Performance Modeling of Nested Transactions in Database Systems

Hossam S. Hassanein
Department of Computing and Information Science
Queen's University
Kingston, Ontario, K7L 3N6 Canada
Hossam@cs.queensu.ca

Mohamed E. El-Sharkawi
Department of Information Systems
Faculty of Computers & Information
Cairo University
Orman, Giza, Egypt
mel_sharkawi@hotmail.com

Abstract

The nested transaction model was introduced to satisfy the requirements of advanced database applications. Moreover, it is currently the basic transaction model for new databases like workflow systems, mobile databases, and object-relational databases. Though there are several performance evaluation studies of different concurrency control mechanisms in nested transactions, effects of transaction parameters on the overall system performance have not received any attention. In this paper, we study the effects of transactions characteristics on system performance. We developed a detailed simulation model and conducted several experiments to measure the impact of transactions characteristics on the performance. First, the effect of the number of leaves on the performance of nested transactions is investigated under different shaping parameters. Also, effects of the depth of the transaction tree on the system performance are investigated.

Key words: Nested Transactions, Two-Phase locking, Performance Evaluation, Simulation.

1. Introduction

Since the mid 80's database applications have changed from the traditional record-keeping applications to more advanced and complex applications, like computer aided design (CAD) systems and office information systems. The nested transaction model [15], originally introduced to increase transaction reliability in distributed systems, proved to be more appropriate for these new applications. Transactions in advanced database applications are characterized of being long and complex [11, 12]. The conventional flat transaction model cannot support all the requirements of these advanced applications [3, 4, 10, 20]. The basic nested transaction model, [15], has been extended and several new models have been introduced. The nested transaction model and its extensions are currently the basic models for several advanced database systems. The transaction model underlying workflow systems is the nested transaction model [19]. As well, transactions in mobile databases are nested transactions [5, 14]. At the data model level, transactions in the object-oriented model are nested. A method applied on an object may invoke another method constituting a hierarchy of method invocations [8].

A nested transaction is a hierarchy of subtransactions, where each subtransaction may contain other subtransactions, or contain the atomic database operations read and write. In other words, a nested transaction is a collection of subtransactions that are composed together to form one whole atomic unit of execution [9, 15]. A nested transaction is represented as a tree, called transaction tree. In the basic nested transaction model and the basic two-phase locking [15], only leaves of the transaction tree can perform read and write operations. Each leaf subtransaction is viewed as a normal flat transaction in the system. The leaves are executed independently. Inner subtransactions cannot request any lock on any data item. Non-leaf subtransactions only organize the control flow and determine when to invoke subtransactions.

To understand the functionality and behavior of advanced database systems and applications it is crucial to get deep understanding of different parameters that affect the performance of systems based on nested transactions. In a recent survey on the advancement in nested transactions research, [13], it is pointed out that the impact of simulation parameters on the performance of nested transaction concurrency control mechanisms has not received enough research attention. In this paper we investigate the performance effects of the transaction shaping characteristics on the performance of the two-phase locking mechanism. The examined characteristics are the number of levels and the number of leaves in a nested transaction..

It can be seen that the leaves are the main parts in the basic nested transaction model, since they request locks on and process database items. Hence, it is expected that the performance effect of the number of leaves is significant. The number of internal subtransactions in a nested transaction depends mainly on the depth of the transaction. Thus, we also expect that the number of levels affects the performance of nested transactions. In this paper we aim to study the performance effects of these two shaping characteristics. The importance of defining the performance effects of these parameters is to give designers of nested transactions some hints in order to improve the system performance. We present a detailed simulation model and results of several experiments that measure the effects of these transaction parameters. It is shown that increasing the number of leaves improves the system's performance as it increases the concurrency level in the system. Though, however, for high level of data contention there is a limit where increasing the number of leaves beyond this limit may cause performance degradation. It is also shown that at high multi-programming levels, the effect of the number of levels on system's performance is insignificant. Details of these results are in Section 4.

The paper is organized as follows. The next section introduces the basic nested transaction model and the two-phase locking mechanism for nested transactions. Section 3 introduces the performance model, which includes database model, the transaction model, the system model, and the experimental settings are introduced. Section 4 gives simulation results of measuring the different transaction characteristics on system's performance. Section 5 is conclusion and future work.

2. The Nested Transaction Model

In this section, we review the basic nested transaction model, as well as the basic two-phase locking concurrency control mechanism for nested transactions as introduced by Moss [15].

2.1 Basic Concepts

A nested transaction is represented as a tree structure where the root of the tree is the main transaction consisting of a number of subtransactions. Each subtransaction can in turn be a nested transaction. Only the leaves of the transaction can access data objects. That is, only leaves can ask for locks on data objects. The root of a transaction tree is assumed to be at level 1. The distance between two vertices i and j in a tree, $d(i, j)$, is defined as the number of edges in the path from i to j. The level of a node n is the distance $d(r, n) + 1$, where r is the root of the tree. The ancestors of a node n are the set of vertices in the path from n to the root. The parent, an, of a node n is the ancestor of n such that $d(an, n)$ is equal to one.

2.2 Two-Phase Locking for Nested Transactions

The work in [15] extended the traditional two-phase locking mechanism to synchronize nested transactions. The mechanism can be summarized as follows:

1. A transaction may hold a lock in WRITE mode if all other transactions holding the lock (in any mode) are ancestors of the requesting transaction.
2. A transaction may hold a lock in READ mode if all transactions holding the lock in write mode are ancestors of the requesting transaction.
3. When a transaction aborts, all its locks, both read and write, are discarded. If any of its

ancestors hold the same lock, they continue to do so, in the same mode as before abort.

4. When a transaction commits, all its locks, both read and write, are inherited by its parent (if any). That is, the parent holds each of the locks, in the same mode as the child held them.

Deadlocks may arise when using the basic two-phase locking mechanism. In case of nested transactions two types of deadlocks may occur, intra-transaction deadlocks and inter-transaction deadlocks. Inter-transaction deadlocks arise between two nested transactions (a situation similar to deadlocks in flat transactions) and intra-transaction deadlocks arise between subtransactions in the same nested transaction.

3 Performance Model

The performance model of a nested transaction system has three components: the database model, the user transactions model, and the system model.

3.1 The database model

The database is modeled as a group of data items where the locking granularity is a page. A data item is the unit of accessing or processing. The database is of size *db_size*, which is a simulation parameter. The data items are chosen uniformly between 1 to *db_size*. In all experiments there is no replacement in choosing items for every subtransaction leaf. The database model is centralized.

3.2 The user transaction model

Users are modeled by a fixed number of terminals that generate transaction trees. The time between the completion of one transaction tree and the submission of another transaction tree from the same terminal is exponentially distributed with mean *trans_ext_think_time*.

The leaves of an active transaction tree arrive independently to the system with respect to the time at which their transaction tree became active. The time between the arrival of any leaf to the system and the activation of its transaction tree is exponentially distributed with mean *leaf_ext_think_time*. This scenario is suitable for modeling nested transactions in computer-aided design systems where designers generate their transactions independently. Moreover, in such systems subtransactions in the same transaction tree do not arrive to the system at the same time. Note that if the value of the parameter *leaf_ext_think_time* is zero all the leaves of the transaction tree arrive to the system at the same time, though, however, they are granted the locks in a random and independent order.

A terminal can only have one pending transaction tree. However, it can have more than one pending leaf that belongs to its transaction tree in the system. The shape of the generated transaction tree is determined by the simulation parameters shown in Table 1. The model is capable of generating transaction trees with same or different shapes. The shapes of the trees can be specific or random. This is determined according to the input shape related parameters.

Shape Parameter	Meaning
max_levels	maximum depth of a generated transaction tree
min_levels	minimum depth of a generated transaction tree
max_child_node	maximum number of children per a non leaf node
min_child_node	minimum number of children per a non leaf node

Table 1 The Transaction Shape Simulation Parameters

3.3 The system model

The system model consists of the disk and CPU. Service times at the disk and CPU are deterministic and are given by *io_obj* for reading or writing a data item and *cpu_obj* for processing a data item. An item needs to be read from the disk if it is not held by a superior of the requesting leaf. The *cpu_obj* time is needed to process an item locked in read or write mode. An item locked in write mode needs time *io_obj* to write the deferred update at commit time.

The simulation model, shown in Figure 1, is based on the ones in [1, 2] with special provisions to accommodate the implementation of the nested two-phase locking (N2PL) protocol. The model represents a closed queuing system of a centralized database management system. The model consists of N terminals, which represent the users that generate the trees. The maximum number of active transaction trees (multiprogramming level) allowed in the system is defined by simulation parameter *max_mpl*. A transaction tree is considered to be active if it has at least one active leaf . A subtransaction is considered to be active if it is receiving service, blocked, or waiting in the queues of the scheduler or disk. We do not control the number of active leaves in the system. The parameter *max_mpl* along with other parameters such *trans_ext_think_time, trans_size,* and *leaf_ext_think_time* control the system load. When a transaction tree is generated from a terminal and the number of active transaction trees in the system is *max_mpl*, the new transaction tree is delayed in the ready queue waiting for an active transaction tree to commit.

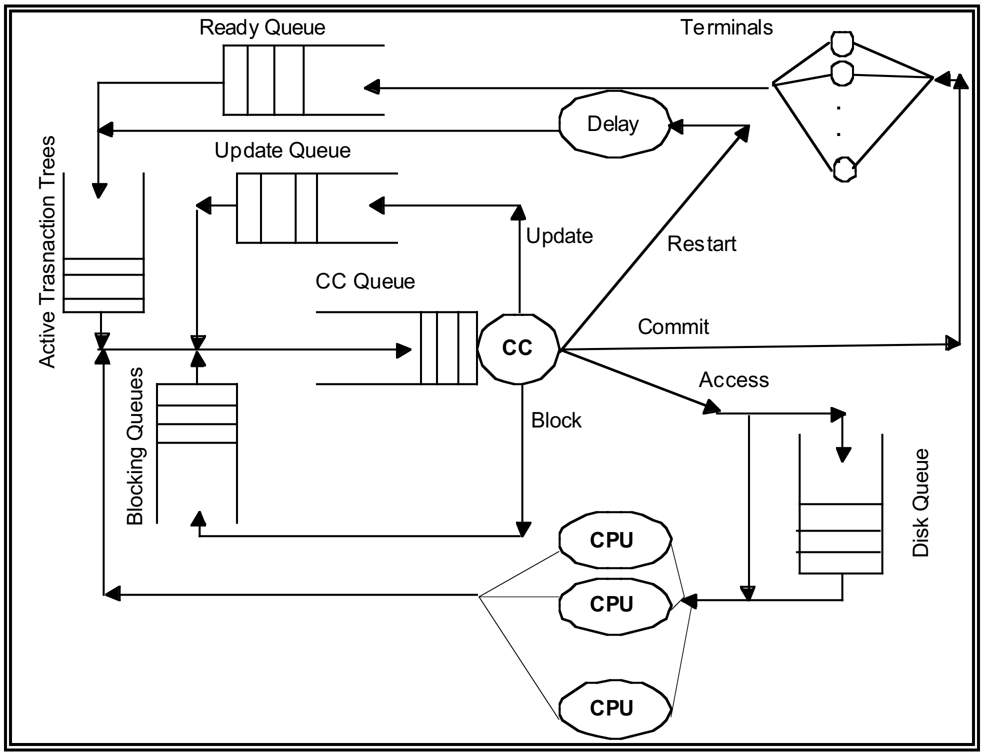

Figure 1 The Logical Queuing Model

When a transaction tree becomes active, its leaves enter the system randomly and independently with rate *1/leaf_ext_think_time* with respect to the arrival of the transaction tree. A new leaf in the system starts its first concurrency control request at the scheduler. The leaves make their lock requests on an item-by-item basis. According to this method, when a lock is granted and the data item is processed, the leaf can request its next item after a mean time *int_req_think_time*. The scheduler or the concurrency control manager is responsible for granting or blocking the requests according to the N2PL concurrency control protocol and the status of the database items. If the request is blocked, it is inserted in the waiting list of the requested item. The blocked leaf waits until it is aborted due to deadlock resolution or is granted the lock. A blocked leaf in a transaction tree may be granted the lock due to the commitment of either another transaction tree or a leaf in the same transaction tree. Whenever a request is blocked, the concurrency control manager invokes the deadlock detection procedure.

According to the adopted deadlock detection algorithm, the system maintains a wait-for-graph-for-nested-transactions, where all the waiting relationships between the subtransactions are presented. The system chooses the subtransaction with the minimum number of locks to be aborted to resolve a deadlock situation. It was found in [6] that this victim selection criterion has the best performance among other tested criteria. If a deadlock is detected, it is resolved by aborting certain number of leaves according to the class-based strategy. According to the two-phase locking mechanism all the inferior leaves of an aborted subtransaction are aborted. However, according to the class-based strategy deadlock resolution strategy in [6] unnecessary abortion of leaves is avoided. The idea of this new strategy, is to partition the leaves of each transaction tree into distinct classes. Leaves in any two different classes do not request items in common. In case of a restart, only the inferior leaves that belong to the class of the awaited item are aborted. While other leaves continue their execution.

The aborted leaves release all their items and are restarted after a delay period uniformly distributed between two simulation parameters, namely *min_restart_time* and *max_restart_time*. The reason behind the choice of a uniformly distributed restart delay is to try to avoid, or at least minimize, repeated deadlock cycles or live lock situations. This problem is discussed in [18] where the restart waiting solution was implemented. According to this solution the restart of the aborted transaction is delayed until the conflicting transactions have left the system [7, 17]. The aborted leaves request the same originally items. In other words there are no fake restarts [1]. As a result of aborting the victim subtree a number of blocked subtransactions may be granted required locks and proceed to the CPU or to the disk queue. The detection of deadlocks takes place if a subtransaction commits and none of the leaves waiting for its locks are granted.

Now if a request is granted, and it was not inherited from a superior, then the granted leaf proceeds to the disk queue to access the granted data item. However, if the lock was inherited from a superior, then the leaf has to access the item from the local space of the transaction tree in memory. Therefore, the granted leaf can realize any uncommitted changes internal to the transaction tree. Hence, if a leaf is granted a lock from a superior, it does not enter the disk queue. This is compatible with the CAD model introduced in [11], where the operation of accessing of a data item by a leaf from a superior's local space, Check-Out operation, takes place in the local memory.

After accessing the granted item, the leaf is assigned a CPU to process the item where the service time is *cpu_obj*. When there is infinite number of processors, the leaf is directly assigned a CPU and starts processing the item without any queuing delay. Therefore, the effect of data contention is isolated from resource contention. When a leaf completes processing the item, it is queued at the end of the concurrency control queue to make its next request. This is repeated until all its requested items are processed. At this point if all the subtransactions in the transaction tree completed their operations, the root commits by writing the deferred updates and then releases all the locked items. When a transaction tree from a terminal commits, another transaction tree is generated from the same terminal after an exponentially distributed time according to the rate *1/trans_ext_think_time*. However, if the raising of locks stops at an uncommitted subtransaction, the waiting queues of the raised items are checked for any request that can be granted. If for any raised item there is not any blocked request to be granted, the deadlock detection procedure is invoked. If a deadlock is found, it is resolved. This way the system is capable of detecting deadlocks as early as they occur. Table 2 below summarizes the parameters, which were described in the simulation model.

3.4 Experimental settings

There are two groups of experiments in this paper. The purpose of the first group is to study the performance effect of the number of leaves, while the purpose of the second group is to investigate the effect of the number of levels. The base parameters in the two groups of experiments are shown in Table 3. These values were chosen to provide a moderate data and resource contention. It should be noted that varying the transaction size and the multiprogramming level values lead to different workloads in the system. Note that varying the number of leaves in transaction trees may lead to different workloads if the leaves in each transaction tree are allowed to request common items. For instance, suppose that the transaction size is 8 and the number of leaves is also 8 then there is a possibility that all the leaves in the transaction tree request the same item. Of course, there is also the possibility that the leaves request totally different items. So, the actual size of the whole transaction can be in the range from 1 to 8 database items. Thus, if the leaves in the same transaction tree are allowed to request common items, the data contention workload may not be equivalent. Therefore, leaves in a transaction tree are not allowed to request common items in the experiments in this section. Consequently, the effect of varying the number of leaves in the transaction tree on the performance of nested transactions can be studied in isolation from the values of the shaping parameters will be specified in each experiment.

Variable	Definition
num_terminals	The number of terminals generating transactions
mpl	The multiprogramming level which is the maximum number of active transactions trees allowed in the system
io_obj	The service time for a single data object in disk server
cpu_obj	The service time for a single data object in the CPU server
db_size	The size of the database which is the number of data items
trans_ext_think_time	The mean of the exponential inter-arrival time of transactions trees
leaf_ext_think_time	The mean of the exponential inter-arrival time of leaves in one transaction tree
int_req_think_time	The mean of exponential inter-arrival time of requests by one leaf
read_prob	The probability of read locks
trans_size	The size of the transaction tree which is the number of data items requested by a transaction tree
leaf_size	The size of a leaf subtransaction which is the number of data items requested by the leaf
max_levels	The maximum depth of a transaction tree
min_levels	The minimum depth of a transaction tree
max_child_node	The maximum number of children for non-leaf subtransactions
min_child_node	The minimum number of children for non-leaf subtransactions
min_restart	The minimum time elapsed before an aborted subtransaction is restarted
max_restart	The maximum time elapsed before an aborted subtransaction is restarted

Table 2 The Simulation Parameters

Parameter	Value
db_size	1000
io_obj	0.03
cpu_obj	0.01
read_prob	0.7
trans_size	16, 24
trans_ext_think_time	1
leaf_ext_think_time	1
int_req_think_time	0
num_terminals	70
max_mpl	10-60

Table 3 The base parameters

The performance metrics measured are the average transaction response time, the system throughput, and the restart ratio. The average transaction response time is the time elapsed from the generation of a transaction tree from a terminal until the commitment of the whole transaction tree. The system throughput is the number of committed transaction trees per time unit. The restart ratio is the average number of times a subtransaction in each transaction tree is rolled back as a result of resolving a deadlock situation. The main metric used in this paper is the response time. The simulation program is written in the C programming language and run on Silicon Graphics Machines.

4 Performance Results

This section investigates effects of two factors: the number of levels and the number of leaves of the transaction tree. The results are obtained for medium and large transaction sizes in order to understand the performance effects of the tree shape accurately. The shape of transaction trees in each experiment is specified. This is necessary to study the effect of each shape on the performance of nested transactions. In the following figures the transaction shape is represented as x-y, where x represents the number of levels and y represents the number of leaves.

Different preliminary experiments were conducted for each of the two sets of experiments where leaves are submitted in parallel or at different times. Similar performance patterns were observed for the two cases. Therefore, it suffices to show the results of the general case where the leaves of each transaction tree enter the system at different times. In other words, the interactive scenario is adopted where the rate is not zero. In general, the obtained results for the two transaction sizes 16 and 24 show the same performance trends. The results obtained for transaction size 16 will only be shown if there is a difference in the performance trends.

4.1 Effect of number of leaves

The number of leaves in the transaction tree is an important factor affecting the performance of nested transactions. On one hand, the number of leaves controls the amount of concurrent work. On the other hand, the number of leaves in the system is related to the frequency of deadlock occurrences and hence to the rollback of subtransactions. In this section, various simulation experiments are performed to study effects of the number of leaves on the performance of nested transactions considering different number of tree levels. The experiments consider nested transactions of 1, 2 and 3 levels. Varying the number of levels yields different numbers of internal subtransactions. Recall that the only task of internal subtransactions is to organize the control flow and determine when to invoke subtransactions. The number of leaves in a transaction tree is varied at different number of levels. Note that the leaf size is determined according to the number of leaves in the transaction tree. Table 4 shows the leaf size corresponding to different possible number of leaves.

Number of leaves	Leaf size
2	12
4	6
6	4
8	3
12	2

Table 4 The number of leaves and the corresponding leaf size for transaction size 24

Response time results obtained for transactions of one level are shown in Figure 2. Note the difference in the relative performance of different transaction shapes before and after mpl = 50. Before this multiprogramming level value, any transaction shape outperforms other shapes with lower number of leaves. At mpl = 60, the response time of some transaction shapes become worse than response time of other transaction shapes having less number of leaves. For instance, transactions with 12 leaves outperform transactions with 6 leaves before mpl = 50 by about 23% on average, but at mpl = 60 transactions with 6 leaves outperform transactions with 12 leaves by about 10%. This can be explained by considering the restart ratio results in Figure 3.

At high mpl values, the restart ratio of transactions having 8 or 12 leaves increases dramatically. The significant increase in the restart ratio adversely affects the response time results. Related to the number of leaves in nested transactions, there are three factors that affect the performance:

1. Increasing the number of leaves increases the amount of concurrent work. This should decrease the response time.
2. Increasing the number of leaves decreases the leaf size, which in turn minimizes the effect of restarts.
3. Increasing the number of leaves increases the number of concurrent independent subtransaction requesting locks, which increases the restart ratio and also the response time for high workload contention.

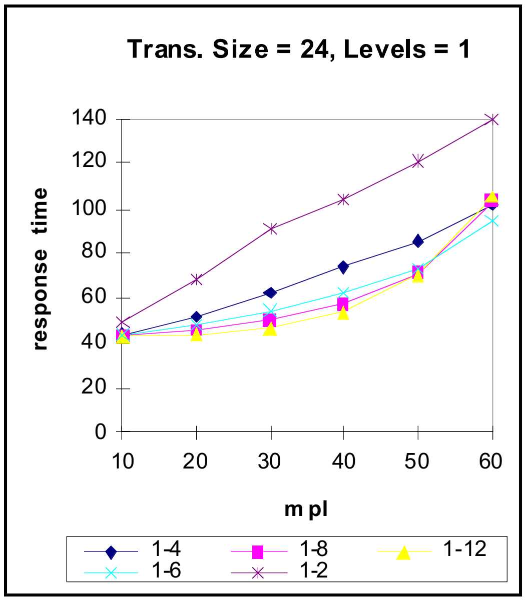

Figure 2 The effect of number of leaves, levels=1, Transaction size =24. (response time)

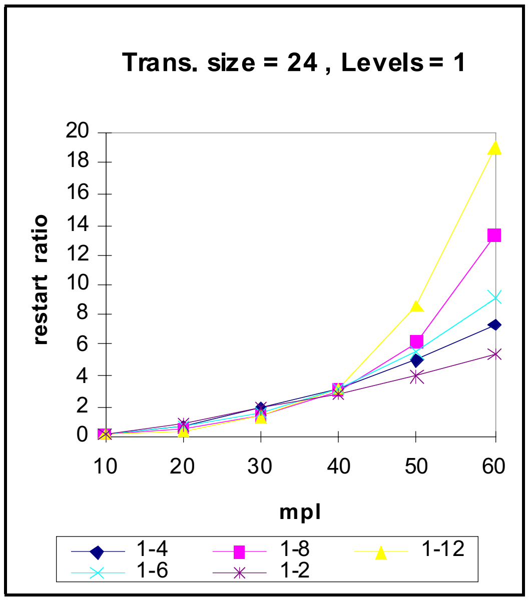

Figure 3 The effect of number of leaves, levels=1, Transaction size =24. (restart ratio)

The impact of the first two factors on the performance of the system is opposite to the impact of the third factor. As the number of leaves is increased, the first two factors enhance the performance of the system while the third factor degrades it. The last factor has little effect on the performance at low values of mpl, as conflicts are rare anyway. At higher mpl, however, the third factor does make an observable difference in the results. Increasing the mpl beyond a certain point with higher number of leaves increases the restart ratio dramatically to the extent that the response time is adversely affected. This can be noted in the performance results of transactions with 8 and 12 leaves. The response time increases significantly beyond mpl = 40 and becomes worse than the response time of other transaction shapes with 4 or 6 leaves.

Figures 4 and 5 illustrate the impact of varying the number of leaves on the response time and the restart ratio, respectively, for different mpl values. Note that for the mpl=40 in Figure 5 increasing the number of leaves decreases the restart ratio, while for mpl=60, increasing the number of leaves significantly increases the restart ratio. This explains the relative performance trends of the response time results change as the number of leaves is increased beyond 6 as shown in Figure 4 for mpl 40 and 60. The adverse effects of increasing the number of leaves are not significant on the performance of medium transactions as the data contention is not high enough. This can be noted in the response time and restart ratio results shown in Figure 6 and Figure 7, respectively, where the transaction size is 16.

Figure 8 and Figure 9 respectively plot the response time and the restart ratio results for transactions of 2 levels. It can be noted that the difference in the response time is not significant between transaction shapes of 4 and 6 leaves and between transaction shapes of 8 and 12 leaves. Unlike transactions of 1 level, the response time of transactions of 2 levels and 8 or 12 leaves remains lower than the response time of transactions of 4 or 6 leaves even at high mpl. This can be explained by noting that the effect of the high restart ratio on the response time is not evident in the results of transactions of 2 levels due to the existence of non-leaf subtransactions. Indeed, increasing the number of leaves in transactions of 2 levels decreases the restart ratio at low mpl and increases the restart ratio with a lower degree than for the 1-level case at high mpl. The final result is that the response time always decreases if the number of leaves is increased. Similar results were observed for transactions of 3 levels.

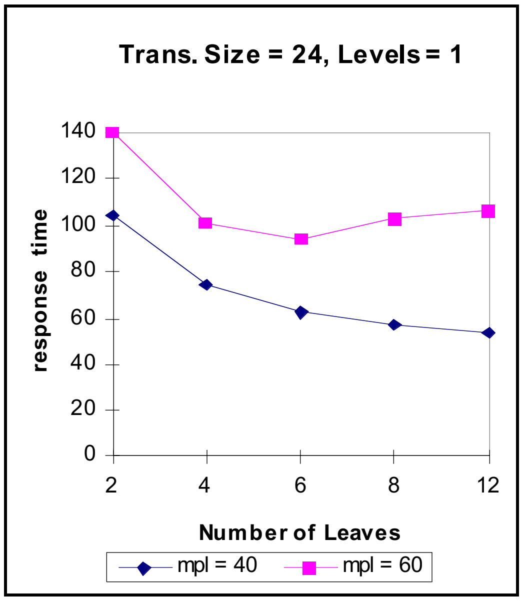

Figure 4 The effect of number of leaves, levels =1, Transactions size = 24, mpl = 40, 60. (response time)

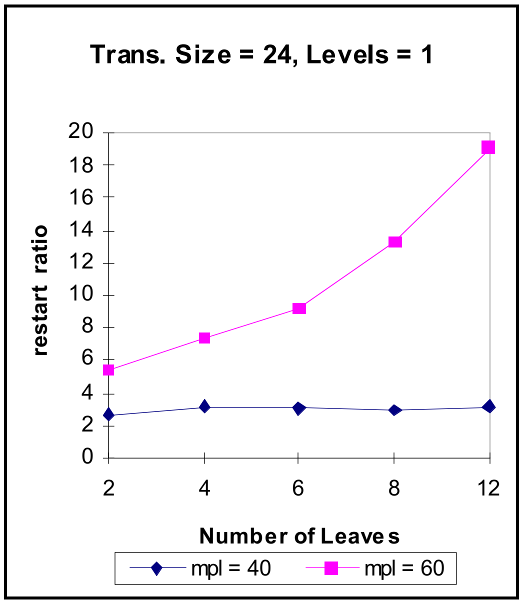

Figure 5 The effect of number of leaves, levels =1, Transactions size = 24, mpl = 40, 60. (restart ratio)

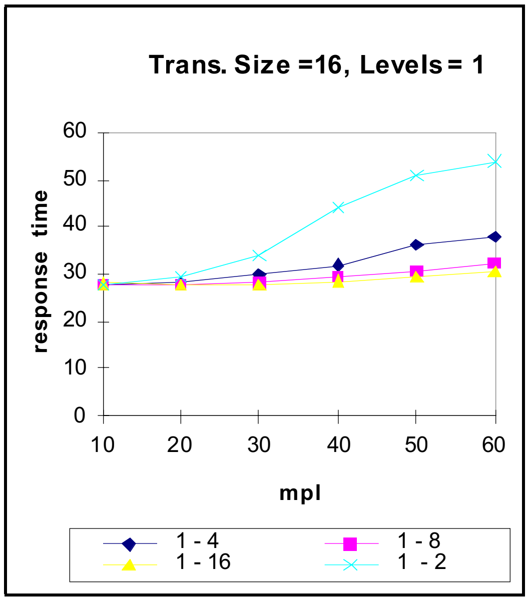

Figure 6 The effect of number of leaves, levels=1, Transaction size =16. (response time)

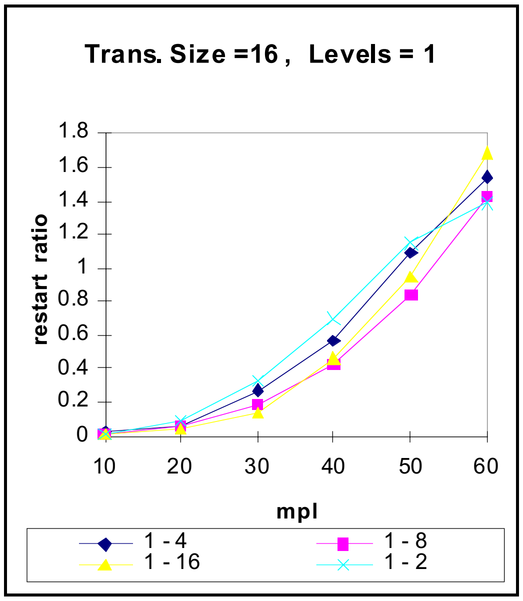

Figure 7 The effect of number of leaves, levels=1, Transaction size =16. (restart ratio)

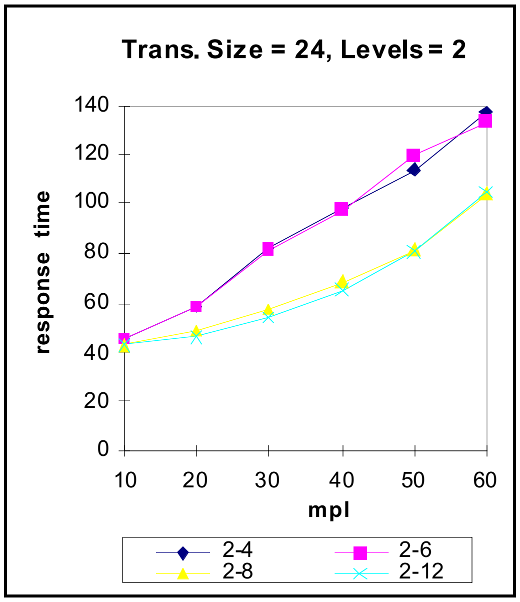

Figure 8 The effect of number of leaves, levels=2, Transaction size =24. (response time)

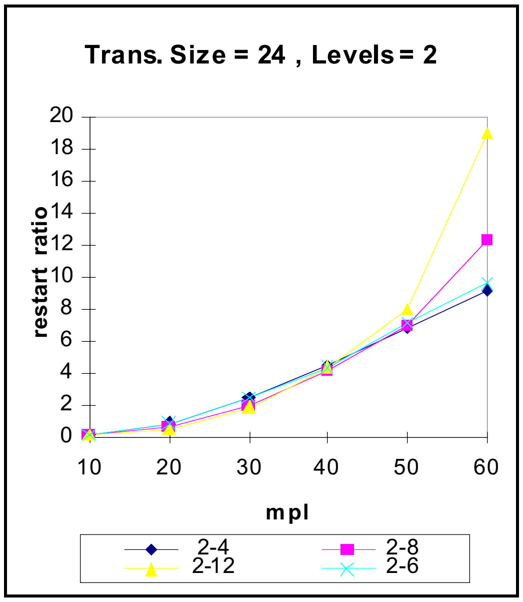

Figure 9 The effect of number of leaves, levels=2, Transaction size =24. (restart ratio)

4.2 Effect of number of levels

Various simulation experiments are performed to examine the effect of the number of levels on the performance of nested transactions. Transactions in each experiment have the same number of leaves. The number of levels is varied according to the number of leaves in each simulation experiment. Table 5 illustrates the different values of the number of levels considered for different number of leaves for transaction size 24.

Figure 10 and Figure 11 plot the response time for transactions of 4 and 6 leaves respectively. The number of levels in these two experiments is 1 and 2. Note that as the mpl is increased the performance difference among different transaction shapes becomes more apparent. This is because transactions of 2 levels have a larger number of internal subtransactions. As the number of internal subtransactions is increased, it takes longer time for the locks to be raised to the ancestors and deadlock cycles are detected. Therefore, the restart ratio and hence the response time of transactions of 2 levels are higher than the restart ratio and the response time results of transactions of 1 level.

Number of leaves	Number of levels
4	1 and 2
6	1 and 2
8	1, 2, and 3
12	1, 2, 3, and 4

Table 5 The number of leaves and the corresponding number of levels for transactions size 24

Figure 12 and Figure 13 respectively plot the response time and the restart ratio results for transactions with 8 leaves. Note that as mpl is increased, transactions of 1 and 2 levels start providing much better performance than transactions of 3 levels. For instance, at mpl = 60 the response time and the restart ratio for transactions of 1 level are respectively 25% and 50% lower than for transactions of 3 levels. Again, the degradation in the performance of transactions of 3 levels is caused by the existence of larger number of internal subtranasctions. The existence of such subtransactions increases the restart ratio, which in turn increases the response time.

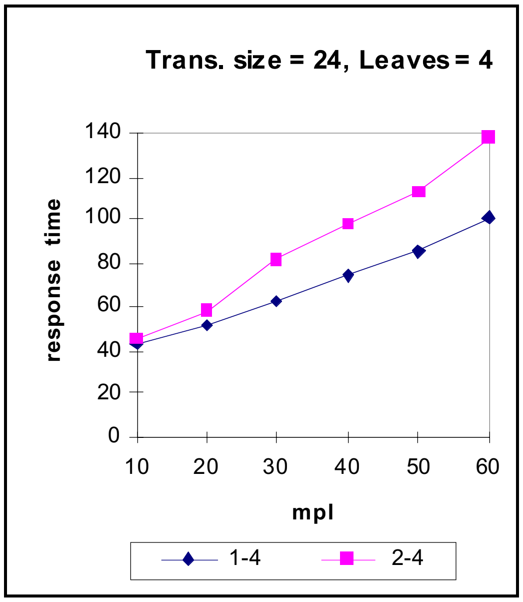

Figure 10 The effect of number of levels, leaves=4, Transaction size = 24. (response time)

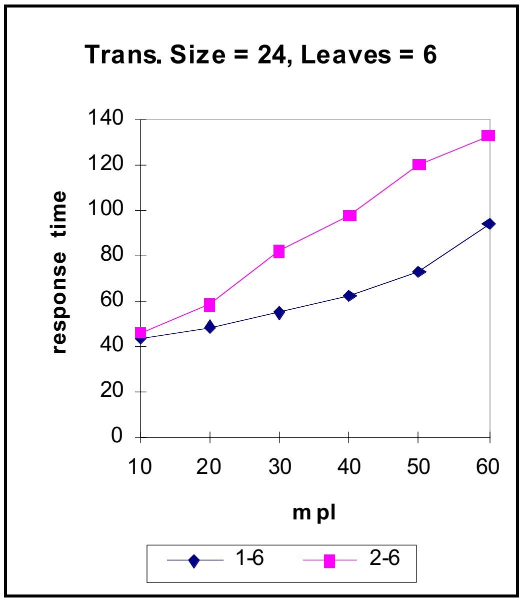

Figure 11 The effect of number of levels, leaves=6, Transaction size = 24. (response time)

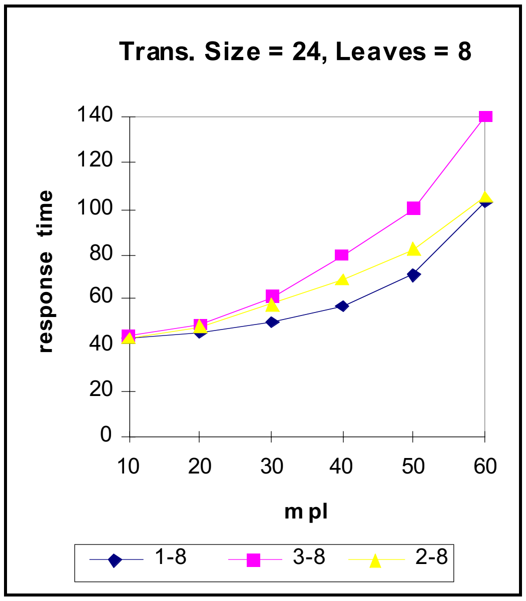

Figure 12 The effect of number of levels, leaves=8, Transaction size = 24. (response time)

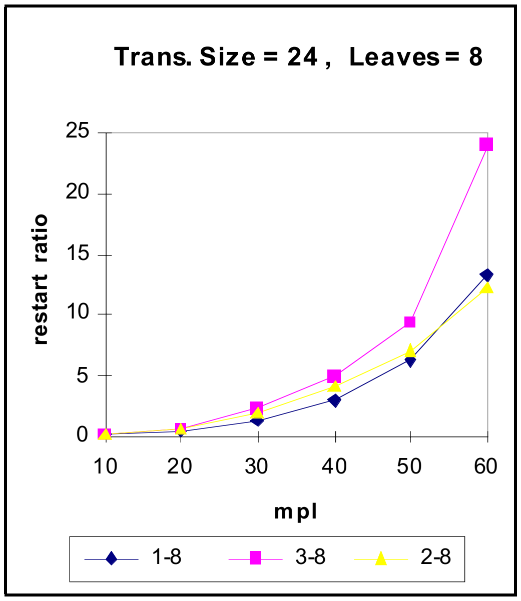

Figure 13 The effect of number of levels, leaves=8, Transaction size = 24. (restart ratio)

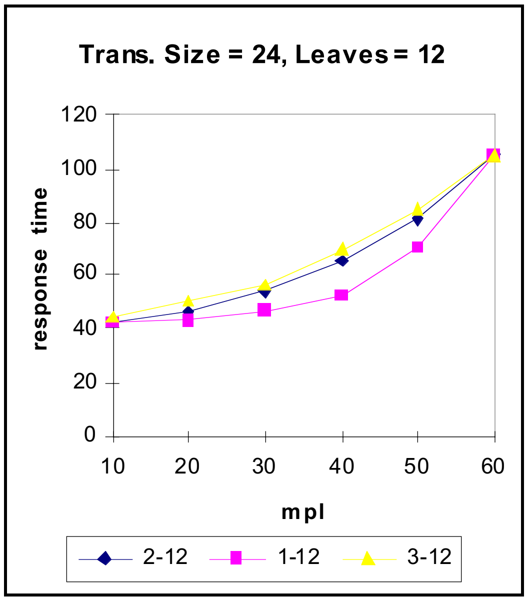

Figure 14 The effect of number of levels, leaves=12, Transaction size=24. (response time)

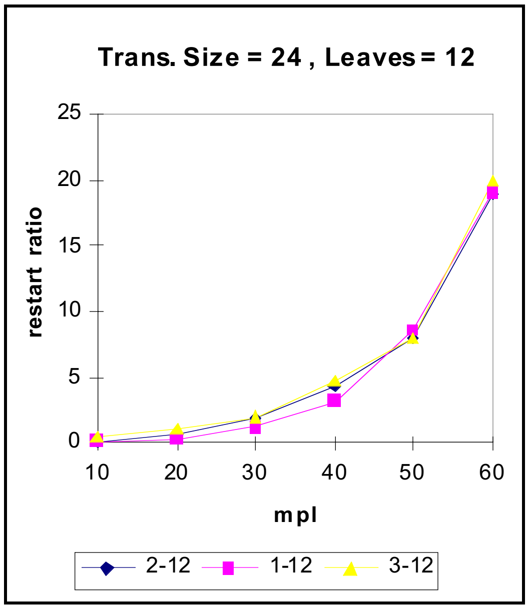

Figure 15 The effect of number of levels, leaves=12, Transaction size=24. (restart ratio)

The response time and the restart ratio results for transactions of 12 leaves are plotted in Figure 14 and Figure 15, respectively. It can be noted that as the mpl is increased beyond 40, the performance difference between transactions of different number of levels becomes less significant, till there is no difference at all at the highest mpl. The restart ratio for the different levels is almost the same for any number of levels and any mpl value (see Figure 15). Indeed, the effect of number of levels on the performance of transactions of 12 leaves (a larger number) is not significant.

5 Conclusions

This paper introduced a comprehensive simulation model for studying the performance of nested transactions in database systems. The model was used to investigate the performance effects of two main factors on a system with nested transactions: the number of levels and the number of leaves of the transaction tree.

It was shown that, in general, increasing the number of leaves improves the performance of the system. This was observed for any number of levels, transaction size, data contention level and leaf arrival time, with or without common items. However, at high data contention levels and large transaction sizes, increasing the number of leaves beyond a certain limit may cause performance degradation. This is due to the significant increase in the restart ratio caused by increasing the number of leaves.

It was also shown that increasing the number of levels of nested transactions degrades the system performance for nested transactions of small number of leaves. The effect of the number of levels on the performance of nested transactions of large number of leaves is insignificant, especially at high mutli-programming levels.

Biographies

Dr. Hassanein is an associate professor in the department of Computing and Information Science at Queen's University. His research interests are in the areas of broadband, wireless and optical networks architecture, protocols, control and performance evaluation. Prior to joining Queen's University in July 1999, he was with departments of Electrical and Computer Engineering at the Univeristy of Waterloo and the department of Mathematics and Computer Science at Kuwait Univeristy. He obtained his B.Sc. from Kuwait University in 1984 and M.Sc. from the University of Toronto in 1986, both in Computer Engineering. He received his Ph.D. in Computing Science at the university of Alberta in 1990. Dr. Hassanein has more than 50 publications in reputable journals, conferences and workshops in the areas of computer networks and performance evaluation. He served on the program committee of a number international conferences and workshops. Dr. Hassanein is the editor of the IEEE 802.6e standard for Slot Reuse in Distributed Queue Dual Bus networks in 1992. He was a voting member of the IEEE 802.6 standardization committee on Metropolitan Area Networks from 1992 to 1995. Dr. Hassanein is the recipient of the 1993 IEE "Hartee Premium" best paper award, for the IEE proceedings on Computers.

Dr. Sharkawi is an assistant professor in the Department of Information Systems at Cairo University. He obtained his B.Sc. from the department of Systems and Computer Engineering, Al-Azhar University, Cairo, Egypt, in 1981, and M.Sc. and Ph.D. from the department of Computer Science and Communication Engineering, Kyushu University, Fukuoka, Japan in 1988 and 1991, respectively. From 1992 to 1999, he was with the department of Mathematics and Computer Science at Kuwait Univeristy. Dr. Sharkawi has served on the program committee of a number international conferences and workshops, including the IEEE conference on distributed computing systems, 1995. His research interests include knowledge discovery and data mining temporal databases groupware, computer supported cooperative work (CSCW) and mobile databases.

References

[1] R. Agrawal, M. Carey, and M. Livny, "Concurrency Control Performance Modeling: Alternatives and Implications", *ACM Transactions on Database Systems*, Vol. 12, No. 4, December 1987, pp. 609-654.

[2] R. Agrawal, M. Carey, and M. Livny, "The Performance of Alternative Strategies for Dealing with Deadlocks in Database Management Systems", *IEEE Transactions on Software Engineering*, Vol. SE-13, No. 12, December 1987.

[3] B.R. Badrinath and K. Ramamritham, "Performance Evaluation of Semantic-based Multilevel Concurrency Control Protocols", *ACM SIGMOD,* 1990.

[4] L. Daynes, O. Gruber, and P. Valduriez, "Locking in OODBMS Client Supporting Nested Transactions", *In Proceeding of eleventh International conference On Data Engineering*, IEEE, 1995.

[5] Dunham, M., Helal, A., Balakrishnan, S, "A Mobile Transaction Model that Captures both the Data and Movment Behavior," Mobile Networks and Applications, Vol.2, 1997, pp. 149 - 162.

[6] A.A. El-Sayed, M.E. El-Sharkawi and H.S. Hassanein, "A New Mechanism for Deadlock Detection and Resolution in Nested Transactions," *Foundations of Database Organization*, Nov. 1998.

[7] P. A. Franaszek, J. T. Robinson, and A. Thomasian, "A Concurrency control for High contention Environments", *ACM Transactions on Database Systems*, Vol. 17, No. 2, June 1992, pp. 304-345.

[8] J.F. Garza and W. Kim, "Transaction Management in an Object-Oriented Database System," *Proc. of ACM SIGMOD Int'l Conf. on Management of Data*, pp. 37- 45, 1988.

[9] J. Gray and A. Reuter, "*Transaction Processing: Concepts and Techniques*", Morgan-Kauffman, San Francisco, 1993.

[10] C. Hasse and G. Weikum, "A Performance Evaluation of Multi-Level Transaction Management", *In Proceedings of the 17th International Conference on VLDB*, Barcelona, September 1991.

[11] H. F. Korth, W. Kim, F. Bancilhon, "On Long-Duration Transactions", *Readings in Object-Oriented Database Systems*, edited by: Zdonik, S. and Maier, D., Morgan Kaufman Publishers, Inc., 1990.

[12] C. Leung and E. Currie, “The Effect of Failures on the Performance of Long-Duration Database Transactions”, *The Computer Journal*, Vol. 38, No. 6, 1995.

[13] [Mad97] S.K. Madria, "A Study of the Concurrency Control and Recovery in Nested Transaction Environment", *The Computer Journal*, Vol. 40, No. 10, 1997.

[14] S.K. Madria and B. Bhargava, “A Transaction Model for Mobile Computing,” *Proceedings of the IEEE Database and Engineering Application Symposium* (IDEAS), pp. 92 – 102, 1998.

[15] J. E. B. Moss. *Nested Transactions: An Approach to Reliable Distributed Computing,* Ph.D. Thesis, MIT Press, Cambridge, Massachusetts, 1985.

[16] R. F. Resende. *Synchronization in Nested Transactions*. Ph.D. thesis, Dept. of Computer Science, Univ. of California - Santa Barbara, 1994.

[17] K. Ryu and A. Thomasian, “Performance Analysis of Dynamic Locking with the No-Waiting Policy”, *IEEE Transactions on Software Engineering*, July 1990, SE-16, pp. 684-698.

[18] Thomasian, “Two-Phase Locking performance and its thrashing behavior”, *Performance of Concurrency control Mechanisms in Centralized Database systems,* V. Kumar (Editor). Prentice Hall, England Cliffs, New Jersey, 1996.

[19] D. Worah and A. Sheth, “Transactions in Transactional Workflows,” in Jajodia, S., Kershberg, L. (Editiors) *Advanced Transaction Models and Architectures*, Kluwer Academic Publishers, pp. 3–33, 1997.

[20] W. Zhou, “Performance Evaluation of Nested Transactions on Locally Distributed Database Systems”, *In Proceedings of 2nd International Symposium on Parallel Architectures, Algorithms, and Networks*, 1-SPAN, IEEE, 1996.