A FORMAL MODEL FOR MAXIMUM CONCURRENCY IN TRANSACTION SYSTEMS WITH PREDECLARED WRITESETS

J. XU

Université Catholique de Louvain, Unité d'Informatique
Chemin du Cyclotron 2, B-1348 Louvain-la-Neuve, Belgium

ABSTRACT: This paper presents a formal model for studying the computation complexity of scheduling a whole set of transactions simultaneously in a transaction system with predeclared writesets. Our study clearly shows that there exists a fundamental tradeoff between the amount of concurrency achieved and the computation overhead necessary to achieve that amount of concurrency. However, it is suggested that based on variants of the model introduced here, schedulers which schedule a whole set of transactions simultaneously may still be able to achieve a higher level of concurrency than conventional schedulers within reasonable computation complexity constraints.

1.INTRODUCTION :

This paper is concerned with the computation complexity of obtaining maximum concurrency in a transaction system with predeclared writesets. Previous work has shown that in transaction systems with predeclared writesets, it is possible to achieve more concurrency than systems where writesets are not predeclared by eliminating restarts [2][3]. This implies that in a transaction system with predeclared writesets, in order to achieve more concurrency, a preventive strategy is one of the best strategies that can be used, i.e., the scheduler puts a requesting transaction into execution only if it determines beforehand that the execution of that transaction will never compromise consistency of the database system.

Previously proposed algorithms for concurrency control in transaction systems with predeclared writesets typically schedule requesting transactions only one at a time, even if a large number of transactions have arrived and are requesting execution simultaneously. In this case, the scheduler may chose for first execution a transaction which precludes the simultaneous execution of any other requesting transaction, while at the same time there may exist among other requesting transactions a large subset which could be simultaneously executed in parallel with all transactions currently in execution if they were chosen for execution first. For this reason, previously proposed algorithms do not achieve the potentional level of concurrency that may possibly be achieved.

In this paper, we present a formal model for studying the computation complexity of achieving maximum concurrency in a transaction system with predeclared writesets. In contrast to the common approach of scheduling transactions only one at a time, our model allows one to find either an optimal solution (at a higher computation cost, but still feasible when the total number of transactions in the system is small), or a suboptimal solution, by analyzing the whole set of requesting transactions to determine the largest subset, or simply any large subset which can be simultaneously put into execution in parallel with all transactions currently in execution in the system.

We begin with a most unrestricted model where the only correctness criterion is serializability and each transaction can read one out of several versions for each data item in its readset. Then we gradually add various restrictions on the model, while studying the effect of these restrictions on concurrency and computation complexity. Finally, we show by example, how a scheduler based on the concepts developed in our model can achieve a higher level of concurrency under reasonable computation complexity constraints, while avoiding certain anomalies which could be present in the less restrictive models.

It is suggested here that although there exists a fundamental tradeoff between the amount of concurrency achieved and the computation overhead necessary to achieve that amount of concurrency, it is still possible, under reasonable computation complexity constraints, to design schedulers in a transaction system with predeclared writesets which achieve a higher level of concurrency by scheduling a whole set of transactions simultaneously instead of scheduling transactions one at a time.

We emphasize that this is only a formal model, an approximation to the way a scheduler may actually function in a transaction system with predeclared writesets. No implementation details are discussed in this paper. We also leave out the problem of preventing starvation in the present discussion.

2. PRELIMINARIES : TRANSACTIONS AND SERIAL SCHEDULES.

In order to develop our model in the following section, we first introduce some basic definitions of transactions, schedules, serial schedules and "read from" relations between transactions in a schedule.

In our model, we consider transactions that consist of two atomic steps : a read of the values of a set of database entities --- called the "readset" of the transaction, followed by a write on a set of database entities --- the "write set". The notation adopted here is similar to that used in [9].

DEFINITION 2.1 : A transaction Ti : ([SRi], [SWi]) is a mapping from a readset SRi of variable names to a writeset SWi of variable names.

The variables are abstractions of data entities, whose granularity is not important for the present discussion. The variables can represent bits, files or records, as long as they are individually accessible. The set of all variable names in the database is denoted by V.

Each transaction Ti can be thought of as first reading a set of values for each variable name in its readset, then performing a possibly lengthy local computation based on that set of values. The results of the computation are finally used to produce a new set of values for each variable name in its writeset. The first step, i.e., the read step is denoted by Ri[SRi], while the last step, i.e., the write step is denoted by Wi[SWi].

DEFINITION 2.2 : A schedule of a set of transactions T = {T1, T2, ..., Tn} : T1 = ([SR1], [SW1]), T2 = ([SR2], [SW2]), ..., Tn = ([SRn], [SWn]) is a permutation of the set Sn = {R1[SR1], W1[SW1], ..., Rn[SRn], Wn[SWn]}, such that for every i : Ri[SRi] precedes Wi[SWi]. We abbreviate Ri[SRi] as Ri and Wi[SWi] as Wi whenever we need not specify SRi and SWi. We also introduce a function π : π is a one to one mapping from a permutation of Sn to the set {1, 2, ..., 2n}, such that for all i, j, αi $\in$ Sn, αj $\in$ Sn, if αi precedes αj in the permutation, then $\pi(\alpha i) < \pi(\alpha j)$.

Below we define a serial schedule, which models the situation where all transactions are executed sequentially.

DEFINITION 2.3 : A schedule of a set of transactions T = {T1, ..., Tn} is a serial schedule of T iff $\pi(Wi) = \pi(Ri) + 1$ for all i = 1, 2, ..., n. i.e. A read Ri always immediately precedes a write Wi of the same transaction.

In the following sections, we shall use " $\pi(Ti) < \pi(Tj)$ " to specify that in a serial schedule the following holds :

$$\pi(Wi) < \pi(Wj) \text{ and } \pi(Ri) < \pi(Rj)$$

DEFINITION 2.4 : We say " Rj reads x from Wi" in schedule s of T = {T1, T2, ..., Tn} iff for some x, i, j, x $\in$ V, $1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant n$:

(A2.1) x $\in$ (SWi $\cap$ SRj) and
(A2.2) $\pi(Wi) < \pi(Rj)$ and
(A2.3) there is no k, $1 \leqslant k \leqslant n$, such that x $\in$ SWk and $\pi(Wi) < \pi(Wk) < \pi(Rj)$

In the following sections, we shall also say "Tj reads x from Ti", when Rj reads x from Wi.

EXAMPLE 2.1.:

s1 = R1[x]W1[y,z,b]R2[z]W2[y]R3[y,b]W3[y]
s2 = R3[y,b]W3[y]R2[z]W2[y]R1[x]W1[y,z,b]

s1 and s2 are both serial schedules of the set of transactions T = {T1, T2, T3} where T1 = ([x], [y,z,b]) T2 = ([z], [y]) T3 = ([y,b], [y]). In serial schedule s1 of T : R2 reads z from W1, R3 reads b from W1, R3 reads y from W2. In serial schedule s2 of T : no transaction reads from any other transaction.

3. THE FORMAL MODEL

In a database system, the task of a scheduler is to maintain consistency of the database system while allowing as many user transactions as possible to simultaneously access the database system. In a transaction system with predeclared writesets, in order to obtain higher concurrency by preventing restarts, the scheduler puts requesting transactions into execution only if it determines beforehand that the execution of those transactions will never compromise consistency of the database system.

Since serializability [5][9] is used as the consistency criterion here, the scheduler must guarantee that the reads and writes of all transactions in the system have the same overall effect as if all transactions were executed in some serial order. (We shall call such an order which is not necessarily identical to the actual time order in which reads and writes are processed a "virtual order").

We model this as the following problem: Given a database system state consisting of 4 elements : an executing set of transactions, a terminated set of transactions, a requesting set of transactions, and a serial schedule defining the virtual ordering of all executing and terminated transactions; construct a new database system state, such that requesting transactions can be put into execution in parallel with all transactions already in execution, while the new virtual ordering is consistent with the previous virtual ordering.

To begin, we start with a most unrestrictive model, where the only correctness criterion is serializability, and each requesting transaction may read any one out of all existing version lues for each variable name in its readset.

DEFINITION 3.1. : A database system state is a quadruple $Q = (TE, TT, TR, s)$, where
TE is called the executing set of transactions
TT is called the terminated set of transactions
TR is called the requesting set of transactions
s is a serial schedule of $T = (TE \cup TT)$, called the virtual schedule.

In definition 3.1., an executing transaction is a transaction which has already been put into execution by reading some existing values of the variable names in its readset, but has NOT yet made the set of values for all variable names in its writeset available for reading by other transactions.

A terminated transaction is a transaction which has made a set of values for all variable names in its writeset available for reading by other transactions.

A requesting transaction is a transaction which has arrived in the system and is requesting execution, but has not been yet put into execution.

A virtual schedule is a serial schedule of all executing and terminated transactions. The scheduler guarantees that the reads and writes of all executing transactions and all terminated transactions will have the same overall effect as if they were executed sequentially in the same order as the virtual schedule. A virtual schedule completely defines which transaction has read the values of which variable names from the writeset of which transaction so far up to the present system state.

DEFINITION 3.2.: Database system state $Q1 = (TE1, TT1, TR1, s1)$ is a valid extension of database system state $Q = (TE, TT, TR, s)$, iff :

(A3.1.) (consistency condition)
(A3.1.1.) $TT1 = TT$ and $TR1 \subseteq TR$, and
(A3.1.2.) $TE1 = (TE \cup TRs)$, where $TRs \subseteq TR$ and $TR1 = (TR - TRs)$, and
(A3.1.3.) For all i, j, x, $Ti \in (TT \cup TE)$, $Tj \in (TT \cup TE)$, $x \in V$:
Rj reads x from Wi in s
<==> Rj reads x from Wi in s1, and
(A3.2.) (existence condition)
For all i, j, x, $Ti \in (TT1 \cup TE1)$, $TJ \in (TT1 \cup TE1)$, $x \in V$:
Rj reads x from Wi in s1 ==> $Ti \in TT1$

Definition 3.2. defines the conditions under which successive new valid system states can be constructed by putting requesting transactions into execution, i.e. by transfering a subset of transactions from the requesting set of the previous system state into the executing set of the new system state.

The "consistency condition" (A3.1.) guarantees that all the effects of the previous system state are preserved in the succeeding system state. Here we only give the minimum consistency conditions, while additional restrictions will be discussed gradually later on. Condition (A3.1.3.) states that if transaction j reads a value from transaction i in the previous virtual ordering, then transaction j should read the same value from transaction i in the new virtual ordering.

The "existence condition" (A3.2.) states that no transactions in the virtual ordering should read a value which has not yet been produced, in other words, transaction j can read a value from transaction i only if transaction i has terminated.

Note that for the same set of executing transactions, more than one virtual schedule can be constructed by rearranging the virtual schedule while keeping all the consistency and existence conditions invariant.

DEFINITION 3.3. : Database system state $Q1 = (TE1, TT1, TR1, s1)$ is a valid termination of database system state $Q = (TE, TT, TR, s)$ iff:

(B3.1.) $TT1 = (TT \cup TEs)$, where $TEs \subseteq TE$ and $TE1 = (TE - TEs)$, and
(B3.2.) $TR1 = TR$, and
(B3.3.) For all i, j, $Ti \in (TE \cup TT)$, $Tj \in (TE \cup TT)$:
$\pi(Tj) < \pi(Ti)$ in s <==> $\pi(Tj) < \pi(Ti)$ in s1

Definition 3.3. defines the conditions in which executing transactions are terminated, i.e., when a subset of transactions are transfered from the executing set of the previous system state into the terminated set of the new system state.

Note that whenever a transaction terminates, it creates a new version value for each variable name in its writeset. Thus, more than one version value may coexist for each variable name in the current database system state, including the initial version value of each variable name of the initial database system state.

DEFINITION 3.4. : Database system state $Q1 = (TE1, TT1, TR1, s1)$ is a valid request of of database system state $Q = (TE, TT, TR, s)$ iff:

(C3.1.) $TT1 = TT$ and $TE1 = TE$, and
(C3.2.) $TR \subseteq TR1$, and
(C3.3.) For all i, j, $Ti \in (TE \cup TT)$, $Tj \in (TE \cup TT)$:
$\pi(Tj) < \pi(Ti)$ in s <==> $\pi(Tj) < \pi(Ti)$ in s1

Definition 3.4. defines the conditions in which new transactions are admitted into the requesting set of transactions.

DEFINITION 3.5. : A database system state $Q = (TE, TT, TR, s)$ is a valid system state iff one of the following conditions are satisfied :

(D3.1.) $TE = 0$ and $TT = 0$ and $TR = 0$ and s is empty (called the initial system state), or
(D3.2.) Q is a valid extension of a valid system state, or
(D3.3.) Q is a valid termination of a valid

system state, or
(D3.4.) Q is a valid request of a valid system state.

Definition 3.5. precisely defines which database system states are considered to be valid, i.e., preserve consistency. Here it is assumed that all valid system states are derived from the initial state, i.e., a state in which no transaction exists within the system.
A valid system state corresponds to a state in which for all transactions currently executing in parallel in the database system, we can guarantee that their final writing on the database global variables can be arranged to produce the same overall effect as if they were executed sequentially in the same order as the serial schedule s.

DEFINITION 3.6.: Given the current valid database system state Q = (TE, TT, TR, s), we call the valid database system state Q1 = (TE1, TT1, TR1, s1) a <u>maximum concurrency extension</u> of Q iff :

(E3.1.) Q1 is a valid extension of Q, and
(E3.2.) No other valid system state Q2 = (TE2, TT2, TR2, s2) exists, such that Q2 is also a valid extension of Q, and |TR2| < |TR1|.

In order to have an intuitive notion of how a scheduler would work according to such a model, let us first examine the following example : (All terminated transactions are primed.)

EXAMPLE 3.1. : Fig 1.demonstrates how a database scheduler may transform one system state to another system state while preserving serializability of all transactions currently executing in parallel.
Below, some explanation may be useful :
In the transformation from database system state 3 to state 4, the scheduler is able to put transaction T2 into execution in parallel with T1, because there exists a virtual ordering identical to the serial schedule s4, where all the values corresponding to the variable names in T2´s readset are available, and T1 reads the

Fig.1.

i : database system state.
TE : executing set of transactions.
TT : terminated set of transactions.
TR : requesting set of transactions.
si : virtual ordering.

i	TE	TT	TR	si
0	0	0	0	----
1	0	0	T1=([x],[y])	----
2	T1	0	0	s2 = T1 = R1[x]W1[y]
3	T1	0	T2=([c,y],[b,y])	s3 = s2
4	T1, T2	0	0	s4 = T2 T1 = R2[c,y]W2[b,y]R1[x]W1[y]
5	T1, T2	0	T3=([y,c],[f,d,c]) T4=([f,y],[x,y]) T5=([y,d],[e,y])	s5 = s4
6	T2	T1	T3, T4, T5	s6 = T2 T1´
7	T2, T4, T5	T1	T3	s7 = T2 T1´T4 T5 = R2[c,y]W2[b,y]R1[x]W1´[y] R4[f,y]W4[x,y]R5[y,d]W5[e,y]
8	T4, T5	T1, T2	T3	s8 = T2´T1´T4 T5
9	T4, T5	T1, T2	T3, T6=([y],[y])	s9 = s8
10	T4, T5, T6	T1, T2	T3	s10 = T2´T6 T1´T4 T5 = R2[c,y]W2´[b,y]R6[y]W6[y]R1[x]W1´[y] R4[f,y]W4[x,y]R5[y,d]W5[e,y]

same values as in the previous virtual ordering s3. Note that T2 is not restricted to read the most recent version value of y in its readset in this first model. Note also that T2 cannot be executed according to the virtual ordering s = T1 T2, because in that virtual ordering T2 is supposed to read the the value of the variable name y in T1´s writeset, which has not yet been produced, since T1 has not yet terminated.

In database system state 5, no transaction among the requesting set can be executed, because none of them can be inserted into the serial schedule without creating a virtual ordering in which at least one transaction is supposed to read a value which has not yet been produced.

In the transformation from database system state 6 to state 7, since T1 has terminated previously, each single transaction in the requesting set can be put in execution by reading the values produced by T1. But because T4 and T5 is the largest subset of all requesting transactions which can be simultaneously put into execution in parallel with T2, we chose T4 and T5 to be executed first. On the contrary, if we chose T3 for execution first, then both T4 and T5 would be blocked from execution, since no virtual ordering can be found which allows T4 and T5 to be executed in parallel with T3.

In the transformation from database system state 9 to state 10, T6 is put into execution by reading an old value of y from T2. Note that there exists no virtual ordering in which T6 can read the most recent value of y (produced by T1) and which is consistent with the previous virtual ordering s9.

And we can continue like this constructing successive new valid system states.

In this example, we obtain the following valid system states :

Q0 = (0, 0, 0, <>,) (the initial system state)

Q1 = (0, 0, {T1}, <>) (T1 requests) Q1 is a valid request of Q0 by definition 3.4.

Q2 = ({T1}, 0, 0, <T1>) (T1 is put into execution) Q2 is a valid extension of Q1 by definition 3.2.

Q3 = ({T1}, 0, {T2},<T1>) (T2 requests)

Q4 = ({T1, T2}, 0, 0, <T2 T1>) (T2 is put into execution in parallel with T1)

Q5 = ({T1, T2}, 0, {T3, T4, T5}, <T2 T1>) (T3, T4, T5 requests)

Q6 = ({T2}, {T1}, {T3, T4, T5}, <T2 T1´>) (T1 terminates) Q6 is a valid termination of Q5 by definition 3.3.

Q7 = ({T2, T4, T5}, {T1}, {T3}, <T2 T1´ T4 T5>) (T4 and T5 are put into execution in parallel with T2) Q7 is a valid extension of Q6 by definition 3.2.

Q8 = ({T4, T5}, {T1, T2}, {T3}, <T2´T1´T4 T5>) (T2 terminates)

Q9 = ({T4, T5} {T1, T2}, {T3, T6}, <T2´T1´T4 T5>) (T6 requests)

Q10 = ({T4, T5, T6}, {T1, T2}, {T3}, <T2´T6 T1´T4 T5> (T6 is put into execution in parallel with T4 and T5).

Note that all valid extensions in this example are maximum concurrency extensions of the previous state.

4. THE COMPUTATION COMPLEXITY

Since we are interested in obtaining practical schedulers, it is necessary to study the computation complexity involved in constructing valid extensions of a given valid system state.

The following theorem suggests that general valid extensions may be impractical to obtain when the number of transactions is very large : (see Appendix for proof)

THEOREM 4.1. : The following problem (VE) is NP complete :

Given a valid system state Q = (TE, TT, TR, s), does there exist a valid system state Q1 = (TE1, TT1, TR1, s1) such that Q1 is a valid extension of Q and TE1 = (TE $\cup$ TR) and TR1 = 0 ?

Theorem 4.1. states that given an arbitrary valid system state Q, the problem of whether there exists a valid system state Q1, such that all requesting transactions in Q are put into execution in parallel with all currently executing transactions while guaranteeing that their final writing on the database global variables can be arranged to produce the same overall effect as if all transactions in the system were executed sequentially is NP complete.

The theorem above suggests that even for the sole reason of reducing computation complexity, it would prove beneficial to investigate possible restrictions to obtain subsets of valid extensions of a given valid system state. Below, we define several subsets of valid extensions in increasing order of restrictiveness and study their implications on performance and computation complexity.

For each restrictive condition defined below, we shall show by example one case in which the restrictive condition prevents a transaction from being immediately put into execution, whereas if the condition is removed, then that transaction could be immediately executed in parallel with all transactions currently in execution.

Conventional concurrency control schemes impose a fixed explicit total ordering of all existing version values of each data variable. This implies a fixed ordering of all terminated transactions which nave produced different version values of the same variable. If we adopt this restriction, then we have the following definition :

DEFINITION 4.1.: Database system state Q1 = (TE1, TT1, TR1, s1) is a <u>valid fixed terminated write position extension</u> (FTWP) of database system state Q = (TE, TT, TR, s), iff :

(F4.1.) Q1 is a valid extension of Q, and

(F4.2.) For all i, j,:

if $SWi \cap SWj \neq 0$ and ($Ti \in TT$ and $Tj \in TT$) then:
$\pi(Wi) < \pi(Wj)$ in s $\Longleftrightarrow$ $\pi(Wi) < \pi(Wj)$ in s1

Condition (F4.2.) states that if two transactions have terminated and their writesets intersect, then their relative ordering in a valid system state is restricted to be kept invariant when constructing the new valid system state, i.e. the FTWP of Q.

Below, we show by an example how condition (F4.2.) restricts concurrency :

EXAMPLE 4.1.:

Suppose we have a valid system state Q = ({T3}, {T1, T2}, {T4}, <T1´T2´T3>) where T1 = ([d], [x,a]), T2 = ([d,f], [x]), T3 = ([d], [f,a]) T4 = ([a], [d]).

One can check (e.g.,by exhaustive checking of all possible serial schedules), that there does not exist any valid system state Q1 = (TE1, TT1, TR1, s1) such that Q1 is a FTWP of Q.

But there exists a valid system state Q2 = ({T3, T4}, {T1, T2}, 0, <T2´T3 T1´T4>) such that Q2 is a valid extension of Q.

Alternatively speaking, condition (F4.2.) prevents T4 from being immediately put into execution, whereas if condition (F4.2.) is removed, then T4 can be executed in parallel with T3, while guaranteeing serializability of all transactions executing in the database system.

In definition 3.2., when a requesting transaction is put into execution, it may read any one out of existing version values for each variable name in its readset.

There may well exist applications in which reading an "old" version of a data item is not acceptable, or in which the extra cost of memory required to store multiversions of data is considered excessive. In such cases, we have the following more restrictive definition of a valid extension :

DEFINITION 4.2.: Database system state Q1 = (TE1, TT1, TR1, s1) is a valid latest read version extension (LTRD) of database system state Q = (TE, TT, TR, s), iff :

(G4.1.) Q1 is a FTWP of Q, and
(G4.2.) For all $Tj \in (TE1 - TE)$, $Ti \in TT$, $x \in V$:
if Rj reads x from Wi in s1, then there exists no k, $Tk \in TT$, such that $x \in SWk$ and $\pi(Wk) > \pi(Wi)$ in s.

Condition (G4.2.) states that when a requesting transaction is put into execution, for each variable name in its readset, it is restricted to read the "latest" available version value in the virtual ordering.

EXAMPLE 4.2.

Suppose we have the valid system state Q = ({T3}, {T1, T2}, {T4}, <T1´T2´T3>) where T1 = ([b], [x]), T2 = ([b], [x,a]), T3 = ([a], [x]), T4 = ([x], [a]). It is easy to check that there does not exist any valid system state Q1 = ({T3, T4}, {T1, T2}, 0, s1) such that Q1 is a LTRD of Q. But there exists a valid system state Q2 = ({T3, T4}, {T1, T2}, 0, <T1´T4 T2´T3>) such that Q2 is a valid extension of Q.

In other words, condition (G4.2.) prevents T4 from being immediately put into execution, whereas if condition (G4.2.) is removed, then T4 can be executed in parallel with T3, while guaranteeing serializability of all transactions executing in the database system.

In conventional concurrency control schemes, the restriction (H4.2) below is also imposed :

DEFINITION 4.3.: Database system state Q1 = (TE1, TT1, TR1, s1) is a valid invariant write position extension (IVWP) of database system state Q = (TE, TT, TR, s), iff :

(H4.1.) Q1 is a valid extension of Q, and
(H4.2.) For all i, j, x, $Ti \in (TT \cup TE)$, $Tj \in (TT \cup TE)$, $x \in V$:
if $SWi \cap SWj \neq 0$ and ($Ti \in TT$ or $Tj \in TT$) then:
$\pi(Wi) < \pi(Wj)$ in s $\Longleftrightarrow$ $\pi(Wi) < \pi(Wj)$ in s1

Condition (H4.2.) states that if the writesets of two transactions intersect, and at least one of them has terminated, then their relative ordering in the previous valid system state is restricted to be kept invariant when constructing the new valid system state, i.e. the IVWP of Q.

EXAMPLE 4.3. : Suppose we have a valid system state Q = ({T2}, {T1}, {T3}, <T1´ T2>) where T1 = ([a], [x]), T2 = ([a], [x]), T3 = ([x], [a]). It is easy to check that there does not exist any valid system state Q1 = ({T2, T3}, {T1}, 0, s1} such that Q1 is a IVWP of Q. But there exists a valid system state Q2 = ({T2, T3}, {T1}, 0, <T2 T1´T3>} such that Q2 is a valid extension of Q.

In this example, condition (H4.2.) prevents T3 from being immediately put into execution, whereas if condition (H4.2.) is removed, then T3 can be executed in parallel with T2, while guaranteeing serializability of all transactions executing in the database system.

In all previous definitions, a requesting transaction may write a value which is to be overwritten by a terminated transaction coming later in the virtual ordering. This may result in new transactions being inserted before a terminated transaction in the virtual ordering. One of the possible ways of preventing this phenomenon is to adopt the following restrictive definition :

DEFINITION 4.4.: Database system state Q1 = (TE1, TT1, TR1, s1) is a valid uptodate write extension (UPDW) of database system state Q = (TE, TT, TR, s), iff :

(I4.1.) Q1 is a valid extension of Q, and
(I4.2.) For all $Tj \in (TE1 - TE)$, $x \in V$:
if $x \in SWj$, then there exists no $Tk \in TT$,

such that $x \in SWk$ and $\pi(Wj) < \pi(Wk)$ in s1.

Condition (I4.2.) states that when a requesting transaction is put into execution, for each variable name in its writeset, if any terminated transaction produced a value for that variable then the requesting transaction must be positioned after (to the right of) that terminated transaction in the schedule. This restricts that no transaction is to produce a value which is to be overwritten by a terminated transaction even before it has started.

EXAMPLE 4.4.
Suppose we have the valid system state Q = ({T2}, {T1}, {T3}, <T1´T2>) where T1 = ([b], [a]), T2 = ([a], [x]), T3 = ([x], [a]) there does not exist any valid system state Q1 = ({T2, T3}, {T1}, 0, s1) such that Q1 is a UPDW of Q. But there exists valid system state Q2 = ({T2, T3}, {T1}, 0, <T3 T1´T2>) such that Q2 is a valid extension of Q.
Here, condition (I4.2.) prevents T3 from being immediately put into execution, whereas if condition (I4.2.) is removed, then T3 can be executed in parallel with T2, while guaranteeing serializability of all transactions executing in the database system.

However, the combination of any two single restrictions given above still does not reduce the computation complexity of our problem to an acceptable level, as evidenced by the following three theorems :
(see Appendix for the proofs)

THEOREM 4.2. : The following problem (LI) is NP complete :
Given a valid system state Q = (TE, TT, TR, s), does there exist a valid system state Q1 = (TE1, TT1, TR1, s1) such that Q1 is both a LTRD and a IVWP of Q and TE1 = (TE $\cup$ TR) ?

THEOREM 4.3. : The following problem (LU) is NP complete :
Given a valid system state Q = (TE, TT, TR, s), does there exist a valid system state Q1 = (TE1, TT1, TR1, s1) such that Q1 is both a LTRD and a UPDW of Q and TE1 = (TE $\cup$ TR) ?

THEOREM 4.4. : The following problem (UI) is NP complete :
Given a valid system state Q = (TE, TT, TR, s), does there exist a valid system state Q1 = (TE1, TT1, TR1, s1) such that Q1 is both a UPDW and a IVWP of Q and TE1 = (TE $\cup$ TR) ?

THEOREM 4.5. : All the problems stated in the theorems 4.2., 4.3., 4.4. above are NP complete, even if there exists no more than two version values for each variable name in the valid system state Q = (TE, TT, TR, s). (including the initial version value of each variable name of the initial database system state). That is, even if the following restriction is imposed in Q:

(L4.1.) for each x, $x \in V$ and for all $Ti \in TT$ such that $x \in SWi$: $|\{Ti\}| \leqslant 1$

Only by combining all the restrictions given above together (with the exception of (L4.1.)), we manage to substantially reduce the computation complexity of our problem :

DEFINITION 4.5.: Database system state Q1 = (TE1, TT1, TR1, s1) is a <u>valid fixed write position uptodate read write extension</u> (FWRW) of database system state Q = (TE, TT, TR, s), iff :
(K4.1.) Q1 is a LTRD of Q, and
(K4.2.) Q1 is a UTDW of Q, and
(K4.3.) Q1 is a IVWP of Q.

DEFINITION 4.6. We call a directed graph G = (X, U), the "<u>FWRW dependency graph</u>" of a valid system state Q = (TE, TT, TR, s) iff in Q :

$X = \{xi \mid Ti \in (TT \cup TE \cup TR)\}$,
$U = \{(xi, xj) \in X \times X \mid (i \neq j)$ and
(
(J4.1.) $(SWj \cap SRi \neq 0)$ and $(Tj \in TT)$ and $(Ti \in TE)$ and $(\pi(Tj) < \pi(Ti)$ in s)
(J4.2.) $(SWj \cap SWi \neq 0)$ and $((Tj \in TT)$ or $(Ti \in TT))$ and $(\pi(Tj) < \pi(Ti)$ in s)
(J4.3.) $(SRi \cap SWj \neq 0)$ and $(Ti \in TR)$ and $(Tj \in TT)$
(J4.4.) $(SWj \cap SWi \neq 0)$ and $(Ti \in TR)$ and $(Tj \in TT)$
(J4.5.) $(SWi \cap SRj \neq 0)$ and $(Ti \in TR)$ and $(Tj \in (TT \cup TE \cup TR))$
(J4.6.) $(SWi \cap SRj \neq 0)$ and $(Tj \in TR)$ and $(Ti \in TE)$ and
(there does not exist k = 1,2,...,p, such that :
$$(SWi \cap SRj) \subseteq (\bigcup_{k=1}^{p} SWk) \text{ and}$$
for all k = 1,2,...,p :
$Tk \in TT$ and $\pi(Ti) < \pi(Tk)$

THEOREM 4.6.
Given a valid system state Q = (TE, TT, TR, s), there exists a valid system state Q1 = (TE1, TT1, TR1, s1) such that Q1 is a FWRW of Q and TE1 = (TE $\cup$ TR) and TR1 = 0, if and only if the FWRW dependency graph of Q is acyclic.

Proof :
Suppose G is acyclic, then according to graph theory [1], the set of nodes in G can be totally ordered, such that if $(xi, xj) \in U$, then xi is ordered after xj. Thus, we can obtain a serial schedule s1 of the set of transactions T = $(TT \cup TE \cup TR)$, in which if $(xi, xj) \in U$, then $\pi(Tj) < \pi(Ti)$. We can prove that the valid system state Q1 = (TE1, TT1, TR1, s1) is a FWRW of Q where TE1 = (TE $\cup$ TR) and TR1 = 0.
To see this, we can verify that the following assertions hold for s and s1 :

for all i, j, $(SWj \cap SRi \neq 0)$ and $(Tj \in TT)$ and $(Ti \in TE)$ and $(\pi(Tj) < \pi(Ti))$ in s :

$\pi(Tj) < \pi(Ti)$ in s1 (e1)
for all i, j, $(SWj \cap SWi \neq 0)$ and $((Tj \in TT)$ or $(Ti \in TT))$ and $(\pi(Tj) < \pi(Ti))$ in s :
$\pi(Tj) < \pi(Ti)$ in s1 (e2)
for all i, j, $(SRi \cap SWj \neq 0)$ and $(Ti \in TR)$ and $(Tj \in TT)$:
$\pi(Tj) < \pi(Ti)$ in s1 (e3)
for all i, j, $(SWj \cap SWi \neq 0)$ and $(Ti \in TR)$ and $(Tj \in TT)$:
$\pi(Tj) < \pi(Ti)$ in s1 (e4)
for all i, j, $(SWi \cap SRj \neq 0)$ and $(Ti \in TR)$ and $(Tj \in (TT \cup TE \cup TR))$:
$\pi(Tj) < \pi(Ti)$ in s1 (e5)
for all i, j, k, $(SWi \cap SRj \neq 0)$ and $(Tj \in TR)$ and $(Ti \in TE)$ and
(there does not exist k = 1,2,...,p, such that $(SWi \cap SRj) \subsetneq (\bigcup_{k=1}^{p} SWk)$ and

for all k = 1,2,...,p : $Tk \in TT$ and $\pi(Ti) < \pi(Tk)$ in s
:
$\pi(Tj) < \pi(Ti)$ in s1 (e6)
)

(e1) and (e2) and (e5) ==> (A3.1.3.)
(e2) ==> (F4.2.) and (H4.2.)
(e5) and (e6) and (A3.1.3.) ==> (A3.2.)
(e3) ==> (G4.2.)
(e4) ==> (I4.2.) (see def.4.4.)

Finally, (A3.1.3.) and (F4.2.) and (H4.2.) and (A3.2.) and (G4.2.) and (I4.2.) together with the fact that all transactions in TR are included in s1 imply that Q1 is a FWRW of Q.

Suppose G contains a loop. Then there does not exist any total ordering at all, such that if $(xi, xj) \in U$, then xi is ordered after xj. This implies that there exists no serial schedule s1 of the set of transactions $T = TT \cup TE \cup TR$, such that if $(xi, xj) \in U$, then $\pi(Tj) < \pi(Tj)$. This in turn implies that (A.3.1.3.) and (F4.2.) and (H4.2.) and (A3.2.) and (G4.2.) and (I4.2.) cannot simultaneously hold for any serial schedule s1, such that Q1 = (TE1, TT1, TR1, s1) is a FWRW of Q.

Q.E.D.

Theorem 4.6. states that given an arbitrary valid system state Q = (TE, TT, TR, s), the problem of deciding whether there exists a valid system state Q1 = (TE1, TT1, TR1, s1), such that Q1 is a FWRW of Q, and all requesting transactions are put into execution in Q1 while guaranteeing serializability can be reduced to the problem of determining whether the FWRW dependency graph G of Q is acyclic. This can be done quite efficiently, the computation time being $O(|V|\ |X|^3)$ [1].

Now suppose the FWRW dependency graph G of Q is cyclic, what can we do then ? An optimal solution can be found by finding the largest subset TRs of the requesting set of transactions TR, such that the subgraph Gs of G is acyclic and Gs is obtained by removing nodes belonging to the requesting set (TR - TRs). Then a virtual schedule s1 in which all requesting transactions in that largest subset TRs can be put into execution in Q1 can be constructed by topological sorting Gs [1] .

This is explained by an example below :

EXAMPLE 4.5.

Suppose initially we have a set of 4 requesting transactions: T1 = ([a], [b]), T2 = ([b,x], [a,y]), T3 = ([b], [a,x]), T4 = ([y,b], [a]). and we start at a time when there is no activity at all in the database.

At this moment, the database system state is Q1 = (0, 0, {T1, T2, T3, T4}, <>)

According to theorem 4.6., since the FWRW dependency graph G1 of Q1 is cyclic, there exists no valid state Q2 = ({T1, T2, T3, T4}, 0, 0, s2) such that Q2 is a FWRW of Q1 and the whole set of requesting transactions TR1 can be put into execution in parallel in Q2 while preserving serializability. Here, the largest subset TR1s of TR1 for which the subgraph G1s of G1 is acyclic and the nodes removed from G1 belong to (TR1-TR1s) contains 3 transactions, i.e. T2, T3, T4. Thus we determine that in order to achieve maximum concurrency, TR1s = {T2, T3, T4} should be put into execution in parallel first.

By topological sorting G1s, we can find at least one schedule s3 such that a new valid system state Q3 = ({T2, T3, T4}, 0, {T1}, s3) can be constructed, in which T2, T3, and T4 can be put into execution in parallel, and Q3 is a FWRW of Q1. In this example

s3 = T4 T2 T3
= R4[y,b]W4[a]R2[x,b]W2[a,y]R3[b]W3[a,x]

and we can continue like this constructing successive new valid system states. Notice that Q3 is a maximum concurrency extension of Q1.

The problem of finding an optimal solution in the example above, can be transformed to the Feedback Vertex Set (FVS) problem [6][8]. Although the FVS problem is known to be NP complete, there exist several factors which imply that higher concurrency by scheduling a whole set of requesting transactions can be achieved at a reasonable computation time cost :
(1) In real world applications, there may exist only a very small number of arcs in the FWRW dependency graph of a valid system state, even if the total number of nodes is very large. Thus, the actual computation time necessary to obtain an optimal solution could be quite short.
(2) We can always limit computation complexity by using efficient heuristics to find good approximations to an optimal solution.
(3) Algorithms actually exist which either find an optimal solution or a suboptimal solution for the FVS problem and which are known by experience to have a good performance. (For example, see [4][7][10]. An algorithm for a suboptimal solution described in [7] has a computation time upper bound of only $O(|X|^3)$.)

In any case, we should be able to obtain more concurrency than any scheduler which schedules only one requesting transaction for execution at a time.

5. SUMMARY

In this paper we presented a formal model for studying the computation complexity of scheduling a whole set of transactions simultaneously in a transaction system with predeclared writesets. Our study clearly shows that there exists a fundamental tradeoff between the amount of concurrency achieved and the computation overhead necessary to achieve that amount of concurrency. However, it is suggested that based on variants of the model introduced here, schedulers which schedule a whole set of transactions simultaneously may still achieve a higher level of concurrency than conventional schedulers within reasonable computation complexity constraints.

ACKNOWLEDGEMENTS

I am greatly indebted to Professors P.J.Courtois and E.Milgrom for reading of this paper and many stimulating discussions. Without their help and encouragement, this paper could not have been written. I also wish to thank all members of the Department of Informatics at U.C.L. for kindly helping in numerous ways. The referees provided valuable comments and suggestions.

REFERENCES :

[1] B.Carre, "Graphs and Networks"
Clarendon Press, Oxford, 1979.

[2] M.A.Casanova, "The Concurrency Control Problem for Database Systems"
Lecture Notes in Computer Science 116, Springer-Verlag, 1981.

[3] M.A.Casanova and P.A.Bernstein, "General Purpose Schedulers for Database Systems".
Acta Informatica 14, 195-220 (1980).

[4] M.Diaz et al, "A note on minimal and quasiminimal essential sets in complex directed graphs"
IEEE Trans. Circuit Theory, vol CT-19, pp.512, Sept, 1972.

[5] K.P.Eswaran et al, "The Notions of Consistency and Predicate Locks in a Database System"
CACM, 19, 11, Nov, 1976. 624-633.

[6] M.R.Garey and D.S.Johnson, "Computers and Intractability : A Guide to the Theory of NP-Completeness".
Freeman, San Francisco, 1979.

[7] G.Guardabassi, "A note on minimal essential sets"
IEEE Trans. Circuit Theory, vol CT-18, pp.557, Sept, 1971.

[8] R.M.Karp, "Reducibility Among Combinatorial Problems"
in R.E.Miller and J.W.Thatcher (eds), "Complexity of Computer Computation", Plenum Press, New York, 85-103. 1972

[9] C.H.Papadimtriou, "The Serializability of Concurrent Database Updates"
JACM, 26, 4, Oct, 1979. 631-653.

[10] G.W.Smith and R.B.Walford, "The Identification of a Minimal Feedback Vertex Set of a Directed Graph".
IEEE Trans. on Circuits and Systems, Vol. Cas-22, 1, Jan, 1975. 9-15.

APPENDIX :

1. PROOF OF THEOREM 4.1.

Proof :

It is easy to see that VE $\in$ NP, because a non deterministic algorithm need only guess a new valid system state Q1, in which s1 is a serial schedule of T = (TE $\cup$ TT $\cup$ TR) and check in polynomial time that Q1 is a valid extension of Q.

Further more, it is easy to see that VE contains LI (see theorem 4.2. and def.4.2. and def.4.3.) as a special case. Since LI is proved below to be NP complete, VE is also a NP complete problem. (Q.E.D.)

2. PROOF OF THEOREM 4.2.

(see example at end of proof)

Proof :

It is easy to see that LI $\in$ NP, because a non deterministic algorithm need only guess a new valid system state Q1, in which s1 is a serial schedule of T = (TE $\cup$ TT $\cup$ TR) and check in polynomial time that Q1 is both a LTRD and a IVWP of Q.

Below, we accomplish our proof by transforming a well known NP complete problem --- the " GRAPH K-COLORABILITY " problem (GKC) [5][6] to LI.

GKC is as follows : Given a graph G = (V, E) and a positive integer K $\leqslant$ N, where N = $|V|$, determine whether graph G is K-colorable, i.e., is it possible to assign each node in G one out of K colors, such that no two connected nodes are assigned the same color ?

Suppose G = (V, E) and a positive integer K $\leqslant$ N, where N = $|V|$, is an arbitrary instance of GKC. We now construct a valid system state Q = (TE, TT, TR, s), such that there exists a valid system state Q1 = (TE1, TT1, TR1, s1) which is both a LTRD and a IVWP of Q and TE1 = (TE $\cup$ TR) if and only if G is K-colorable.

We construct " One in K choice components ". Each " One in K choice component " corresponds to one node in the graph G. We call the " One in K choice component " corresponding to node i in G as " One in K choice component i ". Each One in K choice component i is in turn composed of K " triangle components ". Each triangle component corresponds to one color. We call the triangle component in One in K choice component i corresponding to color j as triangle component [i,j].

Each triangle component [i,j] is composed of 3 transactions : One terminated transaction Tt[i,j] = ([SRt[i,j]], [SWt[i,j]]), one executing transaction Te[i,j] = ([SRe[i,j]], [SWe[i,j]]) and one requesting transaction Tr[i,j] = ([SRr[i,j]], [SWr[i,j]]).

In each triangle component [i,j], i = 1, 2, ..., N, j = 1, 2, ..., K, we let :

(O1) $a[i,j] \in SWt[i,j]$ $SWe[i,j] \cap SRr[i,j] = 0$
$a[i,j] \in SWr[i,j]$ $SWr[i,j] \cap SRt[i,j] = 0$
$a[i,j] \in SRe[i,j]$

In each One in K choice component i, i = 1, 2, ..., N, we let :

(O2) $b[i,j,k] \in SWe[i,j]$
$b[i,j,k] \in SRr[i,j]$
for all j, k : $1 \leqslant j,k \leqslant K$ and $j \neq k$

(O3) $c[i,j+1,j] \in SWr[i,j+1]$
$c[i,j+1,j] \in SRt[i,j]$
for all j : $1 \leqslant j \leqslant K-1$

$c[i,1,K] \in SWr[i,1]$
$c[i,1,K] \in SRt[i,K]$

For all $1 \leqslant p,q \leqslant N$, $p \neq q$ and node p and node q are connected, we let :

(O4) $d[p,q,j] \in SWe[p,j]$
$d[p,q,j] \in SRr[q,j]$
for all j : $1 \leqslant j \leqslant K$

We further construct a serial schedule

s = Tt[1,1]Te[1,1]Tt[1,2]Te[1,2] ...
... Tt[i,j]Te[i,j] ...Tt[N,K]Te[N,K]

Notice that in s :

(P1) for all i, j, i = 1, 2, ..., N,
j = 1, 2, ..., K :
Te[i,j] reads a[i,j] from Tt[i,j]

Then we collect all the transactions constructed above together to form the three sets :

TE = {Te[i,j]} for all i, j :
TT = {Tt[i,j]} i = 1, 2, ..., N
TR = {Tr[i,j]} j = 1, 2, ..., K

It is not difficult to see that the valid system state Q = (TE, TT, TR s) thus constructed can be constructed in polynomial time.

We now prove that there exists a valid system state Q1 = (TE1, TT1, TR1, s1) which is both a LTRD and a IVWP of Q and TE1 = (TE $\cup$ TR) if and only if G is K-colorable.

By the way we have constructed triangle component [i,j], it is easy to verify that in any serial schedule s1 which includes Tr[i,j], Tt[i,j] and Te[i,j] and Q1 is a valid extension of Q, one and only one of the following formulas must hold for the three transactions Tr[i,j], Tt[i,j] and Te[i,j] in each triangle component [i,j] :

(M1) $\pi(Tr[i,j]) < \pi(Tt[i,j]) < \pi(Te[i,j])$

(exclusive) or

(M2) $\pi(Tt[i,j]) < \pi(Te[i,j]) < \pi(Tr[i,j])$

This is because if neither (M1) nor (M2) holds in s1, then (P1) will not hold in s1, which violates (A3.1.3.), and Q1 would not be a valid extension of Q.

We now prove that in any serial schedule s1, such that Q1 is both a LTRD and a IVWP of Q, in each One in K choice component i, (M2) holds for one and only one triangle component [i,j].

Suppose in One in K choice component i, (M2) holds for any two triangle components : triangle component [i,j] and triangle component [i,k].

This means $\pi(Te[i,j]) < \pi(Tr[i,j])$ (1)
and $\pi(Te[i,k]) < \pi(Tr[i,k])$ (2)

But according to (O2) : $b[i,j,k] \in SWe[i,j]$
$b[i,j,k] \in SRr[i,k]$

which implies that in s1 of Q1,
$\pi(Tr[i,k]) < \pi(Te[i,j])$ (3)

must hold, otherwise Te[i,j] will read b[i,j,k] from Tr[i,k], which violates (A3.2.).
Similarly, according to (O2) the following must also hold :

$\pi(Tr[ij]) < \pi(Te[i,k])$ (4)

But (3) and (4) contradicts (1) and (2).

Next, suppose in One in K choice component i, (M2) does not hold for any triangle component, then (M1) must hold for all triangle components :

for j = 1, 2, ..., K :
$\pi(Tr[i,j]) < \pi(Tt[i,j])$ (5)

But according to (O3) :
$c[i,j+1,j] \in SWr[i,j+1]$
$c[i,j+1,j] \in SRt[i,j]$
for all j : $1 \leqslant j \leqslant K-1$

$c[i,1,K] \in SWr[i,1]$
$c[i,1,K] \in SRt[i,K]$

which implies that in any s1 of Q1 :
$\pi(Tt[i,j]) < \pi(Tr[i,j+1])$

for all j : $1 \leqslant j \leqslant K-1$

$$\pi(Tt[i,K]) < \pi(Tr[i,1]) \quad (6)$$

must hold, otherwise Tt[i,j] will read c[i,j+1,j] from Tr[i,j+1] and Tt[i,k] will read c[i,1,k] from Tr[i,1], which violates (A3.2.).
But (5) and (6) leads to a contradiction.

Now we prove that if node p and node q are connected, then in s1 of Q1, (M2) cannot simultaneously hold for any two triangle components in One in K choice components p and q which correspond to the same color x.
Suppose the contrary, then we have

$$\pi(Te[p,x]) < \pi(Tr[p,x]) \quad (7)$$

and $\pi(Te[q,x]) < \pi(Tr[q,x])$ (8)

But according to (04), we have

$d[p,q,j] \in SWe[p,j]$

$d[p,q,j] \in SRr[q,j]$ for all j : $1 \leqslant j \leqslant K$

which implies that in any s1 of Q1 :

$$\pi(Tr[q,x]) < \pi(Te[p,x]) \quad (9)$$

and $\pi(Tr[p,x]) < \pi(Te[q,x])$ (10)

must hold, otherwise either Tr[q,x] will read d[p,q,x] from Te[p,x], or Tr[p,x] will read d[q,p,x] from Te[q,x], which violates (A3.2.).
But (9) and (10) contradicts (7) and (8).

So far we have proved that in any serial schedule s1 of valid system state Q1 = (TE1, IVWP of Q, in each One in K choice component, (M2) must hold for one and only one triangle ed, then (M2) cannot simultaneously hold for two triangle components in One in K choice components p and q which correspond to the same color.
Suppose there exists Q1 which is both a LTRD and a IVWP of Q, we set node i to color j iff (M2) holds for triangle component [i,j] in s1 in Q1. Then each node will be set to one and only one color, and connected nodes will be set to different colors.

Conversely, we show that if the graph G is K-colorable, then for valid system state Q = (TE, TT, TR, s), there exists a valid system state Q1 = (TE1, TT1, TR1, s1) such that Q1 is both a LTRD and IVWP of Q, and TE1 = (TE $\cup$ TR).

Suppose that graph G = (X, U) is K-colorable.
We construct a graph G1 = (X1, U1) as follows:

X1 = ({xr[i,j]} $\cup$ {xt[i,j]} $\cup$ {xe[i,j]}) for $1 \leqslant i \leqslant N$, $1 \leqslant j \leqslant K$.

U1 =

{(xe[i,j], xt[i,j]) ¦ $1 \leqslant i \leqslant N$, $1 \leqslant j \leqslant K$} (Y1)

{(xr[i,j], xe[i,j]) ¦ $1 \leqslant i \leqslant N$, $1 \leqslant j \leqslant K$: node i is colored j} (Y2)

{(xt[i,j], xr[i,j]) ¦ $1 \leqslant i \leqslant N$, $1 \leqslant j \leqslant K$: node i is NOT colored j} (Y3)

{(xe[i,j], xr[i,k]) ¦ $1 \leqslant i \leqslant N$, $1 \leqslant j,k \leqslant K$ and $j \neq k$} (Y4)

{(xr[i,j+1], xt[i,j]) ¦ $1 \leqslant i \leqslant N$, $1 \leqslant j \leqslant K-1$} (Y5)

{(xr[i,1], xt[i,k]) ¦ $1 \leqslant i \leqslant N$} (Y6)

{(xe[p,j], xr[q,j]) ¦ $1 \leqslant i \leqslant N$, $1 \leqslant j \leqslant K$ and node p and node q are connected} (Y7)

First we show that the graph G1 is acyclic. We divide all nodes in G1 into 3 + 2K - 1 disjoint sets.

SET[1] = {xe[i,j] ¦ $1 \leqslant i \leqslant N$, $1 \leqslant j \leqslant K$: node i is NOT colored j}
SET[2] = {xr[i,j] ¦ $1 \leqslant i \leqslant N$, $1 \leqslant j \leqslant K$: node i is colored j}
SET[3] = {xe[i,j] ¦ $1 \leqslant i \leqslant N$, $1 \leqslant j \leqslant K$: node i is colored j}
SET[4] = {xt[i,j] ¦ $1 \leqslant i \leqslant N$, $1 \leqslant j \leqslant K$: node i is colored (j mod K)+1}
SET[5] = {xr[i,j] ¦ $1 \leqslant i \leqslant N$, $1 \leqslant j \leqslant K$: node i is colored (j mod K)+1}
......

SET[3+2m-1] = {xt[i,j] ¦ $1 \leqslant i \leqslant N$, $1 \leqslant j \leqslant K$: node i is colored ((j + m - 1)mod K) + 1}
SET[3+2m] = {xr[i,j] ¦ $1 \leqslant i \leqslant N$, $1 \leqslant j \leqslant K$: node i is colored ((j + m - 1)mod K) + 1}
......

SET[3+2(K-1)-1] = {xt[i,j] ¦ $1 \leqslant i \leqslant N$, $1 < j < K$: node i is colored
SET[3+2(K-1)] = {xr[i,j] ¦ $1 \leqslant i \leqslant N$, $1 < j < K$: node i is colored
SET[3+2K-1] = {xt[i,j] ¦ $1 \leqslant i \leqslant N$, $1 \leqslant j \leqslant K$: node i is colored j}

It should not be difficult to verify that the nodes in SET[1] have no incoming arcs, the nodes in SET[2] have only incoming arcs from SET[1], the nodes in SET[3] have only incoming arcs from SET[1] and SET[2], ..., the nodes in SET[3+2K-1] have only incoming arcs from SET[1], SET[2], ..., SET[3+2K-2], and the nodes in SET[3+2K-1] has no outgoing arcs. According to graph theory [1], this proves that graph G1 is acyclic.
Since G1 is acyclic, it is possible to renumber all nodes in G1, that is, reassign new indices i = 1, 2, ..., NK to each node in X1, such that if (xi, xj) $\in$ U then $j < i$.
We can then obtain a serial schedule s1, in which if $j < i$ then $\pi(Tj) < \pi(Ti)$, where Tj and Ti are the transactions corresponding to the nodes which have been renumbered j and i.
Now we prove that the valid system state Q1 = (TE1, TT1, TR1, s1) thus obtained is both a LTRD and IVWP of Q = (TE, TT, TR, s) as defined before, and TE1 = (TE $\cup$ TR).
From the construction of the graph G1 defined before, the following assertions hold in s1 :

for all i, j, $1 \leqslant i \leqslant N$, $1 \leqslant j \leqslant K$:

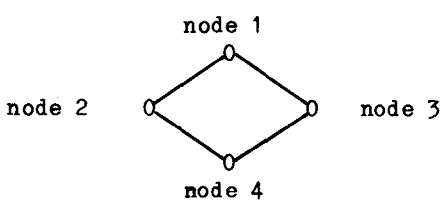
$\pi(Tt[i,j]) < \pi(Te[i,j])$ (y1)
for all i, j, $1 \leq i \leq N$, $1 \leq j \leq K$ and node i is colored j : $\pi(Te[i,j]) < \pi(Tr[i,j])$ (y2)
for all i, j, $1 \leq i \leq N$, $1 \leq j \leq K$ and node i is NOT colored j : $\pi(Tr[i,j]) < \pi(Tt[i,j])$ (y3)
for all i, j, $1 \leq i \leq N$, $1 \leq j,k \leq K$ and $j \neq k$:
$\pi(Tr[i,j]) < \pi(Te[i,k])$ (y4)
for all i, j, $1 \leq i \leq N$, $1 \leq j \leq K-1$:
$\pi(Tt[i,j]) < \pi(Tr[i,j+1])$ (y5)
for all i, j, $1 \leq i \leq N$:
$\pi(Tt[i,K]) < \pi(Tr[i,1])$ (y6)
for all i, j, $1 \leq i \leq N$, $1 \leq j \leq K$ and node p and node q are connected :
$\pi(Tr[q,j]) < \pi(Te[p,j])$ (y7)

From these assertions, we derive :

(y1) and (y2) and (y3) and (01)
==> (P1) holds in s1 in Q1. (y8)
(y4) and (02) ==> for all i, j, k, u, v,
$Tu \in (TE \cup TT \cup TR)$, $Tv \in (TE \cup TT \cup TR)$:
$\neg$(Tu reads b[i,j,k] from Tv in s1) (y9)
(y5) and (y6) and (03) ==> for all i, j, k, u, v, $Tu \in (TE \cup TT \cup TR)$, $Tv \in (TE \cup TT \cup TR)$:
$\neg$(Tu reads c[i,j,k] from Tv in s1) (y10)
(y7) and (04) ==> for all p, q, j, u, v,
$Tu \in (TE \cup TT \cup TR)$, $Tv \in (TE \cup TT \cup TR)$:
$\neg$(Tu reads d[p,q,j] from Tv in s1) (y11)
(y8) and (y9) and (y10) and (y11)
==> (A3.1.) and (A3.2.) (y12)
==> Q1 is a valid extension of Q.
(01) and (02) and (03) and (04)
==> for all u, v, $Tu \in (TE \cup TT \cup TR)$,
$Tv \in (TE \cup TT \cup TR)$: $SWu \cap SWv = 0$
==> (H4.2.) ==> Q1 is a IVWP of Q. (y13)
(y8) and (y9) and (y10) and (y11) and (y12) and (y13) ==> Q1 is a LTRD of Q.

(Q.E.D.)

EXAMPLE :
Suppose we have an instance of the graph G below, and K = 2 :



We construct a valid system state Q = (TE, TT, TR, s), where TE = {Te[i,j]}, TT = {Tt[i,j]}, TR = {Tr[i,j]} $1 \leq i \leq N$, $1 \leq j \leq K$ as following :

Tr[1,1] = ([b[1,2,1], d[2,1,1], d[3,1,1]], [a[1,1], c[1,1,2]])
Tr[1,2] = ([b[1,1,2], d[2,1,2], d[3,1,2]], [a[1,2], c[1,2,1]])
Tr[2,1] = ([b[2,2,1], d[1,2,1], d[4,2,1]], [a[2,1], c[2,1,2]])
Tr[2,2] = ([b[2,1,2], d[1,2,2], d[4,2,2]], [a[2,2], c[2,2,1]])
Tr[3,1] = ([b[3,2,1], d[1,3,1], d[4,3,1]], [a[3,1], c[3,1,2]])
Tr[3,2] = ([b[3,1,2], d[1,3,2], d[4,3,2]], [a[3,2], c[3,2,1]])
Tr[4,1] = ([b[4,2,1], d[2,4,1], d[3,4,1]], [a[4,1], c[4,1,2]])
Tr[4,2] = ([b[4,1,2], d[2,4,2], d[3,4,2]], [a[4,2], c[4,2,1]])

Te[1,1] = ([a[1,1]], [b[1,1,2], d[1,2,1], d[1,3,1]])
Te[1,2] = ([a[1,2]], [b[1,2,1], d[1,2,2], d[1,3,2]])
Te[2,1] = ([a[2,1]], [b[2,1,2], d[2,1,1], d[2,4,1]])
Te[2,2] = ([a[2,2]], [b[2,2,1], d[2,1,2], d[2,4,2]])
Te[3,1] = ([a[3,1]], [b[3,1,2], d[3,1,1], d[3,4,1]])
Te[3,2] = ([a[3,2]], [b[3,2,1], d[3,1,2], d[3,4,2]])
Te[4,1] = ([a[4,1]], [b[4,1,2], d[4,2,1], d[4,3,1]])
Te[4,2] = ([a[4,2]], [b[4,2,1], d[4,2,2], d[4,3,2]])

Tt[1,1] = ([c[1,2,1]], [a[1,1]])
Tt[1,2] = ([c[1,1,2]], [a[1,2]])
Tt[2,1] = ([c[2,2,1]], [a[2,1]])
Tt[2,2] = ([c[2,1,2]], [a[2,2]])
Tt[3,1] = ([c[3,2,1]], [a[3,1]])
Tt[3,2] = ([c[3,1,2]], [a[3,2]])
Tt[4,1] = ([c[4,2,1]], [a[4,1]])
Tt[4,2] = ([c[4,1,2]], [a[4,2]])

The serial schedule s is as following :
s =Tt[1,1]Te[1,1]Tt[1,2]Te[1,2]...Tt[4,2]Te[4,2]

Since the graph G above is K=2 colorable, we can assign color 1 to node 1 and node 4, and assign color 2 to node 2 and node 3. Then we can construct G1 = (X1, U1), and divide all nodes in X1 into 3 + 2(2) - 1 = 6 disjoint sets :

SET[1] = {xe[1,2], xe[2,1], xe[3,1], xe[4,2]}
SET[2] = {xr[1,1], xr[2,2], xr[3,2], xr[4,1]}
SET[3] = {xe[1,1], xe[2,2], xe[3,2], xe[4,1]}
SET[4] = {xt[1,2], xt[2,1], xt[3,1], xt[4,2]}
SET[5] = {xr[1,2], xr[2,1], xr[3,1], xr[4,2]}
SET[6] = {xt[1,1], xt[2,2], xt[3,2], xt[4,1]}

It is easy to verify that the valid system state Q1 = (TE1, TT1, TR1, s1) where TE1 = $(TE \cup TR)$ = $(\{Te[i,j]\} \cup \{Tr[i,j]\})$, TT1 = TT = {Tt[i,j]}, TR1 = 0 and,

s1 = < Tt[i,j] | xt[i,j] $\in$ SET[6] >
< Tr[i,j] | xr[i,j] $\in$ SET[5] >
< Te[i,j] | xe[i,j] $\in$ SET[4] >
< Tt[i,j] | xt[i,j] $\in$ SET[3] >
< Tr[i,j] | xr[i,j] $\in$ SET[2] >
< Tt[i,j] | xt[i,j] $\in$ SET[1] >

= Tt[1,1]Tt[2,2]Tt[3,2]Tt[4,1]
Tr[1,2]Tr[2,1]Tr[3,1]Tr[4,2]
Tt[1,2]Tt[2,1]Tt[3,1]Tt[4,2]

```
Te[1,1]Te[2,2]Te[3,2]Te[4,1]
Tr[1,1]Tr[2,2]Tr[3,2]Tr[4,1]
Te[1,2]Te[2,1]Te[3,1]Te[4,2]
```

is both a LTRD and a IVWP of Q = (TE, TT, TR, s) as defined above.
Note that (M2) i.e. π(Tt[i,j]) < π(Te[i,j]) < π(Tr[i,j]) holds for triangle component [1,1], [4,1], [2,2], [3,2], since node 1 and node 4 was assigned color 1, and node 2 and node 3 was assigned color 2.
Also (M1) i.e. π(Tr[i,j]) < π(Tt[i,j]) < π(Te[i,j]) holds for triangle component [1,2], [4,2], [2,1], [3,1], since node 1 and node 4 was NOT assigned color 2, and node 2 and node 3 was NOT assigned color 1.

3. THE PROOF OF THEOREM 4.3.

proof :

The proof of theorem 4.3 is similar to the proof of theorem 4.2. We transform the " GRAPH K-COLORABILITY " problem to LU in the same fashion as above. Here, for sake of brevity, we shall only show the construction of the valid system state Q = (TE, TT, TR, s), for which there exists a valid system state Q1 = (TE1, TT1, TR1, s1) which is both a LTRD and a UPDW of Q and TE1 = (TE $\cup$ TR) if and only if graph G is K-colorable.

In each triangle component [i,j], i = 1, 2, ..., N, j = 1, 2, ..., K, we let :

(01)´ a[i,j] $\in$ SWt[i,j] SWt[i,j] $\cap$ SRe[i,j] = 0
 a[i,j] $\in$ SRr[i,j] SWe[i,j] $\cap$ SRt[i,j] = 0
 a[i,j] $\in$ SWe[i,j] SWr[i,j] $\cap$ SRe[i,j] = 0

In each One in K choice component i, i = 1, 2, ..., N, we let :

(02)´ b[i,j,k] $\in$ SWr[i,j]
 b[i,j,k] $\in$ SRe[i,j]
 for all j, k : 1 $\leqslant$ j, k $\leqslant$ K and j $\neq$ k

(03)´ c[i,j+1,j] $\in$ SWe[i,j+1]
 c[i,j+1,j] $\in$ SRt[i,j]
 for all j : 1 $\leqslant$ j $\leqslant$ K-1

 c[i,1,K] $\in$ SWe[i,1]
 c[i,1,K] $\in$ SRt[i,K]

For all 1 $\leqslant$ p,q $\leqslant$ N, p $\neq$ q and node p and node q are connected, we let :

(04)´ d[p,q,j] $\in$ SWr[p,j]
 d[p,q,j] $\in$ SRe[q,j]
 for all j : 1 $\leqslant$ j $\leqslant$ K

We further construct a serial schedule

s =
Tt[1,1]Tt[1,2]...Tt[N,K]Te[1,1]Te[1,2]...Te[N,K]

Then we collect all the transactions constructed above together to form the three sets :

TE = {Te[i,j]}
TT = {Tt[i,j]}
TR = {Tr[i,j]}
i = 1, 2, ..., N, j = 1, 2, ..., K

It is not difficult to see that the valid system state Q = (TE, TT, TR s) thus constructed can be constructed in polynomial time.

In the proof of theorem 4.3., the following formulas can be put into one to one correspondence with those in the proof of theorem 4.2. :

(M1)´ π(Te[i,j]) < π(Tt[i,j]) < π(Tr[i,j])
(exclusive) or
(M2)´ π(Tt[i,j]) < π(Tr[i,j]) < π(Te[i,j])

π(Te[i,k]) < π(Tr[i,j]) (3)´
π(Te[i,j]) < π(Tr[i,k]) (4)´

π(Tt[i,j]) < π(Te[i,j+1])
 for all j : 1 $\leqslant$ j $\leqslant$ K-1
and π(Tt[i,K]) < π(Te[i,1]) (6)´

π(Te[q,x]) < π(Tr[p,x]) (9)´
π(Te[p,x]) < π(Tr[q,x]) (10)´

whereas formulas (1)´, (2)´, (5)´, (7)´, (8)´ are directly derived from (M1)´ and (M2)´ above.

The rest of the proof of theorem 4.3. follows exactly the same scheme as that in the proof of theorem 4.2.

4. PROOF OF THEOREM 4.4.

Proof :

The proof of theorem 4.4 is also similar to the proof of theorem 4.2. We transform the " GRAPH K-COLORABILITY " problem to UI in the same fashion as above. Here, for sake of brevity, we shall also only show the the construction of the valid system state Q = (TE, TT, TR, s), for which there exists a valid system state Q1 = (TE1, TT1, TR1, s1) which is both a UPDW and a IVWP of Q and TE1 = (TE $\cup$ TR) if and only if graph G is K-colorable.

In each triangle component [i,j], i = 1, 2, ..., N, j = 1, 2, ..., K, we let :

(01)´´ a[i,j] $\in$ SWt[i,j] SWr[i,j] $\cap$ SRe[i,j] = 0
 a[i,j] $\in$ SRr[i,j] SWr[i,j] $\cap$ SRt[i,j] = 0
 a[i,j] $\in$ SWe[i,j]

In each One in K choice component i, i = 1, 2, ..., N, we let :

(02)´´ b1[i,j,k] $\in$ SWt[i,j]
 b2[i,j,k] $\in$ SRr[i,j]
 for all j, k : 1 $\leqslant$ j, k $\leqslant$ K and j $\neq$ k

(03)´´ c[i,j+1,j] $\in$ SWr[i,j+1]
 c[i,j+1,j] $\in$ SRe[i,j]
 for all j : 1 $\leqslant$ j $\leqslant$ K-1

 c[i,1,K] $\in$ SWr[i,1]
 c[i,1,K] $\in$ SRe[i,K]

For all $1 \leqslant p,q \leqslant N$, $p \neq q$ and node p and node q are connected, we let :

$$(04)'' \quad d1[p,q,j] \in SWt[p,j]$$
$$d2[p,q,j] \in SRr[q,j]$$
$$\text{for all } j : 1 \leqslant j \leqslant K$$

We further construct two sets of auxiliary executing transactions TBE and TDE as follows :

$$TBE = \{Tbe[i,j,k]\} \text{ for all } i,\ j,\ k,\ 1 \leqslant i \leqslant N,$$
$$1 \leqslant j,k \leqslant K \text{ and } j \neq k \text{ and}$$
$$SRbe[i,j,k] = \{b1[i,j,k]\}$$
$$SWbe[i,j,k] = \{b2[i,j,k]\}$$

$$TDE = \{Tde[p,q,j]\} \text{ for all } 1 \leqslant p,q \leqslant N,\ p \neq q$$
and node p and node q are connected, and
$$SWde[p,q,j] = \{d2[p,q,j]\}$$
$$SWde[p,q,j] = \{d2[p,q,j]\}$$

We then construct a serial schedule

$$s = Te[1,1]Te[1,2]\ldots Te[NK]$$
$$Tbe[1,1,2]Tbe[1,1,3]\ldots Tbe[N,K,K-1]$$
$$< Tde[p,q,j] > \quad Tt[1,1]Tt[1,2]\ldots Tt[N,K]$$

Then we collect all the transactions constructed above together to form the three sets :

$$TE = \{Te[i,j]\} \cup TBE \cup TDE$$
$$TT = \{Tt[i,j]\}$$
$$TR = \{Tr[i,j]\}$$
$$i = 1,\ 2,\ \ldots,\ N,\ j = 1,\ 2,\ \ldots,\ K$$

It is not difficult to see that the valid system state Q = (TE, TT, TR s) thus constructed can be constructed in polynomial time.

In the proof of theorem 4.3., the following formulas can be put into one to one correspondence with those in the proof of theorem 4.2. :

$$(M1)'' \quad \pi(Tr[i,j]) < \pi(Te[i,j]) < \pi(Tt[i,j])$$
(exclusive) or
$$(M2)'' \quad \pi(Te[i,j]) < \pi(Tt[i,j]) < \pi(Tr[i,j])$$

$$\pi(Tr[i,k]) < \pi(Tbe[i,j,k]) < \pi(Tt[i,j]) \qquad (3)''$$
$$\pi(Tr[i,j]) < \pi(Tbe[i,k,j]) < \pi(Tt[i,k]) \qquad (4)''$$

$$\pi(Te[i,j]) < \pi(Tr[i,j+1])$$
$$\text{for all } j : 1 \leqslant j \leqslant K-1$$
$$\text{and} \quad \pi(Te[i,K]) < \pi(Tr[i,1]) \qquad (6)''$$

$$\pi(Tr[q,x]) < \pi(Tde[p,q,x]) < \pi(Tt[p,x]) \qquad (9)''$$
$$\pi(Tr[p,x]) < \pi(Tde[q,p,x]) < \pi(Tt[q,x]) \qquad (10)''$$

whereas formulas $(1)''$, $(2)''$, $(5)''$, $(7)''$, $(8)''$ are directly derived from $(M1)''$ and $(M2)''$ above.

The rest of the proof of theorem 4.4. follows exactly the same scheme as that in the proof of theorem 4.2.

5. PROOF OF THEOREM 4.5.

Proof :

It can be easily verified that (L4.1.) holds for each valid system state Q = (TE, TT, TR, s) constructed above in each of the proofs of theorem 4.2., 4.3., and 4.4. (Q.E.D)