Multi-Criteria Transaction for E-Commerce Applications

Peng Li, JayaBharath Goluguri, I-Ling Yen
Department of Computer Science
University of Texas at Dallas
Richardson, TX, USA
{pengli, jayab, ilyen}@utdallas.edu

Ann Tai
IA Tech, Inc.
10501 Kinnard Ave.
Los Angeles, CA 90024
a.t.tai@ieee.org

Abstract

In this paper, we discuss a multi-criteria transaction model to address problems encountered in large-scale E-commerce transaction processing systems. A multi-criteria transaction (MCT) is a generalized transaction where users can specify their preferences in a request by specifying multiple criteria in the transaction. With each criteria specification, a set of alternate execution paths is included. Thus, an MCT is less likely to fail. Moreover, the MCT model allows easy extraction of criteria from a set of transactions and, hence, optimal decisions can be made to achieve better customer satisfaction. We discuss the design of a MCT processing system and compare the MCT model to conventional transaction processing approaches.

1 Introduction

The growth of the Internet and the World Wide Web is making a profound impact on the business world. Currently, the radical shift from conventional business operations to E-commerce is rapidly taking shape. Transaction processing plays a major role in E-commerce applications. Many major companies are developing new transaction technologies to support E-commerce [5] [7]. Due to the high connectivity of the Internet, on-line transaction processing systems are anticipated to deal with highly concurrent accesses that involve multiple database systems. Also, with the sophistication in multimedia technology, future E-commerce customers will be provided with multimedia tools that closely visualize the state of the databases. Thus, the characteristics of the new generation of transaction processing systems will be highly concurrent, highly interactive, and global.

In 90s, research in transaction processing focused on efficient execution of high volume accesses in multiple database systems. The issue in concurrency control in multi-database systems is to achieve the global atomicity while preserving local database autonomy. Conventional transaction processing requires the satisfaction of rigid serializability constraints [1]. The corresponding model for multi-database systems is global consistency. Due to the strong consistency requirement, global consistency has the potential of degraded concurrency, and hence, higher overhead. Several weaker serializability models have been proposed to increase the concurrency for global transaction processing, including two-level serializability model [8][2] and quasi serializability model [3]. Weaker consistency models can increase the degree of concurrency; however, detailed understanding of data dependencies and invariant in each transaction is required in order to take advantage of the models. Thus, transaction programming can be difficult and prone to errors. In addition to the consistency models, algorithms for achieving concurrency control have been proposed. Examples include the site-locking algorithm [6] and ticketing algorithm [4].

Though concurrency control issue for multi-database systems is an important aspect for large-scale E-commerce applications, there are several other problems that need to be addressed. Consider a system where customers are provided the current view of the system state and, based on that, a customer may issue a transaction. Due to the high degree of concurrency, the system state can change when the transaction is issued to the servers. Thus, there is a high probability that the transaction may fail. For example, consider selling tickets for an event. A customer X may want to purchase tickets for some available seats a, b, and c, and X does not mind to get seats d, e, and f, in case seats a, b, and c are not available. When X makes the request, seats a, b, and c are available, so, X submits the transaction for purchasing tickets for seats a, b, and c. Other customers may, at the same time, issue transactions to get seats a, b, and c and obtained those seats before X's request is processed. Now, X is notified that his transaction has failed. However, due to the delay in processing the first transaction, X's second preference, seats d, e, and f are no longer available.

The flexible transaction approach provides a partial solution to the problem discussed above [8][9]. Flexible transactions specify alternative sets of subtransactions. If a more preferable set of subtransactions cannot be committed, the next preferable set of subtransactions will be executed. Semi-atomicity (a weak atomicity requirement) for flexible transactions is defined in [9]. Due to the possibility of executing alternative sets of subtransactions, a flexible transaction is more resilient, i.e., less likely to fail. A site failure will cause one set of subtransactions to abort while still allowing the
transaction to commit. However, two disadvantages exist in this model. First, users must specify each alternative set of subtransactions and their relations. In addition, all of precedence predicates, execution paths, and acceptable states of the transaction should be explicitly specified [8]. In contrast, a criteria-based scheme can greatly ease the specification of alternative sets of subtransactions. Also, as with the conventional transaction processing paradigm, the flexible transaction model can face the problem of poor customer satisfaction. For example, consider the ticket-selling example. Assume that there is only one row, row r, with 10 consecutive seats available, while there are quite a few rows with 5 consecutive-seats. Customer X requests for 10 consecutive seats and customer Y requests for 5 consecutive seats. If Y's transaction arrives first, then it is possible that the system assigns the seats in row r to Y, especially if Y put row r as its first preference in a flexible transaction. When X's transaction arrives, there will be no 10 consecutive seats available to satisfy its request.

In this paper, we discuss a new transaction processing paradigm, namely, multi-criteria transaction processing. A multi-criteria transaction (MCT) is a generalized transaction where users can specify their preferences in a request by specifying multiple criteria in the transaction. The criteria are defined on the data items the transaction operates on. With each criteria specification, a set of alternate execution paths is included. Thus, as with the flexible transactions, an MCT is less likely to fail. Moreover, the MCT model allows easy extraction of criteria from a set of transactions and, hence, optimal decisions can be made to achieve better customer satisfaction.

This paper is organized as follows. In section 2, we give the definition for the multi-criteria transactions (MCT) and discuss the language constructs for MCT specification. In Section 3, we discuss the architecture and design for multi-criteria transaction processing in a single database environment. Section 4 discusses experiments we conducted to evaluate the MCT transaction model. Section 5 states the conclusion of the paper.

2 Multi-criteria Transactions

Let T denotes a transaction and T operates on a set of data items D. T is a multi-criteria transaction if T contains a multi-criteria C that specifies multiple criteria for data items in D and the priorities of the criteria. We have

\[ C = C_1 \text{ or-else } C_2 \text{ or-else } ... \text{ or-else } C_n \]

Where C_i implies C_j for i < j and C_i for 1 ≤ i ≤ n, is a Boolean expression. During transaction processing, the system tries to satisfy C_1 first and when it cannot be satisfied, the system tries to satisfy C_2, and so on.

Concurrent multi-criteria transactions accessing overlapping sets of data objects are processed collectively to maximize the overall customer satisfaction. Consider a group of transactions T_1, T_2, ..., T_n, arrive to the system within a time period t, where t is a reasonably short time period. Let C denote the multi-criteria for T_i. When processing transactions T_1, T_2, ..., T_n, the criteria C, C_2, ..., C_n can be extracted and decisions can be made to satisfy C, C_2, ..., C_n and to allow most number of transactions to succeed for their most desirable criteria possible.

We extend SQL for multi-criteria transaction specification. Two new constructs, namely, select-one-set and or-else, are provided. Construct "select-one-set" is similar to "select" in SQL. As with select, select-one-set construct consists of the keyword select-one-set and following the keyword is the list of data items to be selected. The criteria for selecting the data items, namely, C, is given in the "where" statement (following the keyword "where" defined in SQL). However, select statement returns all the data items that satisfy C while select-one-set returns only one data item that satisfies C. The or-else construct specifies the priority of the criteria as discussed above.

Here we use an example to illustrate the multi-criteria transaction. Assume that a user who wishes to purchase tickets to a sport stadium has the following preferences: He wants to get two consecutive seats in Section 1 of the stadium, between row 5 through 10. If this first criterion cannot be satisfied, then he would like to get two consecutive seats in the first 10 rows of section 2 and the price of the purchase should be less than $80 in total. Assume that there is a table Seats in the database for the stadium seats management, where Seats is defined as Seats (id, section, row_id, column_id, price, flag, owner). A seat s in table Seats has 6 fields. s.id is the seat id. s.section, s.row_id, and s.column_id are the section number, row number, and column number of the seat, respectively. s.price is the price for purchasing the ticket for s. s.flag indicates if the ticket for s is sold. Finally, s.owner is the information of the person who purchased the ticket for s. The transaction specification for the example is given as follows.

1. Select-one-set t1.id, t2.id
2. from seats t1, seats t2
3. where (t1.row_id = t2.row_id)
4. and t1.section = t2.section
5. and t2.column_id = t1.column_id + 1
6. and t1.flag = 0 and t2.flag = 0
7. and t1.price + t2.price < 80
8. and t1.section = 1
9. and t1.row_id between 5 and 10
10. or-else (t1.row_id = t2.row_id)
11. and t1.section = t2.section
12. and t2.column_id = t1.column_id + 1
From line 3 to 9, the first criterion (the most preferable one) is specified. The second criterion is specified in lines 10 through 15. The requirement of two consecutive seats is specified in lines 5 and repeated in the second criterion. "flag = 0" in lines 6 and 13 states the requirement that the selected seats should not be occupied. In line 17 and 18, the database is updated if the seat selection has been successful.

As we can see, MCT approach can help improve customer satisfaction as well as number of satisfied customers. This is achieved not only due to the decision making process for a group of transactions, but also due to the multiple criteria specification in MCT. The user can specify all preferences in a single transaction, instead of having to send alternative transactions to the database, which can cause unnecessary delay and result in losing the time for obtaining the alternative items. MCT has the potential of reducing the network traffic and improving the systems performance due to the same reasons.

3 Multi-criteria Transaction Processing

To allow the database to operate autonomously, we use a multi-criteria transaction processing agent (MTA) to handle MCTs. All MCT transactions issued by the clients are forwarded to the MTA first. The MTA processes them, makes access decisions to satisfy most transactions with the highest possible criteria level, and generates database transactions and interacts with the database to finalize the transaction processing. Figure 1 (a) illustrates the system architecture of multi-criteria transaction processing in a single database environment. Client agent interacts with users and generates transactions, and it sends multi-criteria transactions to MTA and normal transactions directly to the database.

MTA is composed of a request analyzer, a decision-maker, a database state manager, and a group of executors. Figure 1(b) illustrates the architecture of the MTA. The request analyzer, analyzes users requests, extracts the data items to be accessed and criteria specifications. The analysis results of multiple transactions are submitted to the decision maker, which decides how to satisfy the cumulative criteria in order to maximize the satisfaction level of most users. The decision is based on the state information maintained by the database state manager as well as the user criteria. The following subsections discuss the design of each of the MTA components.

3.1 Request Analyzer

For each request issued to MTA, the request analyzer (RA) interprets the request and extracts its criteria and the priority among them. Then, all of these requests will be categorized into different groups based on the data items they need to access. Let $T_1, T_2, ..., T_n$ denote a group of $n$ transactions to be processed by the MTA together. Let $C^i$ denote the multi-criteria for $T_i$ and $D^j$ denote the set of data items accessed by $T_i$. First, the RA extracts $C^1, C^2, ..., C^n$ from $T_1, T_2, ..., T_n$. Then, the RA analyzes $T_1, T_2, ..., T_n$ and obtain $D^1, D^2, ..., D^n$. Finally, RA partitions $T_1, T_2, ..., T_n$ into groups according to $D^1, D^2, ..., D^n$. The algorithm for the partitioning is given below. In this algorithm, $TG$ denotes the set of groups of transactions obtained after partitioning and $TG_i$ denotes the $i$th transaction group. $G^i_{temp}$ is a temporary variable, which is the set of all data items accessed by the transactions in $TG_i$.

\[
g = 1; \quad TG_1 = \{T_1\}; \quad G^1_{temp} = D^1; \quad TG = \{TG_1\};
\]
\[
\text{for } i = 2 \text{ to } n \text{ do}
\]
\[
\text{newgroup = true;}
\]
\[
\text{for } j = 1 \text{ to } g \text{ do}
\]
\[
\text{if } D^j \cap G^i_{temp} \neq \emptyset \text{ then}
\]
\[
TG_j = TG_j \cup \{T_i\}; \quad G^i_{temp} = G^i_{temp} \cup D^j;
\]
\[
\text{newgroup = false; break;}
\]
After partitioning, RA goes through the transactions in each group $T_{Gi}$, for some $i$, to merge the criteria of the transactions. A major step for criteria merging is to resolve the naming conflicts. The merged criteria, $MC^i$, for group $T_{Gi}$, for all $i$, are obtained after merging.

### 3.2 Database State Manager

The database state manager (DSM) maintains the telescopic state information of the underlying database system (or systems in the case of multi-databases). The telescopic state information is less precise and requires less space to maintain. Consider one table, $TB$, in a relational database. Let $TB_{fi}$ denote the $i$th field of $TB$. Also, let $D = \{d_1, d_2, ..., d_m\}$ denote the $m$ data entries in $TB$. The telescopic state information of $TB$ is denoted as $TSF_{TB}$. To obtain $TSF_{TB}$, we first compute the set of data groups $DG$ by applying a grouping function on $D$, i.e.,

$$DG = G(D) = \{dg_1, dg_2, ..., dg_n\},$$

where $G$ is the grouping function and the $i$th data group $dg_i = \{d'_j \mid d'_j = G(d_j) \text{ for all } j\}$. Grouping can be fine-grained or coarse-grained. In a fine-grained grouping, each data group has a smaller number of data items and $DG$ contains more data groups. Thus, the summary state information maintained by the DSM has a higher precision but requires more storage space. In a coarse-grained grouping, each data group has a larger number of data items and $DG$ contains less number of data groups. Subsequently, the state information in DSM has lower precision but requires less storage space. Exact grouping function is application and design specific. Consider the ticket-selling example. In the original database, each seat is a separate data entry. In DSM, we can have the original grouping, each seat is in a group. This is obviously undesirable since in this case, DSM is a replica of the database. We can also group all the seats in one row of a section into a group. The grouping precision can affect the decision making at a later stage.

After obtaining $DG$, we compute the telescopic state information for each data group $dg_i$ in $DG$. Let $TSF_{dg_i}$ denote the telescopic state information for data group $dg_i$. We have

$$TSF_{dg_i} = (\Phi_1(d'_1f_1, d'_2f_1, ..., d'_mf_1), \Phi_2(d'_1f_2, d'_2f_2, ..., d'_nf_2), ..., \Phi_i(d'_1f_i, d'_2f_i, ..., d'_nf_i)).$$

Here, $mi$ denotes the size of $dg_i$ and $k$ denotes the number of fields. $\Phi_i$ is the summary function for field $f_i$ over all the data items in $dg_i$. If a field $f_i$ is not considered in the telescopic state information space, then $\Phi_i(x) = null$, for any $x$, i.e., the field is eliminated from the summary information. The summary function $\Phi_i$ is application dependent.

Consider the ticket-selling example discussed in Section 2. Let $D = \{S_{ijk} \mid \text{for all } i, j, k\}$ denote all the seats in a stadium. Seat $S_{ijk}$ is a seat in Section $i$, row $j$, with a number $k$. Let $DG = G(D) = \{dg_{i1}, dg_{i2}, ..., dg_{im}\}$, where $G$ is the grouping function which groups all the seats in the same row of the same section into a data group and $dg_{ij} = \{S_{ijk} \mid \text{for all } k\}$, for all $i, j$, are the data groups we obtained after grouping. The summary function, $\Phi_i$ for computing the telescopic information $TSF_{dg_{ij}}$ is given as follows. Note that the fields for seat $S_{ijk}$ is the same as those defined in Section 2. The mapping function for obtaining $TSF_{dg_{ij}}$ can be expressed as the set of field mapping functions,

$$\{\Phi_{df}(dg), \Phi_{section}(dg), \Phi_{row_id}(dg), \Phi_{row_num}(dg)\}$$

where $\Phi_{df}(x) = null, \Phi_{column_id}(x) = \Phi_{row_num}(x) = \Phi_{section}(x) = \Phi_{row_id}(x) = null$, for all $x$. Each data group (i.e., each row) keeps the section number and row number for identification and the values can be obtained from the first seat in the data group.

$$dg_{ij}.section = \Phi_{section}(dg_{ij}) = S_{ij}.section,$$

where $S_{ij} \in dg_{ij}$ and $dg_{ij}.row_id = \Phi_{row_id}(dg_{ij}) = S_{ij}.row_id$, and $S_{ij} \in dg_{ij}$. The mapping function for field $flag$ is more complicated. The field $flag$ of a row of seats is mapped to a bag of numbers, where each number is the length of consecutive seats that are available.

$$dg_{ij}.flag = \Phi_{flag}(dg_{ij}) = \{X_{fp} \mid \text{for all } p\}$$

where

$$X_{fp} = \sum_{k=0}^{o(p+1)} f(S_{ijk}.flag) \text{ where } S_{ijk} \in dg_{ij},$$

$$f(S_{ijk}.flag) = \begin{cases} 0 & \text{if } S_{ijk}.flag = \text{occupied} \\ 1 & \text{if } S_{ijk}.flag = \text{free} \end{cases}$$

and $\exists S_{ijk}$ s.t. $o(p) \leq k < o(p+1)$, $\forall S_{ijk}.flag = \text{occupied}$, $\forall S_{ijk}.flag = \text{free}$ for $i \geq 0$.

Where $S_{ij}.flag = \text{occupied}$, $S_{ij}.flag = \text{free}$

Assume that there are 5 consecutive free seats and separately another 2 consecutive free seats in row 1, section A, then we have $x_1 = 5$, $x_2 = 2$, and $dg_{ij}.flag = \{5, 2\}$.

The overall telescopic information $TSF_{TB}$ for table $TB$ is the set of the telescopic information of all data groups derived from $TB$, i.e., $TSF_{TB} = \{TSF_{dg_{ij}} \mid \text{for all } i, dg_{ij} \in DG\}$. $TSF_{TB}$ can be updated either periodically or when needed. The DSM queries the database to obtain the updated raw information and then computes $TSF_{TB}$. 

599
3.3 Decision-Maker

The decision-maker (DMK) narrows down the constraints specified for the data items in multi-criteria transactions and decides specific data items to be accessed for each transaction. RA sends the transaction groups TG and the merged criteria MC for each group TG, to DSM. Based on MC and the state information maintained by DSM, DMK makes the decisions. There are many decision-making strategies such as integer linear programming techniques, branch and bound search, etc., that can be used by the DMK. In our approach, we map the data space to a continuous integer grid space and use integer linear programming to find the solution that satisfies the corresponding set of criteria.

Let TG denote the transaction group we consider and MC is the merged criteria for TG. Assume that TG = {T1, T2, ..., Tn} and MC = C1 ∩ C2 ∩ ... ∩ Cn, where Cj is the original multi-criteria for transaction Tj. Cj can be expressed more specifically as

Cj = Cj,0 or Cj,1 or Cj,2 or ... or Cj,mj.

Our algorithm can be divided into two phases. In the first phase, an appropriate criteria level is selected for each transaction. Since the level 1 criterion is the most important, initially, Cj,0 for all j is selected. Each Cj,0 is tested individually for its satisfiability given the system state TSI maintained by DSM. If Cj,0, for some j, cannot be satisfied, then the level 2 criterion, Cj,1, is selected. The selection process terminates until all Cj,0, for all j, are selected, where Cj,0 is satisfiable given TSI, while Cj,−1, is not. Note that we only consider the satisfiability of individual requests. When these requests are considered together (collectively processed), some criteria may no longer be satisfiable. We can either go to the second phase to solve the set of criteria from all transactions or use additional checks to check whether the combined criteria is satisfiable. Let BC denote the combination of the criteria from all transactions in TG, where BC = C1,n ∩ C2,n ∩ ... ∩ Cn,n, and li is the selected criteria level for the ith transaction. Specific checking scheme for examining combined criteria is application dependent. Here, we consider one general approach that can be applied to many applications. We compare the number of data items requested by the transactions and the number of data items available. Let DG denote the set of data groups whose state information is maintained in TSI. From Cj,0, we can determine DGj = {dgj, l for all xj}, which is the set of data items involved in Tj. Also, from Tj itself, the set of data items, Dij, accessed by Tj can be determined. The criteria evaluation procedure is given in the following.

1. for j = 1 to n do
   determine lj, such that lj is minimum and Cj,0 is satisfiable;
   determine DGj from Cj,0.

end for;

2. DG = ∪j=1,n DGj = {dgj, l for all xj};
   if (Σj |DGj| > Σj |dgj| fj) then
     determine y, where fy = minj=1,n fj
     and-then |DG| = maxj=1,n |DGj|;
     lj = lj + 1;
     determine the new DGj from Cj,0;
     go back to step 2;
   else find a solution for BC = C1 ∩ C2 ∩ ... ∩ Cn;
     if no solution exists then
       determine y, where fy = minj=1,n fj
       and-then |DG| = maxj=1,n |DGj|;
       lj = lj + 1;
       determine the new DGj from Cj,0;
       go back to step 2;
   end if;
end if;

Here, we assume that the number of available data items in each dgj, for some xj, is maintained in TSI by field fj. Thus, the total number of data items available can be computed easily and efficiently. If the condition is not satisfied, then there definitely exists no solution for BC. So, we select a higher criteria level for a selected transaction Tj. Otherwise, we can go ahead to try to find the solution for BC. Tj can be selected in various ways. In the above procedure, we select Tj, whose criteria level being considered is the lowest. If multiple transactions has the same low criteria level being considered, then the Tj with the maximum number of data items involved in it is selected.

In the second phase, the combination of selected criteria BC for transaction group TG is evaluated. We first convert BC into disjunctive normal form. BC, after conversion, can be expressed as

BC = (BC1 ∩ BC2 ∩ ... ∩ BCn) ∨ (BC1 ∩ BC2 ∩ ... ∩ BCn) ∨ ...

Each BCj is an integer inequality and a clause in BC is a set of integer inequalities that can be solved by integer linear programming technique. Since there are multiple clauses in BC, the clauses can be evaluated sequentially until a solution is obtained for one of the clauses.

3.4 Executor

After the DMK completed its decision-making procedure, it sends the MCTs with their final sets of selected data items to the executor pool. When an executor becomes idle, it fetches an MCT from the queue and processes it. The executor generates regular database transactions for the MCT. If the keyword “select-one-set” appears in the MCT, a named cursor (a cursor is a named resource available to a program, which can be specifically used for parsing SQL statements embedded within the application) that just fetches specific data items will be added. After sending out the transaction, the executor waits for the returned results from the database. If the transaction fails, the corresponding MCT will be sent back.
to the DMK. Then the DMK will decide a new set of data items for the returned transaction. The decision will be made with a new group of transactions that arrived more recently. To avoid an MCT from failing continuously, a priority scheme can be used and the previously aborted MCTs could be given a higher priority in terms of assigning the satisfactory data items.

4 Experimental Study

We have conducted preliminary experiments to evaluate the performance of the MCT model and compare it with the traditional transaction model. The simulation is conducted on a single database system. We choose ticket sales, for a large sports stadium, as a case study for our simulation. The stadium consists of 50,000 seats divided into 25 sections. Each section consists of 100 rows with 20 seats in each row. The information for the stadium and all the seats are stored in an Oracle 8i database. The simulation system consists of one server and multiple clients. Clients communicate with the server using the HTTP protocol and the server interacts with the database through JDBC. The clients are simulated by a client program, which issues MCT and conventional transactions to the server continuously. The inter-arrival rate of the transactions generated by the client program follows a Poisson distribution.

The client program generates synthetic transactions to simulate multiple clients. Each MCT is specified by the following parameters: number of seats desired, number of continuous seats desired, section preference (indicates the preferred range of sections), row preference (indicates the preferred range of rows) and price range (the price limit for the purchase). In the case of conventional transactions the parameters would be: number of seats desired, number of continuous seats desired, specific section numbers for the seats, specific row numbers for the seats, and specific seat numbers.

The client program keeps a view of the database and refreshes it periodically. This is to simulate the behavior that the clients may get out of date information when constructing their transactions or due to data caching. The time between two successive refreshes is called “View Refresh Time”. Clients issue transaction based on their current view of the database.

Each of the MCT parameters in our experiments is generated as follows: The number of seats and number of continuous seats desired is randomly selected, from a range, based on a uniform random variable. To generate section range and row range preferences, a row and section are chosen such that they have a possibility of satisfying the seating requirement and as close to the front as possible. This is done to ensure that the regions at the front have a higher chance of being selected than the ones at the back, which simulates the real world behavior of a client preferring a front row to the one at the back. Then a number indicating the range rows is generated as a uniform random variable. The price preference can be set to be from the price range of the selected region.

In the case of conventional transactions, the numbers of seats and number of continuous seats are generated as uniform random variables within a specific range. Since a client would prefer the front rows, we start from the first row and try each section to find a match that will satisfy the seating constraints. On finding such a row-section pair, a conventional transaction with these parameters is issued. Otherwise the search is continued to the next row.

We designed two experimental setups to evaluate the performance of the MCT model and compare to that of the normal transactions:

1. Clients send normal transactions. This experiment is intended to simulate the situation where a client wishes to get one set of the desirable data items that satisfy his needs. All the alternative sets of data items are expressed in terms of multi-criteria. The client first randomly generates the multiple criteria. Based on the state of the database, the client analyzes the criteria and generates a normal transaction that obtains the set of data items satisfying the criteria. If the transaction fails, the client generates another transaction to obtain an alternate set of data items that satisfy the criteria. This continues till one transaction succeeds.

2. Clients send MCTs. In this case, the clients generate MCTs and send it directly to the server. The server process the MCTs by the mechanisms discussed in this paper. This experiment measures the potential gains of the MCT model.

The performance metric used for the purpose of comparison is Mean Extra Time required to succeed. We calculate this time $T$ by

$$T = \frac{TTSu + ATuT}{NoT}$$

where $TTSu$ is the Time Taken by a Transaction (either MCT or Traditional) to succeed, $ATuT$ is the Average Turnaround Time, and $NoT$ is the Total Number of Transaction Issued.

Figure 2 and 3 show the performance results and compare the behavior of normal transactions and MCTs at various transaction rates, for view refresh times 1000 and 500 msec respectively.

There is an exponential increase in $T$ for normal transactions. This could be attributed to the fact that at higher transaction rates more clients are likely to see the same view of the database and thus potentially issue requests for intersecting choice sets, increasing the failure rate of transaction. Also from the figures, we can see that
the higher the view refresh times, the larger the values of $T$. The reason for this is that as view refresh time increases, more clients issue transactions based on a staler view of the database hence increasing the likelihood of failure.

In general the $T$ values for traditional transactions are way above that of MCT, indicating that MCT are more resilient to failure and yields much better performance.

![Graph showing transaction rate vs. time](image1)

**Figure 2**

![Graph showing transaction rate vs. time](image2)

**Figure 3**

### 5 Conclusion

In this paper, we discussed a new transaction model, multi-criteria transactions, for E-commerce applications. The goal of MCT is to reduce the communication cost due to transaction failure and increase the customer satisfaction. Compare to conventional transactions, MCT allows customers to specify their preferences in advance and, hence, avoiding the consequences due to the delay caused by failed transactions. Compare to flexible transaction, the MCT model possesses all the major advantages of flexible transactions. Also, it solves the problem due to high volume of concurrent accesses, avoids compensation actions required in flexible transactions, yields more efficient transaction processing (due to decision making), provides better customer satisfaction (more client requests may be committed), and yields a higher concurrency for transaction processing.

### 6 Acknowledgement

This research was supported in part by the Texas Advanced Technology Program under Grant No. 009741-0143-1999.

### 7 Bibliography


