Introducing QoS to Electronic Commerce Applications

Gregor v. Bochmann[1], Brigitte Kerhervé[2], Hanan Lutfiyya[3], Mohamed-Vall M. Salem[4] and Haiwei Ye[4]

[1] University of Ottawa
[2] University of Quebec at Montreal
[3] University of Western Ontario
[4] University of Montreal

Abstract. Business to consumer is expected to be one of the fastest growing segments of electronic commerce. One important and challenging problem in such context, is the satisfaction of user expectations about the Quality of Service (QoS) provided when applications are deployed on a large scale. In this paper, we will examine the use of dynamic QoS management techniques in combination with replication at the various architectural levels of an electronic commerce application.

1. Introduction

It is expected that business to consumer commerce as well as other forms of electronic commerce will grow at a breakneck pace during the next four years. The value of goods and services traded between companies is expected to skyrocket from $8 billion (U.S.) this year to $327 billion (U.S.) in 2002, according to Sizing Inter-company Commerce, the inaugural report from Forrester Research's Business Trade & Technology Strategies service.

The business to consumer is expected to be the fastest growing segment of electronic commerce. However, there are several impediments that may have an affect on this growth. The most commonly cited problem is that of the lack of trust between businesses and consumers. Thus there is much research in security and payment protocols. Secure payment is only one aspect of concern about doing business on-line. Another, but less discussed problem, is user expectations about the Quality of Service (QoS) provided. By QoS, we are referring to non-functional requirements such as performance, availability and cost of using computing resources. For example, a user is bound to worry if there is a long wait during credit card processing. The user may even perceive the long wait to be a security problem. The time it takes for credit card processing should not exceed a certain threshold where that threshold is determined to be the average amount of time that a consumer is willing to wait on-line before hitting the “panic button”. Satisfying the quality of service expectations of users for electronic commerce sites is becoming a considerable challenge that has not been addressed adequately.

QoS management refers to the allocation and deallocation of computing resources. Static QoS management techniques provide a guarantee that resources will be available when needed. Applying static QoS management techniques for electronic commerce sites typically involves adding additional resources e.g., adding additional

servers or buying faster processors, faster bandwidth connections or more memory. It has been shown that this approach wastes approximately 20% of resource capacity and fails to satisfy quality of service needs in 90% of cases. This suggests that just adding resources is inadequate. In this paper, we will examine the use of dynamic QoS management techniques in combination with replicating web and application servers.

The outline of this paper is as follows. In Section 2 we introduce some typical Web server architectures in order to accommodate a large number of users. We also review the basic QoS parameters, which describe the user's perception of the provided service and the internal performance parameters of the distributed system components. We also consider differentiated service for different classes of users and some policies for managing QoS in this context. We note the importance of monitoring the actual QoS parameters while the system is running in order to provide for dynamic QoS management which automatically adapts to unforeseen situations, such as unexpected bottlenecks, partial system failures or unexpected user demands. In Section 3, we consider system architecture with a single Web server and e-commerce application and partially replicated database servers. We discuss the issues related QoS-aware distributed query processing which is feasible in this context. We consider different optimization criteria that may be related to different user preferences or the trade-off with the preferences of the system administrator. In particular, the network QoS parameters, available throughput and transmission delay, are taken into account. In Section 4, another replicated architecture is considered where the whole e-commerce application are replicated and a so-called broker process distributes the user requests between servers according to some QoS policies. Because of space limitations, only an overview of this approach is given. Section 5 contains the conclusions.

2. Systems Architectures and QoS Parameters

2.1. Basic System Architectures

The simplest form of an electronic commerce application as shown in Figure 1 consists of a web server as the interface for clients, an application server that has the program logic needed for implementation and a database server needed for storage of information.

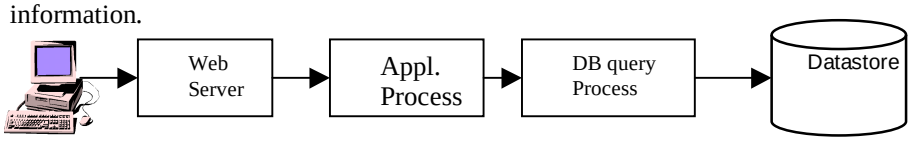

Fig. 1. Simple form of an electronic commerce application

A more complex electronic commerce application has its information reside on databases at different sites. For example, a virtual mall brings together services and inventories of various vendors and allows users to navigate through these vendors, adding items into a virtual shopping cart. The catalogue of services and inventories will often be on the databases at the vendor's site. If the information to be accessed is standardized, the DB query process can provides a uniform interface to the application process that hides the differences of the various catalogs and their access

differences. This would allow the straightforward processing of a query involving several vendors of the same mall, such as the following example query: *Find all sofas which price is less than $1000 and where a matching loveseat, recliner and coffee table can also be found.*

A typical architecture with a distributed data store is shown in Figure 2. Such distribution accommodates the distributed responsibility for the data, which is shared among the different vendors within the shopping mall. The distribution also increases the overall processing capacity since the different local query processes will run in on separate computers with their own data store; this will therefore increase the overall query processing capacity. The problem of QoS optimization in this context will be discussed in more details in Section 3.

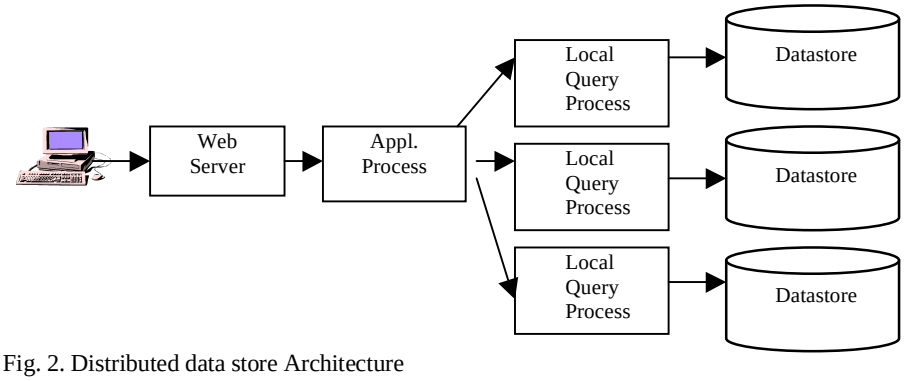

Fig. 2. Distributed data store Architecture

In the case of a very large number of users, a single server is not powerful enough to run the Web server and the application processes for all the users. In order to increase the processing power, one may distribute the Web server, application processes and query processes over different computers. Or one may go one step further and duplicate the whole system architecture, introducing a certain number of independent systems, all providing the same service, as shown in Figure 3. In order to distribute the user requests to different servers, we consider the introduction of a load distribution engine, which we call broker. The broker receives the initial user requests and determines, based on performance management information received from the Web servers, which server should be allocated to serve the given user request and the following interactions occurring within this new shopping session. Some of the issues related to the load sharing and QoS management for this replicated architecture will be discussed in Section 4.

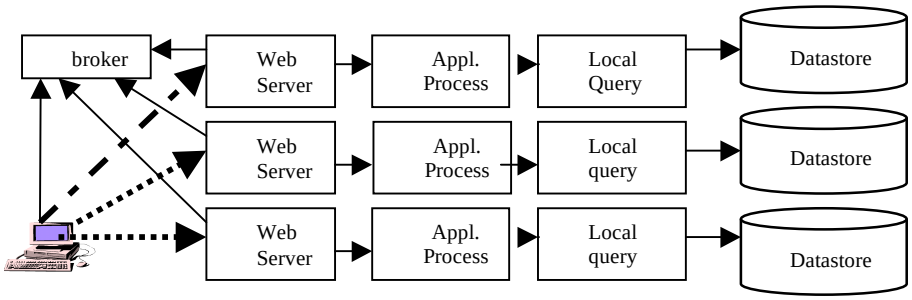

Fig. 3. Full replicated architecture

2.2. Quality of Service Parameters

With respect to performance, a primary user expectation about the QoS is specified in terms of the response time. The response time is affected by the transmission delay of the access network through which the client's workstation accesses the Web server, by the processing times in the Web server, the application process and the DB query process, and possibly the network latency between the Web server, application and database server if they are implemented on separate computers. In order to discuss QoS management in this context, we have to distinguish between the high-level QoS parameters relevant at the user level and the internal QoS parameters pertaining to the different system resources, which are used by the application. The goal of QoS management is to assure the controlled sharing of these resources among the different users according to certain management policies.

2.2.1. User-Level QoS Parameters

The following QoS parameters are important from the user's perspective:

1. Response time: This is the most important QoS parameter from the user's perspective. It is the time between the moment a request is sent to the time that the response has been provided to the user.
2. Availability: Availability is simply a measure of the system's effective "up-time". It represents the percentage of time the server is available during an observation period.
3. Servability: We call servability the percentage of time the server is available and can accept the user request. We assume that, in order to guarantee a certain quality of response time to the active users, the system will refuse new requests when the response time of the system exceeds a certain limit, which we call Rmax.
4. System throughput: This parameter is important from the perspective of the system owner, and it measures the number of user requests that are handled by the system. It is a measure of the amount of service that is provided. It is well known that the response time of a given system increases as the system throughput increases. When the maximum throughput of the system is attained, the response time becomes infinite since the internal queuing delays become arbitrary big. It is interesting to note that different query processing algorithms in distributed databases may lead to different maximum throughput and different response times (at less than maximum throughput). Therefore a compromise must be found

between maximizing the throughput and minimizing the response time. This is an important QoS policy decision to be made by the owner of the e-commerce system.

In the context of databases, the parameters of interest are database server availability, database management throughput and query response time. The throughput is usually measured in transactions per second [TPCW]. We consider the response time as a function of several parameters including the database server load (CPU usage, database connections, disk I/O activities), the network load (the available TCP throughput and delay), as well as the power of client machine.

2.2.2. Internal QoS Parameters

The internal QoS parameters pertain to the performance of the different system components within the considered architecture. They include the following parameters:

1. Network propagation delay: The time between the sending of a packet to its reception by the destination computer.
2. Network access capacity: This is the maximum throughput by which a given computer can send data over the network. It is determined by the network access link, which is relatively limiting in the case of modem access over telephone lines.
3. Effective network maximum throughput: This is the maximum throughput that can be effectively obtained between two computers over a given network. Even if the network access links allow for large throughputs, the effective throughput will be limited in most cases by the flow control mechanism of the TCP protocol since most Web applications and distributed databases use TCP for the transmission of data.
4. Response time of the query process; also its throughput (number of queries processed per second).
5. Processing delay and queuing time in Web server and application process; also their throughput.
6. Availability of the different servers and system components implemented on these servers.
7. Low-level performance parameters, such as CPU and memory utilization of the different servers that are part of the system architecture, as well as statistics related to data transmission over the network access links, etc.

2.3. Differentiated Classes of Users

We think that many future e-commerce systems will be able to provide different levels of service to different classes of users. For instance, the system may distinguish between a casual user (which is not known to the shop's organization) and a registered user who is a regular client of the shop. Some of the registered users may be known to buy many goods in the shop; they may obtain the "Elite" service, while the normal registered user obtains the "Premium" service and the casual user the "Normal" service. These different classes of service may differ in several aspects, such as the following:

- A higher class of service will have a shorter response time.
- A higher class of service may provide facilities, which are not available at the basic level, such as for instance a teleconference chat with a sales person.

- A higher class of service implies higher availability and servability.

2.4. QoS Policies

In the case of a single class of users, the general policy for QoS management may be the following:

Policy 1: At any given point in time, all active users should experience the same average response time. If the average response time reaches the value Rmax, the system should not accept any new user sessions. Note: Rmax may be infinite in the case that no user will be refused. It is clear that the size of the active user population (which varies over time), the system's processing capacity and Rmax determine the servability parameter of the provided service.

In the case of two classes of users A and B (where A is a higher class than B) there are different ways the priority of A over B may be specified. Assuming that all users of a given class should experience the same average response time and servability, and that the average response time for class A is smaller than for class B, we may consider the following policy alternatives:

Policy 2:

1. If the response time for class A reaches $Rmax_A$ then no new users of class A will be accepted.
2. If the response time for class B reaches $Rmax_B$ then no new users of class B will be accepted.
3. $Rmax_A < Rmax_B$

Policy 3:

1. If the response time for class A reaches $Rmax_A$ then no new users of class A nor class B will be accepted.
2. As above
3. As above

Note that policy (3) implies absolute priority of class A users over class B users. If there are enough class A users requesting service, no class B users may enter the system and eventually only class A users will be served. In the case of policy (2), the amount of processing capacity for each of the two classes of service is not specified and may be determined by the system administrator in an arbitrary manner, thus allowing in the case of general overload condition a somehow balanced number of class A and class B users. We note that giving higher CPU priority to the processing of the class A requests as compared with class B requests, may be used to implement both of these policies. However, the policies leave some freedom to the implementation about the amount of resources to be allocated to the different user classes, therefore strict priority may not always be appropriate.

2.5. Fault Management

In any dynamic approach to QoS management, a key problem is detecting why a QoS requirement is not being satisfied. For example, assume that under normal operating conditions the performance requirement for a Web server is as follows: "If the number of requests is less than 100 per second, the time it takes to service an HTTP request should be less than 1.5 seconds". If this requirement is not satisfied, this is a

symptom of a problem. The reason that this requirement may not be satisfied could be due to a number of problems including the following. The host machine that the Web server is running on has a high CPU load, the network between the Web server and the application / database server may be congested or the host that the database server is running on may have a high CPU load or be down. Determining that a QoS requirement is not satisfied is referred to as *fault detection.* Fault diagnosis techniques hypothesize as to a probable cause of the degradation, with subsequent testing used to determine the validity of the hypothesis.

Support of fault detection and diagnosis requires that the architecture supports the following: (1) Resource monitors for the network and the host machines that run the components of the application as well as monitors that measure application behavior that can be compared to the QoS requirements. (2) Management applications that can analyze the monitored information to determine if a QoS requirement has been violated and diagnose the cause.

2.6. Monitoring Application and Resource Behavior

The subsections above illustrate the need for performance monitoring. The broker of Figure 3, the global query process in Figure 2, and the management application described in Section 2.5 all need information about application and system run-time behavior. Thus, for application and system components we need an associated performance monitor. For example, a Web server would have instrumentation for monitoring its behavior and there would be a daemon process monitoring CPU load, incoming network traffic and outgoing network traffic for the computer on which the Web server is running. The monitored information used by the broker, query processor, and the management application all intersect, but are not necessarily the same. In addition, the frequency of the need for information will vary. This suggests that each process be able to register with a resource or application monitor for the timely and periodic sending of information. Alternatively, if a process does not want to register for timely and periodic information, the resource must provide an interface that allows for request of the monitored information when needed.

3. QoS Aware Distributed Query Processing

As shown in Figure 2, a typical real-world back-end database configuration for e-commerce systems consists of multiple database servers for providing stability, load balancing and better performance. In such configurations, the database content can be allocated to different database servers. For example, tables storing product catalog are less likely to be updated as compared to those tables used for order processing, thus these two types of tables can be distributed to different servers to get some performance gain. However, such kind of data distribution requires connecting the two tables when processing queries that need to access information from both tables. This requires the database management system (DBMS), or more specifically the query optimizer, to be aware of the data distribution. The new challenge, as compared to the context for traditional DBMS, consists here on how to take into account the dynamic nature of the underlying network and database server load. In addition, since users may have different expectations from the e-commerce server, while passing down to the database server, these expectations have to be mapped to different

optimization criteria. Therefore, the traditional query optimizer lacks the flexibility to configure to different optimization criteria. In this new context, we revisit distributed database query processing in the presence of information about the network quality of service (delay and available throughput) and user preferences concerning query optimization criteria [Ye99]. We consider such optimization criteria as minimizing the overall resource utilization, lowest response time or lowest cost (based on some tariff structure for network and database utilization) [Ye00]. We work in the context of federated and distributed databases communicating either over a local area network or over the Internet. Accordingly, our distributed query processing strategies can be described as "Quality of Service aware". The QoS aspect here refers to the dynamic nature of network, the server load and the user's preference.

Global query optimization is generally implemented in three steps [Daya85, Meng95, Ozsu99]. 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 problem of *global query decomposition*, *inter-site join ordering* and *join site selection* [Corn88, Du95, Evre97, Urha98].

Global query decomposition is usually complicated by duplication. Therefore this step is usually guided by heuristics. Two alternative heuristics [Ozsu99] can be employed during this step. The first alternative is to decompose a global query into the smallest possible subqueries and the second alternative is to decompose a global query into the largest possible subqueries. In our work, we assume the subqueries are generated based on the second heuristic. The reason is that we wish to push as much processing as possible to the component DBMS so that we could simplify the optimization at the global level and hopefully could reduce the data transmission among different sites. Therefore, the objectives of this step are 1) to reduce the number of subqueries and 2) to reduce the response time of each subquery. The first thing that needs to be done, when a query is given to the optimizer, is to split it into subqueries based on data distribution across multiple nodes. Thus the main task of global query decomposition is to decompose a global query into subqueries so that the tables involved in each subquery target on one local site.

Then in the join-ordering step, the optimizer tries to come up with a good ordering of how to combine those joins between subqueries. The join ordering can be represented as a binary tree, where leaf nodes are the base tables and internal nodes are join operations. Because we want to utilize the distributed nature of multi-database 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. In our work, we first build a left-deep tree using dynamic programming. And then, use some transformation rules to balance the linear tree to a bushy tree. In both steps, we consider both server performance and network performance captured by QoS monitor.

Last, we have to decide where to perform the inter-site join – this is referred as join site selection. In our approach, we implement this step by annotating the binary tree generated from join ordering step. Each node is annotated by a location of where this

join should be performed. The problem is how to integrate QoS information into these phases. Table 1 identifies the related QoS parameters factored into each step.

Global query optimization	Relevant QoS parameters
Decomposition	Server availability Server load
Inter-site join ordering	Server load TCP throughput Network delay
Join site selection	Server load TCP throughput Network delay

Table 1. QoS information relevant to global query optimization

We are presently working on enhancing these algorithms by considering the user preference to provide a differentiate service. For example, for higher priority user, the algorithm could generate a different plan by accessing different set of data nodes so that the result set of the query could have more interesting content such as multimedia data. We could also propose data distribution rules for locating data accessed by higher priority user to high performance database servers. In addition, the issue of how to corporate with other server replication, such as the one introduced in the following section, deserves a careful study.

4. Web Server Replication

A typical approach to improving response time is replication of a service. Replicating the database servers requires a good deal of synchronization of the copies of the data and thus it is not always feasible to replicate the databases. Replication of the web server and associated application servers do not have this overhead and thus this work focuses on web server replication. Client processes connect through a virtual server address to an intermediate host that forwards their requests to a replicated web server. The intermediate host is referred to as a *Broker* (see Figure 3). The *Broker* is used to direct Web traffic to one of a number of web servers. The selection is based on monitored information from the Web servers and administrative policies on how to use the monitored information to select a server. Examples of *administrative policies* include the following:

- **Balanced server performance.** Requests are assigned equally to the replicated Web servers in a round-robin manner or in a weighted manner based on the measured performance of the different servers. Different kinds of such policies are described and evaluated in [Salem 00].
- **Content.** Requests are assigned to Web servers based on the type of request. For example, a request for dynamic content pages may be assigned to one server and a request for static content may be assigned to another server. Another example is that video clips are retrieved from one server while text information is retrieved from another one. Yet another example is using client host information (e.g., host load, connection type) to determine the type of Web page that gets sent to the client.

- **User.** Requests are assigned to a web server based on the user class as discussed in Section 2.3. Users who have paid a fee may be designated as premium users and be assigned to their own server. If the server is in danger of being overloaded, then premium users may also be assigned to a second server, etc; while non-premium (i.e., non-paying users) get assigned to other servers.

5. Conclusion

The discussion in this paper shows that different architectures with server duplication may be adopted for e-commerce applications if the system is designed for a very large user population. In order to provide a service quality and allowing for different classes of users with different expectations, it is important to introduce dynamic QoS management for allocating available resources in the best possible manner. The QoS management decisions should be made dynamically based on measurements of the actually available service qualities from the network and the different server components of the distributed systems. This implies dynamic performance monitoring and making these measurements available to the QoS management processes.

Two distributed system architectures are considered in the paper. The case of distributed query processing in an architecture containing a single Web server and e-commerce application front-end together with a partially replicated database is considered. Another replicated architecture in which the whole e-commerce application (including Web server and database back-end) is replicated on different sites is only discussed without much detail. In our ongoing work, we evaluated the performance of different architectures and the optimization algorithms that can be used in these different contexts.

References

[Corn88] D. W. Cornell, P. S. Yu: Site Assignment for Relations and Join Operations in the Distributed Transaction Processing Environment. ICDE 1988: 100-108

[Daya85] U. Dayal, *Query Processing in a Multidatabase System*, In Query Processing in Database Systems 1985: 81-108

[Du95] W. Du, M.-C. Shan, U. Dayal, *Reducing Multidatabase Query Response Time by Tree Balancing*. SIGMOD Conference 1995: 293-303

[Evre97] C. Evrendilek, A. Dogac, S. Nural, F. Ozcan, *Multidatabase Query Optimization*. Distributed and Parallel Databases 5(1): 77-114 (1997)

[Meng95] W. Meng, and C. Yu, *Query Processing in Multidatabase Systems*, In Modern Database Systems: The Object Model, Interoperability, and Beyond, edited by W. Kim, Addison-Wesley/ACM Press, 1995:551-572

[Ozsu99] M. T. Ozsu, P. Valduriez, *Principles of Distributed Database Systems*, second edition, Chapter 15, Prentice-Hall, 1999

[Salem00] M. Mohamed Salem, G.v.Bochmann and J. Wong: "A Scalable Architecture for QoS Provision in Electronic Commerce Applications, submitted for publication.

[TPCW] TPC-W Benchmark Specification, http://www.tpc.org/wspec.html

[Urha98] T. Urhan, M. J. Franklin, L. Amsaleg: *Cost Based Query Scrambling for Initial Delays*. SIGMOD Conference 1998: 130-141

[Ye99] H. Ye, B. Kerhervé and G. v. Bochmann, Quality of service aware distributed query processing**,** 10th Intern. Workshop on Database & Expert Systems Applications, Florence, Italy, 1-3 Sept. 1999, Proc. published by IEEE Computer Society, 1999.

[Ye00] H. Ye, G.v. Bochmann, B. Kerherv, *An adaptive cost model for distributed query processing*, UQAM Technical Report 2000-06, May 2000