *available bandwidth* and *delay*. For the consideration of server load, we categorize the load into four levels: *no load*, *low*, *medium*, and *high.* They are used to show different levels of resource contention. In addition, a server is also characterized by its availability (*yes* or *no*).
3) Optimization goal. For our prototype implementation, we focus on two optimization goals: minimize the response time and/or the service charge. Basically, we want to demonstrate the integration of the criteria of *time* and *money* into our prototype. Accordingly the

overall optimization goal is calculated by the following formula:

$$\text{Min} \{ \omega_t u_t(\text{response\_time}) + \omega_\$ u_\$(\text{service\_charge}) \}$$

where ω_t and $\omega_\$$ are the weights specified by the users for the response time and service charge, respectively; u_t and $u_\$$ are utility functions used for the response time and service charge respectively. For the purpose of simplicity, we assume the utility function for the response time and the service charge are the utility functions given at the beginning of Section 2.

4) Global cost models. The global cost model (as explained in Section 2.3.1) contains two cost components: response time and service charge. Depending on the optimization goals, three cost models can be selected:
   i. C_{time} = response_time;
   ii. C_{dollar} = service_charge;
   iii. $C_{overall} = W_{time} * u_t(\text{response\_time}) + W_{dollar} * u_\$(\text{service\_charge})$

   The detailed cost model information can be found in [13]. The calculation of the response time is straightforward. The total response time of a query plan (represented as a tree structure) is the sum of the response time on each node along the critical path in the query access tree.

   For the service charge, we are dealing with a pricing issue. Typically, two types of charging schemes are popular today. They are *flat-rate* and *usage-based* [14]. We adopt the usage-based pricing policy for our prototype implementation. We concentrate on network bandwidth utilization. A complete pricing schema, however, should consider all the resources including both the network and the server. The reason for only considering the network resource is not only because we want to simplify the implementation, but also because there have already been many studies for the pricing for the Internet. We assume the service charge of a query plan is proportional to the network resource consumed. Accordingly, this second optimization goal is eventually simplified as the problem of minimizing the network bandwidth utilization.

Prototype architecture. The functional modules of the prototype include the *user interface* part for SQL input and QoS schema selection, the *optimization part* based on the algorithms proposed, the *visualization part* for the query plan and QoS information and the *result display* part. Our prototype offers a simplified GUI for SQL input. This component allows a user to specify a query by selecting the desired attributes and tables as well as join and restriction predicates.

The user can also choose to view the XML representation of the specified query that will be forwarded to the optimizer by clicking the "show query (XML)" button. The other component integrated with the SQL Input GUI is the User Preference manager, shown in the lower part in Figure 4. In this part, the user can select his trade-off between the response time and the service charge. The sliding bars are used for this purpose and this ratio is further integrated in the optimizer to derive the overall optimization goal.

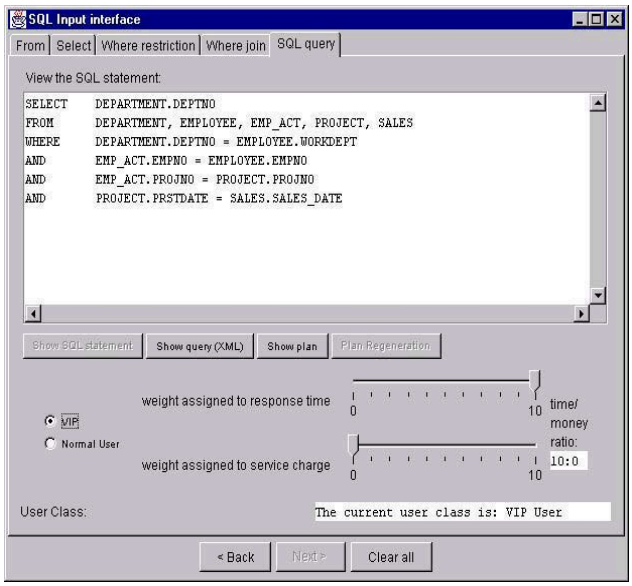


Figure 4 An example of SQL input interface and selection of query preference

When an SQL query and the QoS preferences are specified by the user, he/she can see the generated query access plan. For the query specified as in Figure 4, the query plan shown to the user will look like the one shown in Figure 5.

In short, through the implementation of the prototype, we have demonstrated the following points:

- Different user classes are provided in the prototype. Users are classified based on priority and a system policy is made for each user class;
- Two optimization goals are supported in the current prototype, according to two QoS dimensions: response time and service charge. The overall optimization goal is achieved by using the weighted sum of the resulting utility functions applied for different goals;
- Different query access plans can be generated for different user classes;
- Dynamic QoS conditions for systems may affect the decision. The system parameters include both the network information and server characteristics.

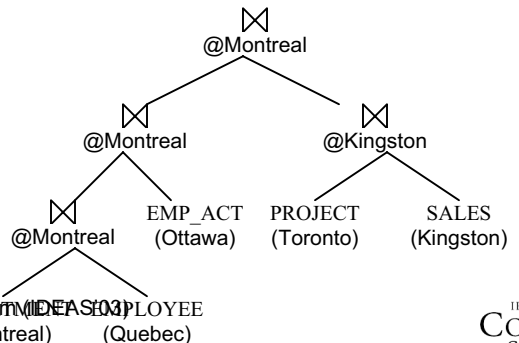


Figure 5 An example of generated query access plan

3. Experimentation

In this section, we evaluate the performance of our QoS-based query processing strategy according to the framework proposed in the previous sections. The objective of our experiment is to show that our query optimizer can adapt itself to workload changes (both server load and network load) and always chooses the best plan for different user classes. In the experiment we simulate two classes of users: *VIP* user and *normal* user.

The specific goals of the experimentation are two-fold: (i) how our estimated plan cost (in terms of response time) is close to the real execution cost; and (ii) what are the quality of service for VIP user and normal user under different workloads (we focus on response time in the experiment). Corresponding to these goals, two sets of experiments were set up. Two types of system loads are used for our measurement, one for network and the other for server load. For network load, we mainly focus on the available bandwidth as the indication of network congestion level. For server load, we concentrate on the CPU utilization as the indication of server load.

In the first experiment, named *estimated vs. executed*, we take the query plan generated by the prototype and execute under different server loads. The network bandwidth used for the plan estimation is 5Mbps since this is the most representative maximum bandwidth during the daytime according to our observation between University of Montreal and University of Ottawa.

The second experiment, named *VIP vs. normal* is designed to measure the response times for VIP users and normal users under different server loads and network congestion levels. The 3-way join with different resulting cardinalities is used for the second experiment.

3.1. Experimental setup and assumptions

All tests were performed under Windows NT 4.0 (SP 6) on a single Pentium III CPU and 192MB RAM. The tables used in this experiment are based on the SAMPLE database provided by the DB2 Universal Database [15]. The size of the database is about 7.5KB.

All the reported execution times of our experiments represent the average of executing the query 20 times. The purpose of this averaging is to avoid the influence of disk I/O to our result. In the measurement of the data transfer times, we have not included the disk I/O time for retrieving a table into memory in order to send it over the network. To simplify the discussion and highlight the points of interest, we disregard the disk I/O.

We mentioned in the previous section that the network congestion level and various server loads are two major system dynamics for our experiment. To study their influence in our prototype, we usually fix one and change the other to collect the performance numbers. It should be noted that as an experimental prototype, our execution engine was designed for ease of implementation and has not been tuned for performance. The main purpose is to demonstrate the feasibility of our ideas in practice.

3.2. Workload classification

The workload in the experiment includes both server load and network load. Concerning the server load, in our experiments we degrade the performance of one server by loading it with additional processes. Each process simply eats up CPU and competes with the database system for CPU utilization. Additional load is quantified by the number of these processes spawned on a server. The reason we concentrate on the CPU is that the buffer pool size for the SAMPLE database is about 1MB (250 pages, with size of 4KB for each page), which is more than enough to hold the whole database. Therefore, the number of disk I/Os does not affect our experiment result very much. As discussed before, we categorized the server load into 4 levels: *no load*, *low load*, *medium load*, and *high load.*

As for the network load, we consider the TCP congestion level. In our global database schema, we assume the data are distributed among different cities in Canada. Because performing experiments directly on the Internet would not provide repeatable results, we instead modeled the behavior of the network using trace data that could be easily relayed. Therefore we need to have knowledge about the available TCP throughput between two cities. We choose Montreal and Ottawa as our experimental base. For this purpose, we observed the TCP traffic using IPERF[16] between UdeM (University of Montreal) and UO (University of Ottawa).

The measurements were made in the morning, in the afternoon and at night each day, and statistics were collected. Based on the observation, we find the maximum bandwidth ranging from 0.2Mbps to 10 Mbps depending on the time of the day. Within this range, 5Mbps is the normal throughput during daytime and 8Mbps is the normal throughput at night. When the network is congested, 2Mbps is the throughput we saw around 4pm to 6pm. Very occasionally, we got 0.1 to 0.2 Mbps. These data are used to define our congestion level. The corresponding congestion level is defined in Table 3. A throughput of 8Mbps is regarded as no congestion.

Table 3 Measured network congestion levels

Network bandwidth	0.1Mbps	0.2Mbps	1Mbps	2Mbps	5Mbps	8Mbps
Congestion level	5	4	3	2	1	0

3.3. Result

We conducted a number of experiments and performance data are collected for the two sets of experiments identified previously.

Estimated versus execution time. In the first set of experiments, *estimated vs. executed*, we first varied the workload of the server. Then under different loads, a plan is generated with an estimated time. This plan is then executed and the observed execution time is recorded for the purpose of comparison. The network congestion level for all links is 1, that is 5Mbps. Figure 6 gives the plots for the comparison of two times under different server loads.

In Figure 6, the estimated times are given in dotted line and the collected times are given in solid line. As it can been seen from the figure, the two curves for each load are very close. We also analyze the result statistically by constructing a linear regression model of these two times. The regression results (detailed in [11]) indicate that the estimated times can explain about 95% of the real execution times.

We compare the execution times for VIP users and normal users under different server loads in Figure 7. The curves marked with square and triangle signs represent the performance for normal users and VIP users, respectively. As we can see from the figure, under no load, all the users will get the same performance (the lowest curve in Figure 6). With the increasing load, the VIP user always stays at the same curve (the same performance), while the normal user will get higher response time (the curves marked with square sign). And the advantage of performance for VIP users increases with increasing server load. In short, Figure 7 shows that the VIP users always get best performance while the normal user will suffer the slow response when the load increases.

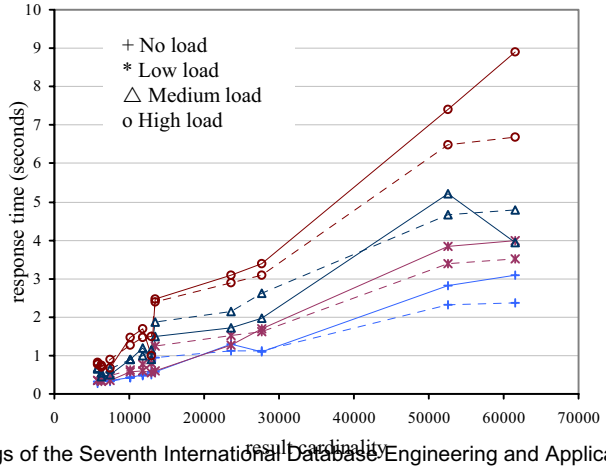


Figure 6. Estimated versus execution time, with various server loads

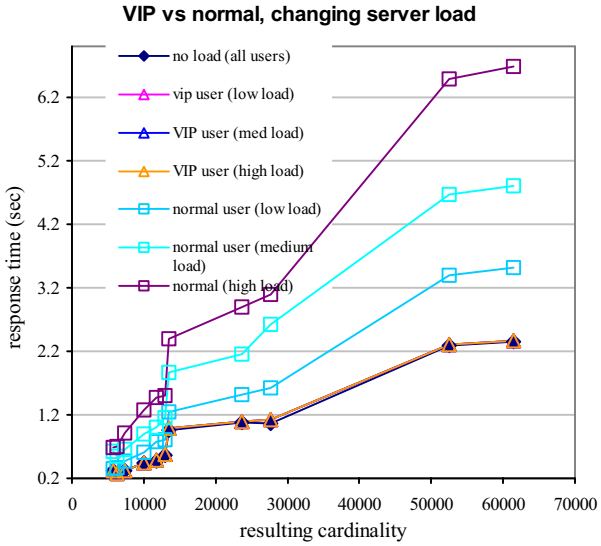
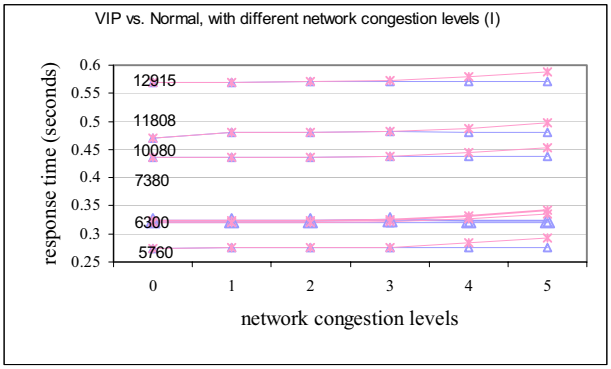
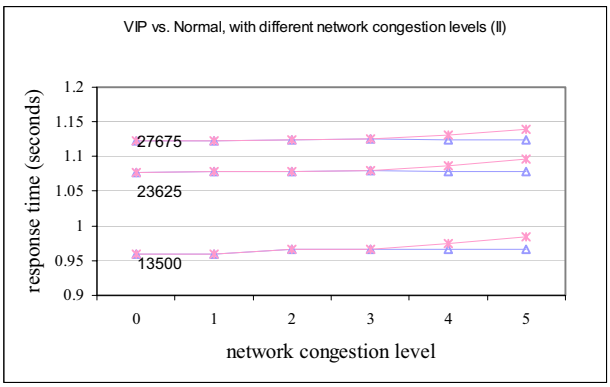


Figure 7. VIP vs. normal user with various server loads

Figure 8 depicts the effect of network congestion on the performance. In this set of tests, we assume that the links among the nodes involved in the join are congested while other links have the normal throughput (5Mbps). In addition, there is no load of the server during the experimental periods. Again, estimated times are used for the comparison of this experiment. Each of the curves in Figure 8 has six data points, which correspond to the six congestion levels. The curves marked with a triangle sign represent the VIP user. The curves marked with a square sign represent the normal user. We observe the same trend as in the load test, whenever the links are congested to a certain level (usually at level 3), the plan for the VIP user can choose another smooth route for data transformation and maintain the fast response time. Since doing so may incur extra data transmission, and this is regarded as "expensive" for normal users, the normal user will experience a slower query response in these cases.



(a)



(b)

Figure 8. VIP vs. normal user, with various network congestion levels

3.4. Experiment summary

In the experiment discussed in this section, we first evaluated how close the estimated query execution time comes to the real execution time. The results shown in Figure 6 illustrates that under various server loads, the observed response time is very close to the estimated time. We then demonstrate, through the second set of experiments, that under all the circumstance the VIP user will get fast response time, or in general the better service.

Using the same experimental setup, we can also compare our algorithm for join site selection (which considers a third candidate site) with the traditional one (which always ships the small table to the large table site). The results in [11] also show the superiority of our algorithm over the traditional algorithm under different system loads. The experiment described in this section attempts to demonstrate the feasibility of the integration of QoS into distributed query processing, which means different treatments for different classes of users. Although our initial experimental result is a very first attempt and is subject to future refinement, this first attempt gives a fairly clear picture showing the capability of delivering QoS differentiation in query processing.

4. Related work

In the last decade, several approaches have been proposed for decomposition and optimization of queries across different data sources. They can be classified into two categories: 1) strategies for providing universal access over multiple information sources and 2) dynamic and adaptive query optimization strategies. Proposals for the first category are based on mediator architectures, where different data sources are described and integrated. Different query capabilities are taken into account during the query optimization. Such approaches include Garlic[17], IRO-DB[18] and Mariposa [19]. The query optimizer implemented in Garlic uses enumeration rules for describing query capabilities and uses dynamic programming to find a good plan. IRO-DB provides federation of object-oriented and relational database systems through the ODMG model and the OQL query language. The global query processor uses services of local cost tuners and their corresponding calibrating procedure to derive the local cost parameters. The originality of the approach proposed in Mariposa is its economic model in the query optimization phase. The bidding mechanism allows sites to observe their environment from query to query, and autonomously restate their costs of operation for subsequent queries.

The approaches proposed in the second category generally provide techniques for dealing with delays in data processing and transfer at remote sites [20][21] and dynamic query processing [22][23]. Our approach falls into the second category and we propose to use QoS monitoring tools to push dynamic properties of the systems into global query optimization. The novelty of our approach lies in the fact that we take the user QoS requirements and the system policies into consideration to support several optimization goals.

5. Conclusion and future work

In this paper, we have proposed a general framework for integrating QoS requirements in a distributed query processing environment. This framework is based on user classes, cost models, utility functions, and policy-based management. Our approach allows to offer differentiated services to different classes of users according to their expectations in terms of QoS. We have presented our QoS-based distributed query processing strategy where we push QoS requirements and information into the different steps of global query optimization: global query decomposition, join ordering and join site selection. We presented the prototype we have developed as well as experimentation we have conducted to validate our approach. The current prototype considers two classes of users as well as two different optimization goals. In the future, we will consider other QoS dimensions to be specified by the user, such as data quality or freshness and will work on rewriting rules to transform specifications on these dimensions into optimization goals and corresponding cost models. To test the feasibility of our method, we designed a very simple scenario. To test our algorithm in a more general case, further experiments should be conducted on a larger and real database system.

References

[1] C. Aurrecoechea, A. Campbell, and, L Hauw, A Survey of QoS Architectures. ACM Multimedia Journal, 6, May 1998, pp. 138-151
[2] H. Ye, B. Kerhervé, G. v. Bochmann, QoS-aware distributed query processing, DEXA Workshop on Query Processing in Multimedia Information Systems (QPMIDS), Florence, Italy, 1-3 September, 1999
[3] D.A. Menasce, V. A.F. Almeida, Scaling for E-Business Technologies, Models, Performance, and Capacity Planning, Prentice Hall Canada, 2000
[4] D. Kossmann, The state of the art in distributed query processing, ACM Computing Surveys (CSUR), Volume 32, Issue 4, December 2000, pp 422 – 469
[5] Frolund, S., & Koistinen, J., Quality-of-Service Specification in Distributed Object Systems. Distributed Systems Engineering Journal (December 1998), vol 5 no 4, pp 179-202.
[6] T. Saaty. Multicriteria Decision Making - The Analytic Hierarchy Process. Technical report, University of Pittsburgh, RWS Publications, 1992
[7] G.v. Bochmann, B. Kerhervé, H. Lutfiyya, M. M. Salem, H. Ye, Introducing QoS to Electronic Commerce Applications, Second International Symposium, ISEC 2001 Hong Kong, China, April 26-28, 2001, pp 138-147
[8] A. Westerinen, J. Schnizlein, J. Strassner, M. Scherling, B. Quinn, S. Herzog, A. Huynh, M. Carlson, J. Perry, S. Waldbusser Terminology for Policy-Based Management, November 2001
[9] Q. Zhu, Y. Sun and S. Motheramgari, Developing Cost Models with Qualitative Variables for Dynamic Multidatabase Environment, Proceedings of IEEE Int'l Conf. On Data Eng. (ICDE2000), San Diego, Feb 29-March 3, 2000, pp 413-424
[10] W. Du, M.-C. Shan, U. Dayal, Reducing Multidatabase Query Response Time by Tree Balancing. SIGMOD Conference 1995, pp 293-303
[11] H. Ye, Integrating Quality of Service Information and Requirements in a Distributed Query Processing Environment, Ph.D thesis (preliminary draft), University of Montreal, 2002
[12] C. Evrendilek, A. Dogac, S. Nural, and F. Ozcan, Multidatabase Query Optimization, Distributed and Parallel Databases, Volume 5, 1997, pp 1-39
[13] H. Ye, G.v. Bochmann, B. Kerhervé, An adaptive cost model for distributed query processing, UQAM Technical Report 2000-06, May 2000
[14] A.M. Odlyzko, Internet pricing and the history of communications**,** *Computer Networks* 36 (2001), pp. 493-517
[15] DB2 UDB Administration Guide V7.2, http://www-4.ibm.com
[16] National Laboratory for Applied Network Research, http://www.nlanr.net/
[17] Hass, L., Kossmann, D., Wimmers, E., Yang, J. Optimizing queries across diverse data sources in Proceedings of tthe Conference on Very Large Data Bases (VLDB), Greece, Aug. 1997, pp276-285
[18] G. Gardarin, F. Sha, and Z. Tang, Calibrating the Query Optimizer Cost Model of IRO-DB, an Object-Oriented Federated Database System, Proceedings of the 22nd VLDB, Mumbai (Bombay), India, 1996, pp 378-389
[19] Stonebraker, M. and al. Mariposa: A Wide-Area Distributed Database System, VLDB Journal, 5,1 (January 1996) pp48-63
[20] Z. Ives, D. Florescu, M. Friedman, A. Levy, D. Weld, An Adaptive Query Execution Engine for Data Integration Proc. of ACM SIGMOD Conf. on Management of Data 1999
[21] J. Hellerstein, M. Franklin, S. Chandrasekaran, A. Deshpande, K. Madden, S. Hildrum, V. Raman, M. Shah, Adaptive Query Processing: Technology in Evolution. In IEEE Bulletin on Data Engineering, vol 23, no 2, 2000, pp 7-18
[22] R. Cole, G. Graefe, Optimization of Dynamic Query Evaluation Plans. SIGMOD Conference 1994, pp 150-160
[23] T. Urhan, M.J. Franklin, and L. Amsaleg, Cost-based Query Scrambling for Initial Delays, SIGMOD'98, Volume 27, Number 2, Seattle, June 1998