Towards Database Scalability through Efficient Data Distribution in E-commerce Environments*

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

Don Bourne
IBM Canada Ltd
dbourne@ca.ibm.com

Abstract

Increasing Web traffic in e-commerce applications poses great challenges to database servers. On one hand, database servers should be able to scale; on the other hand, end users are becoming more and more sensitive to the quality of the offered services. This requires addressing issues such as pushing quality of service (QoS) requirements into database processing and providing database system scalability. In this paper, we discuss scalability issues for back-end database servers used in e-commerce applications. We argue that database scalability cannot be achieved without considering efficient data placement. That leads us to consider the specifics of e-commerce applications as well as user QoS requirements. We propose a generic data distribution strategy integrating user class information and application characteristics. We also present experiments we have conducted to provide practical guidelines to our strategy and to study the impact of data distribution on the behavior of database system in e-commerce applications.

1. Introduction

Performance and scalability are great challenges to be met for making e-commerce successful and largely adopted by customers. Developing strategies and approaches for better performance and scalability are required to offer different QoS levels to the users. Such strategies involve all the components of e-commerce systems, which architecture typically consists of web servers as the interface for clients, application servers having the program logic needed for implementation and database servers needed for storage of information. In this paper we focus on database systems and we investigate data distribution to provide scalability for e-commerce applications.

Data distribution and query optimization are two key techniques that have to be revisited in order to provide QoS support in database systems. In our previous work, we have focused on integrating information on network and server performance for enhancing distributed query-processing algorithms with adequate cost models [1][2][3]. We are now interested in applying such approaches on how to wisely layout the data across the nodes in e-commerce systems since we believe the performance and scalability of a database system are contingent upon data distribution. In addition, a poor data distribution strategy can result in a non-uniform distribution of the load and the formation of bottlenecks.

In this paper, we concentrate on data distribution in e-commerce applications. Such applications are specific compared to those addressed in traditional data distribution strategies since (i) they are less dynamic in the sense that access patterns to the database are static and can be obtained from the application source code; (ii) they need more consideration of the user expectations in terms of quality of the provided service; and (iii) their capability to satisfy these expectations should adapt to the high variations in the number of connected customer. In [4] we defined the general principles of a data distribution strategy that takes user class into account. In this paper, we give a detailed presentation of our strategy and we provide results of experimentation we have conducted to examine the impact of data distribution on the behavior of database system and the e-commerce application in terms of scalability.

The remainder of the paper is organized as follows. Section 2 briefly overviews the related work. Section 3 discusses the scalability issue of the database server. Section 4 demonstrates our approach for the data distribution and proposes a generic methodology for data

**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 and by a student fellowship from IBM.*

distribution in e-commerce applications. In Section 5, we provide our experiment and the implementation concerns. Last, we summarize and point out some future work.

2. Related work

Traditional data placement strategies [5][6] are derived based on application characteristics (including data access pattern and data access frequency) and SQL complexity (referring to the number of tables participating in the query and number of joins involved). These factors are not sufficient to propose an optimal partitioning schema for the e-commerce application if user's concerns are ignored.

Various distribution strategies have been developed for parallel database systems [7][8][9]. They can be classified into three categories according to the criteria used in reducing costs incurred in resource utilization. If the objective is to reduce data transmission over the network, then the policy could be based on *network traffic* [8]. If the goal is to balance the amount of data, or disk I/O access frequency, the strategies are based on *size* [7] and *disk access frequency* [9]. The main idea behind these approaches is either to achieve the minimal load (e.g. network traffic) or a balance of load (e.g. size, I/O access) through database partitioning. Our work differentiated from the previous work in the following ways. First, it considers user class information in the distribution strategy. Second, both replication and partitioning are integrated into the strategy. Last, the strategy is particular tuned for the B2C e-commerce applications.

One way to maintain several copies of data at different sites is replication. We address this issue in the paper. It is also interesting to point out that another way to establish copies of data at different sites of a distributed system is caching. Several differences between data replication and caching are identified in [10]. However, in the rest of this paper, issues related to caching are not addressed. For a detailed discussion between caching and data replication, please refer to [10].

3. Providing scalability for the database server

Scalability refers to the ability of a computer application or product (hardware or software) to adapt to increased demands, but we can find many interpretations to scalability. For many, the top priority for scalability is high-speed processing enabling great numbers of transactions per second. For others, the primary need is a system that can scale up to large user counts or voluminous data storage. Another interpretation of scalability that is being actively promoted in the context of e-business is the capability of a server, application, or Web site not only to function well in the rescaled situation, but also to take full advantage of it.

For the purpose of our work, a system is scalable if there is a "straightforward" way to upgrade it to handle an increase in traffic while maintaining adequate performance. By *straightforward* we mean that no system or software architectural changes should be required to scale the system. Examples of *straightforward* changes are: adding more servers to a system that already employs multiple servers, adding more CPUs to a multiprocessor, and replacing existing servers with faster servers that use the same architecture.

The simplest and most intuitive way to provide a scalable database service is to replicate the database server to different locations, similarly to the way usually proposed for Web and application server replication. This implies that the database content is also replicated and leads us to consider the complex issue of how to efficiently maintain the consistency of several copies of the database stored on different machines.

An alternative way is to deploy a parallel and/or distributed architecture for the back-end database server. Distributed database systems and shared-nothing parallel database systems are increasingly being deployed to support the scalability and performance demands of end-users [11][12]. Since they provide the opportunity to duplicate and/or partition data among multiple nodes, the database system can utilize this data distribution schema to query the database in parallel, and thus enhancing performance. Therefore, two aspects have to be studied: one is how to distribute the data (both duplication and partitioning), which is concerned with data distribution strategy; the other is how to use the existing distribution schema to provide higher performance, which is dealt with distributed query processing strategies. In our previous work [1][13][3], we addressed the second issue. More specifically, we proposed distributed query processing strategies based on Quality of Service (QoS) information describing performance and availability of networks and servers. In this paper, we focus on the first aspect. We examine the specificity of e-commerce applications in order to come up with data distribution strategies allowing better performance and scalability of back-end database systems.

Efficient data distribution is usually closely related to the performance and scalability of a database server. Moreover, data distribution also serves as an important load-balancing mechanism. A poor data distribution strategy can result in a non-uniform distribution of the load and bottlenecks. The initial data distribution should be reorganized in response to skewed workloads and changing access patterns. Therefore, data distribution should be considered in two stages: initial data distribution and data reorganization [5][6]. In our work, attention is focused on the initial data distribution.

However, the approach discussed here is also applicable to data reorganization.

4. Method overview

In this paper we extend traditional way of data distribution by adding user class information. That is, our data distribution strategies should integrate distribution decisions based on user QoS expectations as well as on e-commerce application characteristics. Defining classes of users is a way to differentiate users according to their QoS expectation in order to provide different levels of services based on priorities. This is an important factor to take into account to come up with an optimal data distribution strategy. Under the assumption that a higher priority user class should get better performance in terms of response time, we want to allocate the required data, for example, to more nodes, so that parallelism can be utilized for query execution. This also requires that the DBMS supports such priority awareness so that to route the query from different classes of users to different node groups.

The general method in our study is to allocate different database resources to different classes of users first and then apply the distribution strategy to each user class. Such a procedure is expressed in the following steps:

1) Using resource allocation strategy to decide the number of database nodes of each user class.
2) For each user class, executing the following steps:
   - Deriving database access pattern from the characteristics of this user class.
   - Collecting and analyzing the related statistics information.
   - Applying data distribution algorithm.

4.1. Resource allocation

The first step is a resource allocation issue, which is a question of how to divide up the available resources among available user classes. To simplify the discussion in this paper, we consider one database node as one unit of resource. In the implementation, this abstract resource could be mapped to different hardware and software resources, such as CPU, memory, disk and network bandwidth.

Different QoS levels can be provided by either *shared* resource or *segregated* resource. In the case of shared resource, all the user classes access the same resource. When there is plenty of resource, all the QoS requirements can be satisfied. When the resource utilization reaches to a point (or *threshold*, for example 80%) such that the system cannot guarantee all the QoS requirements, some admission control policies may be triggered to reject or delay the requests from low priority user. In this method, an important step is how to decide the reasonable threshold. A high threshold may lead to service degradation for high priority user. In contrast, a low threshold may reject too many low priority users and lead to under utilization of system resources. This value could be derived from the simulation and should be later tuned for the purpose of changing policy or workload.

In the case of segregated resource, one feasible solution is to keep several copies of an identical database, with each copy for a particular user class, and then allocate the resources to different user classes. If the resource is the number of database nodes, then the allocation is to decide how many disjoint sets of nodes are needed for different classes. In this method, we have to address the problem of how to decide how many nodes to choose for each class. The decision is made depending on several factors such as the workload type, the navigation pattern for each class as well as the total available resources. This is also a policy issue. To do this, we can analyze the HTTP log and monitor resource utilization for each node. For example, from the HTTP log and monitor information, we observe that 80% of the total requests are searching product and only 20% involve payment activity, we could allocate 4/5 of the total data-base nodes to *browser* class and 1/5 to *buyer* class.

No matter which resource allocation strategy is applied, the data distribution strategy should allow for the use of flexible mechanisms that can adapt to workload change. For example, the resource allocation method shown in the above example will not be suitable if the observation of the workload shows that the payment activity increases to 40%. Therefore, with the time goes on, the number of users in each class will change and thus the number of nodes reserved for each class should also be adjusted to the new circumstances.

4.2. Distribution strategy

Ideally, we attempt to duplicate all the tables among all the available nodes for one user class since this will provide the maximum flexibility of utilizing the parallel techniques and reducing data transmission. However, this kind of replication strategy will greatly degrade the system's performance if the environment is updated intensively (such as insert, update or delete SQL commands), as required by the application. Therefore, data partitioning is useful in scenarios where there are frequent updates.

Accordingly, our heuristic is to pick up those tables that can be replicated first and then, for the rest of the tables, decide on the partitioning key. The data distribution algorithm can be summarized in two steps:

1) Grouping tables into two sets for replication (denoted as $\mathbf{S_{rep}}$) and partitioning (denoted as $\mathbf{S_{part}}$), respectively; and

2) Selecting the partitioning key for tables to be partitioned.

The replication strategy can be chosen to duplicate all the tables in $\mathbf{S}_{\mathbf{rep}}$ across all the available nodes for that particular class in the database server. For selecting partitioning key, we can use the frequency information and the response time for each operation as our selection criterion. The partitioning key selection not only depends on the join operation, but also depends on other operations like insert and update. The frequency information refers to the number of an attribute occurrence in the joins/update. For example, in order to choose whether to partition table T on join attribute a or join attribute b, we first need to know the number of occurrences in joins on table T. Suppose attribute a is more frequently invoked than attribute b, then the partitioning key will be attribute a. If the frequencies are the same, the response time is introduced into the decision-making. The attribute that is included in a join with higher response time will be the partitioning key.

In addition, the selection of the partitioning key should also obey the constraints imposed by different implementations of the DBMS. For example, in DB2 EEE [14], there is the constraint that all the columns of the partitioning key must be subsets of the primary key or unique key, and the partitioning key cannot be updated.

4.3. Example for data distribution

We apply our data distribution strategy to a sample database--*Demomall*. The example is taken from Websphere Commerce Suite (WCS 5.1) [15]. The schema of the *Demomall* can be found in [16]. We assume a product browsing scenario. The event monitor [14] is used to collected statistics. The result for tables to be replicated and partitioned is shown in Table 1. Some example partitioning keys are shown in Table 2.

5. Experiment and implementation observations

In this section, we implement the strategy proposed in Section 4 and conduct several experiments. The implementation of the strategy is only used for our study and is not integrated in any IBM product. The purpose of the experiment and implementation is two-fold: (1) to test the feasibility of our approach and (2) to provide some practical guidelines to future prototype development.

Performance analysis is a key technique to understand scalability problems in e-business. In our experiment, we do not measure the database performance directly. Rather, we measure the performance of the whole system from a user's perspective. The two metrics we are concerned with are *HTTP throughput* (HTTP *hits/sec*) and *response time* (*sec*) of each command.

Table 1. The selection of replicated and partitioned tables

Table category	Replicated tables	Partitioning tables
STORE	STORE , MERCHANT, MASSOCCECE	
PRODUCT	CATALOG, CATENTRY, CATGROUP, CATGRPATTR, CATGRPDESC, CATGRPREL, CATTOGRP, INVENTORY, LISTPRICE	
ORDER	CALCODE, CALMETHOD, CCCHECK, CONTRACT, OFFERPRICE, SHPARRANGE SUBORDERS	OFFER, ORDERS, ORDERITEMS, ORDPAYMTHD, TRADEPOS
USER		MEMBER, PERSPROF, USER, USERDEMO
OTHER		IITEM, IITEMLIST

Table 2. The selection of partitioning key

Table	Partitioning Key	SQL Statement collected
ADDRBOOK	ADDRBOOK_ID	Join
ADDRESS	ADDRESS_ID	Update, Join, Insert
IITEM	IITEMLIST_ID + CATENTRY_ID	Update, Join
MEMBER	MEMBER_ID	Update, Join
ORDERS	ORDERS_ID	Update, Insert
USER	USER_ID	Update, Join

5.1. Experiment goal

In our initial performance study, we intend to maintain the adequate performance of the Web server by adding more system resources to database servers. That is, with this increasing resources in database server, the Web server can accept more user connections while remaining the similar performance or not experiencing too much performance punishment. For multiple database machines, we adopt the DB2 UDB EEE (Extended Enterprise Edition) as our database management system.

For the purpose of comparison, we have two sets of configuration: the first one is a centralized database server configuration using DB2 UDB Enterprise Edition (EE), the second one is a parallel database server configuration using DB2 UDB EEE. The DB2 EE setup can be regarded as our baseline measurement. Therefore, we expect for the result are:

(1) To create a scenario that the database is the bottleneck of the whole system performance in a DB2 UDB EE setup.

(2) To apply the same or more workload in the bottleneck scenario to the DB2 UDB EEE setup and we expect that the performance (in terms of http hits / sec) remains the same.

These two points can be regarded as two goals that we want to achieve for the experiment. A third expected result would be that the http hits/sec across the entire site increased in proportion to the number of database servers employed. We did not verify this point (due to the time limit) in our experiment. However, we could derive some useful information by observing the database server utilization as explained in Section 5.3.

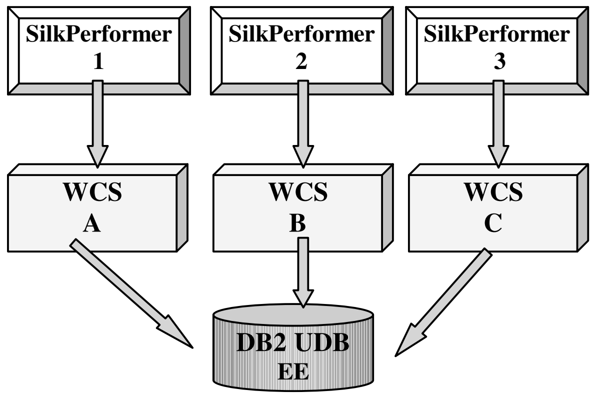
5.2. Environment setup

The environment we set up is a 3-tier system. All the machines are installed on Windows 2000 operating system. The first tier is the web load simulation; we use the SilkPerformer from Segue [17]. Then, we have the web and application servers running WCS Pro 5.1.0.1 for NT 2000 and WAS V3.5. Last, is the DB2 EEE V7.2. Figure 1 depicts the configuration of the experiment.

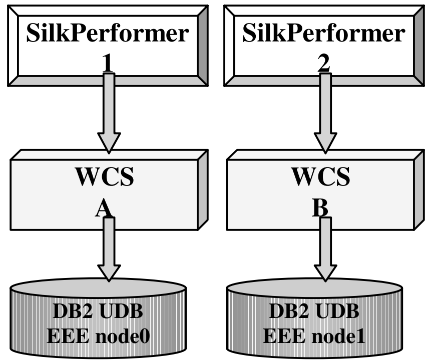
We may notice that in Figure 1b, WCS A and WCS B are connected to different EEE nodes separately for the purpose of the load balance issue as discussed previously. The size of the database is 650MB. The database contains 1,000 categories and 10,000 product items. The workload simulates the browsing shopping scenario, which includes user logon and browsing catalog.

Another important factor that will affect the database performance is the caching mechanism used in the web server. Since the study of cache is out of the scope of this paper, we have to make sure the user's HTTP command will invoke a database call. This requires the cache is not used during our experiment[*]. Therefore, for all the experimental data collected, we are sure that the cache mechanism provided by WCS was turned off. It is also important to point out that a "user" has no think time in our experiment.

[*] Please refer to the WCS installation guide to turn off the cache. In our experiment, we also conduct the experiment with cache on. It turned out that the throughput (Http hits/sec) with cache is more than triple of the result without cache.



(a) DB2 EE configuration



(b) DB2 EEE configuration

Figure 1. Configuration of the experiment

To create the bottleneck scenario for database server, we increase the load on front-end machine to a point that the CPU utilization of the database server is almost 100 percent. At the same time, we should see the performance degradation.

Figure 2 illustrates this scenario. For one Web and application server (that is 30 users) driving the database server, we can get 25 hits/sec for throughput and the CPU utilization for Web and application server is 99%. Since in our experiment the Web server and the application server are sitting on one machine, in the following discussion we just use WCS server to refer both of them. For two WCS servers (a total of 60 users), the database server is 90% used and the average CPU utilization for two WCS servers is 92%. In the case of three WCS servers, we see a bottleneck on database server because the average CPU utilization for each of the three WCS servers is only 88%, for database server is 97% and most important, we only see an average throughput of 22 hits/sec.

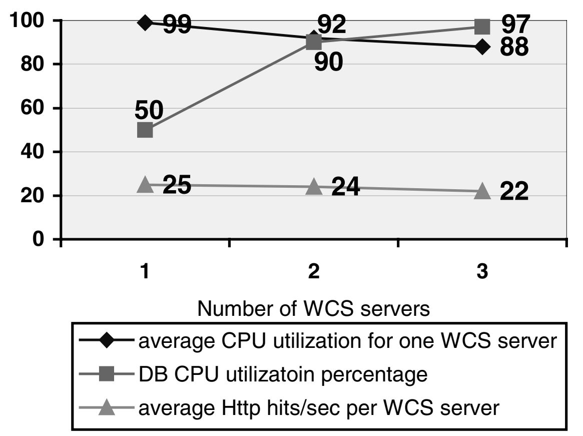


Figure 2. Illustration of the bottleneck scenario

The experimental results indicate that the database server will bottleneck the performance when 90 users (3 WCS machines) access the sample e-store simultaneously. This is just what we expected for the goal 1) as stated in Section 5.1.

5.3. DB2 EEE result

This section gives the result of DB2 EEE deployment (shown in Figure 1b) for the experiment. In the experiment, the tables that are relevant to user information are partitioned. Examples of partitioning tables are USER, ADDRESS, and MEMBER. The partitioning key is **user_id** or **member_id.** Tables related to product information, such as CATALOG and PRODUCT are duplicated among all the EEE nodes.

Because information related to user is partitioned, entries with different **user_id** might locate on different DB2 EEE nodes. Therefore, there will be the case that a user is connected to the wrong database node for the first time and then the database server re-routes him to the correct node. If this is the case, our result could be skewed and not comparable with the DB2 EE result. To avoid this situation, the SilkPerformer script allows us to control the user directing to the correct database node.

In addition, we use the result from 2 WCS servers in bottleneck experiment as our reference for scalability study, as shown in Figure 2. As can be seen from Figure 2, at the point of 2 WCS server, the CPU utilization for the WCS server and database server are over 90%, and the throughput (24 hits/sec) is close to the best case (25 hits/sec). Therefore this type of setup seems to be the optimal point in the performance trend.

Figure 3 shows the DB2 EEE result. For the purpose of comparison, we also plot the DB2 EE result in the figure. From Figure 3, we can see that with the 2 database nodes configured for two WCS servers, we can support 100 users and the throughput is improved to 29 http hits/sec. By comparing this result with DB2 EE (60 users with 24.5 http hits/sec), it is easy to see that using EEE setup:

1) The number of users that the Web server can support is increasing: from 60 users to 100 users.
2) Throughput (Http hits/sec) is also improved: from 24.5 to 29 hits/sec.

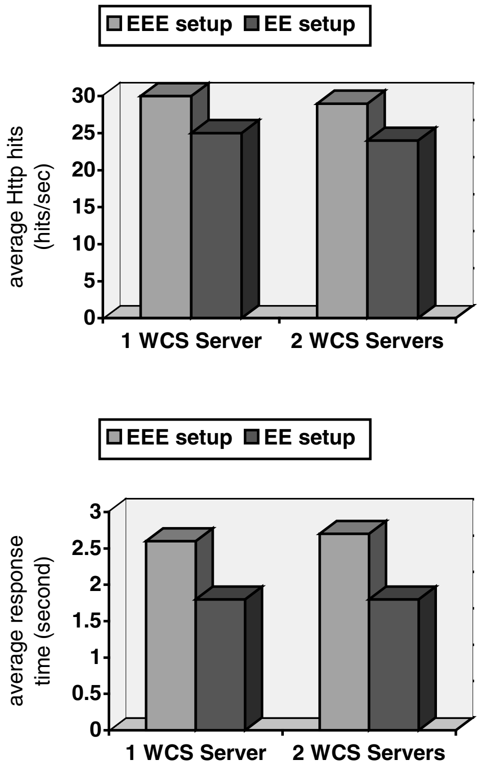


Figure 3. DB2 EEE result

Remember that in the goal defined in Section 5.1, the second one is to study whether the scalability can be achieved by deploying the parallel database server for the E-commerce applications. From the definition of scalability given previously, we can maintain (actually improve) the throughput of the web server with the increasing number of users. We can conclude that the deployment of DB2 EEE for the database server for WCS application will provide the scalability in terms of throughput. Although response time is not the main metric to be measured in this set of experiment, we noticed that it is higher in the parallel database (EEE) setup than in the centralized database (EE) setup. This observation seems reasonable since the use of parallel techniques will introduce some overheads. However, it needs to be further investigated.

As pointed out earlier, another important goal expected from the experiment is that the increasing of the overall throughput of the entire system (across all web servers) would be in proportion to the number of database servers. For example, if we can get a maximum throughput of 22 hits/sec/WCS * 3 WCS = 66 hits/sec by using one database server, then using two database servers, the optimal result would be that we could get 132 hits/sec using about 5.5 web/app servers (132 / 24 = 5.5). This trend can not be directly observed from our experiment. However, a look at the CPU utilization of the database server in the EEE case will be helpful for our analysis. In the EEE case, the CPU utilization of each of the database servers is about 35%. This means about one-third of the database server utilization can serve about 29 hits/sec. It seems that we have a good chance to get more than 70 hits/sec if we add more web/application servers (such as 6 as suggested in the example just mentioned). This is a useful point that could be investigated in the future experiment.

5.4 Implementation observations

The strategy we proposed in the previous section is straightforward. However, implementing it in a real word database system is not a trivial task. This section points out some difficulties we encountered while implementing our strategy. For some of them, we give our solutions and others remain as open issues.

5.4.1. Constraints on replicated table. The replicated table can be implemented by materialized view provided in parallel database systems. A materialized view is designed to improve performance of the database by doing some intensive work in advance. Since our work is based on the duplication of the whole table, we can define the materialized view as a full projection of the whole base table. If the materialized views are allowed for duplication on various nodes in the database systems, we can achieve the purpose of table duplication.

In DB2 EEE, we can use the concept of ***summary*** table for the implementation of materialized view. If DB2 determines that a portion of a query could be resolved using a summary table, the query may be rewritten by the database manager to use the summary table. In a parallel database environment such ad DB2 EEE, summary tables can also be replicated. A replicated summary table is based on a table that may be created in a single-partition nodegroup, but replicated across all of the database partitions in the nodegroup. The *replicated summary tables* may improve query performance in the sense that data shipment is avoided and collocated joins can be formed. In other words, by using replicated summary tables, it is possible that collocation between tables that are not typically collocated can be obtained. Therefore, the replicated summary table reduces the need to retrieve tables that reside remotely.

As a result, the creation of a summary table with the replication option can be used to replicate tables across all nodes in a partitioned database. Some restrictions regarding summary tables have to be aware while designing the data distribution strategy. Some of them are listed below. For a complete set, please refer to the DB2 UDB Administration Guide [14].

1) Summary table cannot be altered.
   In our algorithm, the replicated tables are chosen because they are “static” as regard to update. However, if we do need to update the summary table, such as modify the “product” information, the only way is to first drop the replicated summary table and update the base table, then re-generate the replicated summary table. This might offset the gains brought by the summary table.
2) Primary key cannot be generated for summary table.
   One potential problem without primary key is the primary index cannot be generated automatically since a primary index is automatically created for the primary key. Therefore, we have to manually generate those indexes corresponding to the primary keys for the replicated summary table. Neglect to this point will lead the optimizer fail to use the replicated summary table.
3) Unique index cannot be created.
   In the case that the optimizer will favor a “unique index” over an “index”, the chance for the optimizer to choose the summary table is low. This also relates the issue of optimizer tuning discussed below.

5.4.2. Load balance consideration. Ideally, if we partition the data evenly (in terms of size) across the nodes (assuming these nodes have the same hardware configuration), we expect the load on each node to be balanced. However, in the implementation, other factors will contribute to the unbalanced workload. In DB2 UDB EEE [11], for an application or user session that is connected to a parallel database, the node (partition) at which the CONECT command was processed is called the **coordinator** node. Any database partition can be used as a coordinator node. However, one concern for coordinator node is that it will consume more resources than non-coordinating node since it will interact with the application or user concerning the SQL request and result, it is also responsible for the inter-communication with other EEE nodes.

We conduct another experiment to see this point. The experiment is configured as a two-tier architecture: the Web browser and e-commerce server are set in one NT machine (4-way Pentium III processor), and the database server is installed on 3 nodes (AIX machines) linked by fast Ethernet. The database server and e-commerce application server are connected by a fast Ethernet as well. The workload used is product browsing. In the experiment, we simulate 30 users who access the store Web site to browse the product simultaneously. The load we are concerned with in this setting is mainly the *CPU utilization* of the database node. The performance data is sampled every 5 seconds. The result is an average of 23 minutes observation. The CPU utilization on each node is shown in Figure 4.

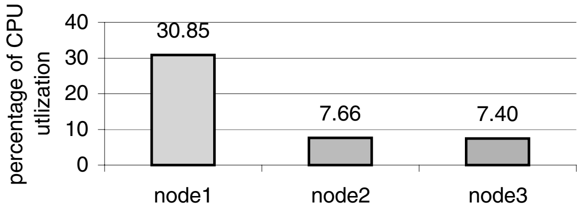


Figure 4. CPU utilization for each node

From Figure 4, we can see that node2 and node3 have nearly identical load in terms of CPU utilization during the whole observation period. There is, however, a noticeable difference between node1 and the other two. This means that the application does not exploit the database system in a very balanced way. Therefore, if multiple application servers connect to one database node, that node will be the coordinator node for all the database transactions. This will eventually lead to the situation that the coordinator node is almost 100% used while other nodes are seldom used. As a result, the coordinator node will bottleneck the whole database performance.

This observation provides us a very important insight into the data distribution issue: the database management system should also take into account the underlying data distribution while constructing the searching strategies. That is, data distributed evenly across nodes cannot guarantee balanced system utilization. For a real application this means that without buying new resources, load balancing can offer more resources and better response times to all users and can effectively avoid the situation where some nodes are idle while others are overloaded. Load balancing becomes more complicated when the system consists of heterogeneous nodes, i.e. faster and slower processors, different amounts of main memory and a different number of CPUs per node. This means that load balancing has to cope with changing, unpredictable load on each node. This also requires the DBMS to be adaptive, that is, to identify the currently important performance factors, to create estimations by profiling the system behavior and to find the trade-off between load balancing overhead and improvement.

For the focus of this paper, we are not trying to modify the DBMS. Instead, we attempt to control the problem brought by coordinator node outside the DBMS, that is the connection between the application server and database server. Accordingly, the overhead of working as a coordinator node can then be spread to more than one node in an instance.

5.4.3. Workload type. Workload plays an important role in collecting the statistics used for our distribution strategy. Different types of workloads could lead to different statistics for the same table. In e-commerce applications, workload could be classified according to different activities such as registration, product browsing, ordering, and fulfillment. Therefore, the derived distribution schema for one workload could unbalance others. We could, accordingly, do the data distribution according to a combined workload, defined as a weighted sum of the different kinds of load. The weight given to each load could be derived experimentally to achieve the maximum optimization goal (e.g. maximum throughput or minimum response time).

In this study, we focus on one type of workload: product browsing. This usually comprises a mixture of less frequent updates or no updates of the product catalog stored in the database, accompanied by a few updates to the order information associated with a given user. This leads to the criterion used to select replicated tables from those tables with read-only access. The tables to be partitioned are those tables with read-write access. In other kind of workload, it is possible that no table is read-only. In such a situation, the ratio of read to write could be used to determine the static aspect of a table. A *threshold* is useful in this case: if the update frequency is smaller than the pre-specified threshold, the table could still be duplicated. For example, if during the observation period, among all the accesses to table *A*, only 5% transactions require a write operations, 95% are read operations. If the threshold is set at 6%, then table *A* could still be regarded as "static" and therefore could be duplicated among database nodes. The best value for the threshold is a trade-off between the performance gain brought by the parallel table access and the update overhead introduced to keep the consistency between base table and replicated table. This value can be derived from the result of the experiment.

6. Conclusions and future work

This paper has discussed issues related to the scalability and performance of back-end database servers used in e-commerce applications. We first argued that the study of database server in e-commerce applications is not isolated from other components: this is a system-wide issue. Therefore, a proper configuration between database servers and other application servers is necessary for the study. We also pointed out that among techniques that should be proposed, data distribution strategies are of prime importance since database scalability cannot be achieved without considering efficient data placement. In particular data distribution strategies should consider the specific e-commerce applications and user expectations in terms of quality of service.

We investigated issues and decision factors related to data distribution strategies. We then proposed strategies for data distribution considering both e-commerce application characteristics as well as user classes. We proposed to differentiate users according to their access patterns to the database. Our strategies include decisions on the configuration of the database in terms of the number of nodes as well as distribution decisions associated to each user class. Last, we conducted some experiment to examine the behavior of the e-commerce application and the database server. This experiment shows that data distribution is valuable if and only if better strategies are proposed for query execution coordination.

Although some experimental results shown in this paper deserve further investigation (remember that the response time is higher when deploy parallel architecture), we still demonstrate the scalability when the throughput is the major performance concern. In addition, these experiments provide a preliminary practical step to study the data distribution strategies for providing scalability. Through the experiment, we also discussed the implementation difficulties in the real-life. In particular, the experimental results are limited by the ability of the optimizer. Lack of knowledge and "control" over the optimizer will also, sometimes, prevent us from deploying the distribution strategies as planned.

References

[1] H. Ye, B. Kerhervé and G. v. Bochmann, *Quality of service aware distributed query processing*, 10th DEXA Workshop on Query Processing in Multimedia Information Systems (QPMIDS), Florence, Italy, 1-3 Sept. 1999, Proc. published by IEEE Computer Society, 1999.

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

[3] H. Ye, *Integrating Quality of Service Information and Requirements in a Distributed Query Processing Environment*, PhD thesis (preliminary draft), University of Montreal, 2002.

[4] H. Ye, B. Kerhervé, G. v. Bochmann, Don Bourne, *Data Distribution Strategies for Providing Scalability in E-Commerce Applications*, Third International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems, MILPITAS, June 21-22, 2001.

[5] M.L. Lee, M. Kitsuregawa, B.C. Ooi, K. Tan. A. Mondal, *Towards Self-tuning data placement in parallel database systems*, SIGMOD 2000, 225-236.

[6] D. C. Zilio, A. Jhingran, S. Padmanabhan, *Partitioning Key Selection for a Shared-Nothing Parallel Database System* IBM Research Report RC 19820 (87739) 11/10/94, T. J. Watson Research Center, Yorktown Heights, NY, October 1994.

[7] K. A. Hua, C. Lee: *An Adaptive Data Placement Scheme for Parallel Database Computer Systems,* VLDB Conference, Brisbane, Australia 1990: 493-506.

[8] P. M. G. Apers, *Data Allocation in Distributed Database Systems* TODS 13(3), 1988, pp. 263-304.

[9] G. P. Copeland, William Alexander, Ellen E. Boughter, Tom W. Keller: *Data Placement in Bubba*, SIGMOD Conference 1988: 99-108.

[10] D. Kossmann, *The state of the art in distributed query processing*, ACM Computing Surveys (CSUR), Volume 32, Issue 4, December 2000, pp 422 – 469.

[11] Baru, G. Fecteau, A. Goyal, H. Hsiao, A. Jhingran, S. Padmanbhan, G.P. Copeland and W.G. Wilson, *DB2 Parallel Edition*, IBM Systems journal, Vol. 34 No. 2, 1995.

[12] Don Chamberlin, *A Complete Guide to DB2 Universal Database*, Morgan Kaufmann Publishers, 1998.

[13] 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.

[14] IBM Corporation: *IBM DB2 Universal Database Administrator Guide*, Version 7.1, IBM Corporation 2000.

[15] http://www-3.ibm.com/software/webservers/commerce/wc_pe/

[16] http://www-4.ibm.com/software/webservers/commerce/wcs_pro/database_schema.pdf

[17] http://www.segue.com/html/s_solutions/s_performer/s_performer.htm