International Journal of Information Science and Management

Parallelism Exploration in B2C and B2B Systems

H. Alipour, M.S.
Shahid Beheshti University, I. R. of Iran
Corresponding Author: hamid.alipour@gmail.com

M. Smaeili, M.S.
Shahid Beheshti University, I. R. of Iran
email: mo.esmaeili@sbu.ac.ir

K. Sheikhi, M.S.
Shahid Beheshti University, I. R. of Iran
email: k.shykhy@sbu.ac.ir

Abstract

As the e-commerce sites are being more secure and reliable in recent years and the number of transactions is rising rapidly, parallelism can help us to reduce response time and increase throughput for e-commerce transactions. This paper will investigate parallelism in on-line transaction processing. It aims to specify those aspects of e-commerce transactions that would profit from parallel processing and analyze current parallel processing techniques to determine those which can be used for e-commerce transactions. The parallel processing techniques proposed in this paper can be easily applied to B2C and B2B on-line transaction processing. Although some parallel implementations of databases have been proposed, to the best of our knowledge, parallel implementations of on-line transaction processing specific to e-commerce are rarely existed.

Keywords: E-Commerce, Parallel Systems, OLTP, Business-to-Consumer (B2C), Business-to-Business (B2B).

Introduction

Electronic commerce (e-commerce) is the use of computers and telecommunication technologies to share business information, maintain business relationships, and conduct business transactions. Its genesis is traced back to the Electronic Data Interchange (EDI) activity in the 1960's. **EDI** refers to the set of activities that are related to the electronic facilitation of the transactions between venders and buyers (purchase orders, waybills, manifests and schedules). Currently, e-commerce depends mostly on the Internet as the underlying platform. Business transactions are events that serve the mission of a business. A transaction provides the primary means by which a business interacts with its suppliers, customers, partners, employees, and the government. Transactions are significant because they capture and/or create data about and for businesses (Whitten & Bently, 1997). Examples of transactions include purchases, orders, sales, reservations,

shipments, invoices, and payment processing.

E-commerce began in the 1990's and was largely driven by the invention of the World Wide Web. The adoption of e-commerce has led to many new business models. The most important models of e-commerce are business-to-business (B2B), business-to-consumer (B2C), and consumer-to-consumer (C2C). *C2C e-commerce* refers to the use of the Internet by consumers to provide goods and information to other consumers; it offers an effective way to exchange goods and information between consumers. *B2C e-commerce* mostly refers to the use of the Internet by a business to provide goods and services to customers; it offers consumers a fast and efficient way to access various products and services from retailers all over the world without leaving home. *B2B e-commerce* refers to the use of the Internet between businesses to order products, receive invoices, and make payments; it reduces production costs, accelerates ordering processes, and improves inventory management. By exploiting efficiency, economy, and speed of the Internet, e-commerce simplifies and reduces the cost of processes involved in business transactions. The parallel processing techniques proposed in this paper can be easily applied to B2C and B2B e-commerce systems. Our prototype e-commerce system implemented in this paper maintains the ACID (*Atomicity, Consistency, Isolation, Durability*) properties of transactions. The transaction processing system is a critical component of any e-commerce system that must manage the transactions between thousands of concurrent clients and back-end systems. Traditional sequential transaction processing techniques may fail to meet e-commerce system requirements such as high throughput and high performance.

To overcome the limitations of sequential processing, such as poor performance and poor throughput, parallel processing techniques could be used to deal with the demands of e-commerce transactions. In this paper, we focus on using parallel processing techniques to improve the performance and throughput of e-commerce transaction processing systems. E-commerce transactions include both on-line analytical processing (OLAP) and on-line transaction processing (OLTP) transactions. The OLTP of e-commerce transactions manages data and processes orders. The OLAP of e-commerce transactions analyzes historical data from OLTP e-commerce systems and provides reports in support of management decisions. In general, OLTP deals with the atomic level of data, needs fast responses, and normally follows standard procedures and well-defined workflows. OLAP focuses on providing analysis capability to management and typically deals with billions or even trillions of transaction records spanning periods from several days to decades. Although some parallel implementations of OLAP transactions (Goil & Choudhary, 1997/1999) and many parallel implementations of databases have been proposed, to the best of our knowledge, parallel implementations of OLTP transactions specific to e-commerce are rarely existed (Furtado, 2004; Dewitt &

Gray, 1992; Raman, Han, & Narang, 2005; Wolf, Turek, Chen, & Ya, 1994).

However, we observe that there are many opportunities to apply parallel processing techniques to OLTP in an e-commerce system. Hence, in this paper, we focus on proposing parallel processing techniques for e-commerce OLTP transactions. To elaborate, typical architecture of an e-commerce system is three-tiered client/server architecture consisting of the GUI tier, the business logic tier, and the database tier. In the business logic tier, one could use a single processor server using multithreading or a multi-processor server to process many transactions concurrently. In the database tier, the database could be fragmented horizontally and distributed to multiple database servers or simply replicated over a number of servers. When a user transaction is processed, the transaction could be processed in parts across multiple database servers simultaneously. For instance, when a user wants to find a particular product, he/she could submit the search criteria to the server. The server in the business logic tier could then transform the search into a query. The query could then be forwarded to different systems for processing. For each system, the query could be executed against parallel databases in the database tier. E-commerce systems are complex. A single transaction may include several logical steps. Some of these steps have dependencies between them, while others do not. The steps that have no dependencies can be executed concurrently. As described above, there are opportunities to apply parallel processing techniques when an e-commerce system processes e-commerce transactions. Identifying what to parallelize and which parallel techniques to use and incorporating them into a flexible and scalable design for a parallel e-commerce system is the focus of this paper. Thus, the aims of this paper are to:

1. Characterize e-commerce transactions with a view to find those aspects that would benefit from parallel processing.

2. Evaluate current parallel processing techniques to determine those techniques that can be applied to e-commerce transactions; and

3. Provide a reliable, flexible, and scalable design of an e-commerce transaction processing system that uses parallel processing techniques to deal with data intensive transactions and process those transactions faster.

This paper describes an e-commerce system design that is three-tiered architecture (the GUI tier, the business logic tier, and the database tier) system using different parallel processing techniques. The design helps e-commerce systems that deal with large dataset to get faster response. This paper also describes an implementation of a prototype e-commerce transaction processing system as a case study that is developed to demonstrate the feasibility of the design. We compare our implementation with the implementation of an e-commerce system that uses traditional *sequential* processing techniques, and highlight the performance improvement brought by our design.

Opportunities that Can Apply Parallel Processing Techniques

The three-tiered architecture separates an application into different blocks and makes the application easier to maintain and upgrade. A three-tiered architecture system can be easily deployed on a distributed environment, which provides opportunities for applying parallel processing.

In following sections, we will explore some opportunities where we can apply parallel processing techniques. Most of these opportunities are in the business logic tier and the database tier. An e-commerce system contains some typical operations such as product search, product comparison, payment processing, and order processing. All these operations can benefit from applying parallel processing techniques. The parallel processing techniques proposed in this paper can be easily applied to B2C and B2B e-commerce systems. Figure 1 indicates the various steps involve in B2C transactions and Figure 2 indicates these steps for B2B.

Parallel search and comparison

The search operation can be divided into several steps. First, in the presentation service tier, a customer provides some search criteria and then submits the search criteria to an e-commerce system. The business logic tier receives the search criteria from the customer and converts the search criteria into database queries, and then submits the queries to the database tier. The database tier processes the queries and returns the result to the presentation service tier. Then, the business logic tier may apply some business logic to the data and returns the result to the customer.

In an e-commerce system, it is quite common to divide a large database into several partitions and distribute these partitions to different database servers. When the database is divided and distributed to different database servers, it is possible to perform *parallel search* on these partitions. We can perform search in different data partitions concurrently and merge the search result, and then return the result to customers.

To provide broader selection of products, some e-commerce systems provide services to let customers search for products from the e-commerce systems of their partners. In these cases, when an e-commerce system receives a search request, it will perform a search on its own local database servers. At the same time, the system will forward the search criteria to its partner systems, and will let partner systems perform their searches. The search in the local e-commerce system and the searches in the partner systems can be executed concurrently. When all searches are completed, the e-commerce system will collect and merge search results from different systems. The merged search result will then be returned to the customer. Figure shows the activity view of a parallel search.

A customer sometimes needs the search result in a certain order. The e-commerce

system needs to compare the search result and return the search result in that specific order. It is also possible for an e-commerce system to apply parallel processing techniques for comparison. The *parallel comparison* is very similar to the parallel search. The difference is when the system merges search result from different partitions or from different systems. The parallel comparison must compare the search result from different partitions or from different systems and return the search result in order.

Parallel payment processing

The payment process consists of processes in at least two different accounts: the first process withdraws money from the bank account of the customer, and the second process deposits money to the bank account of the e-commerce system. In some cases, an e-commerce system may allow a customer to withdraw money from multiple bank accounts. In those cases, the payment process consists of more than two processes.

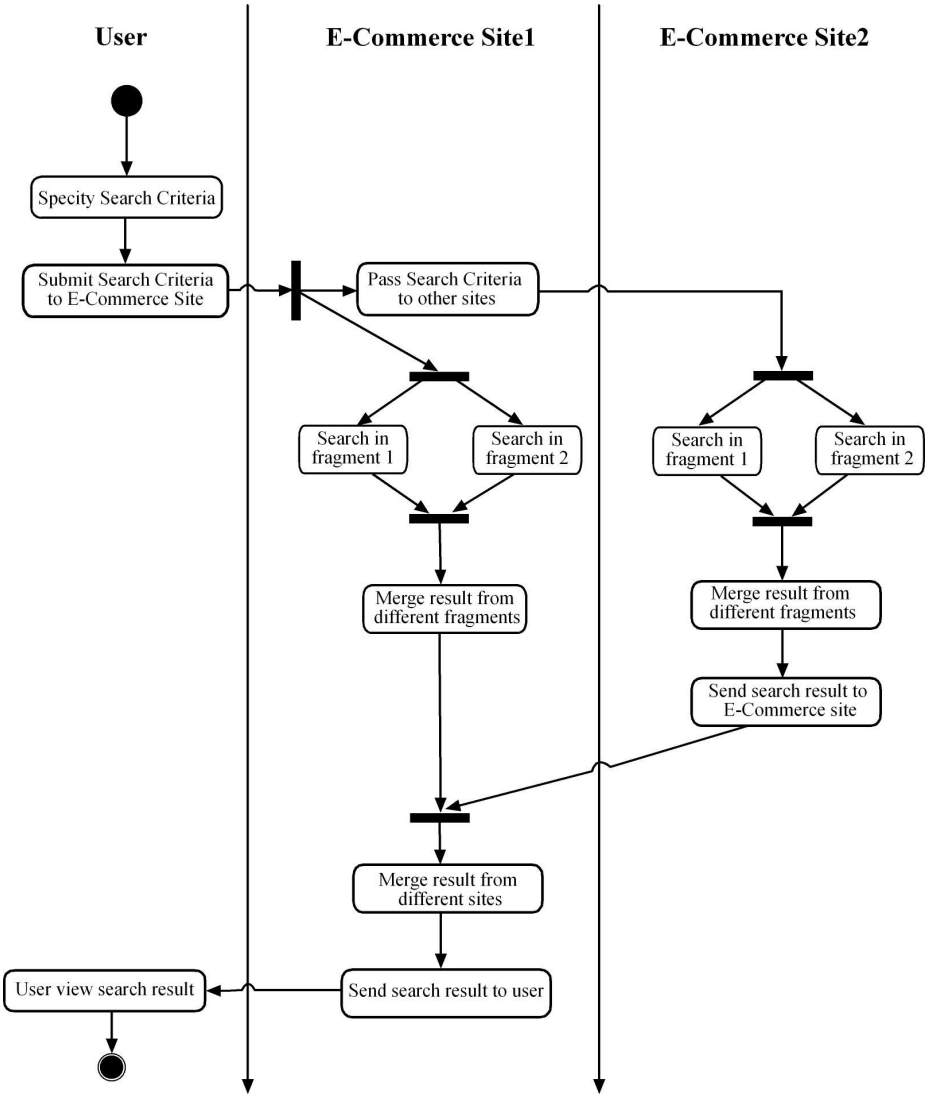

Figure 1. Activity view of parallel search.

The *withdrawal process* consists of three sub-steps. The bank will first validate the customer's bank account, then check the available credit of that account, and finally withdraw money from that account. The *deposit process* consists of two sub-steps. The bank will first validate the merchant account, and deposit money to that account. Because the withdrawal and the deposit processes can be operated in different accounts or different banks, it is possible for us to apply parallel processing techniques in the business logic tier. When the e-commerce system receives the customer payment information, the system initiates two threads. The first thread deposits money to the bank account of the e-commerce system, and the second thread withdraws money from the bank account of the customer. These two threads are executed concurrently.

In the context of parallel processing, it is important to keep the ACID properties of a transaction. If one thread fails in one of its sub-steps, all threads should roll back their changes. If all threads are executed successfully, all their changes should be committed. In either payment processing, if one of the sub-steps fails in the withdrawal thread or the deposit thread, both threads should roll back their changes. Only when both threads are executed successfully, all their changes will be committed. Figure 2 shows an activity view of the process of parallel payment. If the e-commerce system allows a customer to pay from multiple accounts, multiple withdrawal threads should be created.

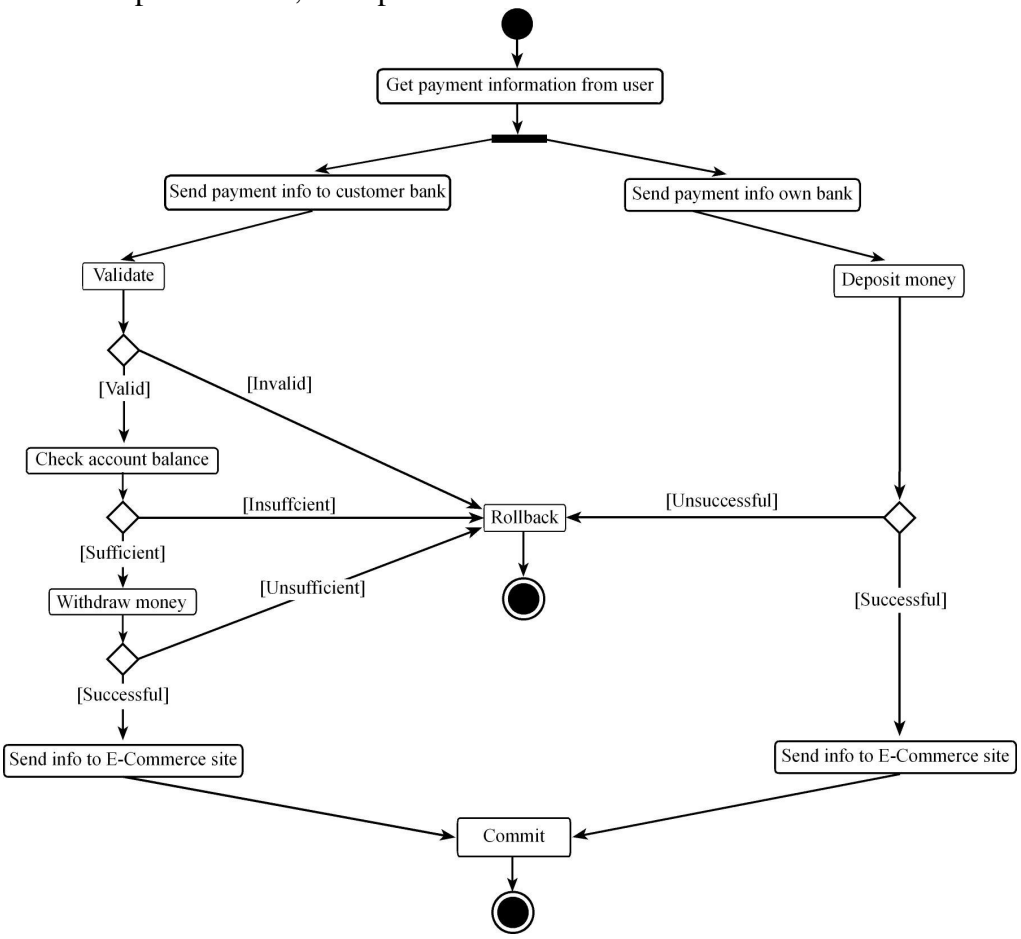

Figure 2. Activity view of parallel payment.

Parallel order processing

The order processing is the core service of an e-commerce system. The order process consists of several processes: the inventory check process, the payment process, and shipping process. For an e-commerce system, there are many concurrent users and the inventory is updated dynamically. For each order, the system should perform an inventory check against each product in the order (the *inventory check process).* The *payment process* was described before. The *shipping process* consists of shipping information confirmation and the shipping arrangement. For these different processes, we can apply parallel processing techniques. We can initialize three different threads for these processes. The first thread checks and updates the inventory. The second thread processes the payment. The third thread processes the shipping. These threads are then executed concurrently. If any of the threads fails in any of its sub-steps, all the threads should roll back their changes, and an error message will be generated and returned to the customer. If all the threads are executed successfully, all the threads will commit their changes. Finally, the system will record the detailed order information, and will return an invoice to the customer for future reference. Figure 3 shows an activity view of order process in parallel.

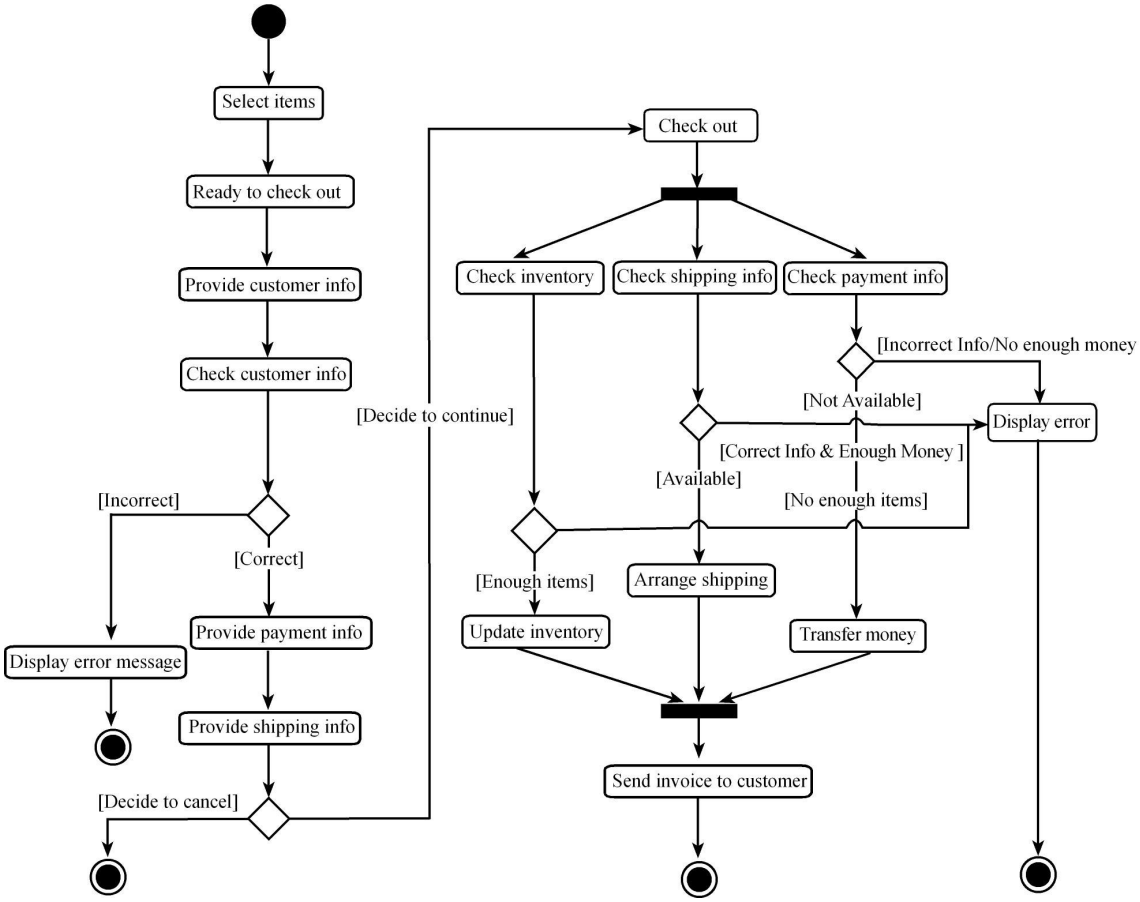

Figure 3. Activity view of parallel order process.

Prototype Implementation & Case study

This Section describes the implementation of our prototype e-commerce transaction processing system.

To discuss how to apply parallel processing techniques in a more specific context, we implemented a prototype e-commerce system in this paper, which is a subset of an online bookstore. In the system, a user may log in as a customer, search for favorite books, add books to cart, check out books, and make payment.

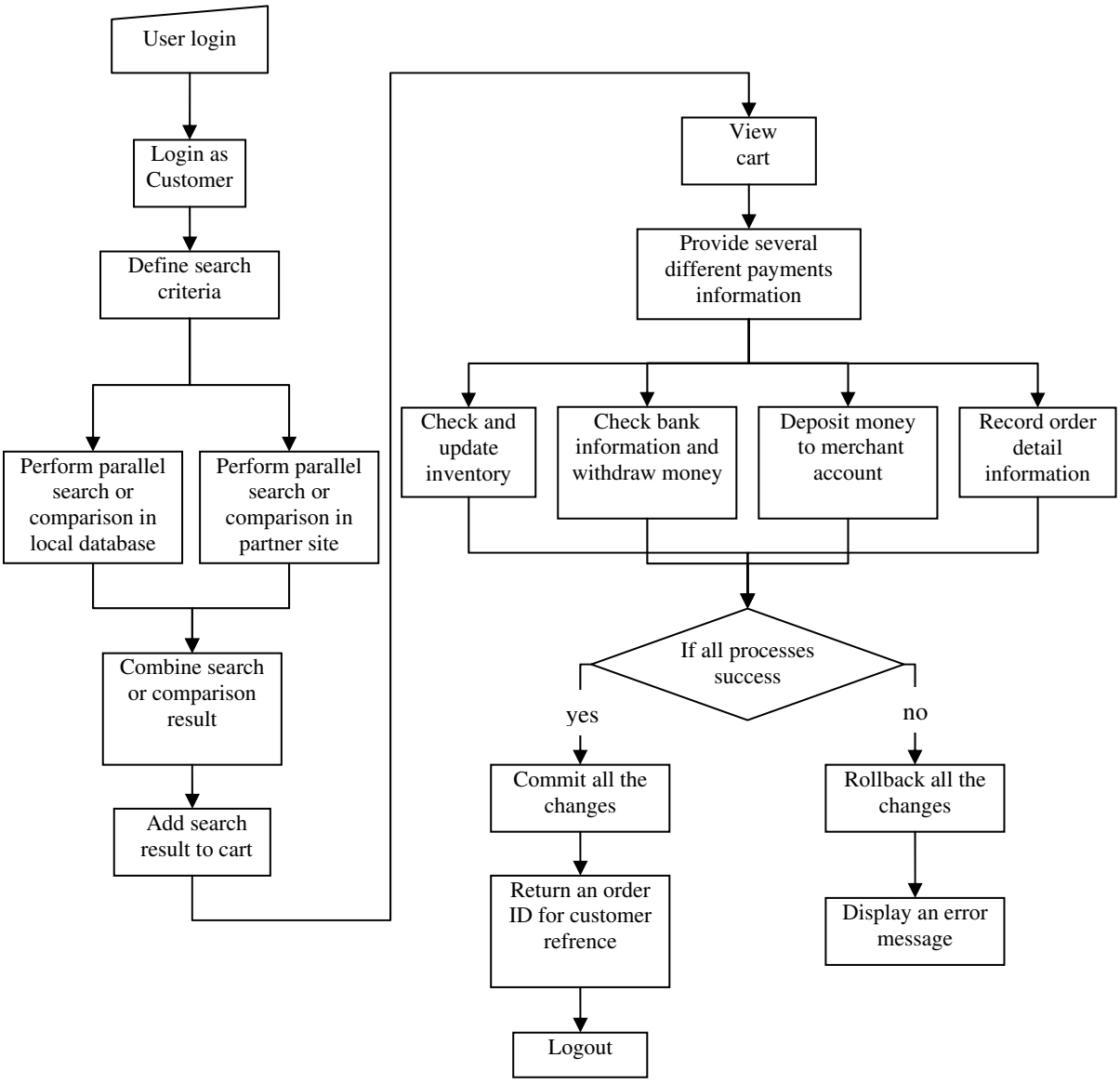

Figure 4. System flowchart.

Figure 4 shows the flowchart of the prototype e-commerce system implemented for this paper. As shown in Figure 4, a user must log in to the system as a customer to buy books from the online bookstore. If the customer's login is successful, the customer could specify the search criteria to find books. If the customer wants the result in a

certain order, he or she could further specify the ordering criteria. Then, the customer could press the search button to perform a search operation. In the search operation, the system initializes several threads to perform concurrent searches in several different sites. One of the threads performs a parallel search in databases of a local system, and other threads perform parallel searches in databases of partner e-commerce systems. After all threads finished their searches, the e-commerce system combines the search results from different systems and returns the combined result to the customer. The customer could then choose books from the combined search result and add the books to a shopping cart. The customer could add more books to the shopping cart, or remove books from the shopping cart. If the customer wants to check out, he or she should first provide the payment information. The payment information may include more than one bank account information. After the payment information is provided, the customer could then press the checkout button to check out. The checkout process uses a two-phase commit mechanism. The two-phase commit splits a commit operation into two parts: the prepare phase and the commit phase. In the *prepare phase* of this system, the system initializes several threads to perform several different jobs. One thread checks and updates the inventory. Several threads check bank accounts of the customer and withdraw money from those accounts. One thread deposits money to the bank account of the e-commerce system.

One thread records the order and detailed order information. All these threads perform their operations concurrently without committing their changes in the prepare phase. In the *commit phase,* the system first checks if operations of all threads have been successful. If any error occurs in any of the threads, the e-commerce system will roll back changes made by all those threads and will display an error message. If operations of all threads are successful, the e-commerce system will commit all the changes and give the customer an invoice for future reference. Finally, the customer could log out by pressing the logout button.

Metrics Used to Evaluate Performance

Metrics such as run-time, speedup, and scale-up are often used to gauge the performance of a parallel implementation. We used these three metrics to analyze the performance of the parallel implementation in this paper.

• **Run-time:** Run-time is the most primitive metric to gauge the performance of a parallel application. We compare the best sequential algorithm run-time *ts* with the parallel run-time *tp.*

• **Speedup:** Speedup is a metric that captures the relative benefit of solving a problem in parallel over using a single processor system for the same problem.

Speedup is defined as:

$$S(P) = T_1/T_P$$

Where T_1 is the time required by the algorithm on one processor, and T_p is the time required on P processors.

• **Scale-up:** Scale-up is the ability of an application to retain response time as the job size or the transaction volume increases by adding additional resources.

The Performance Analysis Environment

The *BookStore* database contains 6 million records (approximately 380 MB in size). Each of the *Bank1, Bank2, Bank3, SaveBank* databases contains 1 million records (approximately 306MB in size). The performance test was based on three kinds of frequently used e-commerce transactions: the search, comparison, and order processing (Because payment processing was included in the order processing of the prototype system, there was no performance test made for payment processing.) For each kind of transactions, the speedup and the scale-up analyses were performed. For the speedup test, the *BookStore* database was partitioned using a range partitioning algorithm on the *BookNo* field and evenly distributed to each node. Table 1 shows how data in the *BookStore* database were partitioned and distributed.

Table 1

BookStore Database for the Speedup & Transaction Volume Scaleup Tests

Test name	Nodes involved	Records in each node	Data size in each node	Total records	Total data size
Test1	1	6M	380MB	6M	380MB
Test2	2	3M	190 MB	6M	380MB
Test3	3	2M	126.7MB	6M	380MB
Test4	4	1.5M	95MB	6M	380MB
Test5	5	1.2M	76MB	6M	380MB
Test6	6	1M	63.3MB	6M	380MB

For the scale-up test, two different tests were performed: (1) the transaction volume test and (2) the response time test. The *transaction volume test* measured how many transactions the system could process per minute when the number of processor nodes increased while the amount of data in the database remained constant. The *BookStore* database used in the transaction volume scale-up test is shown in Table 1.

The *response time test* measured the response time for a transaction when the number of processor nodes increased in proportion to the amount of data in the database. The *BookStore* database used in the response time scale-up test is shown in Table 2.

Table 2

BookStore Database for the Response Time Scaleup Test

Test name	Nodes involved	Records in each node	Data size in each node	Total records	Total data size
Test1	1	1M	63.3MB	1M	63.3MB
Test2	2	1M	63.3MB	2M	126.7MB
Test3	3	1M	63.3MB	3M	190MB
Test4	4	1M	63.3MB	4M	253.3MB
Test5	5	1M	63.3MB	5M	316.7MB
Test6	6	1M	63.3MB	6M	380MB

These two scale-up tests are related. Because the transaction volume could be computed using ***transaction volume = one minute / response time per transaction***, the transaction volume test could be considered as the response time test when the number of processor nodes increased while the amount of data in the database remained constant.

Test Results on Searches and Comparisons

The search and comparison transactions are similar. Comparison transactions first perform a *search* operation, compare the result from the *search* operation, and then return the reordered result to the client. For both search and comparison transactions, different numbers of database servers and different criteria are used for the speedup and the scale-up tests.

Searches based on one criterion

Table shows the search criterion and result set size of Search1 and Comapre1 for the speedup and the scale-up tests.

Figure 5. shows the run-time for different queries in the speedup test for Search1.

Table 3

Search and Compare Criterion for Search1 and Compare1

Query Name	Criteria	Result set size	Ordering field (for compare only)
Search1-a Compare1-a	Title contains "java"	19	Title
Search1-b Compare1-b	Author contains "Chris"	4	Title
Search1-c Compare1-c	Publisher = "Microsoft"	6	Title
Search1-d Compare1-d	Price>10	60	Title

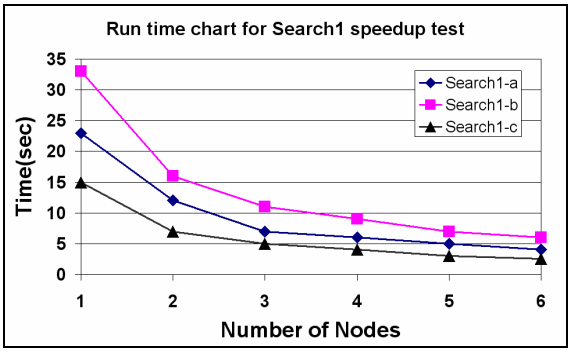

Figure 5.Run time chart for searchl speedup test

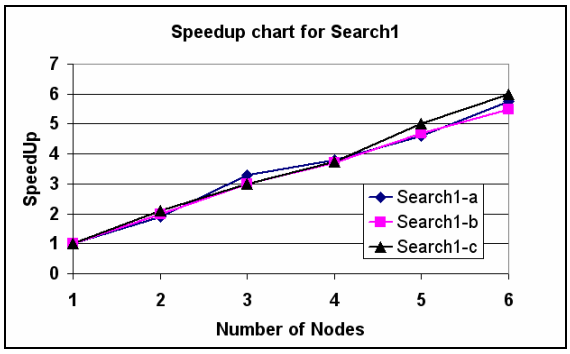

Figure 6. Relative speedup chart for the search1 speedup test

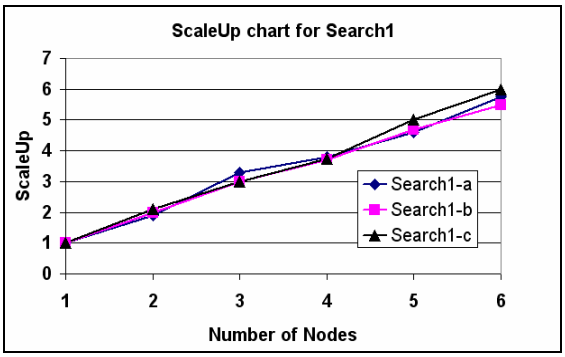

Figure 7. Scaleup chart for the search scaleup test

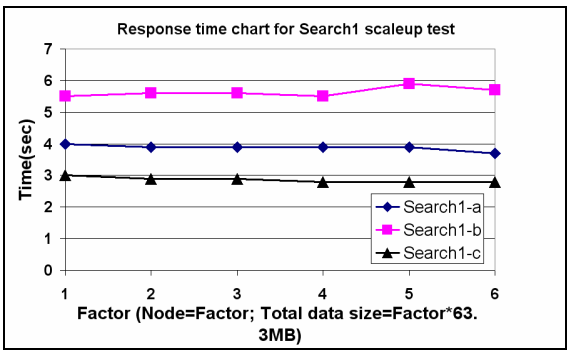

Figure 8. Response time chart for the searchl scaleup test

The run-time of a sequential search transaction (i.e., when the number of nodes = 1) can be computed using:

$$T_{seq\text{-}search} = t_{ss}$$

where $T_{seq\text{-}search}$ stands for the run-time of the sequential search, and t_{ss} stands for the time that used for the sequential search. Similarly, the run-time of a parallel search transaction (i.e., when the number of nodes > 1) can be computed using:

$$T_{par\text{-}search} = t_{start} + t_{ps} + t_{comm}$$

where $T_{par\text{-}search}$ stands for the run-time of the parallel search, t_{start} stands for the time used for initializing the parallel search in different nodes, t_{ps} stands for the time used for the parallel search, and t_{comm} refers to the communication time between nodes (e.g., the time for each node to send result to the originating node). As the search time in each node may be different, t_{ps} is the longest time among all the nodes performing the searches.

Figure 6 shows the relative speedup for different queries in the speedup test for Search1; it shows that linear speedups (i.e., the time taken for searching transactions decreased in proportion to the increase in the number of processor nodes) were achieved

for all searches using one criterion. In our test, since the data for search was large and the result set of the search was small, the communication overhead was small. So,

$$T_{par\text{-}search} = t_{start} + t_{ps} + t_{comm} \approx t_{ps}.$$

Moreover, the number of records in each node was *1/n* of the original data (where *n* = the number of nodes involved in the test). Therefore, the scan time of a parallel search was *1/n* of a sequential search (i.e $t_{ps} \approx t_{ss} / n$). Hence,

$$T_{par\text{-}search} = t_{start} + t_{ps} + t_{comm} \approx t_{ps} \approx T_{seq\text{-}search} / n$$

and

$$Speedup_{search} = T_{seq\text{-}search1} / T_{par\text{-}search} \approx n.$$

This explains why linear speedup was achieved (when result set was small and the data for the search was large). However, when the result set is large, the communication cost t_{comm} will increase, which will then cause a sub-linear speedup.

The transaction volume of a sequential search and a parallel search can be computed, respectively, using:

$$Vol_{tran\text{-}seq} = 1 / T_{seq\text{-}search}$$

and

$$Vol_{tran\text{-}par} = 1 / T_{par\text{-}search}$$

So, the scale-up can be computed using

$$Scaleup_{search} = Vol_{tran\text{-}par} / Vol_{tran\text{-}seq} = T_{seq\text{-}search} / T_{par\text{-}search} \approx n$$

This explains why in Figure 7 linear scale-ups (i.e., the transaction volume was increased in proportion to the number of processor nodes was increased) were achieved for all searches using different one-criterion queries in various tests for Search1.

Figure 8 shows the response times for the response time scale-up test. In the response time scale-up test, because the number of records in each node was constant, the scan time of parallel search was almost the same as the sequential search in one node ($t_{ss} \approx t_{ps}$). So,

$$t_{\text{seq-search}} \approx t_{\text{par-search}}$$

This explains why linear scale-ups (i.e., *search times were sustained when the number of processor nodes was increased in proportion to the amount of data in the database)* were achieved for all searches using different one-criterion queries in various tests for Search1.

For the scale-up tests, linear scale-ups were achieved when result set was small and the data for search was large. However, when the result set was large, the

communication cost t_{comm} increased. The increased communication cost then caused a sub-linear scale-up.

Comparisons based on one criterion

The run-time for comparing transaction can be computed based on the run-time for searching transaction. The run-time for handling sequential comparison transactions (i.e., when the number of nodes =1) can be computed using:

$$T_{seq\text{-}compare} = T_{seq\text{-}search} + t_{compare} = t_{ss} + t_{compare}$$

where $T_{seq\text{-}compare}$ stands for the sequential compare transaction run-time, and $t_{compare}$ refers to the time used to compare and order the searched result. Similarly, the run-time for handling parallel comparison transactions (i.e., when the number of nodes> 1) can be computed using:

$$T_{par\text{-}compare} = T_{par\text{-}search} + t_{compare} = (t_{start} + t_{ps} + t_{comm}) + t_{compare}$$

where the $T_{par\text{-}compare}$ stands for the parallel compare transaction run-time, and $t_{compare}$ refers to the time used to compare and order the searched result.

Here, the data for search was large, and the result set of the search was small. Therefore, the time used to initialize the search, the time used to communicate, and the time used to compare and order the result set were all short. Hence,

$$T_{seq\text{-}compare} = T_{seq\text{-}search} + t_{compare} = t_{ss} + t_{compare} \approx t_{ss}$$

and

$$T_{par\text{-}compare} = T_{par\text{-}search} + t_{compare} = (t_{start} + t_{start} + t_{comm}) + t_{compare} \approx t_{ps}$$

The number of records in each node was *1/n* of the original data (where *n* stands for the number of node involved in the speedup test), and the runtime was *1/n* of the sequential search (i.e., $t_{ps} \approx t_{ss} / n$) Hence,

$$T_{par\text{-}compare} \approx t_{ps} \approx t_{ss} / n \approx T_{seq\text{-}compare} / n$$

and

$$SpeedUp_{compare} = T_{seq-compare} / T_{par\text{-}compare} \approx n.$$

This explains why linear speedup was achieved (when the result set was small and the data for the search was large). However, when the result set was large, the communication cost t_{comm} increased, and the time used to compare and reorder the search result $t_{compare}$ increased. These two increased-costs then caused a sub-linear speedup.

Test Results on Order Processing

For the order transactions, two sets of order tests were performed. In the first set,

each transaction includes *small* number of distinct books; in the second set, each transaction includes *large* number of distinct books. For each set of order transaction, we recorded the run-time, and computed relative speedups and scale-ups.

Small orders

For small orders, we used three different transaction queries with one deposit bank (i.e., one bank for customers to deposit money) and varying number of withdrawal banks (i.e., banks for customers to withdraw money). Query Smallorder-a uses one withdrawal bank for the e-commerce system, query Smallorder-b uses two withdrawal banks, and query Smallorder-c uses three withdrawal banks. Table shows distinct banks involved in each transaction. In each transaction in the test, 60 different books are ordered. All distinct books are evenly distributed into different partitions in different nodes.

Table 4

Banks Involved in Different Queries in Smallorder

Query	Withdraw bank No.	Deposit bank No
Smallorder-a	1	1
Smallorder-b	2	1
Smallorder-c	3	1

The run-time of a sequential order transaction can be computed using:

$$T_{seq} = T_{seq\text{-}update} + T_{b1} + \ldots + T_{bn} + T_{order}$$

where T_{seq} stands for the run-time of a sequential order transaction, $T_{seq\text{-}update}$ stands for the time used for inventory check and update, $T_{b1} \ldots T_{bn}$ stands for the time used for communicating with different banks, and T_{order} stands for the time used for recording the order and detailed order information. The run-time of a parallel order transaction can be computed using:

$$T_{par} = Max\,(T_{par\text{-}update}, T_{b1}, \ldots, T_{bn}, T_{order})$$

where T_{par} stands for the run-time of a parallel order transaction, $T_{par\text{-}update}$ stands for the time used for performing inventory check in different data nodes, $T_{b1} \ldots T_{bn}$; and T_{order} are same as above. Since all the operations (inventory update, different bank processing, and order generation) in parallel order transaction are concurrently performed, the processing time T_{par} is the longest processing time of all operations.

Figure 9 shows the run-time for different queries in a speedup test for Smallorder.

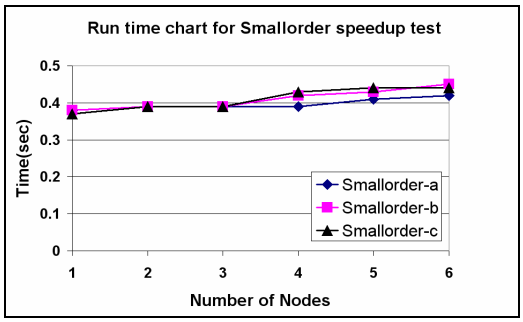

Figure 9. Run-time chart for the smallorder speedup test

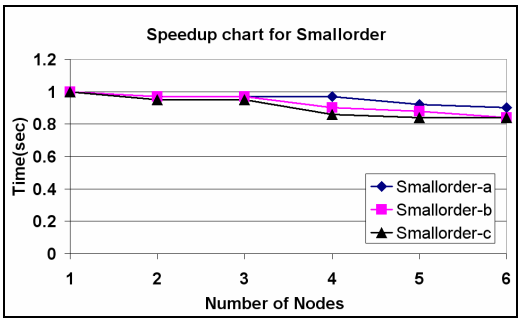

Figure 10. Relative speedup chart for the smallorder speedup test

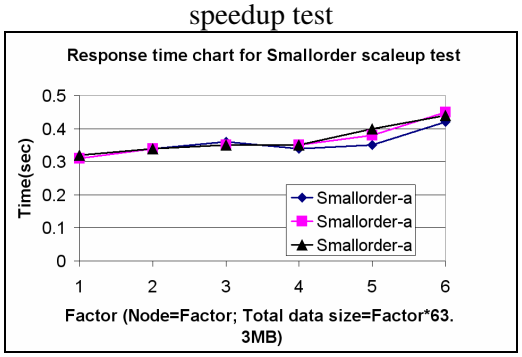

Figure 11. Response time chart for the smallorder scaleup test

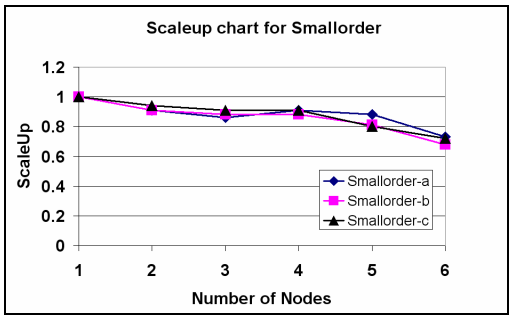

Figure 12. Scaleup chart for the smallorder scaleup test

Because the number of records in one node was *1/n* of original data, the update time was *1/n* of original data. Therefore,

$$T_{par\text{-}update} = T_{seq\text{-}update} / n + T_{comm} + T_{startup}$$

where $T_{startup}$ stands for the time that the originating node to start up the update operation in different nodes, and T_{comm} stands for the time that the originating node sends the ordered books information to other nodes for the update operation.

Hence, the run-time of a parallel order transaction can be computed using:

$$T_{par} = Max(T_{seq\text{-}update} / n + T_{comm} + T_{startup}, T_{b1}, \ldots T_{bn}, T_{order})$$

For the test for small orders (shown in Figure 10), when the involving nodes increased, the communication cost increased and $T_{seq\text{-}update} / n$ decreased. Since $T_{seq\text{-}update}$ was very small, the increase in the communication cost was more than the decrease in $T_{seq\text{-}update} / n$. Therefore, $T_{par\text{-}update}$ increased. Moreover, the speedup test was performed using a local network, the T_{b1}, ... T_{bn}; were very small, and can be ignored. Thus, when the order was small, $T_{par\text{-}update}$ took longer than $T_{seq\text{-}update}$ when node increased. This means that the performance of parallel order processing was worse than sequential order processing. However, in reality, the communication cost with banks T_{b1}....T_{bn}; was high, and was much greater than $T_{par\text{-}update}$ or $T_{seq\text{-}update}$ for small orders. So,

$$T_{seq} = T_{seq-update} + (T_{b1} + \ldots + T_{bn}) + T_{order} \approx T_{b1} + \cdots + T_{bn}$$

and

$$T_{par} = Max(T_{par\text{-}update}, T_{b1}, \cdots, T_{bn}, T_{order}) \approx Max(T_{b1}, \ldots T_{bn})$$

Comparing to the sequential order transaction, the parallel order transaction took less time ($T_{par}<T_{seq}$ because $Max(T_{bl}, \ldots, T_{bn}) < (T_{bl}+ \ldots +T_{bn},)$. However, using more nodes did not lead to high speedup for small orders, because using more nodes did not affect the communication cost with banks.

Figure 11 shows the response times for different queries in the Smallorder's response time scaleup test.

Figure 12 shows that sub-linear scaleups were achieved for orders have small number of distinct books.

When the involving nodes increased, the communication cost increased. Hence, $T_{par\text{-}update}$ increased as well. Because we used a local network, $T_{b1},\ldots,T_{bn};$ were very small, and were ignored. Moreover, when the order was small, *Tpar-update* took longer than $T_{seq\text{-}update}$. (i.e., not lead to high scaleup). Figure 12 shows such a result. However, in reality, the communication with banks $T_{b1},\ldots,T_{bn};$ was long, and was much greater than $T_{par\text{-}update}$ or $T_{seq\text{-}update}$ for small orders. So,

$$T_{seq} = T_{seq-update} + (T_{b1} + \ldots + T_{bn}) + T_{order} \approx T_{b1} + \cdots + T_{bn}$$

and

$$T_{par} = Max(T_{par\text{-}update}, T_{b1}, \cdots, T_{bn}, T_{order}) \approx Max(T_{b1}, \ldots T_{bn})$$

Compared to the sequential processed order, the parallel processed order used less time. However, the scaleup was *not* high because using more nodes did not affect the processing time. Thus, the number of transactions processed per minute did not increase when the involving nodes increased.

Large orders

Similar to the queries used for small orders, we also used three transaction queries with one deposit bank and varied number of withdrawal banks (where Queries Largeorder-a, -b, & -c uses one, two, & three withdrawal banks respectively) for large orders. Table 5 shows different banks involved in each transaction. In each transaction in the test, 6060 distinct books were ordered.

Table 5

Banks Involved in Different Queries in Largeorder

Query	Withdrow bank No.	Deposit bank No
Largeorder-a	1	1
Largeorder-b	2	1
Largeorder-c	3	1

Figure 13 shows the run-time for different queries in the speedup test for Largeorder.

For large order, the communication cost with banks can be ignored. Hence,

$$T_{Seq} = T_{\text{sec}-update} + (T_{b1} + \cdots + T_{bn}) + T_{order} \approx T_{\text{sec}-update} + T_{order}$$

and

$$T_{Par} = Max(T_{par-update}, T_{b1}, \cdots, T_{bn}, T_{order}) \approx Max(T_{par-update}, T_{order})$$

Because in our implementation, the order table was not partitioned and distributed, T_{order} remained the same and no communication was involved. Moreover, in most cases, $T_{par\text{-}update} > T_{order}$ So,

$$T_{par} = Max(T_{par-update}, T_{b1}, \cdots, T_{bn}, T_{order}) \approx Max(T_{par-update}, T_{order}) \approx T_{par-update}$$

In the large-order speedup test, when the number of nodes increased, T_{comm} increased and $T_{seq\text{-}update} / n$ decreased. Since the decrease in $T_{seq\text{-}update} / n$ was more than the increase in T_{comm}, a sub-linear speedup was achieved (as shown in Figure 14) for orders having large number of distinct books.

Figure 15 shows the response times for different queries in the Largeorder's response time scaleup test.

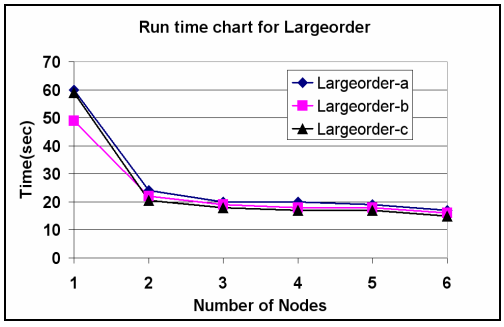

Figure 13. Run-time chart for the Largeorder speedup test

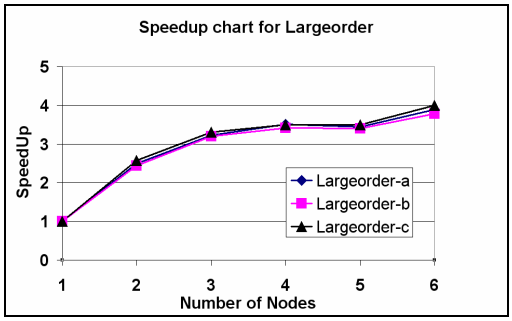

Figure 14. Relative speedup chart for the Largeorder speedup test

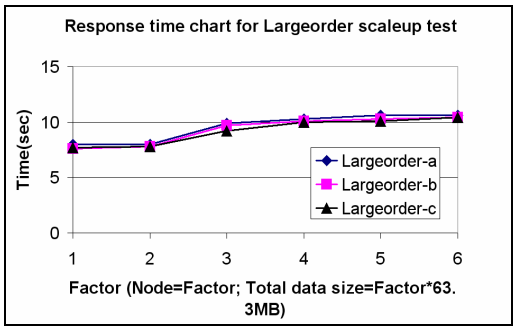

Figure 15. Response time chart for the Largeorder scaleup test

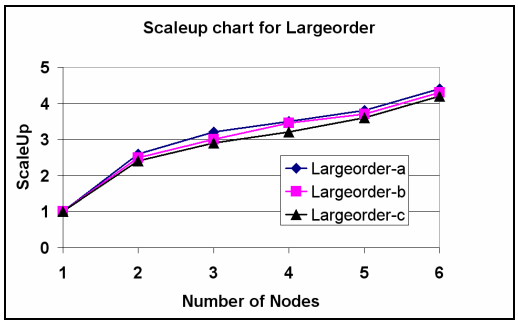

Figure 16. Scaleup chart for the Largeorder scaleup test

Figure 16 shows the scaleup for different queries in the scaleup test for Largeorder.

We can see that sub-linear scaleup has been achieved for orders have large number of distinct books. Here, when the number of nodes increased, T_{comm} increased and $T_{seq\text{-}update} / n$ decreased. Since the decrease in $T_{seq\text{-}update} / n$ was more than the increase in T_{comm}, the number of order transactions processed per minute increased. As shown in Figure 16, a sub-linear speedup was achieved.

Conclusions

The wide acceptance and use of the World Wide Web has significantly expanded the horizons of commerce, and has changed the face of commercial transactions we used to know to a new form of transactions (namely, e-commerce transactions). An e-commerce transaction processing system, which processes e-commerce transactions, requires high throughput and high performance. Traditional sequential transaction processing techniques fails to meet these e-commerce system requirements. However, parallel processing techniques could be used to deal with the demands of e-commerce system. In this paper, our goal was to (a) investigate typical e-commerce transactions, (b) identify the aspects in those transactions that could benefit from parallel processing (c) apply parallel processing techniques suitable for those transactions to e-commerce system, and (d) provide a reliable, flexible, and scalable e-commerce transaction system design.

We analyzed some typical e-commerce transactions such as the search, compare, payment, and order transactions. Each of these transactions can be benefited by applying parallel processing techniques in their implementations. Each of the transactions was analyzed using the UML activity diagram to isolate opportunities that are amenable to parallel processing techniques.

Finally, we implemented a prototype e-commerce system that applies parallel processing techniques designed in this paper. Performance test results of the prototype e-commerce system showed that applying parallel processing techniques results in better performance and higher throughput than using the sequential processing techniques.

References

DeWitt, D. L. & Gray, J. (1992). Parallel database systems: The future of high performance database systems. *Communications of the ACM, 35*(6), 85-98.

Furtado, P. (2004). Workload-based placement and join processing in node-partitioned data warehouses. In *Proceedings of the 6th DaWaK Conference,* 38-47.

Goi1, S. & Choudhary, N. (1997). High performance OLAP and data mining on parallel computers. *Data Mining and Knowledge Discovery. 1*(4), 391-417.

Goi1, S. & Choudhary, A. N. (1999). A Parallel scalable infrastructure for OLAP and data mining. In *Proceedings of International Database Engineering and Applications Symposium,* 178-186.

Raman, V., Han, W., & Narang, I. (2005). Parallel querying with non-dedicated computers. In *Proceedings of the 31st VLDB Conference,* 61-72.

Whitten, J. L. & Bentley, L. D. (1997). *Systems analysis and design methods.* (4th ed.). Columbus: OH.

Wolf, J. L. Turek, J., Chen, M., & Yu, P. S. (1994). Scheduling multiple queries on a parallel machine. In *Proceedings of the 1994 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems,* 45-55.