View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Scholarly Commons @ MiamiOH (Miami University)

Computer Science and Systems Analysis

Computer Science and Systems Analysis Technical Reports

Miami University *Year* 1992

An Empirical Investigation of Four Strategies for Serializing Schedules in Transaction Processing

Terri Johnson
Miami University, commons-admin@lib.muohio.edu

This paper is posted at Scholarly Commons at Miami University.
http://sc.lib.muohio.edu/csa_techreports/45

MIAMI UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE & SYSTEMS ANALYSIS

TECHNICAL REPORT: MU-SEAS-CSA-1992-014

An Empirical Investigation of Four Strategies for Serializing Schedules in Transaction Processing

Terri Johnson

School of Engineering & Applied Science | Oxford, Ohio 45056 | 513-529-5928

An Empirical Investigation of Four Strategies for Serializing Schedules in Transaction Processing

by

Terri Johnson
Systems Analysis Department
Miami University
Oxford, Ohio 45056

Working Paper #92-014 12/92

An Empirical Investigation of Four Strategies for Serializing Schedules in Transaction Processing

By

Terri Johnson

December 2,1992

ABSTRACT

A database management system (DBMS) is a very large program that allows users to create and maintain databases. A DBMS has many capabilities. This study will focus on the capability known as transaction management, the capability to provide correct, concurrent access to the database by many users at the same time. If a DBMS did not provide transaction management, livelocks, deadlocks, and non-serializable schedules could occur. A *livelock* can occur when a transaction is waiting on a locked data item, and another transaction appears. After the data item is unlocked, the second transaction locks the data item, which causes the first transaction to continue waiting. Conceivably, the first transaction could wait indefinitely to lock the data item. This situation is called livelock. *Deadlock* is a situation in which each member of a set of two or more transactions is waiting to lock an item currently locked by some other transaction in the set. None of the transactions can proceed, so they all wait indefinitely. A schedule is *serial* if for every pair of transactions, all of the operations of one transaction execute before any of the operations of the other transaction. A schedule is *serializable* if its effect on the database is the same as some serial execution of the same set of transactions. A schedule is *non-serializable* if its effect on the database is not equivalent to that of any serial schedule which processes the same transactions. The scheduler is a component of the DBMS, and it is responsible for resolving any livelocks, deadlocks, or non-serializable schedules that occur. This study looks specifically at non-serializable schedules. There are many methods by which the scheduler can serialize non-serializable schedules. This study proposes and examines four strategies to detect and resolve non-serializable schedules. Computer

simulation is used to examine the four strategies. These strategies reduce a non-serializable schedule to a serializable or a serial schedule, thus eliminating the possibility of incorrectly updating data items within a database. It is shown experimentally that, of the four strategies, the one that delays the transaction which has executed the least number of steps until non-serializability is detected is the best.

1. INTRODUCTION

A database system is a system which involves humans and computers. It has been compared to a very complex and involved file system. The five components of a database system are people, data resource, hardware, software, and procedures. A special software system involved in the database system is the database management system. The DBMS is responsible for overseeing almost every component and process within the database system. One of the most important functions of the DBMS is to provide control over concurrent database operations. This is the focus of this research.

In a DBMS, if no concurrency control exists, a number of undesirable situations could occur, including livelocks, deadlocks, and non-serializable schedules, as defined earlier. This study takes a closer look at non-serializable schedules. Non-serializable schedules could result in incorrect updates of data items within a database. If a schedule turns out to be non-serializable, it is necessary to transform it into a serializable or a serial schedules. This eliminates the possibility of incorrect updates to data items.

The scheduler and the lock manager are components of the DBMS which work together to resolve non-serializable schedules. The scheduler has the responsibility to arbitrate between conflicting requests. The lock manager keeps track of how many transactions are reading or writing a given data item. It also prohibits another transaction from gaining access to a data item, if that access could cause a conflict. A transaction will request access to a data item through the scheduler. The scheduler then checks with the lock manager to determine if the request can be granted. Then, the scheduler relays a message of grant access, wait, or abort to the transaction.

In one approach, in order for the scheduler to determine if requests from transactions are conflicting, it generates a directed graph. This directed graph is examined for cycles. The nodes of this graph represent the transactions of the schedule and the arcs represent their dependencies. This directed graph is referred to as a waits-for graph or a serialization graph. A waits-for graph shows which transactions are "waiting" on other transactions. If a cycle exists in the waits-for graph, then the transactions involved in the cycle yield conflicting requests. In this study, an algorithm has been developed to detect cycles in a waits-for graph. If this algorithm detects a conflict, then it is necessary for the scheduler to determine which transaction in the conflict should be delayed until the remainder of the schedule has been executed.

In this study, we consider four possible strategies to determine the transaction that should be delayed so that a detected cycle can be broken. We will call such a transaction the *victim transaction*. The four strategies are:

1. Transaction which has executed the least number of steps is the victim;
2. Transaction which has most recently entered the cycle is the victim;
3. Transaction which has requested the most number of data items is the victim; and
4. The non-two-phase transaction, described later in Section 2, in the cycle is the victim. However, if two or more transactions in the cycle are non-two-phase, then randomly choose which transaction will be the victim.

A program was developed in order to test these strategies. The strategy which results in the smallest average wait time for the delayed transactions will be considered

as the best strategy of the four tested. The wait time is the amount of time a given delayed transaction will have to wait to restart from this transaction's initial beginning execution time. The objective is to minimize the wait time, thus the entire schedule can be completed in a minimum amount of time. Many experiments are executed in order to determine the best strategy.

In Section 2 of this paper, fundamental concepts of transaction processing are discussed. This includes discussion of the database management system, its capabilities and descriptions of the different types of schedules. Section 3 presents an algorithm to detect cycles in an undirected graph, an algorithm to test for serializability, and an algorithm to detect cycles in a directed graph. Section 4 describes four strategies for serializing a non-serializable schedule. In section 5, a description of the program and the experiments is given. The results of the experiments, the conclusions, and further research directions are discussed in section 6.

2. FUNDAMENTAL CONCEPTS OF TRANSACTION PROCESSING

A database management system is a collection of programs that allows users to create and maintain a database. Two capabilities which are fundamental to any DBMS are:

1. The ability to manage persistent data.
2. The ability to access large amounts of data efficiently.

In addition to these, the following are functions which are expected of DBMSs:

1. Support for at least one data model, or mathematical abstraction through which

the user perceives the data.

2. Support for certain very high-level, and desirably non-procedural languages that allow the user to define the structure of data, access data, and manipulate data.
3. Transaction management, the capability to provide correct, concurrent access to the database by many users at once.
4. Access control, the ability to limit access to data by unauthorized users, and the ability to check the validity of data.
5. Resiliency, the ability to recover from system failures without losing data.

The transaction management capability allows the DBMS to manage concurrent transactions, which may access and/or alter data items. If concurrency is not controlled, livelocks, deadlocks, and non-serializable schedules can occur. Incorrect updates to data items could result if non-serializable schedules are produced. A good example of the necessity for this capability are systems used in the banking industry. These database systems are accessed nearly simultaneously by numerous automated teller machines and bank employees. For example, if you are depositing money to your account through a bank teller and at the same time, your spouse is withdrawing money from an automated teller machine, the DBMS needs to make certain that both transactions correctly affect your account balance. If these transactions happen at exactly the same moment, then an invalid result may occur and your account balance could be incorrect. As one can see from this example, transaction management is a major issue in any DBMS.

A database system processes many transactions. A *transaction* is the execution of a program that accesses and/or changes the contents of the database. A set of concurrent

transactions is called a *schedule*. From a database schedule, we can determine which transactions are affecting which data items, at what time unit. There are three types of schedules: serial, serializable, and non-serializable [2,9,10]. A schedule is *serial* if for every pair of transactions, all of the operations of one transaction execute before any of the operations of the other transaction. Figure 1 gives an example of a serial schedule. A schedule is *serializable* if its effect on the database is the same as some serial execution of the same set of transactions. Figure 2 gives an example of a serializable schedule. The effect of Figure 2 is the same as a serial schedule in which transaction 2 precedes transaction 1. A schedule is *non-serializable* if its effect on the database is not equivalent to that of any serial schedule which processes the same transactions. Figure 3 gives an example of a non-serializable schedule. Some non-serializable schedules may produce results which are equivalent to a serial schedule. However, if the results are not produced in precisely the same order of operations as some serial schedule, then the schedule is considered to be non-serializable[10]. For example, if the end result of a schedule is to subtract 10 from the variable A, suppose a serial schedule produces this result by (A+10)-20. If a schedule being tested for serializability produces the same result by (A+20)-30, then the schedule is considered to be non-serializable.

T1	T2
READ A	
A:=A-10	
WRITE A	
READ B	
B:=B+10	
WRITE B	
	READ B
	B:=B-20
	WRITE B
	READ C
	C:=C+20
	WRITE C

Figure 1. Example of a Serial Schedule [10].

T1	T2
READ A	
	READ B
A:=A-10	
	B:=B-20
WRITE A	
	WRITE B
READ B	
	READ C
B:=B+10	
	C:=C+20
WRITE B	
	WRITE C

Figure 2. Example of a Serializable Schedule [10].

T1	T2
READ A	
A:=A-10	
	READ B
WRITE A	
	B:=B-20
READ B	
	WRITE B
B:=B+10	
	READ C
WRITE B	
	C:=C+20
	WRITE C

Figure 3. Example of a Non-Serializable Schedule [10].

Transactions must place locks on data items in order to access and/or update these data items. There are two types of locks: read locks and write locks. Multiple, concurrent read locks on the same data item are allowable, since a read lock only allows the transaction to read that data item. However, if a transaction has a write lock on a data

item, then no other transaction can place a lock of any kind on that data item. This helps protect the data item from incorrect updates. In this study, we will consider all locks to be write locks.

An important protocol when discussing database schedules is the Two-Phase Protocol. This protocol requires that within a given transaction, all locks precede all unlocks. Transactions that follow this protocol, are said to be Two-Phase. The first phase contains all the locks, the locking phase. The second phase contains all the unlocks, the unlocking phase. In Figure 4, transaction 1 and transaction 3 are Two-Phase transactions, while transaction 2 is not a Two-Phase transaction. The Two-Phase protocol is important to database scheduling due to the following theorem: "If S is any schedule of two-phase transactions, then S is serializable."[10]. Thus, if we can show that all transactions in a schedule are Two-Phase, then we have shown that the schedule is serializable.

T1	T2	T3
		LOCK C
LOCK A		LOCK D
LOCK B		
		UNLOCK C
	LOCK C	UNLOCK D
UNLOCK A		
UNLOCK B	UNLOCK C	
	LOCK A	
	UNLOCK A	

Figure 4. Two-Phase Transactions (T1 and T3).

As were defined previously, the lock manager and the scheduler are components of the database management system that work together to detect problems such as non-

serializable schedules, and transform them into serializable or serial schedules. The lock manager keeps track of how many transactions are reading or writing a given data item. The scheduler arbitrates between conflicting transaction requests. It controls the relative order of transactions by delaying or rejecting some transactions. A technique that helps the scheduler determine which transactions will be delayed or rejected, is to examine the waits-for graph of the schedule. A waits-for graph is a partial directed graph, or a digraph, whose nodes are labelled by transaction names; it contains an edge TI --> TJ whenever TI is waiting for TJ to release a lock on a data item. A theorem by R.C. Holt [7], states that, "In a waits-for graph, a cycle is a necessary condition for non-serializability". The next section describes concepts of graph theory that are related to digraphs, along with the algorithm developed to detect cycles in a digraph.

3. AN ALGORITHM FOR TESTING SERIALIZABILITY

A digraph is a pair (N,E), where N is a non-empty set of nodes and E is a set of edges. Each edge in E is an ordered pair (a,b), where a and b are nodes in N. An edge (a,b) is described as being directed from node a to node b [6,7]. A waits-for graph by its definition is a digraph. Its nodes are transactions and the edges are the dependencies between those transactions, due to locks on data items. Figure 6 gives an example of a waits-for graph. The nodes, or transactions, in Figure 6 are 1, 2, 3, 4, and 5. The edges, or dependencies, are (1,2), (2,3), (3,4), (4,1) and (3,5).

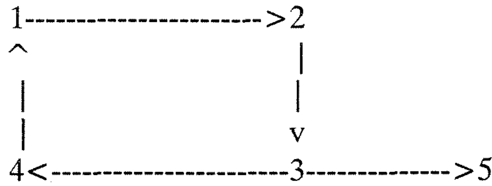

Figure 6. Example of a Waits-For Graph.

Since this study uses computer simulation, the computer representation of digraphs is now discussed. We can represent a digraph by a NxN matrix, A, called the adjacency matrix of the digraph. Here, N represents the number of nodes in the digraph. An entry in the matrix, Aij = 1, if an edge connecting nodes i and j exists; otherwise Aij = 0.

The following definitions of adjacency matrices are used in determining if a cycle exists in a digraph.

Definition 1 - The sum of a column gives the *indegree* of the corresponding node.

Definition 2 - The sum of a row gives the *outdegree* of the corresponding node.

Definition 3 - A *source* can be identified by a column of all zeros, i.e. its indegree is zero.

Definition 4 - A *sink* can be identified by a row of all zeros, i.e. its outdegree is zero.

Definition 5 - An *isolated point* can be detected by a column and corresponding row which both contain only zeros, i.e. both its indegree and outdegree are equal to zero.

Since a waits-for graph is a digraph, the above definitions are applicable to it. For example, Aij = 1 would imply that transaction i is waiting on transaction j to release a given data item.

We need to derive an algorithm to detect cycles in directed graphs. Three

algorithms are presented. The first algorithm detects cycles in undirected graphs. This algorithm will be the basis from which we will develop an algorithm to detect cycles in directed graphs. The second algorithm tests a schedule for serializability. We use the concept of transaction ordering to determine serializability from this algorithm in conjunction with the first algorithm and previously defined graph definitions, to help derive the algorithm which detects cycles in directed graphs. The three algorithms are as follows.

Algorithm to Detect Cycles in an Undirected Graph

INPUT: An undirected graph in which each node is connected to at least one other node in the graph.

OUTPUT: Generated cycles OR if no cycles have been generated, then no cycles exist.

ALGORITHM:

Let G be a given undirected graph of N nodes. First, find all the connected components of G. Then, the fundamental set of cycles can be found for each component H of G as follows.

Step 1 - Let E be the set of edges and V the set of nodes of H. Take any node v from V as the root of the tree consisting of the single node. Set T = {v}, S = V.

Step 2 - Let X be any node in $T \cap S$. If such a node does not exist, then stop.

Step 3 - Consider each edge (X,Y) in E.

If Y is in T, then generate the fundamental cycle consisting of edge (X,Y) together with the unique path between X and Y in the tree, and delete the edge (X,Y) from E.

If Y is not in T, then add the edge (X,Y) to the tree, add the node Y to T, and

delete the edge (X,Y) from E.

Step 4 - Remove the node X from S and return to Step 2.

The algorithm for testing serializability of a schedule [10], is as follows:

Algorithm for Serializability Testing

INPUT: A schedule S for a set of transactions T1,..,TK.

OUTPUT: A determination whether S is serializable.

If so, a serial schedule equivalent to S is produced.

ALGORITHM:

Create a directed graph G (called a serialization graph), whose nodes correspond to the transactions. To determine the arcs of the graph G, let S be a1;a2;...;an, where each ai is an action of the form:

Tj:LOCK Am or Tj:UNLOCK Am.

Tj indicates the transaction to which the step belongs. If ai is Tj:UNLOCK Am, look for the next action ap following ai that is of the form Ts:LOCK Am. If there is one, and $s \neq j$, then draw an arc from Ts to Tj. The intuitive meaning of this arc is that in any serial schedule equivalent to S, Tj must precede Ts.

If G has a cycle, then S is not serializable. If G has no cycles, then find a linear order for the transactions such that Ti precedes Tj whenever there is an arc Tj --> Ti. This ordering can always be done by the process known as *topological sorting,* defined as follows. There must be some node Ti with no entering arcs, else we can prove that G has a cycle. List Ti and remove Ti from G. Then repeat the process on the remaining graph until no nodes remain. The order in which the nodes are listed is a serial order for the

transactions.

Algorithm to Detect Cycles in a Directed Graph

INPUT: A directed graph in which each node is connected to at least one other node in the graph.

OUTPUT: Generated cycles. If cycles were generated, then the tree contains the arcs in the cycle and T contains the nodes in the cycle. If no cycles were generated, no cycles exist.

ALGORITHM:

Step 1 - Determine all edges, A:(X,Y). (A is the adjacency matrix of the digraph.)

Step 2 - Determine all nodes, V = S. T = null set.

Step 3 - If T intersect S = null set then choose a node v, from S, to be the root. v becomes an element of T.

Step 3a- If S is empty, then stop.

Step 4 - Choose a node 'NextNode' such that, 'NextNode' is in T $\cap$ S.

Step 4a- If such a 'NextNode' does not exist, then stop.

Step 5 - Consider each edge (NextNode,Y) in A.

Step5a- If no (NextNode,Y) exists in A, then delete NextNode from T and delete any (*,NextNode) from A.

else

If no (*,NextNode) exists in A or Tree, then delete NextNode from T and delete (NextNode,*) from A.

else

If Y is in T and (*,NextNode) is in the Tree, then add (NextNode,Y) to the Tree, generate the cycle consisting of the edges in the Tree, and delete (NextNode,Y) from A.

else

If Y is not in T and (Y,*) exists in A or Tree, then add the edge (NextNode,Y) to the Tree, add the node Y to T, and delete the edge (NextNode,Y) from A.

else

Delete (NextNode,Y) from A and if for all (NextNode,y), no (y,*) exists, then delete NextNode from T.

Step6 - Delete NextNode from S and go to Step 3.

Note: * indicates any node.

In the final algorithm, we can see the basic steps of the algorithm for detecting cycles in undirected graphs. In step 5a of the final algorithm, we see the concept of ordering transactions to determine serializability. If a cycle is generated, we know that the order of the transactions is that of a non-serializable schedule. Also in step 5a, the digraph definitions mentioned earlier are used. For a given node, if its indegree or outdegree equals zero, then the node being tested is not in a cycle. Thus, any arc containing that node can be disregarded.

4. FOUR STRATEGIES FOR SERIALIZING A NON-SERIALIZABLE SCHEDULE

There are multiple ways to break cycles in schedules. This study takes a look at four cycle breaking strategies and attempts to reach specific conclusions about their

ranking. In this study, a step is considered to be a lock or an unlock of any data item. The four strategies are:

1. Transaction which has executed the least number of steps is the victim. The idea in this strategy is to lose as little processing as possible, thus the transaction which has executed the least number of steps is the victim.
2. Transaction which has most recently entered the cycle is the victim. This strategy determines which transaction has transformed a path into a cycle and that transaction becomes the victim. The idea behind this strategy is that if the responsible transaction is removed, then there is a good chance the remaining transactions of the cycle will not form a subsequent cycle.
3. Transaction which has requested the most number of data items is the victim. This transaction has the potential to cause further cycles since it has requested locks on many data items. Postponing this transaction could decrease the number of cycles in the schedule.
4. The non-two-phase transaction in the cycle is the victim. However, if two or more transactions in the cycle are non-two-phase, then randomly choose which transaction will be the victim. This strategy was derived from the following theorem: "If S is any schedule of two-phase transactions, then S is serializable." [10]

Strategies 1, 2, and 3 assume two-phase transactions occur, but they do not affect which transaction becomes the victim. A victim transaction is a transaction whose operations are delayed and it must wait until the remainder of the transactions in the

schedule have completed before it may restart its sequence of operations.

5. DESCRIPTION OF THE PROGRAM AND THE EXPERIMENTS

A program was developed to test the four previously described strategies. The program randomly generates a database schedule from two inputs: the number of transactions in the schedule and an initial seed value. For each transaction, a random number between 1 and 4 is generated to represent the number of data items that a given transaction will request. For each data item within a given transaction, a random number between 1 and 10 will be generated to determine which data item that transaction will request. So far, we have the number of data items and which data items the transactions will request. The next step is to determine when these data items will be locked and unlocked. These times are also generated randomly. Finally, the program presents all of this information in the form of a schedule.

Next the program proceeds to determine if any cycles exist. This is accomplished by examining the adjacency matrix at each time step. At each time step, an adjacency matrix is generated by examining the schedule up to and including the current time step. The adjacency matrix is then tested for cycles by using the algorithm derived to detect cycles in a digraph, discussed earlier in Section 3. If a cycle is not detected, then the time step is increased by one and another adjacency matrix is generated for examination. If a cycle is detected, then the strategy selected by the user is utilized to determine which transaction, TI, will be delayed until after the remainder of the schedule has completed. The program determines the maximum of the end times for all of the transactions in the

schedule, except for the transaction TI. This maximum end time is the end of the schedule. Transaction TI can be restarted at maximum end time + 1. The program calculates the wait time each time a cycle is detected. The wait time is the difference between the maximum end time and the start time of TI. The start time of TI is the time when the first lock is requested by TI. If subsequent cycles occur between the remaining transactions in the schedule, then those wait times are added to the previous wait time. The objective is to minimize the wait time for the entire schedule. The smaller the wait time, the faster the schedule as a whole can complete its processing and the less time the user has to wait for his/her job to complete. Appendix C contains an example run of the program.

In trying to complete the objective of this study, to determine which strategy is the best, 18 experiments were executed. Figure 7 is an example of an experiment. In this figure, we have 25 runs of each strategy with 15 transactions. One hundred passes through the program are shown. In run #1 of Figure 7, strategy #2 has the smallest wait time. The average wait time is calculated for each strategy. These averages are then ranked from lowest to highest. The strategy with the lowest average wait time is considered to be the best in that experiment. In Figure 7, strategy #3 is the best. All 18 experiments are examined and the number of times each strategy comes in first, second, third, and fourth is tabulated and the percentages calculated. Appendix A contains charts which express these calculations.

RUN #	# OF TRANS	INIT SEED	STRAT #1	STRAT #2	STRAT #3	STRAT #4
1	15	38	444	359	521	413
2	15	185	529	647	552	690
3	15	288	115	99	99	118
4	15	231	312	310	173	172
5	15	315	309	96	40	309
6	15	300	312	310	173	172
7	15	380	115	99	99	118
8	15	415	312	310	173	172
9	15	457	182	170	183	183
10	15	402	206	190	154	194
11	15	7	62	62	76	62
12	15	163	269	134	168	273
13	15	259	449	477	478	471
14	15	42	362	505	383	411
15	15	650	431	252	265	377
16	15	357	115	99	99	118
17	15	111	168	181	217	166
18	15	43	234	401	196	402
19	15	49	291	380	192	420
20	15	609	494	543	397	213
21	15	190	189	189	162	195
22	15	180	323	637	613	613
23	15	222	37	112	120	37
24	15	333	206	190	154	194
25	15	444	87	80	83	83

	STRAT #1	STRAT #2	STRAT #3	STRAT #4
AVERAGE:	262.12	273.28	230.8	263.04
RANKING:	2nd	4th	**1st**	3rd

Figure 7. An example experiment.

Appendix B contains all 18 experiments. The number of transactions per schedule used are 5, 7, 9, 11, 13, and 15. For each number of transactions, 7, 15, and 25 runs were executed. Each run contains 4 passes of the program, one for each strategy. Thus, we have obtained [(7x6) + (15x6) + (25x6)] x 4 = 1,128 pieces of data. The next section discusses the results of the experiments and the conclusions which were reached.

6. CONCLUSIONS

In this paper, we discussed the capabilities of the DBMS. Transaction management was discussed in some detail. When considering transaction processing, undesirable situations could occur if transaction management was not present. Situations such as livelock, deadlock, and non-serializable schedules are possible. Serializability was studied in detail. The scheduler, lock manager, and their responsibilities were discussed. The scheduler utilizes a waits-for graph to detect non-serializable schedules. A waits-for graph is equivalent to a digraph, thus digraph theory was studied. Two algorithms were studied along with several graph definitions, to derive an algorithm to detect cycles in directed graphs. This study uses computer simulation, so we represent the digraphs as adjacency matrices.

Four strategies to break cycles in the waits-for graph were defined. A program was developed which implemented the algorithm to detect cycles in the waits-for graph and the four strategies to break those cycles. Numerous runs were made for each strategy. Appendix B contains the data collected from these runs. After all of the runs were completed, an average wait time was calculated for each strategy in each experiment. The

strategies were then ranked according to the average wait time. Finally, it was determined what percentage of the time each strategy ranked first, second, third, and fourth. Appendix A contains charts describing the results. From Chart #1 in Appendix A, it is clear that strategy #1 is the best. Strategy #1 came in first place 67% of the time. Strategy #3, although not as good as strategy #1, is clearly better than strategies #2 and #4. The experiments were then divided into three groups: small, medium, and large number of transactions. These group labels, small, medium, and large number of transactions, are relative to these experiments, since there are no standards for a small, medium, or large schedule of transactions. Chart #2 of Appendix A describes the results for experiments with five and seven (small) transactions. Again, strategy #1 is the best and strategy #3 comes in second place. Chart #3 of Appendix A describes the results for experiments with nine and eleven (medium) transactions. Similarly, strategy #1 is the best and strategy #3 comes in second place. Chart #4 of Appendix A describes the results for experiments with thirteen and fifteen (large) transactions. Once again, the same results have occurred. Strategy #1 is the best and strategy #3 comes in second place. Strategy #1 is the transaction which has executed the least number of steps is the victim and strategy #3 is the transaction which has requested the most number of data items is the victim. From these results, strategy #1 would be recommended for a database system whose schedules are random. Implementing strategy #1 in a database system would minimize the wait time so users transactions would finish faster.

As we progressed through this study, other research directions were discovered. This study considered all locks to be write locks. A study similar to this situation, but also

considering read locks could give another perspective. Also, the program used in this study checked the adjacency matrix at every time step. One could check the adjacency matrix at every N time steps. If a cycle exists, then do a binary search to determine at what exact time unit the cycle occurred. This type of search would speed up the processing time for the database schedule. However, the wait time for a given schedule and strategy would remain the same. Of course, the four strategies are not all inclusive. One could develop more strategies and test them against these four strategies or other developed strategies.

REFERENCES

1. P.A. Bernstein, V. Hadzilacos, N. Goodman, *Concurrency Control and Recovery in Database Systems,* Addison-Wesley Publishing Company, Reading, MA, 1987.

2. P.A. Bernstein, D.W. Shipman, W.S. Wang, "Formal Aspects of Serializability in Database Concurrency Control", *IEEE Transaction on Software Engineering.* Vol. V, No. 3, May 1979, pp.203-216.

3. B. Carre, *Graphs and Networks,* Clarendon Press, 1979.

4. R. Elmasri, S.B. Navathe, *Fundamentals of Database Systems,* The Benjamin/Cummings Publishing Company, Inc., Redwood City, CA, 1989.

5. S. Even, *Graph Algorithms,* Computer Science Press, Potomac, MD, 1979.

6. L.R. Foulds, D.F. Robinson, *Digraphs: Theory and Techniques,* Gordon and Breach Science Publishers, New York, NY, 1980.

7. R.C. Holt, "Some Deadlock Properties of Computer Systems", *ACM Computing Surveys.* Vol. IV, No. 3, September 1972, pp. 180-196.

8. H.T. Lau, *Algorithms on Graphs,* TAB Professional and Reference Books, Blue Ridge Summit, PA, 1989.

9. C.H. Papadimitriou, *The Theory of Concurrency Control,* Computer Science Press, Rockville, MD, 1986.

10. J.D. Ullman, *Principles of Database and Knowledge-Base Systems,* Vol. I, Computer Science Press, Rockville, MD, 1988.

11. A.A. Zykov, *Fundamentals of Graph Theory,* BCS Associates, 1990.

Appendix A - Charts describing results of the experiments

Chart #1 All 18 experiments used

	Strategy #1	Strategy #2	Strategy #3	Strategy #4
# times in 1st place	12 67%	1 5%	5 28%	0 0%
# times in 2nd place	2 11%	3 17%	8 44%	5 28%
# times in 3rd place	2 11%	5 28%	5 28%	6 33%
# times in 4th place	2 11%	9 50%	0 0%	7 39%

Chart #2 Small (5 and 7 Transactions), 6 experiments

	Strategy #1	Strategy #2	Strategy #3	Strategy #4
# times in 1st place	6 100%	0 0%	0 0%	0 0%
# times in 2nd place	0 0%	2 33%	4 67%	0 0%
# times in 3rd place	0 0%	1 17%	2 33%	3 50%
# times in 4th place	0 0%	3 50%	0 0%	3 50%

Chart #3 Medium (9 and 11 Transactions), 6 experiments

	Strategy #1	Strategy #2	Strategy #3	Strategy #4
# times in 1st place	3 50%	1 17%	2 33%	0 0%
# times in 2nd place	0 0%	1 17%	3 50%	2 33%
# times in 3rd place	2 33%	2 33%	1 17%	1 17%
# times in 4th place	1 17%	2 33%	0 0%	3 50%

Appendix A (continued)

Chart #4 Large (13 and 15 Transactions), 6 experiments

	Strategy #1	Strategy #2	Strategy #3	Strategy #4
# times in 1st place	3 50%	0 0%	3 50%	0 0%
# times in 2nd place	2 33%	0 0%	1 17%	3 50%
# times in 3rd place	0 0%	2 33%	2 33%	2 33%
# times in 4th place	1 17%	4 67%	0 0%	1 17%

Appendix B - Data collected from multiple runs of the program

RUN #	# OF TRANS	INIT SEED	STRAT #1	STRAT #2	STRAT #3	STRAT #4
1	5	213	16	16	16	16
2	5	319	11	15	11	11
3	5	523	17	12	17	17
4	5	460	17	11	17	17
5	5	109	17	17	15	15
6	5	164	15	15	15	13
7	5	197	15	50	50	52

	STRAT #1	STRAT #2	STRAT #3	STRAT #4
AVERAGE:	15.43	19.43	20.14	20.14
RANKING:	**1st**	2nd	3rd	4th

RUN #	# OF TRANS	INIT SEED	STRAT #1	STRAT #2	STRAT #3	STRAT #4
1	5	215	16	16	16	16
2	5	285	11	11	11	11
3	5	229	14	43	45	45
4	5	333	11	11	11	11
5	5	624	17	17	17	15
6	5	80	13	16	16	16
7	5	93	13	14	13	13
8	5	147	18	52	18	54
9	5	176	13	41	44	18
10	5	263	11	11	11	11
11	5	316	17	12	17	17
12	5	379	11	11	11	11
13	5	505	11	14	11	11
14	5	650	13	15	13	13
15	5	150	18	50	18	52

	STRAT #1	STRAT #2	STRAT #3	STRAT #4
AVERAGE:	13.8	22.27	18.13	20.93
RANKING:	**1st**	4th	2nd	3rd

Appendix B (continued)

RUN #	# OF TRANS	INIT SEED	STRAT #1	STRAT #2	STRAT #3	STRAT #4
1	5	90	12	14	12	12
2	5	156	13	14	13	13
3	5	173	18	50	18	52
4	5	185	19	22	22	22
5	5	207	21	21	20	20
6	5	231	11	11	17	17
7	5	92	13	14	13	13
8	5	271	11	15	11	11
9	5	348	12	12	12	12
10	5	360	17	16	16	17
11	5	386	11	15	11	11
12	5	5	18	16	18	18
13	5	79	13	16	16	13
14	5	555	12	12	12	12
15	5	432	11	15	11	11
16	5	91	13	14	13	13
17	5	198	18	18	15	15
18	5	1234	18	18	20	20
19	5	2468	12	12	12	12
20	5	160	15	15	22	22
21	5	412	11	14	11	11
22	5	108	15	15	15	15
23	5	145	13	16	16	13
24	5	147	18	52	18	54
25	5	151	18	50	18	52

	STRAT #1	STRAT #2	STRAT #3	STRAT #4
AVERAGE:	14.52	19.48	15.28	19.24
RANKING:	**1st**	4th	2nd	3rd

RUN #	# OF TRANS	INIT SEED	STRAT #1	STRAT #2	STRAT #3	STRAT #4
1	7	197	67	131	131	152
2	7	229	24	55	57	57
3	7	80	124	122	28	124
4	7	190	13	13	19	19
5	7	176	20	10	20	20
6	7	150	20	53	20	20
7	7	85	63	52	61	63

	STRAT #1	STRAT #2	STRAT #3	STRAT #4
AVERAGE:	47.29	62.29	48	65
RANKING:	**1st**	3rd	2nd	4th

Appendix B (continued)

RUN #	# OF TRANS	INIT SEED	STRAT #1	STRAT #2	STRAT #3	STRAT #4
1	7	33	25	25	25	75
2	7	44	16	16	16	17
3	7	120	18	14	18	18
4	7	148	19	19	76	76
5	7	189	13	13	20	20
6	7	216	27	77	79	79
7	7	39	19	17	19	19
8	7	67	16	16	16	17
9	7	88	58	48	58	58
10	7	130	77	75	75	79
11	7	157	19	17	19	19
12	7	240	13	13	13	13
13	7	160	17	17	21	21
14	7	20	18	16	16	18
15	7	99	26	71	25	72

	STRAT #1	STRAT #2	STRAT #3	STRAT #4
AVERAGE:	25.4	30.27	33.07	40.07
RANKING:	**1st**	2nd	3rd	4th

Appendix B (continued)

RUN #	# OF TRANS	INIT SEED	STRAT #1	STRAT #2	STRAT #3	STRAT #4
1	7	173	20	53	20	20
2	7	386	20	71	73	71
3	7	79	120	118	28	120
4	7	432	20	71	73	71
5	7	415	17	17	19	19
6	7	145	63	80	84	84
7	7	151	20	53	20	20
8	7	149	57	97	53	53
9	7	89	58	48	58	58
10	7	507	17	17	19	19
11	7	657	87	14	54	54
12	7	5	29	24	29	29
13	7	16	19	17	19	19
14	7	11	20	71	73	71
15	7	22	16	16	16	17
16	7	234	18	16	16	18
17	7	295	20	71	73	71
18	7	268	54	21	21	24
19	7	325	18	16	18	18
20	7	357	29	24	29	29
21	7	409	20	71	73	71
22	7	447	13	13	13	13
23	7	196	15	50	52	52
24	7	525	20	71	73	71
25	7	555	18	16	18	18

	STRAT #1	STRAT #2	STRAT #3	STRAT #4
AVERAGE:	32.32	45.44	40.96	44.4
RANKING:	**1st**	4th	2nd	3rd

RUN #	# OF TRANS	INIT SEED	STRAT #1	STRAT #2	STRAT #3	STRAT #4
1	9	24	26	121	63	69
2	9	57	17	17	61	62
3	9	91	73	59	59	59
4	9	13	23	23	23	25
5	9	37	24	24	24	26
6	9	134	101	101	100	107
7	9	215	74	74	69	142

	STRAT #1	STRAT #2	STRAT #3	STRAT #4
AVERAGE:	48.29	59.86	57	70
RANKING:	**1st**	3rd	2nd	4th

Appendix B (continued)

RUN #	# OF TRANS	INIT SEED	STRAT #1	STRAT #2	STRAT #3	STRAT #4
1	9	4	114	106	65	65
2	9	31	77	75	74	74
3	9	49	69	117	120	120
4	9	62	80	78	80	85
5	9	79	31	79	31	31
6	9	83	57	47	109	108
7	9	95	53	52	70	70
8	9	101	83	82	29	84
9	9	116	22	50	50	55
10	9	123	72	59	67	67
11	9	127	91	89	58	58
12	9	148	70	138	134	134
13	9	156	24	22	24	24
14	9	163	23	26	26	26
15	9	189	85	138	140	170

	STRAT #1	STRAT #2	STRAT #3	STRAT #4
AVERAGE:	63.4	77.2	71.8	78.07
RANKING:	**1st**	3rd	2nd	4th

Appendix B (continued)

RUN #	# OF TRANS	INIT SEED	STRAT #1	STRAT #2	STRAT #3	STRAT #4
1	9	200	26	25	25	26
2	9	210	194	222	86	193
3	9	225	78	153	150	155
4	9	237	31	26	31	31
5	9	250	24	24	24	26
6	9	257	26	23	26	26
7	9	261	26	121	63	69
8	9	270	53	50	50	64
9	9	274	27	25	27	27
10	9	281	19	49	49	19
11	9	289	45	43	21	18
12	9	12	23	70	66	70
13	9	295	23	70	66	70
14	9	301	73	73	73	80
15	9	45	33	28	28	33
16	9	60	27	30	27	27
17	9	135	57	43	45	45
18	9	86	50	47	47	50
19	9	204	67	130	133	76
20	9	315	16	16	21	21
21	9	472	63	58	70	70
22	9	90	54	74	71	54
23	9	21	19	49	49	19
24	9	69	32	27	32	32
25	9	5	63	58	70	70

	STRAT #1	STRAT #2	STRAT #3	STRAT #4
AVERAGE:	45.96	61.36	54	54.84
RANKING:	**1st**	4th	2nd	3rd

RUN #	# OF TRANS	INIT SEED	STRAT #1	STRAT #2	STRAT #3	STRAT #4
1	11	222	158	132	219	151
2	11	60	35	35	39	35
3	11	83	162	127	131	162
4	11	79	305	190	195	202
5	11	4	37	32	37	37
6	11	188	77	58	58	72
7	11	247	118	122	118	74

	STRAT #1	STRAT #2	STRAT #3	STRAT #4
AVERAGE:	127.43	99.43	113.86	104.71
RANKING:	4th	**1st**	3rd	2nd

Appendix B (continued)

RUN #	# OF TRANS	INIT SEED	STRAT #1	STRAT #2	STRAT #3	STRAT #4
1	11	140	231	134	85	72
2	11	5	27	65	22	70
3	11	20	69	128	69	73
4	11	155	171	238	152	162
5	11	132	19	19	65	19
6	11	138	180	161	126	198
7	11	144	65	91	114	114
8	11	163	24	70	24	26
9	11	196	91	90	79	148
10	11	555	75	75	35	35
11	11	225	163	207	246	246
12	11	12	172	142	144	103
13	11	286	113	108	97	97
14	11	80	75	145	95	75
15	11	48	105	94	96	96

	STRAT #1	STRAT #2	STRAT #3	STRAT #4
AVERAGE:	105.33	117.8	96.6	102.27
RANKING:	3rd	4th	**1st**	2nd

Appendix B (continued)

RUN #	# OF TRANS	INIT SEED	STRAT #1	STRAT #2	STRAT #3	STRAT #4
1	11	15	95	77	79	95
2	11	29	130	196	82	208
3	11	62	20	18	18	20
4	11	8	177	162	148	177
5	11	95	39	98	109	109
6	11	73	65	63	122	122
7	11	35	172	142	144	103
8	11	48	105	94	96	96
9	11	57	59	59	118	128
10	11	88	30	27	30	30
11	11	150	170	170	104	128
12	11	123	164	70	70	175
13	11	176	127	105	119	130
14	11	182	24	15	24	24
15	11	206	80	72	74	82
16	11	309	113	108	97	97
17	11	219	77	58	58	72
18	11	267	145	134	75	145
19	11	242	27	65	22	70
20	11	293	118	122	118	74
21	11	327	30	29	30	30
22	11	345	102	91	100	102
23	11	365	29	67	67	29
24	11	416	132	71	71	146
25	11	392	26	26	26	29

	STRAT #1	STRAT #2	STRAT #3	STRAT #4
AVERAGE:	90.24	85.56	80.04	96.84
RANKING:	3rd	2nd	**1st**	4th

RUN #	# OF TRANS	INIT SEED	STRAT #1	STRAT #2	STRAT #3	STRAT #4
1	13	95	79	143	77	33
2	13	73	147	221	222	93
3	13	242	192	158	157	157
4	13	365	121	124	74	74
5	13	20	42	87	97	97
6	13	144	180	236	259	259
7	13	80	197	273	281	278

	STRAT #1	STRAT #2	STRAT #3	STRAT #4
AVERAGE:	136.86	177.43	166.71	141.57
RANKING:	**1st**	4th	3rd	2nd

Appendix B (continued)

RUN #	# OF TRANS	INIT SEED	STRAT #1	STRAT #2	STRAT #3	STRAT #4
1	13	225	331	563	310	563
2	13	261	46	45	45	45
3	13	281	288	249	264	290
4	13	12	247	242	376	374
5	13	204	196	271	271	161
6	13	90	161	256	421	259
7	13	49	168	258	91	91
8	13	79	220	386	283	285
9	13	83	260	490	395	543
10	13	95	79	143	77	33
11	13	116	102	171	183	183
12	13	148	85	80	64	79
13	13	189	104	31	31	100
14	13	57	155	118	118	144
15	13	386	254	206	145	145

	STRAT #1	STRAT #2	STRAT #3	STRAT #4
AVERAGE:	179.73	233.93	204.93	219.67
RANKING:	**1st**	4th	2nd	3rd

Appendix B (continued)

RUN #	# OF TRANS	INIT SEED	STRAT #1	STRAT #2	STRAT #3	STRAT #4
1	13	432	254	206	145	82
2	13	145	263	341	233	343
3	13	418	250	217	174	248
4	13	196	106	177	106	106
5	13	525	316	281	317	342
6	13	160	522	337	152	172
7	13	99	158	156	90	158
8	13	197	412	424	291	291
9	13	229	337	311	262	334
10	13	150	95	223	160	167
11	13	61	154	146	154	93
12	13	176	165	77	156	156
13	13	650	248	146	289	387
14	13	319	121	124	74	74
15	13	275	105	95	92	92
16	13	237	35	87	35	35
17	13	26	39	30	39	39
18	13	34	254	206	145	82
19	13	173	95	223	160	167
20	13	309	39	30	39	39
21	13	357	192	158	157	157
22	13	28	192	158	157	157
23	13	41	186	236	96	239
24	13	399	46	45	45	45
25	13	30	21	25	25	25

	STRAT #1	STRAT #2	STRAT #3	STRAT #4
AVERAGE:	184.2	178.36	143.72	161.2
RANKING:	4th	3rd	**1st**	2nd

RUN #	# OF TRANS	INIT SEED	STRAT #1	STRAT #2	STRAT #3	STRAT #4
1	15	35	310	240	291	309
2	15	80	247	258	418	268
3	15	16	214	308	189	313
4	15	45	449	477	478	471
5	15	96	333	416	393	308
6	15	116	311	550	432	275
7	15	137	431	252	265	377

	STRAT #1	STRAT #2	STRAT #3	STRAT #4
AVERAGE:	327.86	357.29	352.29	331.57
RANKING:	**1st**	4th	3rd	2nd

Appendix B (continued)

RUN #	# OF TRANS	INIT SEED	STRAT #1	STRAT #2	STRAT #3	STRAT #4
1	15	206	379	366	292	194
2	15	61	225	300	433	319
3	15	99	495	494	526	295
4	15	107	566	502	431	701
5	15	15	444	359	521	413
6	15	167	115	163	190	165
7	15	125	267	309	492	370
8	15	25	119	110	226	314
9	15	53	396	376	244	378
10	15	71	101	85	87	87
11	15	87	567	447	355	449
12	15	134	368	681	204	689
13	15	228	649	604	362	375
14	15	197	460	550	525	619
15	15	252	444	359	521	413

	STRAT #1	STRAT #2	STRAT #3	STRAT #4
AVERAGE:	373	380.33	360.6	385.4
RANKING:	2nd	3rd	**1st**	4th

Appendix B (continued)

RUN #	# OF TRANS	INIT SEED	STRAT #1	STRAT #2	STRAT #3	STRAT #4
1	15	38	444	359	521	413
2	15	185	529	647	552	690
3	15	288	115	99	99	118
4	15	231	312	310	173	172
5	15	315	309	96	40	309
6	15	300	312	310	173	172
7	15	380	115	99	99	118
8	15	415	312	310	173	172
9	15	457	182	170	183	183
10	15	402	206	190	154	194
11	15	7	62	62	76	62
12	15	163	269	134	168	273
13	15	259	449	477	478	471
14	15	42	362	505	383	411
15	15	650	431	252	265	377
16	15	357	115	99	99	118
17	15	111	168	181	217	166
18	15	43	234	401	196	402
19	15	49	291	380	192	420
20	15	609	494	543	397	213
21	15	190	189	189	162	195
22	15	180	323	637	613	613
23	15	222	37	112	120	37
24	15	333	206	190	154	194
25	15	444	87	80	83	83

	STRAT #1	STRAT #2	STRAT #3	STRAT #4
AVERAGE:	262.12	273.28	230.8	263.04
RANKING:	2nd	4th	**1st**	3rd

Appendix C - An example run of the program

Enter the number of transactions
7
Enter the initial seed value
35
Transaction #1 has 1 Data Items
Transaction #2 has 2 Data Items
Transaction #3 has 4 Data Items
Transaction #4 has 3 Data Items
Transaction #5 has 2 Data Items
Transaction #6 has 4 Data Items
Transaction #7 has 4 Data Items

Transaction #	Data Item #	Lock Time	Unlock Time
1	2	8	12
2	5	14	20
	7	8	16
3	9	21	23
	1	2	6
	3	10	16
	6	7	14
4	8	9	19
	10	5	8
	2	3	7
5	5	6	13
	7	17	26
6	10	1	4
	2	13	18
	4	5	11
	6	15	23
7	9	10	20
	1	7	11
	3	4	9
	5	21	28

SCHEDULE

	T1	T2	T3	T4	T5	T6	T7
1						L10	
2			L1				
3				L2			
4						U10	L3
5				L10		L4	
6			U1		L5		
7			L6	U2			L1
8	L2	L7		U10			
9				L8			U3

Note:
Li = Lock Time for Ti
Ui = Unlock Time for Ti

Appendix C (continued)

	T1	T2	T3	T4	T5	T6	T7
10			L3				L9
11						U4	U1
12	U2						
13					U5	L2	
14		L5	U6				
15						L6	
16		U7	U3				
17					L7		
18						U2	
19				U8			
20		U5					U9
21			L9				L5
22							
23			U9			U6	
24							
25							
26					U7		
27							
28							U5

Choose a method to delay a transaction in a cycle.
1. Least Steps Performed
2. Most Recent Entry to the Cycle
3. Maximum Data Items Requested
4. Non 2-Phase Transaction

1

	T1	T2	T3	T4	T5	T6	T7
T1	0	0	0	1	0	0	0
T2	0	0	0	0	0	0	0
T3	0	0	0	0	0	0	1
T4	0	0	0	0	0	1	0
T5	0	0	0	0	0	0	0
T6	0	0	0	0	0	0	0
T7	0	0	1	0	0	0	0

Tree

3	7
7	3

TNodes Array
3
7

Cycle? Yes
Time is 10

Appendix C (continued)

STARTABTRAN: 2
MAXENDTIME: 28
ABWAITTIME: 26
PREVWAITTIME: 26

Transaction 3 has been delayed.

SCHEDULE

	T1	T2	T3	T4	T5	T6	T7
1						L10	
2							
3				L2			
4						U10	L3
5				L10		L4	
6					L5		
7				U2			L1
8	L2	L7		U10			
9				L8			U3
10							L9
11						U4	U1
12	U2						
13					U5	L2	
14		L5					
15						L6	
16		U7					
17					L7		
18						U2	
19				U8			
20		U5					U9
21							L5
22							
23						U6	
24							
25							
26					U7		
27							
28							U5
29			L1				
30							
31							
32							
33			U1				
34			L6				
35							
36							

Appendix C (continued)

	T1	T2	T3	T4	T5	T6	T7
37			L3				
38							
39							
40							
41			U6				
42							
43			U3				
44							
45							
46							
47							
48			L9				
49							
50			U9				

	T1	T2	T3	T4	T5	T6	T7
T1	0	0	0	1	0	0	0
T2	0	0	0	0	0	0	0
T3	0	0	0	0	0	0	0
T4	0	0	0	0	0	1	0
T5	0	0	0	0	0	0	0
T6	1	0	0	0	0	0	0
T7	0	0	0	0	0	0	0

Tree

1	4
4	6
6	1

TNodes Array

1
4
6

Cycle? Yes
Time is 13

STARTTRAN:	8
MAXENDTIME:	50
WAITTIME:	42
SCHEDWAITTIME:	68

Transaction 1 has been delayed.

Appendix C (continued)

SCHEDULE

	T1	T2	T3	T4	T5	T6	T7
1						L10	
2							
3				L2			
4						U10	L3
5				L10		L4	
6					L5		
7				U2			L1
8		L7		U10			
9				L8			U3
10							L9
11						U4	U1
12							
13					U5	L2	
14		L5					
15						L6	
16		U7					
17					L7		
18						U2	
19				U8			
20		U5					U9
21							L5
22							
23						U6	
24							
25							
26					U7		
27							
28							U5
29			L1				
30							
31							
32							
33			U1				
34			L6				
35							
36							
37			L3				
38							
39							
40							
41			U6				
42							
43			U3				

Appendix C (continued)

	T1	T2	T3	T4	T5	T6	T7
44							
45							
46							
47							
48			L9				
49							
50			U9				
51	L2						
52							
53							
54							
55	U2						

	T1	T2	T3	T4	T5	T6	T7
T1	0	0	0	0	0	0	0
T2	0	0	0	0	1	0	0
T3	0	0	0	0	0	0	0
T4	0	0	0	0	0	1	0
T5	0	0	0	0	0	0	0
T6	0	0	0	1	0	0	0
T7	0	0	0	0	0	0	0

Tree

4	6
6	4

TNodes Array

4

6

Cycle? Yes

Time is 14

STARTTRAN:	3
MAXENDTIME:	55
WAITTIME:	52
SCHEDWAITTIME:	120

Transaction 4 has been delayed.

SCHEDULE

	T1	T2	T3	T4	T5	T6	T7
1						L10	
2							
3							
4						U10	L3

Appendix C (continued)

	T1	T2	T3	T4	T5	T6	T7
5						L4	
6					L5		
7							L1
8		L7					
9							U3
10							L9
11						U4	U1
12							
13					U5	L2	
14		L5					
15						L6	
16		U7					
17					L7		
18						U2	
19							
20		U5					U9
21							L5
22							
23						U6	
24							
25							
26					U7		
27							
28							U5
29			L1				
30							
31							
32							
33			U1				
34			L6				
35							
36							
37			L3				
38							
39							
40							
41			U6				
42							
43			U3				
44							
45							
46							
47							
48			L9				

Appendix C (continued)

	T1	T2	T3	T4	T5	T6	T7
49							
50			U9				
51	L2						
52							
53							
54							
55	U2						
56				L2			
57							
58				L10			
59							
60				U2			
61				U10			
62				L8			
63							
64							
65							
66							
67							
68							
69							
70							
71							
72				U8			

	T1	T2	T3	T4	T5	T6	T7
T1	0	0	0	0	0	0	0
T2	0	0	0	0	1	0	0
T3	0	0	0	0	0	0	0
T4	0	0	0	0	0	0	0
T5	0	1	0	0	0	0	0
T6	0	0	0	0	0	0	0
T7	0	0	0	0	0	0	0

Tree
2 5
5 2
TNodes Array
2
5
Cycle? Yes
Time is 17

Appendix C (continued)

STARTTRAN: 8
MAXENDTIME: 72
WAITTIME: 64
SCHEDWAITTIME: 184

Transaction 2 has been delayed.

SCHEDULE

	T1	T2	T3	T4	T5	T6	T7
1						L10	
2							
3							
4						U10	L3
5						L4	
6					L5		
7							L1
8							
9							U3
10							L9
11						U4	U1
12							
13					U5	L2	
14							
15						L6	
16							
17					L7		
18						U2	
19							
20							U9
21							L5
22							
23						U6	
24							
25							
26					U7		
27							
28							U5
29			L1				
30							
31							
32							
33			U1				
34			L6				
35							
36							

Appendix C (continued)

	T1	T2	T3	T4	T5	T6	T7
37			L3				
38							
39							
40							
41			U6				
42							
43			U3				
44							
45							
46							
47							
48			L9				
49							
50			U9				
51	L2						
52							
53							
54							
55	U2						
56				L2			
57							
58				L10			
59							
60				U2			
61				U10			
62				L8			
63							
64							
65							
66							
67							
68							
69							
70							
71							
72				U8			
73		L7					
74							
75							
76							
77							
78							
79		L5					
80							

Appendix C (continued)

	T1	T2	T3	T4	T5	T6	T7
81		U7					
82							
83							
84							
85		U5					

Number of transactions delayed: 4
Cycle detected at time 10
Cycle detected at time 13
Cycle detected at time 14
Cycle detected at time 17
The total wait time for the schedule: 184

Do you want to generate another schedule?
No