Integrating Quality of Service into Database Systems*

Haiwei Ye[1], Brigitte Kerhervé[2], and Gregor v. Bochmann[3]

[1] Département d'IRO,
Université de Montréal, CP 6128
Succ. Centre-ville,
Montréal Québec, Canada H3C 3J7
ye@iro.umontreal.ca
[2] Département d'informatique,
Université du Québec à Montréal, CP 8888,
Succ. Centre-ville,
Montréal Québec, Canada H3C 3P8
Kerherve.Brigitte@uqam.ca
[3] School of Information Technology & Engineering,
University of Ottawa
P.O. Box 450, Stn A,
Ottawa Ontario, Canada K1N 6N5
bochmann@site.uottawa.ca

Abstract. Quality of Service (QoS) management has attracted a lot of research interests in the last decade, mainly in the fields of telecommunication networks and multimedia systems. With the recent advance in e-commerce deployment, it clearly appears that today's web applications will require the integration of QoS mechanisms to specify, declare and support the different service levels they can provide. Such mechanisms should therefore be integrated in the different components of the core technology and more specifically in database systems. In this paper, we present an approach to push QoS inside database systems. This approach integrates QoS requirements into distributed query processing and considers also the dynamic properties of the system. We propose a query optimization strategy where multiple goals may be considered with separate cost models.

1 Introduction

To support QoS activities, mechanisms have been mainly provided for individual components such as operating systems, transport systems, or multimedia storage servers and integrated into QoS architectures for end-to-end QoS provision [1]. None of these proposals takes database systems into consideration. However, database systems are an important component of today's distributed systems. We consider them a major player in QoS management.

* 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.

In the context of a research project funded by the Canadian Institute for Telecommunication Research (CITR), we investigate how to support different QoS levels in distributed systems [2] and more specifically for e-commerce applications. We address two complementary issues. On one hand, we study the management of QoS information to support distributed QoS decision models. For that purpose, we are currently designing and implementing an extensible QoS Information Base (QoSIB) manager offering basic services to store, access, share, transfer, produce or analyze QoS information [3]. On the other hand, we work on pushing QoS inside database systems [4][5][11]. More specifically, 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.

In this paper, we focus on the second issue of our research. We present the general principles of our approach and we describe the query optimization strategy we propose where multiple goals may be considered with several cost models. A complete description of this work, together with experimental results can be found in [6].

The rest of the paper is organized as follows. The next section introduces the QoS concepts related to the proposed approach. Section 3 describes our QoS-based distributed query processing strategy. Section 4 provides a short conclusion.

2 Pushing QoS inside Database Systems

Many efforts have been directed at the provision of QoS at the network level. We believe that QoS concepts could and should be broadened from networking to the database area. However, there has been a lack of discussion of QoS issues in database systems. This observation opens possibilities to introduce QoS concepts in database systems.

What are the QoS issues in database system? The quality of service provided by the database systems in an e-commerce application is less likely perceivable by the end user. However, the implementation of QoS for the whole system requires each system component to be QoS responsive. In our work, different QoS requirements and constraints specified by end users are mapped to different optimization goals for the database system. Traditional database systems cannot be directly used for this purpose since they are designed to provide a single optimization goal. Another reflection of QoS requirements is the integration of dynamic system properties. The consideration of the user's QoS preferences is modeled into user classes. In this section, we concentrate on concepts that are important to our work: classes of users, multiple optimization goals, and dynamic system properties.

2.1 Classes of Users

The involvement of the user is crucial in e-commerce applications; accordingly various user requirements and QoS should also be available from the underlying DBMS. Defining user classes is a way of differentiating users according to their QoS expectation in order to provide different levels of service. 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 [7]. 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*. Another example of segregating users could be based on *priority*. In this classification, we differentiate users into different classes according to, for example, their profit brought to an e-store. Thus an e-commerce site tries to provide *higher priority* users with better service than *lower priority* users.

2.2 Various Optimization Goals

Conventional distributed/parallel database query optimization was primarily aimed at either minimizing the response time or system resource utilization. However, in the context of emerging applications, such as e-commerce, this provision of a single optimization goal is not adequate. Therefore, other possible optimization goals should be proposed and integrated into the optimizer. Table 1 lists some optimization goals that are useful for our study. They are grouped into different categories. Some optimization goals are *performance oriented*, for example, the response time, and the throughput of the database system. Others are *money oriented*, one example is the service charge for a particular service.

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. The weight assigned to each goal is explicitly specified by the user.

The satisfaction for each optimization goal or QoS metric can be captured by using a *utility* function. By indicating different utility functions, the user expresses his/her individual tastes. 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 access 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 they are used to compare the quality of the access plans. Utility functions also provide an important link between the quality of a query plan and the user satisfaction.

2.3 Dynamic System Properties

Two main challenges imposed by today's e-commerce applications are *diversity* and *unpredictability*. Diversity includes various user expectations and the heterogeneity of the database systems, network types and the machine power. The unpredictable nature comes from the varying network performance (especially for the Internet-based networks) and server load at different times. To capture these dynamic properties of the systems, we rely on the QoS monitor to offer the time-changing information. This information is then modeled in the relevant cost models of query processing.

2.4 An Example

In order to illustrate the idea of user classes and multiple optimization goals, we give an example. Assume we are interested in two QoS dimensions: response time and money. Suppose that the query optimizer compares two query access plans. The related information is given in Table 2.

Table 2. An example of related information for two query plans

	Plan a	**Plan b**	**User 1 Weight**	**User 2 Weight**
Response time	0.01s	0.009s	0.8	0.2
Money	\$0.05	\$0.10	0.2	0.8

Also assume that the utility functions are the formula defined in Section 2.2, that is $u_t(t) = 1 / t$ and $u_\$(x) = 1 / x$. In our approach, weighed sum of utilities is used to achieve the overall optimization. For each user, the optimizer selects the maximum utility values between two query access plans. For User 1, the overall utilities for Plan *a* and *b* are 84 and 90.8, respectively. Accordingly, Plan *b* will be chosen for User 1 since it has higher utility. Similarly, Plan *a* is optimal from User 2's perspective.

From this example, we can see that corresponding to different user's QoS requirements/preferences, the optimizer should be able to choose different query access plans.

3 QoS-Based Query Processing

To support QoS in database systems, we propose to enrich query processing by investigating how the construction and the selection of query access plans can be enriched with QoS features. We build our framework along the issues addressed above. More specifically, we consider QoS factors such as user requirements, dynamic network performance and dynamic server load in the procedure of global query processing. The main objective is to provide a flexible QoS model for multidatabase management systems and to offer differentiated services. Due to space limitation, in this section we present the general principles of our approach, addressing relevant issues and sketching the general methodology used to tackle the problem. The reader can refer to [6] for a detailed presentation of the approach and corresponding algorithms.

3.1 Query Processing and Optimization Revisited

To address the dynamically changing requirements of the user and the unpredictable performance of the underlying systems, we propose the integration of QoS within distributed query processing. Specifically, we are guided by two main goals when designing the QoS-based query processor: 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, which shows the relationships between QoS management and distributed query processing.

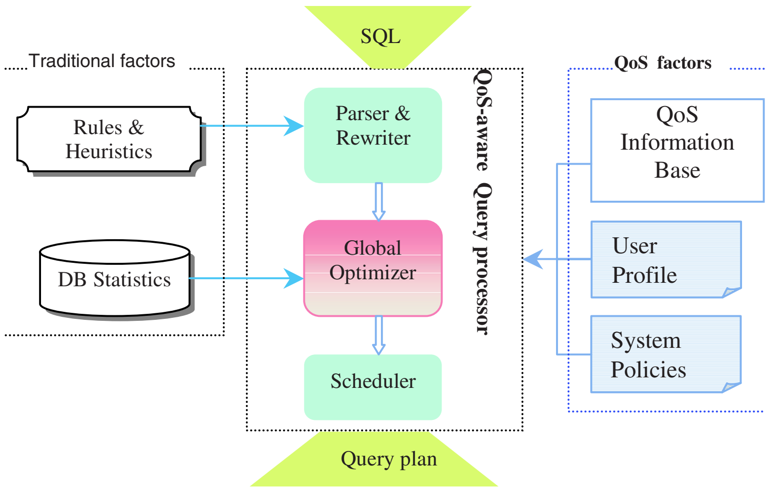

Fig. 1. A big picture for QoS-aware distributed query processor

We follow the conventional steps for distributed query processing: parsing, optimizing, and scheduling. In this framework, we include the typical components introduced in [8]. We keep the traditional factors (such as table, column, and index statistics) considered in the query processor. More importantly, we include the QoS factors, which are information from the *QoS Information Base* (QoSIB), *User Profile* and *System Policies.* The QoS information base (QoSIB) stores information about the service level offered by the different system components. A user profile is built to 1) store the user's QoS expectation for a particular service, and 2) to derive the trade-off between QoS dimensions, which is represented by the weight assigned to each dimension. The class of a user is used in the system policies. The system policies also determine the constraints under which the system resources can be used for providing services to the users. Although our focus is to push these QoS factors in the phase of optimization, the similar treatment can be applied to other steps. As a result, the global optimizer chooses a query access plan which will satisfy multiple optimization objectives derived from the user's requirements specified by the user. The dynamic system status is captured by QoS monitoring such that the selected query access plan takes into consideration the available system resources.

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 new algorithms for the query optimization. We address these problems in [6]. In this paper, we give a brief discussion on cost models used in our work and general description of our approach.

3.2 Cost Models

We propose a new approach to the problem of evaluating the cost of a query plan in a multidatabase system. Our approach relies on QoS monitoring to provide dynamic system status and on 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. Global optimization involves evaluation of the trade-offs between the amount of work to be done by the global level and the amount of communication and processing done by different component databases. Similar trade-offs exist in a distributed DBMS, but the heterogeneity and autonomy add considerably to the complexity.

Global cost models are the essential parts for the global query optimizer. Therefore, the information for network performance and load of database server is more crucial for the global cost model. The global cost model is defined as follows:

$$max \ \{ \sum_{i=1}^{n} \omega_i \cdot u_i(\mathrm{C}_i) \} \ ,$$

where $u_j()$ is the utility function for cost component $\mathrm{C_i}$ (based on one of the QoS dimensions i); ω_i is the weighting factor assigned to the cost component C_i, where $0 \leq \omega_i \leq 1$ and the sum of ω_i equals to 1.

Plan cost model. A query access plan is represented by a binary tree. Each internal node is an inter-site binary operation (such as join or union) 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 3 lists the cost functions for time, dollar, and availability. We use join operation as our study focus. The join time for each node is determined by the load of the server and the current TCP performance. Note that the QoS monitor captures this information. The formula for each join is:

$$\mathrm{T}_{\mathrm{join}} = \mathrm{local}\ (site,\ query) + \mathrm{net}\ (site_i,\ site_j)$$

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 3. 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.

Local cost model. As just mentioned, 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. The price and the availability must be reported by each local database server. 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). Due to space limitation, we will not give detailed information here. A complete discussion can be found in[6].

3.3 System Overview

After we have obtained the user's SQL query and QoS requirements, two major steps are used in our approach. One is the selection of the cost model, the other one is the global query processing.

The selection of the cost model includes the selection of cost components and the choice of utility functions. Different optimization goals may correspond to different cost models or query processing strategies. Our general heuristic for the selection of the cost model is that one optimization goal usually corresponds to one cost component. Therefore, how many cost components are included in the general cost model is determined by the number of optimization goals. For example, in the performance category given in Table 1, the cost factors comprise the measures of local processing time, communication time as well as some overhead due to parallelism. For the optimization goals related to cost (in terms of dollars), the cost measures include information on the resource usage and the pricing scheme. For the category of data quality, special attention should be given to query rewrite techniques to locate the best target databases. The selection of utility functions relies on the cost component and the application. For example, if the cost component is the image resolution, the available utility functions are usually non-decreasing (which means the more the better). On the other hand, if the cost components involve time or money, the utility functions are usually decreasing (which means the less the better).

Three steps are deployed for global query processing. They are global query decomposition, join ordering and join site selection. 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. The cost model used for this step mainly depends on the local information, depending 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 subquery 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.

Global query decomposition 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 algorithms for join ordering are based on the traditional algorithms [10] but enriched with QoS features. The cost models used in this step consist of both global cost model and local cost model. A new transformation rule, called flex-transformation, is defined for the step of bushy tree generation as discussed above. The algorithms for global query decomposition and third site consideration for join site selection are new. The QoS aspects include the provision of different query execution plans for different user classes. The construction of the global query plan also takes into consideration the dynamic system properties, such as server load and network available throughput, with the help of QoS monitoring. Some of this support comes from our QoS-based cost model used in the algorithm, others are directly infused in the algorithms.

In case of data duplication, one subquery might have several potential locations, thus the optimizer should decide at which location this subquery would 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. This process may be done in a bottom-up fashion. We propose a new algorithm where we use post order tree traversal to visit the internal nodes of the tree[6].

3.4 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 implementation 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) 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 adopt the utility functions given in the example in section 2.3.

3) Cost models. The general cost model 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. $C_{overall} = W_{time} * u_t(\text{response\_time}) + W_{dollar} * u_\$(\text{service\_charge})$

We also 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 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. Under different system loads, the results [6] show that the VIP user always enjoy the fast response time while the normal user will get slower response when the system is heavily loaded.

4 Conclusions

In this paper, we have presented 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 offering differentiated services to different classes of users according to their expectations in terms of QoS. We also developed a prototype to verify the effectiveness of the idea. Our current prototype supports two QoS dimensions: response time and service charge. In the future, we will consider other QoS dimensions to be specified by the user, such as data quality or freshness. We will also be interested in working on rewriting rules to transform specifications on these dimensions into optimization goals and corresponding cost models.

References

[1] C. Aurrecoechea, A. Campbell, L Hauw, A Survey of QoS Architectures. ACM Multimedia Journal, 6, May 1998, pp. 138–151

[2] 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
[3] K.K. Nguyen, F. Fetjah, B. Kerhervé, Quality of Service Information Base (QoSIB) Manager for Electronic Commerce Applications, Poster presented at the CITR Annual Conference, August 2001
[4] 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
[5] H. Ye, B. Kerhervé, G. v. Bochmann, V. Oria, Pushing Quality of Service Information and Requirements into Global Query Optimization, the Seventh International Database Engineering and Applications Symposium (IDEAS 2003), Hong Kong, China, July 16–18
[6] H. Ye, Integrating Quality of Service Information and Requirements in a Distributed Query Processing Environment, Ph.D. thesis, University of Montreal, May 2003
[7] D. A. Menasce, V. A. F. Almeida, Scaling for E-Business Technologies, Models, Performance, and Capacity Planning, Prentice Hall Canada, 2000
[8] D. Kossmann, The state of the art in distributed query processing, ACM Computing Surveys (CSUR), Volume 32, Issue 4, December 2000, pp 422–469
[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, B. Kerhervé, G. v. Bochmann, Revisiting Join Site Selection in Distributed Database Systems. International Conference on Parallel and Distributed Computing, 26th–29th August 2003, Klagenfurt, Austria