Acta Informatica 29, 121–160 (1992)

Acta Informatica

© Springer-Verlag 1992

On-line multiversion database concurrency control

J. Xu*

Department of Computer Science, York University, 4700 Keele Street, North York, Ontario, Canada M3J 1P3

Received July 10, 1990 / September 10, 1991

Abstract. This paper presents a new model for studying the concurrency vs. computation time tradeoffs involved in on-line multiversion database concurrency control. The basic problem that is studied in our model is the following:

Given: *a current database system state* which includes information such as which transaction previously read a version from which other transaction; which transaction has written which versions into the database; and the ordering of versions previously written; and *a set of read and write requests* of requesting transactions.

Question: Does there exist a new database system state in which the requesting transactions can be immediately put into execution (their read and write requests satisfied, or in the case of predeclared writeset transactions, write requests are guaranteed to be satisfied) while preserving consistency under a given set of additional constraints? (The amount of concurrency achieved is defined by the set of additional constraints).

In this paper we derive "limits" of performance achievable by polynomial time concurrency control algorithms. Each limit is characterized by a minimal set of constraints that allow the on-line scheduling problem to be solved in polynomial time. If any one constraint in that minimal set is omitted, although it could increase the amount of concurrency, it would also have the dramatic negative effect of making the scheduling problem NP-complete; whereas if we do not omit any constraint in the minimal set, then the scheduling problem can be solved in polynomial time. With each of these limits, one can construct an efficient scheduling algorithm that achieves an optimal level of concurrency in polynomial computation time according to the constraints defined in the minimal set.

* Current address: Department of Computer Science, York University, 4700 Keele Street, North York, Ontario, Canada M3J 1P3

1 Introduction

This paper studies the performance vs. computation time tradeoff involved in on-line database concurrency control.

The objective of database concurrency control is to allow many users to simultaneously read existing information in a common database, perform local computation based on previously read information, and write new information resulting from local computation into the database, while guaranteeing correctness of the database system.

The database concurrency control problem has received considerable attention in recent years, and a great number of algorithms have been proposed (e.g. [1–4, 7, 10–18, 23]). A principal measure of the performance of a database concurrency control algorithm is the amount of concurrency it achieves while guaranteeing serializability [13, 15, 20]. Generally speaking, the higher the amount of concurrency achieved by a concurrency control algorithm, the greater the computation time required to achieve that amount of concurrency. In fact, the computation time required to achieve concurrency beyond a certain limit may become intolerably high, such that it does not pay off any more to achieve concurrency beyond that limit. Generally, we would want to maximize concurrency while restricting the required computation time to be polynomial (as opposed to exponential). Hence it is important to study the performance vs. computation time tradeoff involved in database concurrency control.

Previous work on this subject includes [14–16]. In [15], the computational complexity involved in database concurrency control is studied in a two step model of transactions. In [14], the two step model of transactions in [15] is extended to a multistep model of transactions. [16] considers the case where multiversions of data exist in the database.

This paper presents a new model for studying the concurrency vs. computation time tradeoffs involved in on-line multiversion database concurrency control. A major difference between our model and previous models is the basic problem that is studied. In previous models, the basic problem that is studied is the following:

Given: a schedule s of a set of transactions representing the *output* of *some* concurrency control algorithm; and a class C of serializable schedules.

Question: Is schedule s in class C? (The amount of concurrency achieved is defined by the restrictions on membership in C).

In contrast the basic problem that is studied in our model is the following:

Given: (i) *a current database system state* which includes information such as which transaction previously read a version from which other transaction; which transaction has written which versions into the database; and the ordering of versions previously written; and (ii) *a set of read and write requests* of requesting transactions. (Note that (i) and (ii) above represent the *input* to a concurrency control algorithm).

Question: Does there exist a new database system state in which the requesting transactions can be immediately put into execution (their read and write requests satisfied, or in the case of predeclared writeset transactions, write requests are

guaranteed to be satisfied) while preserving consistency under a given set of additional constraints? (The amount of concurrency achieved is defined by the set of additional constraints).

In our model, whenever the existence of the new database system state can be determined in polynomial time, a corresponding on-line polynomial time multiversion concurrency control algorithm that actually constructs the new database system state is shown.

In order to obtain a higher amount of concurrency, various optimizing strategies have been designed for database concurrency control. Below, we briefly introduce some of the optimizing strategies studied in this model.

In many cases, user transactions are able to specify in advance the names of the database entities they intend to write new information into. Such transactions are called predeclared writeset transactions. Previous work has shown that it is possible to increase the performance of a database system by using a preventive strategy to eliminate restarts of predeclared writeset transactions, i.e.: the scheduler puts a predeclared writeset transaction into execution only if it determines beforehand that the future write of that transaction will never compromise correctness of the database system [4, 21].

In practice, it is often the case that more than one read or write request may have arrived in the system within a same short period of time and it may be either impossible (in the case of distributed systems that do not have a global time reference) or not important to distinguish them in terms of real arrival time. In such cases, it seems perfectly logical to allow the concurrency control algorithm to have the freedom to schedule a whole set of read or write requests in a logical order that optimizes performance, i.e., achieves more concurrency. Previously proposed algorithms for concurrency control in database systems typically schedule requesting transactions one at a time, or some times even one step of a transaction at a time, even if a large number of transactions have arrived in the system and are requesting execution simultaneously. In this case, the scheduler may chose for first execution a transaction whose execution precludes the simultaneous execution of any other transaction. In such situations, it is possible to achieve higher performance of the database system by analyzing the whole set of requesting transactions to chose the largest subset or simply any large subset which can be simultaneously executed in parallel with all transactions currently executing in the system [21].

The performance of a database system can also be increased if the database system supports multiversions of data. That is, each time a user transaction writes new information into some database entity with name x, a new version of x is produced. If a user transaction requests to read information in the database entity bearing name x, then the scheduler selects one of the existing versions of x to be read. Since writes do not overwrite each other, and since reads can read any existing version (additional constraints can be specified on which version should be read), a higher level of concurrency can be achieved [2, 16, 21].

However, in order to use any of the optimizing strategies described above, the concurrency control algorithm must spend additional computation time. In our model, we study what additional constraints we need to add in order to restrict the computation time to be polynomial.

Our model is capable of handling the various optimizing concurrency control strategies above within a unified model.

In the next section we define transactions and serial schedules. In our model transactions consist of a read and write step. The complexity results that are obtained in this two step model can be easily extended to a more general n-step model of transactions, as will be discussed at the end of this paper.

In Sect. 3 we introduce the formal model for on-line multiversion concurrency control. In our model there are two types of transactions: predeclared writeset transactions and non-predeclared writeset transactions. Predeclared writeset transactions declare the entities they intend to read and write at submission time. Non-predeclared transactions declare only their read sets at submission time; their write sets become known only at the end when they have finished their computation.

Our paper studies the following two basic problems:

(*1*) Preventive scheduling of predeclared writeset transactions, so that they never have to be restarted. That is, given the current database system state where some transactions are currently executing and some transactions are already completed, determine if one or more predeclared writeset transactions can start execution (i.e., perform their read steps), while guaranteeing that the final results of their computation can always be written into the database while preserving serializability.

(*2*) When one or more non-predeclared writeset transactions finish their computation and announce their writesets, determine if they can write into the database (while preserving serializability) without aborting any transactions.

It will be shown that the first problem is actually equivalent to the second problem when the constraint *DSRD* introduced in Sect. 4 is imposed on the first problem.

It is assumed that the system supports multiple versions of data. The model contains only a minimum set of built-in constraints (precisely those necessary to guarantee serializability). The possibility of using various optimizing strategies (such as predeclare writesets [4, 21], or schedule a whole set of read or write requests simultaneously [21], or use multiversion data [2, 16, 21], etc.) is inherent in the formalism. In fact, denying the scheduler the possibility of using any single optimizing strategy is formally defined to be an additional constraint. The performance achieved by a scheduler with no additional constraints on its input or output represents the upper bound of performance that can ever be achieved by a scheduler when serializability is used as the correctness criterion.

In Sect. 4, we introduce additional constraints that we have found to have a significant effect on the computational complexity of the on-line concurrency problem. That is, the addition or omission of any single one of these constraints, can make an otherwise NP-complete problem polynomial time solvable, or vice-versa.

The constraints that we study include: imposing a fixed explicit total ordering of all existing versions for each variable name (*FTWP*); restricting each transaction to read the "latest" available version (*LTRD*) or a designated version (*DSRD*), or write an "up-to-date" version (*UPDW*) or a designated version (*DSWT*); imposing an invariant ordering between existing versions and versions to be written by predeclared writeset transactions (*IVWP*) for each variable name; and, imposing an upper bound on the number of transactions being scheduled each time (*UB*).

It is interesting to note that these constraints that affect computational complexity are also constraints that may naturally arise in practical applications.

In this paper we derive "limits" of performance achievable by polynomial time concurrency control algorithms. Each limit is characterized by a minimal set of constraints that allow the on-line scheduling problem to be solved in polynomial time. If any one constraint in that minimal set is omitted, although it could increase the amount of concurrency, it would also have the dramatic negative effect of making the scheduling problem NP-complete; whereas if we do not omit any constraint in the minimal set, then the scheduling problem can be solved in polynomial time.

We also introduce a constraint ($2V$) that restricts the number of versions for each variable name in a previous database system state. The main purpose of this constraint is to make the NP-completeness results as strong as possible and show that in most cases restricting the number of version values for each variable name in the database will not help to substantially reduce the required computation time (unless the number of version values is restricted to be 1).

In Sects. 5 and 6, we prove that the minimal set of restrictions $\{FTWP$, ($LTRD$ or $DSRD$), $IVWP$, ($UPDW$ or $DSWT$)$\}$ and the minimal set of restrictions $\{FTWP, IVWP, UB\}$ constitute two fundamental limits of performance achievable by polynomial-time concurrency control algorithms. With each of these limits, one can construct an efficient scheduling algorithm that achieves an optimal level of concurrency in polynomial computation time according to the constraints defined in the minimal set.

In Sect. 7 we provide a summary of the complexity results. Finally, in Sect. 8, we discuss how existing concurrency control algorithms fit into our framework and how to extend our results to an n-step model of transactions.

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 on a set of database entities – called the "readset" of the transaction, followed by a write on a set of database entities – the "writeset". The notation adopted here is similar to that used in [15].

A *database system* consists of a set V of variable names and a set $T = T_1, T_2, \ldots, T_n$ of transactions. A *transaction* T_i is a pair ($[SR_i]$, $[SW_i]$), where SR_i is a subset of V called the *readset* of T_i, and SW_i is a subset of V called the *writeset* of T_i.

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.

It is assumed that the system supports multiple versions of data [2, 16], i.e., there may co-exist one or more values, called *"versions"* for each variable name in the database.

Each transaction T_i can be thought of as first reading a set of versions for each variable name in its readset, then performing a possibly lengthy local computation based on that set of versions. The results of the computation are

finally used to produce a new set of versions for each variable name in its writeset. The first step, i.e., the read step is denoted by $R_i[SR_i]$, while the last step, i.e., the write step is denoted by $W_i[SW_i]$.

A *schedule* of a set of transactions $T=\{T_1, T_2, \ldots, T_n\}$: $T_1=([SR_1], [SW_1])$, $T_2=([SR_2], [SW_2])$, ..., $T_n=([SR_n], [SW_n])$ is a permutation of the set $S_n=\{R_1[SR_1], W_1[SW_1], \ldots, R_n[SR_n], W_n[SW_n]\}$, such that for every i: $R_i[SR_i]$ precedes $W_i[SW_i]$. We abbreviate $R_i[SR_i]$ as R_i and $W_i[SW_i]$ as W_i whenever we need not specify SR_i and SW_i. Associated with a schedule we have a one to one function $\pi: S_n \to \{1, 2, \ldots, 2n\}$, such that for all i, j, $\alpha_i \in S_n$, $\alpha_j \in S_n$, if α_i precedes α_j 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.

A schedule of a set of transactions $T=\{T_1, \ldots, T_n\}$ is a *serial schedule* of T iff $\pi(W_i)=\pi(R_i)+1$ for all $1, 2, \ldots, n$, i.e. a read R_i always immediately precedes a write W_i of the same transaction. We abbreviate the serial schedule $R_{i_1}[SR_{i_1}]\, W_{i_1}[SW_{i_1}]\, R_{i_2}[SR_{i_2}]\, W_{i_2}[SW_{i_2}] \ldots R_{i_n}[SR_{i_n}]\, W_{i_n}[SW_{i_n}]$ as $\langle T_{i_1}\, T_{i_2} \ldots T_{i_n}\rangle$ whenever we need not specify the readset and writeset of each transaction in the serial schedule. We associate with a serial schedule a one to one function $\mu: T \to \{1, 2, \ldots, n\}$, such that for all i, j, $T_i \in T$, $T_j \in T$, if $\pi(W_i) < \pi(W_j)$ and $\pi(R_i) < \pi(R_j)$ in the serial schedule, then $\mu(T_i) < \mu(T_j)$.

We say R_j *reads* x *from* W_i *in schedule* s if $x \in (SW_i \cap SR_j)$ and $\pi(W_i) < \pi(R_j)$ in s and there exists no W_k such that $x \in SW_k$ and $\pi(W_i) < \pi(W_k) < \pi(R_j)$ in s.

In the following sections, we shall also say T_j *reads* x *from* T_i in schedule s when R_j reads x from W_i in s.

Example 2.1

$$s_1 = R_1[x]\, W_1[y, z, b]\, R_2[z]\, W_2[y]\, R_3[y, b]\, W_3[y]$$
$$s_2 = R_3[y, b]\, W_3[y]\, R_2[z]\, W_2[y]\, R_1[x]\, W_1[y, z, b]$$

s_1 and s_2 are both serial schedules of the set of transactions $T=\{T_1, T_2, T_3\}$ where $T_1=([x], [y, z, b])$, $T_2=([z], [y])$ and $T_3=([y, b], [y])$. In serial schedule s_1 of T: R_2 reads z from W_1, R_3 reads b from W_1, R_3 reads y from W_2. In serial schedule s_2 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.

Serializability [3, 6, 15] is used as the consistency criterion here. If an interleaved execution of a set of transactions produces the same overall effect as a serial execution of the same set of transactions, then the execution is called *serializable*. (We shall call the order of the transactions in such a serial execution, which is not necessarily identical to the actual time order in which the reads and writes of those transactions are processed, a "virtual order".)

We model this as the following problem: Given a database system state consisting of four elements: an executing set of predeclared writeset transactions, an executing set of non-predeclared writeset transactions, a terminated set of transactions, and a serial schedule defining the virtual order 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, or transactions that have finished their computation can write new versions of data into the database, while the reads and writes of all transactions in the new virtual order are consistent with the previous virtual order.

To begin, we start with a most unrestrictive model, where the only correctness criterion is serializability.

Definition 3.1 A *database system state* is a quadruple $Q=(PE, NE, TT, s)$ where

(a) PE is called the *set of predeclared writeset (P-)executing transactions;*

(b) NE is called the *set of nonpredeclared writeset (N-)executing transactions*;

(c) TT is called the *set of terminated transactions;*

(d) s is a serial schedule of $T=(PE \cup NE \cup TT)$ called the *virtual schedule.* □

A *P-executing transaction* is a transaction which has been put into execution by reading a set of existing versions for each variable name in its readset, but has not yet completed its local computation, thus has not yet made a set of versions for each variable name in its writeset available for reading by other transactions. An *N-executing transaction* is a transaction which has been put into execution by reading a set of existing versions for each variable name in its readset, but has not yet completed its local computation, and its writeset has not yet been defined (its writeset is assumed to be empty). A *terminated transaction* is a transaction which has completed its local computation and has written a set of versions for each variable name in its writeset into the database. The set of versions for each variable name in the writeset of a terminated transaction can be read by other transactions. The *virtual schedule* is a serial schedule of all P-executing transactions, N-executing transactions and terminated transactions. The scheduler guarantees that the reads and writes of all P-executing transactions, N-executing transactions and terminated transactions will have the same 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 versions of which variable names from the writeset of which transaction so far up to the present system state.

Definition 3.2 Given a set PR of predeclared writeset transactions that are requesting to be put into execution (PR is called a *P-requesting set*), and a database system state $Q=(PE, NE, TT, s)$, a new database system state $Q_1=(PE_1, NE_1, TT_1, s_1)$ is a *valid P-execution* of Q and PR, whenever

(a) s_1 is a serial schedule; and

(b) $NE_1=NE$ and $TT_1=TT$ and for some nonnull subset $PR_s \subseteq PR$: $PE_1=(PE \cup PR_s)$; and

(c) (*consistency condition*:) for all i, j, x, $T_i \in (PE \cup NE \cup TT)$, $T_j \in (PE \cup NE \cup TT)$, $x \in V$: T_j reads x from T_i in s iff T_j reads x from T_i in s_1 (i.e., every read from relation in s is preserved in s_1); and

(d) (*existence condition*:) for all i, j, x, $T_i \in (PE_1 \cup NE_1 \cup TT_1)$, $T_j \in (PE_1 \cup NE_1 \cup TT_1)$, $x \in V$: if T_j reads x from T_i in s_1 then $T_i \in TT_1$ (i.e., T_j can read from T_i only if T_i has terminated). □

Definition 3.2 defines the conditions under which a set of predeclared writeset requesting transactions can be put into execution. P-requesting transactions

are scheduled as follows: the concurrency control algorithm or scheduler must find a new virtual schedule s_1 which includes both the P-requesting transactions and all the transactions in the previous virtual schedule s, such that the "consistency condition" is satisfied, i.e.: all the read from relations in s are preserved in s_1. In addition, the existence condition must be satisfied, i.e.: no transaction can read from a predeclared writeset transaction before it has terminated and has produced a version value for each variable name in its writeset. If a new virtual schedule s_1 can be found that satisfies these conditions, then a new valid database system state Q_1 can be constructed, where each P-requesting transaction is put into execution by reading the version values specified by the virtual schedule s_1 and by transferring that transaction from the requesting set PR to the executing set PE. The "existence condition" states that no transaction in the virtual order should read a version which has not yet been produced. Note that for the same P-executing set, N-executing set and terminated set of transactions, more than one virtual schedule can be constructed by rearranging the virtual schedule while satisfying all the consistency and existence conditions. In our model we do not allow transactions to read version values written by "uncommitted" transactions, i.e., transactions that have not terminated. This is mainly to prevent "cascading aborts", i.e., situations where in the event that some transaction has to be aborted, all transactions that read from that transaction must also be aborted.

Definition 3.3 Given a set NR of non-predeclared writeset transactions that are requesting to be put into execution (NR is called a *N-requesting set*), and a database system state $Q=(PE, NE, TT, s)$, a new database system state $Q_1=(PE_1, NE_1, TT_1, s_1)$ is a *valid N-execution* of Q and NR, whenever

(a) s_1 is a serial schedule; and

(b) $PE=PE$ and $TT_1=TT$ and for some nonnull subset $NR_s \subseteq NR$: $NE_1 = (NE \cup NR_s)$; and

(c) (*consistency condition*:) for all i, j, x, $T_i \in (PE \cup NE \cup TT)$, $T_j \in (PE \cup NE \cup TT)$, $x \in V$: T_j reads x from T_i in s iff T_j reads x from T_i in s_1 (i.e., every read from relation in s is preserved in s_1); and

(d) (*existence condition*:) for all i, j, x, $T_i \in (PE_1 \cup NE_1 \cup TT_1)$, $T_j \in (PE_1 \cup NE_1 \cup TT_1)$, $x \in V$: if T_j reads x from T_i in s_1 then $T_i \in TT_1$ (i.e., T_j can read from T_i only if T_i has terminated). □

Definition 3.4 Let $PE_c \subseteq PE$ be a nonnull subset of P-executing transactions that have completed their local computation and are requesting to write a new version for each variable name in their respective writeset into the database. (PE_c is called a *P-commit set*). A new database system state $Q_1=(PE_1, NE_1, TT_1, s_1)$ is a *valid P-termination* of a current database system state $Q=(PE, NE, TT, s)$ and PE_c, whenever

(a) s_1 is a serial schedule; and

(b) $NE_1=NE$ and $TT_1=(TT \cup PE_c)$ and $PE_1=(PE-PE_c)$ and

(c) (*consistency condition*:) for all i, j, x, $T_i \in (PE \cup NE \cup TT)$, $T_j \in (PE \cup NE \cup TT)$, $x \in V$: T_j reads x from T_i in s iff T_j reads x from T_i in s_1 (i.e., every read from relation in s is preserved in s_1). □

Since a preventive strategy was used for putting P-requesting transactions into execution, a set of versions produced by the local computation of a P-commit transaction for each variable name in its writeset can always be immedi-

ately written into the database while preserving consistency of the database system.

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

Definition 3.5 Let NE'_c be a set of nonpredeclared writeset transactions where every transaction $T'_i \in NE'_c$ has a writeset defined and corresponds to a N-executing transaction $T_i \in NE$ that has completed its local computation and is requesting that the set of versions in the writeset SW'_i produced by its local computation be written into the database. For corresponding transactions T_i and T'_i: $SR_i = SR'_i$. (NE'_c is called a *N-commit set*). Let $NE_c \subseteq NE$ be the set of N-executing transactions that correspond to the set NE'_c. A new database system state $Q_1 = (PE_1, NE_1, TT_1, s_1)$ is a *valid N-termination* of a current database system state $Q = (PE, NE, TT, s)$ and NE'_c whenever

(a) s_1 is a serial schedule; and

(b) $PE_1 = PE$ and

(c) for some nonnull subset $NE'_{c_s} \subseteq NE'_c$ and $NE_{c_s} \subseteq NE_c$ where NE_{c_s} is the set of N-executing transactions that correspond to the set NE'_{c_s}: $TT_1 = (TT \cup NE'_{c_s})$ and $NE_1 = (NE - NE_c)$; and for all i, j, x, $T'_i \in (TT_1 - TT)$ and $T_j \in TT$ and $T_i \in NE_{c_s}$, $x \in V$: T'_i reads x from T_j in s_1 iff T_i reads x from T_j in s (i.e., every terminated nonpredeclared writeset transaction T'_i must read the same set of versions in s_1 that its corresponding N-executing transaction T_i read in s); and

(d) for all i, j, x, $T_i \in (PE_1 \cup NE_1 \cup TT)$, $T_j \in (PE_1 \cup NE_1 \cup TT)$, $x \in V$: T_i reads x from T_j in s_1 iff T_i reads x from T_j in s (i.e., every other transaction must read the same set of versions in s_1 that was read in s). □

In Definition 3.5, it is possible that some transactions (in the subset $NE_c - NE_{c_s}$) may have to be aborted if it is determined that they cannot write into the database without compromising consistency.

Definition 3.6 A database system state $Q = (PE, NE, TT, s)$ is a *valid database system state* iff:

(a) $PE = \emptyset$ and $NE = \emptyset$ and $TT = \emptyset$ and s is empty (called the initial database system state); or,

(b) Q is a valid P-execution of a valid database system state and a P-requesting set of transactions; or,

(c) Q is a valid N-execution of a valid database system state and a N-requesting set of transactions; or,

(d) Q is a valid P-termination of a valid database system state and a P-commit set of transactions; or,

(e) Q is a valid N-termination of a valid database system state and a N-commit set of transactions. □

Definition 3.6 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 or terminated in the database

system, their reads and writes on the database have the same overall effect as if they were executed sequentially in the same order as the serial schedule s.

Definition 3.7 Given a P-requesting set PR and a current database system state $Q=(PE, NE, TT, s)$, we call the valid database system state $Q_1=(PE_1, NE_1, TT_1, s_1)$ a *maximum concurrency* P-*execution* of Q and PR iff: Q_1 is a valid P-execution of Q and PR; and no other database system state $Q_2=(PE_2, NE_2, TT_2, s_2)$ exists such that Q_2 is also a valid P-execution of Q and PR; and $|PE_1|<|PE_2|$. □

Definition 3.8 Given a current database system state $Q=(PE, NE, TT, s)$ and a N-commit set NE_c' of nonpredeclared writeset transactions, we call the valid database system state $Q_1=(PE_1, NE_1, TT_1, s_1)$ a *maximum concurrency N-termination* of Q and NE_c' iff: Q_1 is a valid N-termination of Q and NE_c'; and no other database system state $Q_2=(PE_2, NE_2, TT_2, s_2)$ exists such that Q_2 is also a valid N-termination of Q and NE_c'; and $|TT_1|<|TT_2|$. □

Note that in Definition 3.5 and 3.8 above we did not allow P-executing transactions to be aborted in favor of terminating non-predeclared writeset transactions. In other words we have guaranteed that once a predeclared writeset transaction is put into execution, it will never be aborted. (If a system designer wishes to change that policy, then the requirement for maximum concurrency in Definition 3.8 could be changed to $|TT_1 \cup PE_1|<|TT_2 \cup PE_2|$ besides changing the condition $PE=PE_1$ in Definition 3.5.)

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:

Figure 3.1

PR_i: P-requesting set of transactions.
NR_i: N-requesting set of transactions.
PEc_i: P-commit set of transactions.
NEc_i': N-commit set of transactions.
Q_i: database system state.
PE_i: P-executing set of transactions.
NE_i: N-executing set of transactions.
TT_i: terminated set of transactions.
s_i: virtual ordering.

Q_0: $PE_0=\emptyset$ $NE_0=\emptyset$ $TT_0=\emptyset$
$s_0=\langle\ \rangle$.

(The initial database system state).

$PR_1=\{T_1=([b], [a])\}$,
Q_1: $PE_1=\{T_1\}$ $NE_1=\emptyset$ $TT_1=\emptyset$
$s_1=\langle T_1\rangle=R_1[b]\ W_1[a]$.

(T_1 is put into execution in Q_1. Q_1 is a valid execution of PR_1 and Q_0 by Definition 3.2.)

$PR_2=\{T_2=([c, a], [d, a])\}$;
Q_2: $PE_2=\{T_1, T_2\}$ $NE_2=\emptyset$ $TT_2=\emptyset$
$s_2=\langle T_2\ T_1\rangle=R_2[c, a]\ W_2[d, a]\ R_1[b]\ W_1[a]$.

(T_2 is put into execution in Q_2. Q_2 is a valid execution of PR_2 and Q_1 by Definition 3.2.)

$PR_3 = \{T_3 = ([a, c], [f, g, c]), T_4 = ([f, a], [b, c]), T_5 = ([a, g], [e, a])\}$;
Q_2: $PE_2 = \{T_1, T_2\}$ $NE_2 = \emptyset$ $TT_2 = \emptyset$
$s_2 = \langle T_2\ T_1 \rangle = R_2[c, a]\ W_2[d, a]\ R_1[b]\ W_1[a]$.

(There does not exist any valid P-execution of PR_3 and Q_2 in which any transaction in PR_3 can be put in execution.)

$PR_3 = \{T_3, T_4, T_5\}$ $PEc_1 = \{T_1\}$;
Q_3: $PE_3 = \{T_2\}$ $NE_3 = \emptyset$ $TT_3 = \{T_1\}$
$s_3 = s_2$.

(T_1 is terminated in Q_3. Q_3 is a valid P-termination of PEc_1 and Q_2 by Definition 3.4.)

$PR_3 = \{T_3, T_4, T_5\}$;
Q_4: $PE_4 = \{T_2, T_4, T_5\}$ $NE_4 = \emptyset$ $TT_4 = \{T_1\}$
$s_4 = \langle T_2\ T_1\ T_4\ T_5 \rangle$
$= R_2[c, a]\ W_2[d, a]\ R_1[b]\ W_1[a]\ R_4[f, a]\ W_4[b, c]\ R_5[a, g]\ W_5[e, a]$.

(T_4, T_5 are put into execution in Q_4. Q_4 is a valid P-execution of PR_3 and Q_3 by Definition 3.2.)

$PR_4 = \{T_3\}$ $NR_1 = \{T_6 = ([a], \square),$ $T_7 = ([a, b], \square)\}$,
Q_5: $PE_5 = \{T_2, T_4, T_5\}$ $NE_5 = \{T_6, T_7\}$ $TT_5 = \{T_1\}$
$s_5 = \langle T_2\ T_1\ T_6\ T_7\ T_4\ T_5 \rangle$
$= R_2[c, a]\ W_2[d, a]\ R_1[b]\ W_1[a]\ R_6[a]\ W_6 \square R_7[a, b]\ W_7 \square R_4[f, a]$
$W_4[b, c]\ R_5[a, g]\ W_5[e, a]$.

(T_6 and T_7 are put into execution in Q_5. Q_5 is a valid N-execution of NR_1 and Q_4 by Definition 3.3.)

$PR_4 = \{T_3\}$ $NEc'_1 = \{T'_6 = ([a], [f]),\ T'_7 = ([a, b], [f, g])\}$;
Q_6: $PE_6 = \{T_2, T_4, T_5\}$ $NE_6 = \emptyset$ $TT_6 = \{T_1, T'_6\}$
$s_6 = \langle T_2\ T_1\ T_4\ T'_6\ T_5 \rangle = R_2[c, a]\ W_2[d, a]\ R_1[b]\ W_1[a]\ R_4$
$[f, a]\ W_4[b, c]\ R'_6[a]$
$W'_6[f]\ R_5[a, g]\ W_5[e, a]$.

(T'_6 is terminated but T_7 is aborted in Q_6. Q_6 is a valid N-termination of NEc_1 and Q_5 by Definition 3.5.)

Example 3.1 Figure 3.1 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 Q_1 to state Q_2, the scheduler is able to put transaction T_2 into execution in parallel with T_1, because there exists a virtual ordering identical to the serial schedule s_2, where a set of versions corresponding to the variable names in T_2's readset are available, and T_1 reads the same versions as in the previous virtual ordering s_1. Note that T_2 cannot be executed according to the virtual ordering $s = \langle T_1\ T_2 \rangle$, because in that virtual ordering T_2 is supposed to read a version of the variable name

a in T_1's writeset, which has not yet been produced, since T_1 has not yet terminated.

With database system state Q_2, no transaction in the P-requesting set PR_3 can be executed, because no transaction in PR_3 can be inserted into the serial schedule without creating a virtual ordering in which at least one transaction is supposed to read a version which has not yet been produced.

In the transformation from database system state Q_3 to state Q_4, since T_1 has terminated previously, each single transaction in the P-requesting set PR_3 can be put in execution by reading the version of a produced by T_1. But because T_4 and T_5 is the largest subset of all P-requesting transactions in PR_3 which can be simultaneously put into execution in parallel with T_2, we chose T_4 and T_5 to be executed first. On the contrary, if we chose T_3 for execution first, then both T_4 and T_5 would be blocked from execution, since no virtual ordering can be found which allows T_4 and T_5 to be executed in parallel with T_3.

In the transformation from database system state Q_5 to Q_6, two N-executing transactions T_6 and T_7 have completed their local computation. Their corresponding N-commit transactions T_6' and T_7' each has a writeset defined. Among the two N-commit transactions T_6' and T_7' in the N-commit set NEc_1', only T_6' can be committed, without aborting other transactions; whereas if T_7' is committed, then two P-executing transactions T_4 and T_5 must be aborted. This is because T_7' cannot be positioned after T_4, nor positioned before T_4 in any virtual ordering, otherwise either T_7' would read b from T_4, or T_4 would read f from T_7'. But T_7 – the N-completed transaction corresponding to T_7', did not read b from T_4, neither did T_4 read f from T_7' in the previous virtual schedule s_5. Similarly, T_7' cannot be positioned after T_5, nor positioned before T_5, otherwise T_7' would read a from T_5, or T_5 would read g from T_7', which both violates the conditions in Definition 3.5. Thus, in order to minimize the total number of aborted transactions, we chose to abort and restart the single N-completed transaction T_7. Note that T_6' must be positioned after T_4 and before T_5 in the new virtual ordering s_6, otherwise either T_4 would read f from T_6', or T_6' would read a from T_5, which also violates the conditions in Definition 3.5. And we can continue like this constructing successive new valid database system states.

Note that in this example all valid P-executions are maximum concurrency P-executions of the previous state, and the valid N-termination Q_6 is a maximum concurrency N-termination of NEc_1' and Q_5. □

Two problems, modeled as state transformations above, involve major performance vs. computation time tradeoffs. The first is the problem of constructing valid P-executions of a P-requesting set and a valid database system state, which formalizes the problem of determining whether a given set of predeclared writeset transactions can be put into execution while guaranteeing that their future writes on the database will never compromise correctness of the database system. That is, the predeclared writeset transactions are guaranteed to be "committed" (terminated) without aborting any of the executing transactions. The second is the problem of constructing valid N-terminations of a N-commit set and a given valid database system state, which formalizes the problem of determining whether the writesets of a given set of non-predeclared writeset transactions can be written into the database while preserving correctness of the database system.

As will be shown in the next section, the second problem is equivalent to the first problem if a particular additional constraint called the "designated read (*DSRD*)" constraint is imposed on the first problem. For this reason, we need only study in detail the performance vs. computation time tradeoff involved in solving the first problem. That is, all the results that we derive for the problem of constructing valid P-executions of a P-requesting set and a valid database system state under the *DSRD* constraint can be directly applied to the problem of constructing valid N-terminations of a N-commit set and a valid database system state. Since the problem of constructing a valid N-execution or a valid P-termination of a given valid database system state can be trivially solved in polynomial time, we do not address these problems any further.

4 Additional constraints

Since the general problem of constructing valid P-executions or valid N-terminations of a given valid database system state when no additional constraints are imposed is NP-complete (even under a combination of various additional constraints, the problem is still NP-complete, as we will show in the following sections), even for the sole reason of reducing computational complexity, it would prove beneficial to investigate possible additional constraints to obtain subsets of valid P-executions or N-terminations of a valid database system state and a P-requesting set or N-commit set in polynomial time.

The constraints that we introduce below, are constraints that we have found to have a significant effect on the computational complexity of the on-line concurrency problem. That is, the addition or omission of any single one of these constraints, can make an otherwise NP-complete problem polynomial time solvable, or vice-versa. At the same time, it is interesting to note that these constraints that affect computational complexity are also constraints that may naturally arise in practical applications.

Given a P-requesting set PR (or a N-commit set NE_c) of transactions and a valid database system state $Q=(PE, NE, TT, s)$, we may impose that one or more of the constraints introduced below are to be satisfied by a valid P-execution (or valid N-termination) $Q_1=(PE_1, NE_1, TT_1, s_1)$ of PR (or NE_c) and Q.

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

1. Fixed terminated write position (FTWP) constraint: for all i, j, if $(SW_i \cap SW_j) \neq \emptyset$ and $T_i \in TT$ and $T_j \in TT$ then: $\mu(T_i) < \mu(T_j)$ in s iff $\mu_1(T_i) < \mu_1(T_j)$ in s_1 (i.e., if two transactions have terminated and their writesets intersect, then their relative ordering in the virtual schedule s of the previous database system state Q must be kept invariant in the virtual schedule s_1 of the new database system state Q_1).

In conventional concurrency control schemes, the constraint below is also imposed:

2. Invariant write position (IVWP) constraint: for all i, j, $T_i \in (PE \cup TT)$, $T_j \in (PE \cup TT)$, if $(SW_i \cap SW_j) \neq \emptyset$ and $((T_i \in TT$ and $T_j \in PE)$ or $(T_j \in TT$ and $T_i \in PE))$ then $\mu(T_i) < \mu(T_j)$ in s iff $\mu_1(T_i) < \mu_1(T_j)$ in s_1 (i.e., if the writesets of two transactions intersect, and one is a terminated transaction and the other is a P-executing transaction, then their relative ordering in the virtual schedule s of the previous database system state Q must be kept invariant in the virtual schedule s_1 of the new database system state Q_1).

In Definitions 3.2 and 3.3, when a P-requesting or an N-requesting transaction is put into execution, it may read any one out of existing versions 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. In such cases, we have the following constraint:

3. Latest read version (LTRD) constraint: for all i, j, x, $T_j \in (PE_1 - PE)$ (or $T_j \in (NE_1 - NE)$), $T_i \in TT$, $x \in V$: if T_j reads x from T_i in s_1, then there exists no k, $T_k \in TT$, such that $x \in SW_k$ and $\mu_1(T_i) < \mu_1(T_k)$ in s_1 (i.e., when a predeclared writeset transaction is put into execution in Q_1, for each variable name in its readset, it must read the "latest" available version in the virtual ordering).

The following constraint states that when a P-requesting transaction is put into execution, for each variable name in its writeset, it must finally write a version which is ordered after all currently existing versions of that variable name. The same constraint can be applied to N-commit transactions.

4. Up to date write (UPDW) constraint: for all j, x, $T_j \in (PE_1 - PE)$ (or $T_j \in (TT_1 - TT)$ if Q_1 is a valid N-termination), $x \in V$: if $x \in SW_j$, then there exists no k, such that $T_k \in TT$ and $x \in SW_k$ and $\mu_1(T_j) < \mu_1(T_k)$ in s_1 (i.e., for every predeclared writeset transaction that is put into execution in Q_1 (or for every non-predeclared writeset transaction that is currently being terminated in Q_1), for each variable name in its writeset, if any terminated transaction produced a version for that variable name then the predeclared writeset transaction (or non-predeclared writeset transaction currently being terminated) must be positioned after (not necessarily immediately after) that terminated transaction in the virtual schedule s_1).

In the *LTRD* constraint, a P-requesting or N-requesting transaction is restricted to read the "latest" available version for each variable name in its readset when put into execution. In the following constraint, instead of restricting a P-requesting or N-requesting transaction to read the "latest" available version, we can restrict it to read *a* pre-designated version for each variable name in its readset.

5. Designated read (DSRD) constraint: for all i, j, x, such that $T_i \in (PE_1 - PE)$ (or $T_i \in (NE_1 - NE)$) and $((dr(i, x) = T_j$ and $T_j \in TT$ and $x \in (SR_i \cap SW_j))$ or $(dr(i, x) = v_0))$: if $dr(i, x) = T_j$ then T_i reads x from T_j in s_1; else if $dr(i, x) = v_0$ then for all v, $T_v \in TT$: $\neg(T_i$ reads x from T_v in $s_1)$ (i.e., a P-requesting transaction is restricted to read a designated version for each variable name in its readset when put into execution. "$dr(i, x)$" designates the terminated transaction from which P-requesting transaction T_i must read *a* version of x. "$dr(i, x) = v_0$" designates that T_i must read the initial version of x).

Similarly, instead of restricting a P-requesting or N-commit transaction to be positioned after all terminated transactions which have produced a version for any variable name in that P-requesting or N-commit transaction's writeset, we can restrict a P-requesting or N-commit transaction to be positioned after a pre-designated terminated transaction which has produced a version for a variable name in that P-requesting or N-commit transaction's writeset.

6. *Designated write* (*DSWT*) *constraint:* for all i, j, x, such that $T_i \in (PE_1 - PE)$ (or $T_i \in (TT_1 - TT)$ in the case of a valid N-termination) and $((dw(i,x) = T_j$ and $T_j \in TT$ and $x \in (SW_i \cap SW_j))$ or $(dw(i,x) = v_0))$: if $dw(i,x) = T_j$ then: $\mu_1(T_j) < \mu_1(T_i)$ in s_1 and there exists no k, such that $T_k \in TT$ and $x \in (SW_j \cap SW_k \cap SW_i)$ and $\mu_1(T_j) < \mu_1(T_k) < \mu_1(T_i)$ in s_1; else if $dw(i,x) = v_0$ then: there exists no k, such that $T_k \in TT$ and $x \in (SW_k \cap SW_i)$ and $\mu_1(T_k) < \mu_1(T_i)$ in s_1 (i.e., a P-requesting or N-commit transaction is restricted to be positioned after a designated terminated transaction which has produced a version for a variable name in that P-requesting or N-commit transaction's writeset. "$dw(i,x)$" designates the terminated transaction that has produced a version for the variable name x in T_i's writeset and that should be positioned before T_i. "$dw(i,x) = v_0$" designates that T_i must be positioned before all terminated transactions which have produced a version for the variable name x in T_i's writeset).

There may also exist situations where one would prefer to limit the number of transactions being scheduled each time. This leads to the following constraint:

7. *Upper bound* (*UB*) *constraint:* $|PR| \leqq C$ or $|NE'_c| \leqq C$ (i.e., the number of P-requesting transactions or N-commit transactions scheduled each time cannot exceed a constant C).

In order to examine the effect of limiting the number of versions that exists for each variable name in the database system, we specify the following constraint:

8. *Two version data* (*2V*) *constraint:* for all x, i, if $x \in SW_i$ and $T_i \in TT$ then there exists no k, such that $T_k \in TT$ and $x \in (SW_k \cap SW_i)$ and $k \neq i$ (i.e., there exists no more than two versions – the initial version plus another version created by a terminated transaction for each variable name in the datebase).

We need not explicitly define a "one version data (*1V*) constraint here, since if only one version exists for each variable name in the database, then the problem of constructing valid P-executions or valid N-terminations of a given valid database system state is trivially polynomial time solvable.

In this paper only NP-completeness theorems use the 2-version (*2V*) constraint. The *2V* constraint is only defined on the "previous" database system state Q in those theorems. The main purpose of the *2V* constraint is to make the NP-completeness results as strong as possible. The *2V* constraint in Theorems 5.1–5.6 in the next section essentially states that "even if only two version values for each variable name exist in the previous database system state, ... (under the other constraints) ..., the problem is still NP-complete". This tells us that if only those other constraints are enforced, then restricting the number of version values for each variable name in the database will not help to substantially reduce the required computation time (unless the number of version values is restricted to be 1). None of the polynomial time result theorems in this paper use any version constraints. The on-line polynomial time multiversion concur-

rency control algorithms presented in this paper do *not* use or depend on the definition of the n-version constraint.

In the definition of the *UPDW* and *DSWT* constraints above, for P-requesting transactions, the *UPDW* or *DSWT* constraint is checked at the time when the transaction is being scheduled for execution. If one also wishes to enforce the *UPDW* or *DSWT* constraint when a P-executing transaction terminates, then one may have to abort the P-executing transaction if the constraint cannot be satisfied at that time.

Note that implementation details of the on-line multiversion concurrency control algorithms are not explicitly addressed in this paper. For example, in an actual implementation, instead of allowing the number of versions for each variable name to grow forever, one would put a limit on the total number of versions for each variable name that is allowed to exist in the database. One could achieve this by "purging" old versions (or "terminated transactions") from the system when the total number of versions exceeds the specified limit. Thus the so-called "initial version" (which is really the "oldest version" among all versions that are still kept in the database) and other existing versions for each variable name will be continuously "updated" throughout the life time of the database. By doing so, we could easily solve the problem of the scheduler getting slower as the number of versions increase over time. In such an implementation, the $2V$ constraint would mean that there exists no more than the latest two versions for each variable name in the database. Note that in the model, the set of "terminated transactions" defines which version values currently exist in the database, which transaction wrote those version values, while the ordering of the existing versions is defined by the ordering of the terminated transactions which wrote each version in the virtual schedule.

In an actual implementation, it is also possible to allow the constraints above to be applied differently for different transactions. The complexity results in the next three sections are still applicable in such cases. That is, if any subset of the transactions have the same or fewer constraints compared with each set of constraints defined in Theorems 5.1–5.6 in the next section, then the scheduling problem would still be NP-complete.

As we mentioned in the previous section, the problem of constructing valid N-terminations of a N-commit set and a valid database system state is equivalent to the problem of constructing valid P-executions of a P-requesting set and a valid database system state under the *DSRD* constraint. To see this, note that given a valid N-commit set NE_c' and a valid database system state $Q=(PE, NE, TT, s)$, in any valid N-termination $Q_1=(PE_1, NE_1, TT_1, s_1)$ of NE_c' and Q, every nonpredeclared writeset transaction T_i' in NE_c' which is terminated in Q_1 must read the same set of versions in s_1 that its corresponding N-executing transaction T_i read in s. We can construct an equivalent problem of finding valid P-executions of a P-requesting set PR^* and a valid database system state $Q^*=(PE^*, NE^*, TT^*, s^*)$ under the *DSRD* constraint as follows: let $PR^*=NE_c'$; let $PE^*=PE$ and $TT^*=TT$; let $NE^*=(NE-\{T_i \mid T_i' \in NE_c'\})$; let the ordering between every pair of transactions T_i and T_j in s^* be the same as in s, i.e.: for all i, j, T_i, $T_j \in (PE^* \cup NE^* \cup TT^*)$: $\mu^*(T_i) < \mu^*(T_j)$ in s^* iff $\mu(T_i) < \mu(T_j)$ in s. We specify the *DSRD* constraint as follows: for every P-requesting transaction T_i' in PR^*, let $dr(i, x)=T_j$ iff T_i reads x from T_j in s. It is straightforward to verify that there exists a valid N-termination $Q_1=(PE_1, NE_1, TT_1, s_1)$ of Q and NE_c' in which the set NE_c' of N-commit transactions is terminated in Q_1,

that is, $TT_1=(TT\cup NE_c')$ if and only if there exists a valid P-execution $Q_1^*=(PE_1^*, NE_1^*, TT_1^*, s_1^*)$ of Q^* and PR^* in which the set $PR^*=NE_c'$ of P-requesting transactions is put into execution in Q_1^*, i.e.: $PE_1^*=(PE^*\cup PR^*)$ under the specified *DSRD* constraint.

If we are given any problem of terminating non-predeclared transactions under a set of additional constraints and we have an algorithm for solving the problem of scheduling predeclared writeset transactions for execution under the same set of additional constraints plus the *DSRD* constraint, then we can always solve the former problem by first transforming it to an equivalent latter problem by constructing the set PR^*, the database system state $Q^*=(PE^*, NE^*, TT^*, s^*)$ and the *DSRD* constraints as defined above (if any additional constraints are specified in the former problem then the same set of additional constraints plus the *DSRD* constraint defined above should be specified for the equivalent latter problem), then use the algorithm to obtain a valid P-execution $Q_1^*=(PE_1^*, NE_1^*, TT_1^*, s_1^*)$ of Q^* and PR^* in which the subset $PR^*=NE_c'$ of P-requesting transactions is put into execution in Q_1^*, i.e.: $PE_1^*=(PE^*\cup PR^*)$ under the constructed *DSRD* constraint. We can then obtain the valid N-termination $Q_1=(PE_1, NE_1, TT_1, s_1)$ of Q and NE_c' in which the set NE_c' of N-commit transactions is terminated in Q_1, that is, $NE_c'\subseteq TT_1$, as follows: let $PE_1=PE$; let $NE_1=(NE-\{T_i|\, T_i'\in NE_c'\})$, let $TT_1=(TT\cup NE_c')$; let the ordering between every pair of transactions T_i and T_j in s_1 be the same as in s_1^*.

5 Scheduling predeclared writeset transactions for execution

In this and the following section, we shall study the computational complexity of the problem of determining whether a set of predeclared writeset transactions can be put into execution while guaranteeing that their future writes will never compromise correctness of the database system under any combination of the constraints *FTWP*, *LTRD* or *DSRD*, *IVWP*, *UPDW* or *DSWT*, 1*V*, 2*V* and *UB*.

For each scheduling problem that is proved to be NP-complete below, the scheduling problem contains a "maximal" set of constraints. That is, if one more additional constraint is imposed, then the problem would become polynomial time solvable. Whereas for each scheduling problem that is proved to be polynomial time solvable below, the scheduling problem contains a "minimal" set of restrictions. That is, if one restriction is removed from that set, then the scheduling problem would become NP-complete. We construct an efficient scheduling algorithm for each problem that can be solved in polynomial time.

Theorem 5.1 *The following problem (LI) is* NP*-complete:*
Given a P*-requesting set PR and a valid database system state* $Q=(PE, NE, TT, s)$ *that satisfies the* 2*V constraint, does there exist a valid database system state* $Q_1=(PE_1, NE_1, TT_1, s_1)$ *such that* Q_1 *is a valid* P*-execution of PR and Q under the FTWP, IVWP and LTRD constraints and* $PE_1=(PR\cup PE)$*?* □

Theorem 5.2 *The following problem (UI) is* NP*-complete: Given a* P*-requesting set PR and a valid database system state* $Q=(PE, NE, TT, s)$ *that satisfies the*

2V constraint, does there exist a valid database system state $Q_1=(PE_1, NE_1, TT_1, s_1)$ *such that* Q_1 *is a valid* P-*execution of PR and Q under the FTWP, IVWP and UPDW constraints and* $PE_1=(PR \cup PE)$? □

(See the Appendix for the proofs of Theorems 5.1 and 5.2.)

Theorem 5.3 *The following problem (LU1) is* NP-*complete:*
Given a P-*requesting set* $PR=\{T_r\}$ *and a valid database system state* $Q=(PE, NE, TT, s)$ *that satisfies the 2V constraint, does there exist a valid database system state* $Q_1=(PE_1, NE_1, TT_1, s_1)$ *such that* Q_1 *is a valid* P-*execution of PR and Q under the FTWP, LTRD and UPDW constraints and* $PE_1=(\{T_r\} \cup PE)$? □

(See the Appendix for the proof of Theorem 5.3.)

Theorem 5.4 *The following problem is* NP-*complete:*
Given a P-*requesting set PR and a valid database system state* $Q=(PE, NE, TT, s)$ *that satisfies the 2V constraint, does there exist a valid database system state* $Q_1=(PE_1, NE_1, TT_1, s_1)$ *such that* Q_1 *is a valid* P-*execution of PR and Q under the FTWP, IVWP and DSRD constraints and* $PE_1=(PR \cup PE)$? □

Theorem 5.5 *The following problem is* NP-*complete:*
Given a P-*requesting set PR and a valid database system state* $Q=(PE, NE, TT, s)$ *that satisfies the 2V constraint, does there exist a valid database system state* $Q_1=(PE_1, NE_1, TT_1, s_1)$ *such that* Q_1 *is a valid* P-*execution of PR and Q under the FTWP, IVWP and DSWT constraints and* $PE_1=(PR \cup PE)$? □

Theorem 5.6 *The following problem is* NP-*complete*:
Given a P-*requesting set* $PR=\{T_r\}$ *and a valid database system state* $Q=(PE, NE, TT, s)$ *that satisfies the 2V constraint, does there exist a valid database system state* $Q_1=(PE_1, NE_1, TT_1, s_1)$ *such that* Q_1 *is a valid* P-*execution of PR and Q under the FTWP, (LTRD or DSRD) and (UPDW or DSWT) constraints and* $PE_1=(\{T_r\} \cup PE)$? □

The proof of Theorems 5.4–5.6 immediately follows from Theorems 5.1–5.3 and the fact that the constraint *LTRD* is a special case of the constraint *DSRD*, and the constraint *UPDW* is a special case of the constraint *DSWT*. One can replace the *LTRD* constraint by the *DSRD* constraint as follows: for all i, x, such that $T_i \in PR$ and $x \in SR_i$, let $dr(i, x)$ designate the terminated transaction which is rightmost in the virtual schedule s and contains x in its writeset. If no terminated transaction exists in TT which contains x in its writeset, then let $dr(i, x)=v_0$. Similarly, one can replace the *UPDW* constraint by the *DSWT* constraint as follows: for all i, x, such that $T_i \in PR$ and $x \in SW_i$, let $dw(i, x)$ designate the terminated transaction which is rightmost in the virtual schedule s and contains x in its writeset. If no terminated transaction exists in TT which contains x in its writeset, then let $dw(i, x)=v_0$.

If we combine the constraints *FTWP*, *IVWP*, (*LTRD* or *DSRD*), (*UPDW* or *DSWT*) together, the computational complexity of our problem can be substantially reduced.

Definition 5.1 We call a directed graph $G=(N, A)$ the "*IDRW dependency graph*" of a valid database system state $Q=(PE, NE, TT, s)$ whenever in G there is the set of nodes $N=\{T_i \mid T_i \in (PR \cup PE \cup NE \cup TT)\}$; and the set of arcs $A=\{(T_i, T_j) \in (X \times X) \mid (i \neq j)$ and

((a) (($(SW_j \cap SR_i) \neq \emptyset$) or ($(SW_j \cap SR_j) \neq \emptyset$)) and ($T_j \in (PE \cup NE \cup TT)$) and ($T_i \in (PE \cup NE \cup TT)$) and ($\mu(T_j) < \mu(T_i)$ in s); or

(b) ($(SW_j \cap SW_i) \neq \emptyset$) and ($T_j \in (PE \cup TT)$) and ($T_i \in (PE \cup TT)$) and (($T_j \in TT$) or ($T_i \in TT$)) and ($\mu(T_j) < \mu(T_i)$ in s); or

(c) ($(SW_j \cap SR_i) \neq 0$) and ($T_i \in PR$) and ($T_j \in TT$) and there exists some x, such that ($T_j = dr(i, x)$); or

(d) ($(SR_j \cap SW_i) \neq \emptyset$) and ($T_j \in PR$) and ($T_i = (TT \cup PE)$) and there exists some x, such that (($\mu(dr(j, x)) < \mu(T_i)$ in s) or ($dr(j, x) = v_0$)); or

(e) ($(SW_j \cap SW_i) \neq \emptyset$) and ($T_i \in PR$) and ($T_j \in TT$) and there exists some x, such that ($T_j = dw(i, x)$); or

(f) ($(SW_j \cap SW_i) \neq \emptyset$ and ($T_j \in PR$) and ($T_i \in TT$) and there exists some x, such that (($\mu(dw(j, x)) < \mu(T_i)$ in s) *or* ($dw(j, x) = v_0$)); or

(g) ($(SR_j \cap SW_i) \neq \emptyset$) and ($T_i \in PR$) and ($T_j \in (PE \cup NE \cup TT)$) and ((there exists some t, x, such that ($T_t \in TT$) and ($T_t = dw(i, x)$) and (T_j reads x from T_t in s)) or (there exists some x, $x \in SW_i$ such that for all v, $T_v \subset TT$: $\neg$(T_j reads x from T_v in s))); or

(h) ($(SR_j \cap SW_i) \neq \emptyset$) and ($T_i \in PR$) and ($T_j \in PR$) ((there exist some t, x, such that ($T_t \in TT$) and ($T_t = dw(i, x)$) and ($T_t = dr(j, x)$)) or ($dr(j, x) = v_0$))) □

Theorem 5.7 *Given a* P*-requesting set PR and a valid database system state* $Q = (PE, NE, TT, s)$*, there exists a valid database system state* $Q_1 = (PE_1, NE_1, TT_1, s_1)$ *such that* Q_1 *is a valid* P*-execution of PR and Q under the FTWP, IVWP, DSRD and DSWT constraints and* $PE_1 = (PR \cup PE)$ *if and only if the IDRW dependency graph of Q and PR is acyclic.*

Proof. Suppose G is acyclic, by topologically sorting G, we can obtain a serial schedule s_1 of the set of transactions $T = (PR \cup PE \cup NE \cup TT)$, such that if the arc($T_i$, T_j) exists in U, then T_i is ordered after T_j. One can verify that the set of arcs (b) in the *IDRW* dependency graph G imposes an ordering of all transactions in s_1 that satisfies the *FTWP* and *IVWP* constraints. The set of arcs (a), (b) and (g) in G imposes an ordering in s_1 that preserves all the read from relations in s. The set of arcs (a), (c), (d) and (h) in G imposes an ordering in s_1 that satisfies the *DSRD* constraint. The set of arcs (b), (e) and (f) in G imposes an ordering in s_1 that satisfies the *DSWT* constraint.

Conversely, if there exists a valid database system state $Q_1 = (PE_1, NE_1, TT_1, s_1)$ such that Q_1 is a valid P-execution of PR and Q under the *FTWP*, *IVWP*, *DSRD* and *DSWT* constraints and $PE_1 = (PR \cup PE)$, then the graph indicating the order of every pair of transactions in s_1, which we call G_1, must be acyclic. Furthermore, every arc in the *IDRW* dependency graph G must be included in G_1. This is because: if condition (a) in Definition 5.1 is true, but the arc (T_i, T_j) $\notin G_1$, i.e., $\mu(T_j) < \mu(T_i)$ in s_1 is not true, then either the consistency condition in Definition 3.2 or the *FTWP* constraint will be violated. If condition (b) is true, but (T_i, T_j) $\notin G_1$, then the *IVWP* constraint will be violated. If condition (c) is true, but (T_i, T_j) $\notin G_1$, then the *DSRD* constraint will be violated. If condition (d) is true, but (T_i, T_j) $\notin G_1$, then either the *DSRD* or the *FTWP* or the *IVWP* constraint will be violated. If condition (e) is true but (T_i, T_j) $\notin G_1$, then the *DSWT* constraint will be violated. If condition (f) is true, but (T_i, T_j) $\notin G_1$, then either the *DSWT* or the *FTWP* constraint will be violated. If condition (g) is true, but (T_i, T_j) $\notin G_1$, then either the *DSWT* constraint or the consistency condition in Definition 3.2 will be violated. If condition (h) is true, but

$(T_i, T_j) \notin G_1$, then either the *DSRD* or the *DSWT* constraint will be violated. Thus G_1 is a supergraph of G, and since G_1 is acyclic, G must also be acyclic. □

Theorem 5.8 *Given a* P-*requesting set PR and a valid database system state* $Q=(PE, NE, TT, s)$, *the problem of determining whether there exists a valid database system state* $Q_1=(PE_1, NE_1, TT_1, s_1)$ *such that* Q_1 *is a valid* P-*execution of PR and Q under the FTWP, IVWP, (DSRD or LTRD) and (DSWT or UPDW) constraints and* $PE_1=(PR \cup PE)$ *can be determined in* $O(|V||X|^2)$ *time.* □

The proof of Theorem 5.8 also immediately follows from the fact that the constraint *LTRD* is a special case of the constraint *DSRD*, and the constraint *UPDW* is a special case of the constraint *DSWT*. Again, one can replace the *LTRD* constraint by the *DSRD* constraint and replace the *UPDW* constrint by the *DSWT* constraint in exactly the same way as explained before. Then one can construct the *IDRW* dependency graph and determine whether it is acyclic in $O(|V||X|^2)$ time.

In the case where one wishes to achieve more concurrency by allowing more than one read or write request to be scheduled simultaneously, the NP-completeness results in Theorems 5.1–5.2 and 5.4–5.5 in combination with the polynomial time result in Theorem 5.7–5.8 indicate what additional constraints must be added in order to obtain an on-line concurrency control algorithm that achieves an optimal level of concurrency in polynomial time and whose computation time is not exponential relative to the number of simultaneously scheduled requesting transactions.

Now suppose the *IDRW* dependency graph G of Q is cyclic, what can we do then? An optimal solution can be found by finding a largest subset PRs of the P-requesting set of transactions PR, such that the subgraph Gs of G is acyclic and Gs is obtained by removing nodes belonging to the P-requesting set $(PR-PRs)$. Then a virtual schedule s_1 in which all P-requesting transactions in that largest subset PRs are put into execution in Q_1 can be constructed by topologically sorting Gs.

This is explained by an example below:

Example 5.1 Suppose we wish to construct a valid maximum concurrency P-execution of the P-requesting set PR_3 and the valid database system state Q_3 in Example 3.1 under the *FTWP*, *IVWP*, *LTRD* and *UPDW* constraints. Then $dr[3, a]=dr[4, a]=dr[5, a]=T_1$; $dr[3, c]=dr[4, f]=dr[5, g]=v_0$; $dw[3, f]=dw[3, g]=dw[3, c]=dw[4, b]=dw[4, c]=dw[5, e]=v_0$; $dw[5, a]=T_1$; The *IDRW* dependency graph of Q_3 and PR_3 is shown in Fig. 1.

Since the *IDRW* dependency graph $G1$ of Q_3 and PR_3 is cyclic, there exists no valid database system state Q'_4 such that Q'_4 is a valid P-execution of Q_3 and PR_3 under the *FTWP*, *IVWP*, *LTRD* and *UPDW* constraints, and the whole set of P-requesting transactions $PR_3=\{T_3, T_4, T_5\}$ can be put into execution in parallel in Q'_4 while preserving serializability. Here, the largest subset PR_{3_s} of PR_3 for which the subgraph G_{1_s} of $G1$ is acyclic and the nodes removed from $G1$ belong to $(PR_3-PR_{3_s})$ contains trwo transactions, i.e.: T_4 and T_5. Thus we determine that in order to achieve maximum concurrency, $PR_{3_s}=\{T_4, T_5\}$ should be put into execution in parallel first.

By topologically sorting G_{1_s} (Fig. 2), we obtain the serial schedule s_4 with which we can construct the new valid database system state $Q_4=(\{T_2, T_4, T_5\}, \emptyset, \{T_1\}, \langle T_2\ T_1\ T_4\ T_5\rangle)$ shown in Example 3.1 where T_4 and T_5 can

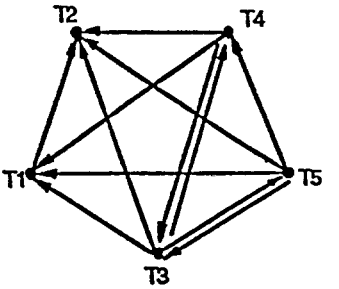

Fig. 1

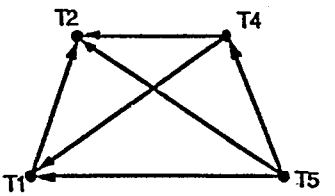

Fig. 2

be put into execution in parallel with T_2, and Q_4 is a valid P-execution of PR_3 and Q_3 under the *FTWP*, *IVWP*, *LTRD* and *UPDW* constraints. And we can continue like this constructing successive new valid system states. Notice that Q_4 is a maximum concurrency P-execution of PR_3 and Q_3 by Definition 3.7. □

The Feedback Vertex Set (FVS) problem [8] can be transformed to the problem of finding an optimal solution in the example above. Although the FVS problem is known to be NP-complete, we can always limit computational complexity by using efficient heuristics to find good approximations to an optimal solution. 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 [5, 9, 19]. An algorithm for a suboptimal solution described in [19] has a computation time upper bound of only $O(|X^3|)$.

We emphasize that in our model achieving more concurrency by scheduling more than one read or write request simultaneously is only one among many possible options which the algorithm designer has the freedom to choose from. The model certainly does not preclude the possibility of enforcing a condition that certain transactions must be scheduled in an order that is identical to their arrival order – it is a very simple task to add such a facility to an actual implementation. For example, in an algorithm based on Theorem 5.7, if it is desired that some transaction T_i be scheduled before another transaction T_j, all we have to do is add the arc(T_i, T_j) to the *IDRW* dependency graph to achieve the desired effect.

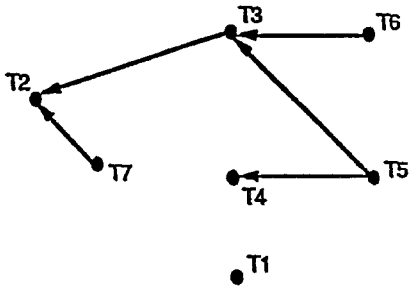

Fig. 3

In [3], a class of schedules called CPSR (Conflict-preserving serializable) is described. CPSR requires that the order of every pair of writes be preserved in the serialization whenever there is a "conflict" between the two writes, i.e., their write sets intersect. The set of constraints $\{FTWP, IVWP, (DSRD \text{ or } LTRD)$ and $(DSWT \text{ or } UPDW)\}$ is less restrictive than CPSR in the sense that it does not require that the order of two P-executing transactions be preserved whenever their writesets intersect.

6 Scheduling transactions under the *FTWP*, *IVWP* and *UB* constraints

In this section, we shall construct a polynomial time algorithm for solving the problem of scheduling transactions with predeclared writesets one at a time in a database system which supports multiple versions of data when the *FTWP* and *IVWP* constraints (if the writeset of any two transactions intersect and at least one of them has terminated, then their relative ordering must be kept invariant) is imposed.

At the end of this section, we show that these results can be generalized to the case where an upper bound is imposed on the number of P-requesting transactions scheduled each time.

Definition 6.1 We call a directed graph $G=(X, Z)$ the *IVWP* P-*dependency graph* of a valid database system state $Q=(PE, NE, TT, s)$ whenever in G there is the set of nodes $X=\{T_i | \text{all } i\colon T_i \in (PE \cup NE \cup TT)\}$, and the set of arcs $Z=\{(T_i, T_j) \in X \times X | \text{all } i, j\colon i \neq j$ and $T_i \in (PE \cup NE \cup TT)$ and $T_j \in (PE \cup NE \cup TT)$ and

(P1) $((((SW_i \cap SR_j) \neq \emptyset)$ or $(SW_j \cap SR_i) \neq \emptyset)$ and $\mu(T_j) < \mu(T_i)$ in $s)$ or

(P2) $((SW_i \cap SW_j) \neq \emptyset$ and $(T_i \in TT$ or $T_j \in TT)$ and $\mu(T_j) < \mu(T_i)$ in $s))\}$. □

Lemma 6.1 *In valid database system state $Q=(PE, NE, TT, s)$, for all i, j, such that T_i, $T_j \in (PE \cup NE \cup TT)$, if there exists a path from T_j to T_i in the IVWP* P-*dependency graph of Q, then in any valid database system state $Q_1=(PE_1, NE_1, TT_1, s_1)$ such that Q_1 is a valid* P-*execution of $\{T_r\}$ and Q under the FTWP and IVWP constraints, $\mu_1(T_i) < \mu_1(T_j)$ in s_1 must be satisfied in Q_1.*

Proof. It should be easy to see that for any two transactions $T_i, T_j \in (PE \cup NE \cup TT)$, if condition *(P1)* and *(P2)* is satisfied in Q, but $\mu_1(T_i) < \mu_1(T_j)$ in s_1 is not satisfied in Q_1, then either the *FTWP* and *IVWP* constraints would not be satisfied, or at least one read from relation in s would be changed in s_1, which violates the consistency condition in Definition 3.2. □

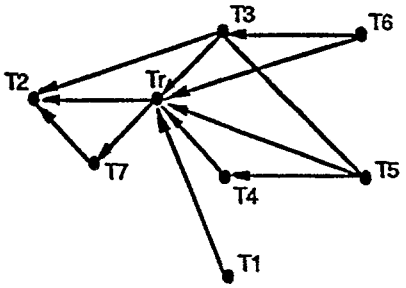

Fig. 4

Example 6.1 The *IVWP* P-dependency graph $G=(X, Z)$ of valid database system state $Q=(PE, NE, TT, s)$ where $PR=\{T_r\}$, $PE=\{T_1, T_4, T_6\}$, $NE=\{T_7\}$, $TT=\{T_2, T_3, T_5\}$, $s=T_1\ T_2\ T_3\ T_4\ T_5\ T_6\ T_7$ and $T_1=([\,], [z])$, $T_2=([a], [b])$, $T_3=([c], [a])$, $T_4=([\,], [y])$, $T_5=([\,], [c, y])$, $T_6=([a], [x])$, $T_7=([b], [\,])$ and $T_r=([x, y, z], [a, b])$ is shown in Fig. 3. □

Below we introduce a set $PB(T_r)$ called the "P-boundary set of T_r". $PB(T_r)$ is the set of transactions that must be serialized after T_r in any valid execution of Q and T_r under the *FTWP* and *IVWP* constraints.

Definition 6.2 In valid database system state $Q=(PE, NE, TT, s)$ for each P-requesting transaction $T_r \in PR$, we define the *P-boundary set of* T_r: $PB(T_r)$ as follows: for all i, j, such that $T_i \in (PE \cup NE \cup TT)$ and $T_j \in (PE \cup NE \cup TT)$:

(*initial set*) $T_i \in PB(T_r)$ if

(a) $T_i \in PE$ and there exists some x, such that $x \in (SW_i \cap SR_r)$ and there exists no t, such that $T_t \in TT$ and $x \in SW_t$ and $\mu(T_i) < \mu(T_t)$ in s.

(*inductive set*) $T_i \in PB(T_r)$ if $T_j \in PB(T_r)$ and

(b) $T_j \in (PE \cup NE \cup TT)$ and $T_i \in (PE \cup NE \cup TT)$ and there exists a path from T_i to T_j in the *IVWP* P-dependency graph of Q, or

(c) $T_j \in (PE \cup NE \cup TT)$ and $T_i \in TT$ and there exists some x, such that $x \in (SW_i \cap SW_r \cap SR_j)$ and T_j reads x from T_i in s, or

(d) $T_j \in TT$ and $T_i \in PE$ and there exists some x, such that $x \in (SW_i \cap SR_r \cap SW_j)$ and $\mu(T_i) < \mu(T_j)$ in s and there exists no t, $T_t \in TT$ such that $x \in SW_t$ and $\mu(T_i) < \mu(T_t) < \mu(T_j)$ in s. □

Lemma 6.2 *In valid database system state* $Q=(PE, NE, TT, s)$, *for all* i, r *such that* $T_i \in (PE \cup NE \cup TT)$ *and* $T_r \in PR$: *if* $T_i \in PB(T_r)$, *then in any valid database system state* $Q_1=(PE_1, NE_1, TT_1, s_1)$ *such that* Q_1 *is a valid* P-*execution of* $\{T_r\}$ *and* Q *under the FTWP and IVWP constraints and* T_r *is put into execution in* Q_1 : $\mu_1(T_r) < \mu_1(T_i)$ *in* s_1 *must be satisfied in* Q_1.

Proof. We prove Lemma 6.2 induction. For all T_i in the initial set, if $\mu_1(T_r) < \mu_1(T_i)$ in s_1 is not satisfied, then Condition (a) and the *FTWP* and *IVWP* constraints imply that either T_r reads x from $T_i \in PE$ in s_1, or T_r reads x from some other transaction $T_i' \notin TT$ in s_1, which violates the existence condition in Definition 3.2.

As an induction hypothesis, suppose that $\mu_1(T_r) < \mu_1(T_j)$ in s_1 must be satisfied for all $T_j \in PB(T_r)$. Condition (b) and Lemma 6.1 imply that $\mu_1(T_j) < \mu_1(T_i)$ in s_1 must be satisfied. This and our induction hypothesis imply that $\mu_1(T_r) < \mu_1(T_i)$ in s_1 must be satisfied. Condition (c) implies that T_j must read x from

T_i in s_1. This and our induction hypothesis imply that $\mu_1(T_r)<\mu_1(T_i)$ in s_1 must be satisfied, otherwise T_j will read x from T_r, which violates the consistency condition in Definition 3.2. Condition (d) and the *FTWP* and *IVWP* constraints and our induction hypothesis also imply that $\mu_1(T_r)<\mu_1(T_i)$ in s_1 must be satisfied, otherwise either T_r reads x from $T_i \in PE$, or T_r reads x from some other transaction $T_i' \notin TT$, which violates the existence condition in Definition 3.2. □

The P-boundary set $PB(T_r)$ of a given valid database system state $Q=(PE, NE, TT, s)$ where $T_r \in PR$ can be found by the following procedure (Suppose $PB'(T_r)$ is the set of transactions already known to be in $PB(T_r)$ at any intermediate stage of the computation):

1) Find the initial P-boundary set of T_r in Q according to condition (a) in Definition 6.2. If the initial P-boundary set of T_r in Q is empty then terminate with $PB(T_r)=\emptyset$.

2) Construct the *IVWP* P-dependency graph $G=(X, Z)$ of Q according to Definition 6.1.

3) For each $T_j \in PB'(T_r)$ such that the current step has not been performed before for T_j, find all T_i such that $T_i \in ((PE \cup NE \cup TT)-PB'(T_r))$ and T_i satisfies (b), (c) and (d) related to T_j in Definition 6.2 and put T_i in $PB'(T_r)$. When no new transaction T_i in $((PE \cup NE \cup TT)-PB'(T_r))$ which satisfies (b), (c) and (d) can be found for any $T_j \in PB'(T_r)$, then $PB(T_r)=PB'(T_r)$.

Example 6.2 The P-boundary set $PB(T_r)$ of the same valid database system state Q as given in Example 6.1 can be found as follows:

First we find all transactions belonging to the initial P-boundary set of T_r in Q by checking condition (a) in Definition 6.2 on all T_i such that $T_i \in PE$ in Q. Here $T_6 \in PE$ and $x \in (SW_6 \cap SR_r)$ and there does not exist any $T_t \in TT$ such that $x \in SW_t$ and $\mu(T_6)<\mu(T_t)$ in s. Similarly, $T_1 \in PE$ and $z \in (SW_1 \cap SR_r)$ and there does not exist any $T_t \in TT$ such that $z \in SW_t$ and $\mu(T_1)<\mu(T_t)$ in s. According to condition (a) in Definition 6.2: $T_6 \in PB'(T_r)$ and $T_1 \in PB'(T_r)$.

Next we construct the *IVWP* P-dependency graph $G=(X, Z)$ of Q (Fig. 3).

Then for all $T_j \in PB'(T_r)$ such that the same step has not been performed before for T_j, we check conditions (b), (c) and (d) on all $T_i \in ((PE \cup NE \cup TT)-PB'(T_r))$. From $T_6 \in PB'(T_r)$, we find $T_3 \in TT$ and $a \in (SW_3 \cap SW_r \cap SR_6)$ and T_6 reads a from T_3 in s. According to condition (c): $T_3 \in PB'(T_r)$. From $T_3 \in PB'(T_r)$, we find T_5 such that there exists a path from T_5 to T_3 in the *IVWP* P-dependency graph G of Q. According to condition (b): $T_5 \in PB'(T_r)$. From $T_5 \in PB'(T_r)$ and $T_5 \in TT$, we find $T_4 \in PE$ and $y \in (SW_4 \cap SR_r \cap SW_5)$ and $\mu(T_4)<\mu(T_5)$ in s and there does not exist any $T_t \in TT$ such that $y \in SW_t$ and $\mu(T_4)<\mu(T_t)<\mu(T_5)$ in s. According to condition (d): $T_4 \in PB'(T_r)$. After $PB'(T_r)=\{T_6, T_1, T_3, T_5, T_4\}$ is found, no other transaction in $((PE \cup NE \cup TT)-\{T_r\})$ can be found which satisfies (b), (c) and (d) related to any transaction $T_j \in PB'(T_r)$. Thus $PB(T_r)=\{T_6, T_1, T_3, T_5, T_4\}$. □

Definition 6.3 Given valid database system state $Q=(PE, NE, TT, s)$ and P-requesting transaction $T_r \in PR$, we call the directed graph $G_1=(X_1, Z_1)$ the *P-boundary graph of* T_r in Q whenever in G there is the set of nodes $X_1=(\{T_r\} \cup PE \cup NE \cup TT)$; and the set of arcs $Z_1=(Z \cup Z' \cup Z'')$; where

(a) Z is the set of arcs in the *IVWP* P-dependency graph $G=(X, Z)$ of Q;

(b) $Z'=\{(T_i, T_r) \mid \text{all } i,\ T_i \in PB(T_r)\}$;

(c) $Z''=\{(T_r, T_i) \mid \text{all } i,\ T_i \in ((PE \cup NE \cup TT)-PB(T_r))\}$. □

Example 6.3 The P-boundary graph $G_1=(X_1, Z_1)$ of T_r in the valid database system state Q given in Example 6.1 is shown in Fig. 4. □

Theorem 6.1 *Given valid database system state $Q=(PE, NE, TT, s)$ and* P-*requesting transaction $T_r \in PR$, there exists a valid database system state $Q_1=(PE_1, NE_1, TT_1, s_1)$ such that Q_1 is a valid* P-*execution of $\{T_r\}$ and Q under the FTWP and IVWP constraints and T_r is put into execution in Q_1, i.e.: $PE_1=(PE \cup \{T_r\})$, iff in Q:*

(PI) *for all i, $T_i \in (PE \cup NE \cup TT)$: if there exists some x, such that $x \in (SR_i \cap SW_r)$, and for all v such that $T_v \in (PE \cup NE \cup TT)$: $\neg$(T_i reads x from T_v in s) then $T_i \notin PB(T_r)$.*

Condition (PI) states that for all transactions T_i, if T_i reads the initial version for some variable name x in s, i.e.: it does not read x from any other transaction in s, and x also belongs to the writeset of P-requesting transaction T_r, then T_i should not belong to the P-boundary set of T_r in Q.

Proof. (PI) is a necessary condition because from Lemma 6.3, $T_i \in PB(T_r)$ implies that $\mu_1(T_r) < \mu_1(T_i)$ in s_1 must be satisfied. If (PI) is not satisfied, then T_i would read x from T_r, which violates the consistency condition in Definition 3.2.

Now we show that (PI) is a sufficient condition. From Definition 6.3, any serial schedule s_1 obtained by topologically sorting G_1 has the following form: $s_1 = s'' T_r s'$, where all transactions in the sub-schedule s' belong to the set $PB(T_r)$, and all transactions in the sub-schedule s'' belong to the set $((PE \cup NE \cup TT) - PB(T_r))$. Furthermore, since all arcs in the *IVWP* dependency graph of Q are included in the P-boundary graph G of T_r in Q, the *FTWP* and *IVWP* constraints are satisfied in s_1. No $T_j' \in PB(T_r)$ in s' can read any x from T_r in s_1, because if $T_j' \in PB(T_r)$ read the initial version of x in s, then (PI) will not hold; whereas if T_j' read x from some transaction $T_i'' \notin PB(T_r)$ in s, then condition (c) in Definition 6.2 implies $T_i'' \in PB(T_r)$ which is a contradiction. T_r cannot read from any $T_i \in PE$ in s_1, where $T_i \notin PB(T_r)$, otherwise this and the *FTWP* and *IVWP* constraints together with condition (d) in Definition 6.2 imply that $T_i \in PB(T_r)$, which is also a contradiction. Hence Q_1 is a valid P-execution of $\{T_r\}$ and Q under the *FTWP* and *IVWP* constraints. □

In the proof of Theorem 6.1, we simultaneously proved the following:

Theorem 6.2 *Given valid database system state $Q=(PE, NE, TT, s)$ and* P-*requesting transaction $T_r \in PR$, if* (PI) *is satisfied in Q, then the following database system state Q_1 is a valid* P-*execution of $\{T_r\}$ and Q under the FTWP and IVWP constraints: $Q_1=(PE_1, NE_1, TT_1, s_1)$ where $PE_1=(PE \cup \{T_r\})$ and $TT_1=TT$ and s_1 is a serial schedule of $(\{T_r\} \cup PE \cup NE \cup TT)$ constructed by topologically sorting the* P-*boundary graph $G_1=(X_1, Z_1)$ of T_r in Q.* □

The proof of Theorem 6.1 leads to the following algorithm for scheduling one P-requesting transaction $T_r \in PR$ at a time in a given valid database system state $Q=(PE, NE, TT, s)$ (Suppose $PB'(T_r)$ is the set of transactions already known to be in $PB(T_r)$ at any intermediate stage of the computation):

1) Find the initial P-boundary set of T_r in Q according to condition (a) in Definition 6.2. If the initial P-boundary set of T_r in Q is empty then terminate with $s_1 = s T_r$.

2) Construct the *IVWP* P-dependency graph $G=(X, Z)$ of Q according to Definition 6.1.

3) For each $T_j \in PB'(T_r)$ such that the current step has not been performed before for T_j, find all T_i such that $T_i \in ((PE \cup NE \cup TT) - PB'(PR))$ and T_i satisfies (b), (c) and (d) related to T_j in Definition 6.2 and put T_i in $PB'(T_r)$. If (PI) is not satisfied for some $T_j \in PB'(T_r)$, then terminate $-T_r$ cannot be put into execution in any valid P-execution of $\{T_r\}$ and Q under the *FTWP* and *IVWP* constraints. When no new transaction T_i in $((PE \cup NE \cup TT) - PB'(T_r))$ which satisfies (b), (c) and (d) can be found for any $T_j \in PB'(T_r)$, then $PB(T_r) = PB'(T_r)$.

4) Construct the P-boundary graph $G_1 = (X_1, Z_1)$ of T_r in Q according to Definition 6.3.

5) Topologically sort G_1 to obtain the serial schedule s_1. Terminate.

If the computation of the algorithm terminates in either Step 1 or Step 5, then T_r can be put into execution in the database system state $Q_1 = (PE_1, NE_1, TT_1, s_1)$ where $PE_1 = (PE \cup \{T_r\})$, $TT_1 = TT$ and s_1 is obtained by Step 1 or Step 5 above and Q_1 is a valid P-execution of $\{T_r\}$ and Q under the *FTWP* and *IVWP* constraints.

Corollary 6.1 *Given valid database system state $Q = (PE, NE, TT, s)$ and* P-*requesting transaction $T_r \in PR$, the problem of determining whether there exists at least one database system state $Q_1 = (PE_1, NE_1, TT_1, s_1)$ such that Q_1 is a valid* P-*execution of $\{T_r\}$ and Q under the FTWP and IVWP constraints and $PE_1 = (PE \cup \{T_r\})$ can be determined in $O(|V||PE \cup NE \cup TT|^2)$ time.*

Furthermore, if (PI) *is satisfied in Q, then a database system state $Q_1 = (PE_1, NE_1, TT_1, s_1)$ such that Q_1 is a valid* P-*execution of $\{T_r\}$ and Q under the FTWP and IVWP constraints and $PE_1 = (PE \cup \{T_r\})$ can be constructed in $O(|V||PE \cup NE \cup TT|^2)$ time.* □

Example 6.4 Given the same valid database system state Q as in Example 6.1, we can obtain the following serial schedule s_1 by topologically sorting the P-boundary graph $G_1 = (X_1, Z_1)$ of T_r in Q: (Fig. 4) $s_1 = \langle T_2\ T_7\ T_r\ T_1\ T_3\ T_4\ T_5\ T_6 \rangle = R_2[a]\ W_2[b]\ R_7[b]\ W_7 \square R_r[x, y, z]\ W_r[a, b]\ R_1 \square W_1[z]\ R_3[c]\ W_4[a]\ R_4 \square W_4[y]\ R_5 \square W_5[c, y]\ R_6[a]\ W_6[x]$. The new database system state $Q_1 = (PE_1, NE_1, TT_1, s_1)$ where $PE_1 = \{T_1, T_4, T_6, T_r\}$, $NE_1 = \{T_7\}$, $TT_1 = \{T_2, T_3, T_5\}$ and $s_1 = \langle T_2\ T_7\ T_r\ T_1\ T_3\ T_4\ T_5\ T_6 \rangle$ is a valid P-execution of $\{T_r\}$ and Q under the *FTWP* and *IVWP* constraints and T_r is put into execution in Q_1, i.e.: $PE_1 = (PE \cup \{T_r\})$. □

Corollary 6.1 indicates that if one imposes the constraint of scheduling one requesting transaction at a time, then one only needs two other additional constraints $-$ *FTWP* and *IVWP*, i.e., one does not need to add the constraints ((*LTRD* or *DSRD*) and (*UPDW* or *DSWT*)) as in Theorem 5.8 in order to obtain a polynomial time algorithm.

The results obtained above can be generalized to the case where an upper bound is imposed on the number of predeclared writeset transactions scheduled each time.

Theorem 6.3 *The following problem can be determined in polynomial time: Given valid database system state $Q = (PE, NE, TT, s)$ and* P-*requesting set PR such that for some constant C, $|PR| < C$; Does there exist a valid database system state $Q_1 = (PE_1, NE_1, TT_1, s_1)$, such that $PE_1 = (PE \cup PR)$ and Q_1 is a valid* P-*execution of PR and Q under the FTWP and IVWP constraints?*

Proof. The following procedure can be used to determine the existence of Q_1:

1. Generate all permutations of the set of P-requesting transactions PR: $\{s_i\}$
2. For each permutation $s_i = T_i^n\, T_i^{n-1} \ldots, T_i^2\, T_i^1$, $n = |PR|$, $i = 1, 2, \ldots, n!$, try to put each P-requesting transaction into the P-executing set and the virtual schedule one by one according to the order of the permutation as follows:

2.1. Use condition (PI) in Theorem 6.1 to determine whether there exists a valid database system state $Q_i^1 = (PE_i^1, NE_i^1, TT_i^1, s_i^1)$ such that $PE_i^1 = (PE \cup \{T_i^1\})$ and Q_i^1 is a valid P-execution of $\{T_i^1\}$ and Q under the *FTWP* and *IVWP* constraints. If (PI) is satisfied for $PB(T_i^1)$ in Q, then construct Q_i^1 according to Theorem 6.2.

2.2. For each j, $j = 2, 3, \ldots, n$, respectively do the following:

Redefine the P-boundary set $PB(T_i^j)$ in Q_i^{j-1} as follows: Let the initial P-boundary set of T_i^j in Q_i^{j-1} be the union of $\{T_i^{j-1}, T_i^{j-2}, \ldots, T_i^1\}$ and the set defined by condition (a) in Definition 6.2. Let all other members of $PB(T_i^j)$ in Q_i^{j-1} be still defined by (b), (c) and (d) in Definition 6.2. Use this new definition of $PB(T_i^j)$ in Q_i^{j-1} together with condition (PI) in Theorem 6.1 to determine whether there exists a valid database system state $Q_i^j = (PE_i^j, NE_i^j, TT_i^j, s_i^j)$ such that $PE_i^j = (PE_i^{j-1} \cup \{T_i^j\})$ and Q_i^j is a valid P-execution of $\{T_i^j\}$ and Q_i^{j-1} under the *FTWP* and *IVWP* constraints and for all k, $k = j-1, \ldots, 1$: $\mu_i^j(T_i^j) < \mu_i^j(T_i^k)$ in s_i^j (including $T_i^{j-1}, \ldots, T_i^1$ in the initial P-boundary set of T_i^j in Q_i^{j-1} implies that the latter condition must be satisfied in Q_i^j). If (PI) is satisfied for the redefined P-boundary set $PB(T_i^j)$ in Q_i^{j-1}, then construct Q_i^j according to Theorem 6.2.

By reasoning similar to that in the proof of Theorem 6.1 and 6.2, it is straightforward to prove that there exists a valid database system state $Q_1 = (PE_1, NE_1, TT_1, s_1)$ such that $PE_1 = (PE \cup PR)$ and Q_1 is a valid P-execution of PR and Q, if and only if for at least one i, $1 \leqq i \leqq n!$: Q_i^1 is a valid P-execution of $\{T_i^1\}$ and Q under the *FTWP* and *IVWP* constraints and for all j, $j = 2, 3, \ldots, n$: Q_i^j is a valid P-execution of $\{T_i^j\}$ and Q_i^{j-1} under the *FTWP* and *IVWP* constraints and for all k, $k = j-1, \ldots, 1$, $\mu_i^j(T_i^j) < \mu_i^j(T_i^k)$ in s_i^j. Furthermore, if this condition is satisfied, then Q_i^n is a valid P-execution of PR and Q under the *FTWP* and *IVWP* constraints and $PE_i^n = (PE \cup PR)$.

Note that this procedure requires the same order of computation time as the algorithm presented above for scheduling P-requesting transactions one at a time, since the same algorithm can be used here without substantial modification for constructing $Q_i^1, \ldots, Q_i^n$ (the only difference is that $T_i^{j-1}, \ldots, T_i^n$ is added to the initial P-boundary set of T_i^j in Q_i^{j-1} in Step 2.2), and the number of times the algorithm must be applied is bounded above by $n \times n!$, where $n = |PR| < C$. □

Example 6.5 Suppose we are given the valid database system state $Q = (PE, NE, TT, s)$ where $TT = \{T_1\}$, $PE = NE = \emptyset$, $s = \langle T_1 \rangle$, $T_1 = ([\,], [b])$, $T_2 = ([a], [b])$, $T_3 = ([b], [a])$ and a set of P-requesting transactions $PR = \{T_2, T_3\}$.

The set of all permutations of PR has two elements: $s_1 = T_2\, T_3$ and $s_2 = T_3\, T_2$. For the permutation $s_1 = T_2\, T_3$, from Step 2.1, one can obtain the valid database system state $Q_1^1 = (\{T_3\}, \emptyset, \{T_1\}, \langle T_1\, T_3 \rangle)$ such that Q_1^1 is a valid P-execution of $\{T_3\}$ and Q under the *FTWP* and *IVWP* constraints and $PE_1^1 = (PE \cup \{T_3\})$. By adding T_3 to the initial P-boundary set of T_2 in Q_1^1 and applying the algorithm, from Step 2.2, one can obtain the valid database system state $Q_1^2 = (\{T_2, T_3\}, \emptyset, \{T_1\}, \langle T_2\, T_1\, T_3 \rangle)$ such that Q_1^2 is a valid P-execution of $\{T_2\}$ and Q_1^1 under the *FTWP* and *IVWP* constraints and $PE_1^2 = (PE_1^1 \cup \{T_2\})$ and $\mu_1^2(T_2)$

$<\mu_1^2(T_3)$ in s_1^2. Q_1^2 is also a valid P-execution of PR and Q under the $FTWP$ and $IVWP$ constraints where $PE_1^2=(PE\cup PR)$. Note that applying the Steps 2.1 and 2.2 according to the permutation $s_2=T_3\,T_2$ will not lead to any valid database system state Q_1, such that Q_1 is a valid P-execution of PR and Q under the $FTWP$ and $IVWP$ constraints and $PE_1=(PE\cup PR)$. □

Finally, we mention that if we wish to solve the same problem as in Theorem 6.2 or the same problem as in Theorem 6.3 but with the *DSRD* constraint added, all we have to do is add the set of arcs (c), (d) and (h) of the *IDRW* dependency graph defined in Definition 5.1 to the P-boundary graph defined in Definition 6.3 and follow the same procedure as described earlier.

If we wish to solve the same problem as in Theorem 6.2 or the same problem as in Theorem 6.3 but with the *DSWT* constraint added, then we should add the set of arcs (b), (e), (f) and (g) of the *IDRW* dependency graph defined in Definition 5.1 to the P-boundary graph defined in Definition 6.3 and follow the same procedure as described earlier.

If only the *DSRD* constraint is added then the corresponding algorithm can be used to solve the problem of terminating nonpredeclared writeset transactions under the *FTWP*, *IVWP* and *UB* constraints as described in Sect. 4. The algorithm will achieve an optimal level of concurrency in polynomial time for the problem of terminating nonpredeclared writeset transactions since the versions that must be read by each transaction in the N-commit set are fixed already – they must be the same versions that were read by each corresponding transaction in the N-executing set.

7 Summary of complexity results

Table 1 below summarizes the computational complexity of the problem of determining whether a set of predeclared writeset transactions can be put into execution while guaranteeing that their future writes will never compromise correctness of the database system under any combination of the constraints *FTWP*, *LTRD* or *DSRD*, *IVWP*, *UPDW* or *DSWT*, 1*V*, 2*V* and *UB*.

The results in Table 1 are respectively proved in Theorems 5.1 and 5.4; 5.2 and 5.5; 5.3 and 5.6; 6.1; 5.7 and 5.8.

In the following tables, "yes" signifies that the corresponding constraint is imposed, while " – " signifies that the corresponding constraint is not imposed.

For each NP-complete problem:
If "yes" is replaced by " – ", then problem remains NP-complete.
If " – " is replaced by "yes", then problem becomes polynomial time solvable.

For each polynomial time solvable problem:
If " – " is replaced by "yes", then problem remains polynomial time solvable.
If "yes" is replaced by " – ", then problem becomes NP-complete.

This suggests that each NP-complete problem below contains a maximal subset of constraints, while each polynomial time solvable problem below contains a minimal subset of constraints.

Table 1

FTWP	*LTRD* or *DSRD*	*IVWP*	*UPDW* or *DSWT*	1*V*	2*V*	*UB*	
yes	yes	yes	—	—	yes	—	NP
yes	yes	—	yes	—	yes	yes	NP
yes	—	yes	yes	—	yes	—	NP
yes	—	yes	—	—	—	yes	P
yes	yes	yes	yes	—	—	—	P

Note that here the constraint 1*V* trivially implies the constraints *FTWP*, *LTRD*, *IVWP* and *UPDW*. The constraint 2*V* trivially implies the constraint *FTWP*

Table 2

FTWP	*IVWP*	*UPDW* or *DSWT*	1*V*	2*V*	*UB*	
yes	yes	—	—	yes	—	NP
yes	—	yes	—	yes	yes	NP
yes	yes	—	—	—	yes	P
yes	yes	yes	—	—	—	P

Note that the constraint 1*V* trivially implies the constraints *FTWP*, *IVWP* and *UPDW*. The constraint 2*V* trivially implies the constraint *FTWP*

Table 2 below summarizes the computational complexity of the problem of determining whether the writesets of a set of non-predeclared writeset transactions can be written in the database while preserving correctness under any combination of the constraints *FTWP*, *IVWP*, *UPDW* or *DSWT*, 1*V*, 2*V* or *UB*.

These results are proved by the fact that the problem of constructing valid N-terminations of a N-commit set and a valid database system state is equivalent to the problem of constructing valid P-executions of a P-requesting set and a valid database system state under the *DSRD* constraint.

8 Conclusions

In this paper, we presented a new model for studying the concurrency vs. computation time tradeoffs involved in on-line multiversion database concurrency control. The major difference between our model and previous models is the basic problem that is studied.

While most previous models study the problem of determining whether a schedule s representing the output of a concurrency control algorithm belongs to a certain class C of serializable schedules, our model studies the following problem: Given some previous state of the database system determine whether a new database system state exists (and construct the new database system state if it does exist) in which a set of read and write requests can be satisfied while preserving consistency of the database system.

In our model the following two basic problems were studied: (1) Preventive scheduling of predeclared writeset transactions, so that they never have to be restarted. (2) When one or more non-predeclared writeset transactions finish their computation and announce their writesets, determine if they can write into the database (while preserving serializability) without aborting any transactions.

It was shown that the first problem is actually equivalent to the second problem when the constraint *DSRD* is imposed on the first problem. Consequently, all the complexity results that we have derived for the first problem under the *DSRD* constraint can be directly applied to the second problem.

We proved that the set of restrictions {*FTWP*, (*LTRD* or *DSRD*), *IVWP*, (*UPDW* or *DSWT*)} and the set of restrictions {*FTWP*, *IVWP*, *UB*} define two fundamental limits of performance achievable by polynomial time concurrency control algorithms. If any one constraint in the minimal sets is omitted, although it could increase the amount of concurrency, it would also have the dramatic negative effect of making the scheduling problem NP-complete; whereas if we do not omit any constraint in the minimal sets, then the scheduling problem can be solved in polynomial time. With each one of these minimal set of restrictions, we constructed an efficient scheduling algorithm that achieves an optimal level of concurrency in polynomial computation time.

In the following we briefly discuss how previously proposed concurrency control algorithms fit into our framework. By examining previously proposed concurrency control algorithms in the literature one can observe that in general they do not allow more than one read or write request to be scheduled simultaneously. Furthermore, previously proposed multiversion concurrency control algorithms do not exhaustively examine *all* possibilities of allowing each transaction to read or write *any* version, and they impose a fixed ordering of all versions for each variable name in the database. This implies that in general, previously proposed algorithms impose at least all the following constraints (in fact, each previously proposed concurrency control algorithm individually imposes more constraints than the common subset listed below): (i) $|PR|=1$ or $|NE'_c|=1$; (ii) (*DSWT* or *UPDW*) and (*DSRD* or *LTRD*); (iii) *FTWP* and *IVWP*.

Thus it should be easy to see that previously proposed algorithms achieve less concurrency than the polynomial time algorithms corresponding to the two minimal sets in this paper. This does not mean that one can always achieve more concurrency than previous algorithms for a particular database application because in many cases the application itself may require that certain constraints be imposed. However, our results do provide interesting insight in determining whether and what additional constraints must be enforced to obtain a polynomial time concurrency control algorithm that achieves an optimal level of concurrency for a given set of application imposed constraints.

Finally, we mention that the complexity results obtained within the two step transaction model can be easily extended to an n-step transaction model [22]. In an n-step model of transactions a transaction can be associated with three sets of variable names: a readset, a writeset and a predeclared writeset which respectively are the set of variable names for which that transaction has previously read, written or intends to write in future a version. These three sets are not invariant as in our two step model. They may all grow as new read, write or predeclared write requests are satisfied. However, when these three sets become known dynamically then one cannot guarantee anymore that

a transaction will never abort. (One can only guarantee that a transaction will never be aborted at the time a request is accepted if that transaction does not make any unacceptable further requests to read or write additional version values). Despite these differences the conditions that must be satisfied when scheduling requests in an n-step transaction model are basically the same as when constructing valid P-(N-)executions and P-(N-)terminations of a current database system state in the two step model and consequently the complexity results inTheorems 5.1–5.8 and Theorems 6.1–6.3 in the two step model carry over to the n-step model.

Acknowledgements. The author is greatly indebted to Pierre-Jacques Courtois. Without his help, encouragement and support this work could never have been done. The author also wishes to express his gratitude to Michel Sintzoff and Elie Milgrom for their support, encouragement and suggestions. Philippe van Bastelaer, Gérard Roucairol provided helpful comments on earlier versions of this work. Kenneth C. Sevcik also provided helpful comments and suggestions during the author's presentations of earlier versions of this work. The author is grateful to the anonymous referees, whose comments and suggestions have led to important improvements in the presentation of this paper.

References

1. Bernstein, P.A., Goodman, N.: Concurrency control in distributed database systems. ACM Comput. Surv. **13**, 185–221 (1981)
2. Bernstein, P.A., Goodman, N.: Multiversion concurrency control – Theory and algorithms. ACM Trans. Database Syst. **8**, 465–483 (1983)
3. Bernstein, P.A., Shipman, D.W., Wong, S.W.: Formal aspects of serializability in database concurrency control. IEEE Trans. Software Eng. **SE-5**, 203–216 (1979)
4. Casanova, M.A.: The concurrency control problem for database systems. (Lect. Notes Comput. Sci., Vol. 116) Berlin Heidelberg New York: Springer 1981
5. Diaz, M. et al.: A note on minimal and quasi-minimal essential sets in complex directed graphs. IEEE Trans. Circuit Theory **CT-19**, 512–513 (1972)
6. Eswaran, K.P., Gray, J.N., Lorie, R.A., Traiger, I.L.: The notions of consistency and predicate locks in a database system.CAM **19**, 624–633 (1976)
7. Franaszek, P., Robinson, J.T.: Limitations on concurrency in transaction processing. ACM Trans. Database Syst. **10**, 1–28 (1985)
8. Garey, M.R., Johnson, D.S.: Computers and intractability: A guide to the theory of NP-completeness. San Francisco: Freeman 1979
9. Guardabassi, G.: A note on minimal essential sets. IEEE Trans. Circuit Theory **CT-18**, 557–560 (1971)
10. Ibaraki, T., Kameda, T., Katoh, N.: Cautious transaction schedulers for database concurrency control. IEEE Trans. Software Eng. **14**, 997–1009 (1988)
11. Ibaraki, T., Kameda, T., Minoura, T.: Serializability with constraints. ACM Trans. Database Syst. **12**, 429–452 (1987)
12. Katoh, N., Ibaraki, T., Kameda, T.: Cautious transaction schedulers with admission control. ACM Trans. Database Syst. **10**, 205–229 (1985)
13. Kung, H.T., Papadimitriou, C.H.: An optimality theory of concurrency control for databases. Acta Inf. **19**, 1–11 (1983)
14. Krishnamurthy, R., Dayal, U.: Theory of serializability for a parallel model of transactions. Proceedings ACM Symposium on Principles of Database Systems, California, pp. 293–305, 1982
15. Papadimitriou, C.H.: The serializability of concurrent database updates. J. ACM **26**, 631–653 (1979)
16. Papadimitriou, C.H., Kanellakis, P.: On concurrency control by multiple versions. ACM Trans. Database Syst. **9**, 89–99 (1984)
17. Rosenkrantz, D.J., Stearns, R.E., Lewis, P.M.: System level concurrency control for distributed database systems. ACM Trans. Database Syst. **3**, 178–198 (1978)

18. Silberschatz, A., Kadem, Z.: Consistency in hierarchical database systems. J. ACM **27**, 72–80 (1980)
19. Smith, G.W., Walford, R.B.: The identification of a minimal feedback vertex set of a directed graph. IEEE Trans. Circuits Syst. **Cas-22**, 9–15 (1975)
20. Vidyasankar, K.: Generalized theory of serializability. Acta Inf. **24**, 105–119 (1987)
21. Xu, J.: A formal model for maximum concurrency in transaction systems with predeclared writesets. Proceedings of the 8th International Conference on Very Large Data Bases, pp. 77–90, 1982
22. Xu, J.: The complexity of database concurrency control. Doctorate Dissertation, Unité d'Informatique, Université Catholique de Louvain, Belgium, 1984
23. Yannakakis, M.: A theory of safe locking policies in database systems. J. ACM **29**, 718–740 (1982)

Appendix

Proof of Theorem 5.1

Proof. It is easy to see that *LI* is in *NP*, because a non deterministic algorithm need only guess a new valid database system state Q_1, in which s_1 is a serial schedule of $T=(PR \cup PE \cup NE \cup TT)$ and check in polynomial time that Q_1 is a valid P-execution of *PR* and *Q* under the *LTRD* and *IVWP* constraints.

Below, we accomplish our proof by transforming a well known NP complete problem – the "GRAPH K-COLORABILITY" problem (GKC) [8] to *LI*.

GKC is as follows: Given a graph $G=(V, E)$ and a positive integer $K \leqq N$, 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 $K \leqq N$, where $N=|V|$, is an arbitrary instance of GKC. We now construct a valid database system state $Q=(PE, NE, TT, s)$ and a set *PR* of P-requesting transactions, such that there exists a valid database system state $Q_1=(PE_1, NE_1, TT_1, s_1)$ which is a valid P-execution of *PR* and *Q* under the *LTRD* and *IVWP* constraints if and only if G is K-colorable.

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

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

In each color component $[i,j]$, $i=1, 2, \ldots, N$, $j=1, 2, \ldots, K$, we let: $a[i,j] \in SWt[i,j]$, $a[i,j] \in SWr[i,j]$, $a[i,j] \in SRe[i,j]$.

In each node component i, $i=1, 2, \ldots, N$, we let: $b[i,j,k] \in SWe[i,j]$, $b[i,j,k] \in SRr[i,k]$ for all j, $k: 1 \leqq j, k \leqq K$ and $j \neq k$; $c[i,j+1,j] \in SWr[i,j+1]$, $c[i,j+1,j] \in SRt[i,j]$ for all $j: 1 \leqq j \leqq K-1$; and $c[i, 1, K] \in SWr[i, 1]$, $c[i, 1, K] \in SRt[i, K]$.

For all p, q such that $1 \leqq p, q \leqq N$, $p \neq q$ and node p and node q are connected, we let: $d[p,q,j] \in SWe[p,j]$, $d[p,q,j] \in SRr[q,j]$ for all $j: 1 \leqq j \leqq K$.

We further construct a serial schedule $s = Tt[1,1]\ Te[1,1]\ Tt[1,2]\ Te[1,2]\ \ldots\ Tt[i,j]\ Te[i,j]\ \ldots\ Tt[N,K]\ Te[N,K]$.

Then we collect all the transactions constructed above together to form the three sets: $PE = \{Te[i,j]\}$ $TT = \{Tt[i,j]\}$ $PR = \{Tr[i,j]\}$ for all i, j, $i = 1, 2, \ldots, N$, $j = 1, 2, \ldots, K$. We let NE be empty.

It is not difficult to see that the valid database system state $Q = (PE, NE, TT, s)$ and the requesting set PR of P-transactions thus constructed can be constructed in polynomial time.

Notice that in s, in every color component $[i,j]$, $Te[i,j]$ reads $a[i,j]$ from $Tt[i,j]$. Also, in s no transaction reads any of the variable names $b[i,j,k]$, $c[i,j,k]$, $d[p,q,k]$ from any other transaction. From the consistency condition of Definition 3.2, in any valid P-execution $Q_1 = (PE_1, NE_1, TT_1, s_1)$ of PR and Q, the same read from relations must be satisfied, i.e., in s_1, for every color component $[i,j]$, $Te[i,j]$ must read $a[i,j]$ from $Tt[i,j]$. Also, in s_1 no transaction should read any of the variable names $b[i,j,k]$, $c[i,j,k]$, $d[p,q,k]$ from any other transaction.

We now prove that there exists a valid database system state $Q_1 = (PE_1, NE_1, TT_1, s_1)$ which is a valid P-execution of PR and Q under the *LTRD* and *IVWP* constraints if and only if G is K-colorable.

By the way we have constructed color component $[i,j]$, it is easy to verify that in any serial schedule s_1 which includes $Tr[i,j]$, $Tt[i,j]$ and $Te[i,j]$ and Q_1 is a valid P-execution of PR and Q, one and only one of the following formulas must be satisfied for the three transactions $Tr[i,j]$, $Tt[i,j]$ and $Te[i,j]$ in each color component $[i,j]$: NOTCOLORED(i,j): $\mu_1(Tr[i,j]) < \mu_1(Tt[i,j]) < \mu_1(Te[i,j])$ in s_1 (exclusive) or COLORED(i,j): $\mu_1(Tt[i,j]) < \mu_1(Te[i,j]) < \mu_1(Tr[i,j])$ in s_1. This is because if neither NOTCOLORED(i,j) nor COLORED(i,j) is satisfied in s_1, then $Te[i,j]$ would not read $a[i,j]$ from $Tt[i,j]$ in s_1.

In each node component i, for any two color components $[i,j]$ and $[i,k]$, $k \neq j$, $\mu_1(Tr[i,k]) < \mu_1(Te[i,j])$ in s_1 must hold, otherwise $Tr[i,k]$ would read $b[i,j,k]$ from $Te[i,j]$ in s_1. For a similar reason, $\mu_1(Tr[i,j]) < \mu_1(Te[i,k])$ in s_1 must hold. This implies that COLORED(i,j) and COLORED(i,k) cannot hold simultaneously, otherwise they would generate the following cycle: $\mu_1(Te[i,j]) < \mu_1(Tr[i,j]) < \mu_1(Te[i,k]) < \mu_1(Tr[i,k]) < \mu_1(Te[i,j])$.

In each node component i, $\mu_1(Tt[i,j]) < \mu_1(Tr[i,j+1])$ in s_1 for all j: $1 \leqq j \leqq K-1$, and $\mu_1(Tt[i,K]) < \mu_1(Tr[i,1])$ in s_1 must hold, otherwise for some j, $Tt[i,j]$ would read $c[i,j+1,j]$ from $Tr[i,j+1]$ in s_1 or $Tt[i,K]$ would read $c[i,1,K]$ from $Tr[i,1]$ in s_1. This implies that for at least one j, COLORED(i,j) must be true, otherwise the following cycle would be created: $\mu_1(Tr[i,1]) < \mu_1(Tt[i,1]) < \mu_1(Tr[i,2]) < \mu_1(Tt[i,2]) \ldots < \mu_1(Tr[i,K]) < \mu_1(Tt[i,K]) < \mu_1(Tr[i,1])$.

Suppose that node p and node q are connected, then $\mu_1(Tr[q,k]) < \mu_1(Te[p,k])$ in s_1 must hold, otherwise $Tr[q,k]$ would read $d[p,q,k]$ from $Te[p,k]$. Similarly, $\mu_1(Tr[p,k]) < \mu_1(Te[q,k])$ in s_1 must hold. This implies that COLORED(p,k) and COLORED(q,k) cannot simultaneously hold, otherwise this would generate the following cycle: $\mu_1(Te[p,k]) < \mu_1(Tr[p,k]) < \mu_1(Te[q,k]) < \mu_1(Tr[q,k]) < \mu_1(Te[p,k])$.

So far we have proved that in any serial schedule s_1 of valid database system state $Q_1 = (PE_1, NE_1, TT_1, s_1)$ such that Q_1 is a valid P-execution of PR and Q, in each node component i, COLORED(i,j) must be satisfied for one and

only one color component j, and if node p and node q are connected, then COLORED(p,j) and COLORED(q,j) cannot simultaneously be satisfied for two color components in node components p and q which correspond to the same color j.

Suppose that there exists Q_1 which is a valid P-execution of PR and Q. We set node i to color j iff COLORED(i,j) is satisfied for color component $[i,j]$ in s_1 in Q_1. Then each node will be set to one and only one color, and connected nodes will be set to different colors.

Conversely, if the graph G is K-colorable, then we divide all transactions in $(PR \cup PE \cup NE \cup TT)$ into $3+2K-1$ disjoint sets: $SET[1]=\{Te[i,j]|$node i is NOT colored $j\}$; $SET[2]=\{Tr[i,j]|$node i is colored $j\}$; $SET[3]=\{Te[i,j]|$ node i is colored $j\}$; $SET[4]=\{Tt[i,j]|$node i is colored $(j \bmod K)+1\}$; $SET[5]=\{Tr[i,j]|$node i is colored $(j \bmod K)+1\}$; ... $SET[3+2m-1]=\{Tt[i,j]|$ node i is colored $((j+m-1) \bmod K)+1\}$; $SET[3+2m]=\{Tr[i,j]|$node i is colored $((j+m-1) \bmod K)+1\}$; ... $SET[3+2(K-1)-1]=\{Tt[i,j]|1 \leqq i \leqq N$, node i is colored $((j+(K-1)-1) \bmod K+1)\}$; $SET[3+2(K-1)]=\{Tr[i,j]|1 \leqq i \leqq N$, node i is colored $((j+(K-1)-1) \bmod K+1)\}$; $SET[3+2K-1]=\{Tt[i,j]|$node i is colored $j\}$ where $1 \leqq i \leqq N$, $1 \leqq j \leqq K$.

We can construct a valid database system state $Q_1=(PE_1, NE_1, TT_1, s_1)$ as follows: $PE_1=(PE \cup PR)$, $NE_1=NE$, $TT_1=TT$, and $s_1=\langle SET[3+2K-1]\rangle$ $\langle SET[3+2(K-1)]\rangle$ $\langle SET[3+2(K-1)-1]\rangle$ ... $\langle SET[3+2m]\rangle$ $\langle SET[3+2m-1]\rangle$ ... $\langle SET[5]\rangle$ $\langle SET[4]\rangle\langle SET[3]\rangle$ $\langle SET[2]\rangle$ $\langle SET[1]\rangle$. ("$\langle SET[k]\rangle$" denotes any serial schedule of the transactions in $SET[k]$). One can verify that in s_1: for all i, j, $Te[i,j]$ reads $a[i,j]$ from $Tt[i,j]$, which is the same as in s, and for all i, j, $Tr[i,j]$ reads the initial version for every variable name in its read set. Also in s_1, as in s, no transaction reads any of the variable names $b[i,j,k]$, $c[i,j,k]$, $d[p,q,k]$ from any other transaction. Thus both the consistency condition and existence condition of Definition 3.2 are satisfied, and Q_1 is a valid P-execution of PR and Q. Since the write sets of all transactions in $(PE \cup TT)$ do not intersect with each other, the $FTWP$, $IVWP$ and $2V$ constraints are satisfied in Q_1. Since none of the terminated transactions in TT have written a version for any variable name that is in a read set of any P-requesting transaction in PR, the $LTRD$ constraint is also satisfied in Q_1. □

Proof of Theorem 5.2

Proof. We transform the "GRAPH K-COLORABILITY" problem to UI in the same fashion as in the proof of Theorem 5.1. Below, we show the construction of the valid database system state $Q=(PE, NE, TT, s)$, for which there exists a valid database system state $Q_1=(PE_1, NE_1, TT_1, s_1)$ which is a P-execution of Q and PR under the $FTWP$, $IVWP$, $UPDW$ and $2V$ constraints and $PE_1=(PE \cup PR)$ if and only if graph G is K-colorable.

Here, each color component $[i,j]$ is also composed of three transactions: one terminated transaction $Tt[i,j]=([SRt[i,j]], [SWt[i,j]])$, one P-executing transaction $Te[i,j]=([SRe[i,j]], [SWe[i,j]])$ and one P-requesting transaction $Tr[i,j]=([SRr[i,j]], [SWr[i,j]])$.

In each color component $[i,j]$, $i=1, 2, \ldots, N$, $j=1, 2, \ldots, K$, we let: $a[i,j] \in SWt[i,j]$, $a[i,j] \in SRr[i,j]$, $a[i,j] \in SWe[i,j]$.

In each node component i, $i=1, 2, \ldots, N$, we let: $b1[i,j,k] \in SWt[i,j]$, $b2[i,j,k] \in SRr[i,k]$ for all j, k: $1 \leqq j,k \leqq K$ and $j \neq k$; $c[i,j+1,j] \in SWr[i,j+1]$, $c[i,j+1,j] \in SRe[i,j]$ for all j: $1 \leqq j \leqq K-1$, and $c[i,1,K] \in SWr[i,1]$, $c[i,1,k] \in SRe[i,K]$.

For all $1 \leqq p,q \leqq N$, $p \neq q$ and node p and node q are connected, we let: $d1[p,q,j] \in SWt[p,j]$, $d2[p,q,j] \in SRr[q,j]$ for all j: $1 \leqq j \leqq K$.

We further construct two sets of auxiliary P-executing transactions TBE and TDE as follows: $TBE=\{Tbe[i,j,k]\}$ for all i, j, k, $1 \leqq i \leqq N$, $1 \leqq j,k \leqq K$ and $j \neq k$, where $SRbe[i,j,k]=\{b1[i,j,k]\}$ and $SWbe[i,j,k]=\{b2[i,j,k]\}$;

$TDE=\{Tde[p,q,j]\}$ for all $1 \leqq p,q \leqq N$, $p \neq q$ and node p and node q are connected, where $SRde[p,q,j]=\{d1[p,q,j]\}$ and $SWde[p,q,j]=\{d2[p,q,j]\}$.

We then construct a serial schedule $s=Te[1,1]\,Te[1,2] \ldots Te[N,K]$ $Tbe[1,1,2]$ $Tbe[1,1,3] \ldots Tbe[N,K,K-1]\langle\{Tde[p,q,j]\}\rangle$ $Tt[1,1]\,Tt[1,2] \ldots Tt[N,K]$.

Then we collect all the transactions constructed above together to form the three sets: $PE=\{Te[i,j]\} \cup TBE \cup TDE$, $TT=\{Tt[i,j]\}$, $PR=\{Tr[i,j]\}$ for $i=1, 2, \ldots, N$, $j=1, 2, \ldots, K$, we let $NE=\emptyset$.

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

Notice that in s: for all i, j, $i=1, 2, \ldots, N$, $j=1, 2, \ldots, K$: $a[i,j] \in (SWe[i,j] \cap SWt[i,j])$ and $\mu(Te[i,j])<\mu(Tt[i,j])$. Also, in s no transaction reads any of the variable names $b1[i,j,k]$, $b2[i,j,k]$, $c[i,j,k]$, $d1[p,q,k]$ $d2[p,q,k]$ from any other transaction; from the consistency condition of Definition 3.2, no transaction should read any of these variable names from any other transaction in s_1.

In each color component $[i,j]$, one and only one of the following formulas must be satisfied for the three transactions $Tr[i,j]$, $Te[i,j]$ and $Tt[i,j]$: NOTCOLORED(i,j): $\mu_1(Tr[i,j])<\mu_1(Te[i,j])<\mu_1(Tt[i,j])$ in s_1 (exclusive) or COLORED(i,j): $\mu_1(Te[i,j])<\mu_1(Tt[i,j])<\mu_1(Tr[i,j])$ in s_1.

Either NOTCOLORED(i,j) or COLORED(i,j) must be satisfied in each color component $[i,j]$ because: $\mu_1(Te[i,j])<\mu_1(Tt[i,j])$ must be satisfied in s_1, otherwise the $IVWP$ constraint will not be satisfied in Q_1; and: $Tr[i,j]$ cannot be positioned between $Te[i,j]$ and $Tt[i,j]$, otherwise $Tr[i,j]$ will read $a[i,j]$ from $Te[i,j]$, which violates the existence condition in Definition 3.2.

In each node component i, for any two color components $[i,j]$ and $[i,k]$, $k \neq j$, $\mu_1(Tr[i,k])<\mu_1(Tbe[i,j,k])<\mu_1(Tt[i,j])$ in s_1 must hold. Otherwise $Tr[i,k]$ will read $b2[i,j,k]$ from $Tbe[i,j,k]$, which violates the existence condition in Definition 3.2; or $Tbe[i,j,k]$ will read $b1[i,j,k]$ from $Tt[i,j]$ in s_1, which violates the consistency condition in Definition 3.2. Similarly, $\mu_1(Tr[i,j])<\mu_1(Tbe[i,k,j])<\mu_1(Tt[i,k])$ in s_1 must hold. This implies that COLORED(i,j) and COLORED(i,k) cannot hold simultaneously.

In each node component i, $\mu_1(Te[i,j])<\mu_1(Tr[i,j+1])$ in s_1 for all j, such that $1 \leqq j \leqq K-1$ and $\mu_1(Te[i,K])<\mu_1(Tr[i,1])$ in s_1 must hold. Otherwise for some j, $Te[i,j]$ will read $c[i,j+1,j]$ from $Tr[i,j+1]$ in s_1 or $Te[i,K]$ will read $c[i,1,K]$ from $Te[i,1]$ in s_1. This implies that for at least one j, COLORED(i,j) must be true.

For all p, q, such that $1 \leqq p,q \leqq N$, $p \neq q$ and node p and node q are connected: $\mu_1(Tr[q,k])<\mu_1(Tde[p,q,k])<\mu_1(Tt[p,k])$ in s_1 for all k, $1 \leqq k \leqq K$ must hold. Otherwise $Tr[q,k]$ will read $d2[p,q,k]$ from $Tde[p,q,k]$ in s_1, or $Tde[p,q,k]$ will read $d1[p,q,k]$ from $Tt[p,k]$ in s_1. Similarly, $\mu_1(Tr[p,k])<$

$\mu_1(Tde[q,p,k]) < \mu_1(Tt[q,k])$ in s_1 for all k, $1 \leqq k \leqq K$ must hold. This implies that COLORED(p,k) and COLORED(q,k) cannot simultaneously hold.

Conversely, if the given graph G is K-colorable, then we can construct a database system state $Q_1 = (PE_1, NE_1, TT_1, s_1)$, where $PE_1 = (PE \cup PR)$, $NE_1 = \emptyset$, $TT1 = TT$ and $s1 = \langle \{Te[i,j] | \text{node } i \text{ is colored } j\} \rangle \langle \{Tr[i,j] | \text{node } i \text{ is colored } ((j+K-2) \bmod K+1)\} \rangle \langle \{Te[i,j] | \text{node } i \text{ is colored } ((j+K-2) \bmod K+1)\} \rangle \langle \{Tr[i,j] | \text{node } i \text{ is colored } ((j+K-3) \bmod K+1)\} \rangle \langle \{Te[i,j] | \text{node } i \text{ is colored } ((j+K-3) \bmod K+1)\} \rangle \ldots \langle \{Tr[i,j] | \text{node } i \text{ is colored } ((j+K-l) \bmod K+1)\} \rangle \langle \{Te[i,j] | \text{node } i \text{ is colored } ((j+K-l) \bmod K+1)\} \rangle \ldots \langle \{Tr[i,j] | \text{node } i \text{ is colored } ((j \bmod K)+1)\} \rangle \langle \{Te[i,j] | \text{node } i \text{ is colored } ((j \bmod K)+1)\} \rangle \langle \{Tbe[i,j,k] | \text{node } i \text{ is colored } j\} \rangle \langle \{Tde[p,q,k] | \text{node } p \text{ is colored } k \text{ and node } p \text{ and node } q \text{ are connected}\} \rangle \langle \{Tt[i,j] | \text{node } i \text{ is colored } j\} \rangle \langle \{Tr[i,j] | \text{node } i \text{ is colored } j\} \rangle \langle \{Tbe[i,j,k] | \text{node } i \text{ is NOT colored } j\} \rangle \langle \{Tde[p,q,k] | \text{node } p \text{ is NOT colored } k \text{ and node } p \text{ and node } q \text{ are connected}\} \rangle \langle \{Tt[i,j] | \text{node } i \text{ is NOT colored } j\} \rangle$ where $1 \leqq i \leqq N$, $1 \leqq j \leqq K$, $1 \leqq p, q \leqq N$, $1 \leqq k \leqq K$.

One can verify that in s_1: for all i, j, either $Tr[i,j]$ reads $a[i,j]$ from $Tt[i,j]$, or $Tr[i,j]$ reads the initial version of $a[i,j]$. Also, in s_1 no transaction reads any of the variable names $b1[i,j,k]$, $b2[i,j,k]$, $c[i,j,k]$, $d1[p,q,k]$, $d2[p,q,k]$ from any other transaction, which is the same as in s. Thus both the consistency condition and existence condition of Definition 3.2 are satisfied, and Q_1 is a valid P-execution of PR and Q. Since the write sets of all transactions in $(PE \cup TT)$ do not share any other variable names except for $a[i,j] \in (SWe[i,j] \cap SWt[i,j])$ and $\mu(Te[i,j]) < \mu(Tt[i,j])$ in s_1, which is the same as in s, the $FTWP$, $IVWP$ and $2V$ constraints are satisfied in Q_1. Since none of the terminated transactions in TT have written a version for any variable name that is in a write set of any P-requesting transaction in PR, the $UPDW$ constraint is also satisfied in Q_1. □

Proof of Theorem 5.3

Proof. It is easy to see that $LU1$ is in NP, because given any valid database system state $Q = (PE, NE, TT, s)$ and the set of P-requesting transactions $PR = \{Tr\}$, a non deterministic algorithm need only guess a new valid database system state $Q_1 = (PE_1, NE_1, TT_1, s_1)$ such that $PE_1 = PE \cup \{Tr\}$, $NE_1 = NE$, $TT_1 = TT$, and s_1 is a serial schedule of $T = (\{Tr\} \cup PE \cup NE \cup TT)$, and check in polynomial time that Q_1 is a valid P-execution of $PR = \{Tr\}$ and Q under the $FTWP$, $LTRD$ and $UPDW$ constraints.

Below, we accomplish our proof by transforming a well known NP-complete problem – 3-satisfiability of Boolean formulas (3-SAT) [8] to $LU1$.

3-SAT is as follows: given a set of clauses C on a finite set of variables U such that each clause in C contains three literals, does there exist a truth assignment for U that satisfies all the clauses in C?

Let $C = \{c[1], c[2], \ldots, c[M]\}$ be the set of clauses and $U = \{u[1], u[2], \ldots, u[N]\}$ be the set of variables in an arbitrary instance of 3-SAT. We now construct a valid database system state $Q = (PE, NE, TT, s)$ and a P-requesting transaction Tr such that the $2V$ constraint is satisfied in Q and there exists a valid system state $Q_1 = (PE_1, NE_1, TT_1, s_1)$ which is a valid P-execution of $PR = \{Tr\}$ and Q under the $FTWP$, $LTRD$ and $UPDW$ constraints if and only if C is satisfiable.

For each variable $u[i] \in U$, $i=1, 2, \ldots, N$, we construct a "variable component i" that consists of two terminated transactions: $Tt[i, 1]=([SRt[i, 1]], [SWt[i, 1]])$ and $Tt[i, 2]=([SRt[i, 2]], [SWt[i, 2]])$.

For each literal $z[j, k]=u[i]$ or $z[j, k]=\bar{u}[i]$ in each clause $c[j]=(z[j, 1], z[j, 2], z[j, 3])$, $k=1, 2, 3$, $j=1, 2, \ldots, M$, we construct a "literal component $[j, i]$" which is composed of four P-executing transactions: $Te[j, i, 1]=([SRe[j, i, 1]], [SWe[j, i, 1]])$, $Te[j, i, 2]=([SRe[j, i, 2]], [SWe[j, i, 2]])$, $Te[j, i, 3]=([SRe[j, i, 3]], [SWe[j, i, 3]])$ and $Te[j, i, 4]=([SRe[j, i, 4]], [SWe[j, i, 4]])$.

For each literal component $[j, i]$, and its corresponding variable component i, $j=1, 2, \ldots, M$, $i=1, 2, \ldots, N$, we let: $a[j, i] \in SWt[i, 1]$, $a[j, i] \in SRe[j, i, 1]$, $a[j, i] \in SWe[j, i, 2]$; $b[j, i] \in SWt[i, 2]$, $b[j, i] \in SRe[j, i, 3]$, $b[j, i] \in SWe[j, i, 4]$; $c[j, i] \in SWe[j, i, 4]$, $c[j, i] \in SRt[i, 1]$; $d[j, i] \in SWe[j, i, 2]$, $d[j, i] \in SRt[i, 2]$; $e[j, i] \in SWe[j, i, 3]$, $e[j, i] \in SRe[j, i, 2]$; $f[j, i] \in SWe[j, i, 1]$, $f[j, i] \in SRe[j, i, 4]$;

We construct one P-requesting transaction $Tr=(SRr, SWr)$.

For each clause $c[j]=(z[j, 1], z[j, 2], z[j, 3])$, where $(z[j, 1]=u[i_1]$ or $z[j, 1]=\bar{u}[i_1])$ and $(z[j, 2]=u[i_2]$ or $z[j, 2]=\bar{u}[i_2])$ and $(z[j, 3]=u[i_3]$ or $z[j, 3]=\bar{u}[i_3])$ for $j=1, 2, \ldots, M$, $1 \leqq i_1, i_2, i_3 \leqq N$, we have one "clause component j" that consists of the three literal components $[j, i_1]$, $[j, i_2]$, $[j, i_3]$. For every literal component $[j, i_k]$, $k=1, 2, 3$, we define one "output node" $Tout[j, i_k]=([SRout[j, i_k]], [SWout[j, i_k]])$ and one "input node" $Tin[j, i_k]=([SRin[j, i_k]], [SWin[j, i_k]])$ as follows: iff $z[j, k]=u[i_k]$ then $Tout[j, i_k]=Te[j, i_k, 1]$ and $Tin[j, i_k]=Te[j, i_k, 2]$, else iff $z[j, k]=\bar{u}[i_k]$ then $Tout[j, i_k]=Te[j, i_k, 3]$ and $Tin[j, i_k]=Te[j, i_k, 4]$. For every clause component j, we let: $w[j, i_1, 0] \in SRr$, $w[j, i_1, 0] \in SWout[j, i_1]$, $w[j, i_3, i_2] \in SRin[j, i_2]$, $w[j, i_3, i_2] \in SWout[j, i_3]$, $w[j, i_2, i_1] \in SRin[j, i_1]$, $w[j, i_2, i_1] \in SWout[j, i_2]$, $w[j, 0, i_3] \in SRin[j, i_3]$, $w[j, 0, i_3] \in SWr$.

We construct a serial schedule s of all P-executing and terminated transactions constructed above: $s=\langle\{Tt[i, 2]\}\rangle\langle\{Te[j, i, 2]\}\rangle\langle\{Tt[i, 1]\}\rangle\langle\{Te[j, i, 3] | z[j, 1]=\bar{u}[i]\}\rangle\langle\{Te[j, i, 4] | z[j, 1]=\bar{u}[i]\}\rangle\langle\{Te[j, i, 3] | z[j, 2]=\bar{u}[i]\}\rangle\langle\{Te[j, i, 4] | z[j, 2]=\bar{u}[i]\}\rangle\langle\{Te[j, i, 3] | z[j, 3]=\bar{u}[i]\}\rangle\langle\{Te[j, i, 4] | z[j, 3]=\bar{u}[i]\}\rangle\langle\{Te[j, i, 3] | z[j, k]=u[i]\}\rangle\langle\{Te[j, i, 4] | z[j, k]=u[i]\}\rangle\langle\{Te[j, i, 1]\}\rangle$ where $1 \leqq j \leqq M$, $1 \leqq i \leqq N$ and $1 \leqq k \leqq 3$. ($\langle T\rangle\langle T'\rangle \ldots$ means a serival schedule obtained by a concatenation of the serial schedules of the sets T, $T', \ldots$ of transactions).

Then we collect all the transactions constructed above to form the four sets: $PR=\{Tr\}$; $PE=\{Te[j, i, 1], Te[j, i, 2], Te[j, i, 3], Te[j, i, 4] |$ for all j, k, i, $j=1, 2, \ldots, M$, $k=1, 2, 3$, such that $c[j]=(z[j, 1], z[j, 2], z[j, 3])$ in C and $(z[j, k]=u[i]$ or $z[j, k]=\bar{u}[i])$ and $1 \leqq i \leqq N\}$; $NE=\emptyset$; $TT=\{Tt[i, 1], Tt[i, 2] |$ for all i, $i=1, 2, \ldots, N\}$.

It is not difficult to see that the system state $Q=(PE, NE, TT, s)$ thus constructed is a valid database system state which satisfies the $2V$ constraint and can be constructed in polynomial time.

Notice that in s, in every literal component $[j, i]$ and its corresponding variable component i, $Te[j, i, 1]$ reads $a[j, i]$ from $Tt[i, 1]$ and $Te[j, i, 3]$ reads $b[j, i]$ from $Tt[i, 2]$ in s. Also in s, no transaction reads any of the variable names $c[j, i]$, $d[j, i]$, $e[j, i]$, $f[j, i]$, $w[j, i_1, i_2]$, $1 \leqq j \leqq M$, $1 \leqq i, i_1, i_2 \leqq N$ from any other transaction. From the consistency condition of Definition 3.2, in any valid P-execution $Q_1=(PE_1, NE_1, TT_1, s_1)$ of PR and Q, the same read from relations must be satisfied, i.e., in s_1, for every literal component $[j, i]$, $Te[j, i, 1]$

must read $a[j,i]$ from $Tt[i,1]$ and $Te[j,i,3]$ must read $b[j,i]$ from $Tt[i,2]$ in s. Also, in s_1 no transaction should read any of the variable names $c[j,i]$, $d[j,i]$, $e[j,i]$, $f[j,i]$, $w[j,i_1,i_2]$, $1 \leqq j \leqq M$, $1 \leqq i, i_1, i_2 \leqq N$ from any other transaction.

We now prove that there exists a valid database system state $Q_1 = (PE_1, NE_1, TT_1, s_1)$ such that Q_1 is a valid P-execution of PR and Q if and only if C is satisfiable.

By the way we have constructed each literal component $[j,i]$, it is easy to verify that in any serial schedule s_1 which includes $Tt[i,1]$, $Te[j,i,1]$ and $Te[j,i,2]$ and Q_1 is a valid P-execution of PR and Q, one and only one of the following formulas must be satisfied for the three transactions $Tt[i,1]$, $Te[j,i,1]$ and $Te[j,i,2]$ in each literal component $[j,i]$: $T1(j,i)$: $\mu_1(Te[j,i,2]) < \mu_1(Tt[i,1]) < \mu_1(Te[j,i,1])$ in s_1 (exclusive) or $F1(j,i)$: $\mu_1(Tt[i,1]) < \mu_1(Te[j,i,1]) < \mu_1(Te[j,i,2])$ in s_1. This is because if neither $T1(j,i)$ or $F1(j,i)$ is satisfied in s_1, then $Te[j,i,1]$ will not read $a[j,i]$ from $Tt[i,1]$ in s_1. Similarly, one and only one of the following two formulas must be satisfied for the transactions $Tt[i,2]$, $Te[j,i,3]$ and $Te[j,i,4]$ in literal component $[j,i]$: $T2(j,i)$: $\mu_1(Tt[i,2]) < \mu_1(Te[j,i,3]) < \mu_1(Te[j,i,4])$ in s_1 (exclusive) or $F2(j,i)$: $\mu_1(Te[j,i,4]) < \mu_1(Tt[i,2]) < \mu_1(Te[j,i,3])$ in s_1. This is because if neither $T2(j,i)$ nor $F2(j,i)$ is satisfied, then $Te[j,i,3]$ will not read $b[j,i]$ from $Tt[i,2]$ in s_1.

We now prove that for every literal component $[j,i]$ and its corresponding variable component i, one and only one of the following two formulas must be satisfied for the six transactions $Tt[i,1]$, $Te[j,i,1]$, $Te[j,i,2]$, $Tt[i,2]$, $Te[j,i,3]$ and $Te[j,i,4]$: TRUE(j,i): $T1(j,i)$ and $T2(j,i)$ (exclusive) or FALSE (j,i): $F1(j,i)$ and $F2(j,i)$.

Suppose the contrary, then either ($T1(j,i)$ and $F2(j,i)$) or ($T2(j,i)$ and $F1(j,i)$) must be satisfied in s_1.

Notice that $\mu_1(Tt[i,1]) < \mu_1(Te[j,i,4])$ must be satisfied in s_1, otherwise $Tt[i,1]$ would read $c[j,i]$ from $Te[j,i,4]$ in s_1; $\mu_1(Tt[i,2]) < \mu_1(Te[j,i,2])$ must also be satisfied in s_1, otherwise $Tt[i,2]$ would read $d[j,i]$ from $Te[j,i,2]$ in s_1; But this and $F2(j,i)$ and $T1(j,i)$ lead to a contradiction, since they generate the following cycle: $\mu_1(Tt[i,1]) < \mu_1(Te[j,i,4]) < \mu_1(Tt[i,2]) < \mu_1(Te[j,i,2]) < \mu_1(Tt[i,1])$. Similarly, notice that $\mu_1(Te[j,i,2]) < \mu_1(Te[j,i,3])$ must be satisfied in s_1, otherwise $Te[j,i,2]$ would read $e[j,i]$ from $Te[j,i,3]$ in s_1; $\mu_1(Te[j,i,4]) < \mu_1(Te[j,i,1])$ must also be satisfied in s_1, otherwise $Te[j,i,4]$ would read $f[j,i]$ from $Te[j,i,1]$ in s_1. But this and $F1(j,i)$ and $T2(j,i)$ lead to a contradiction, since they generate the following cycle: $\mu_1(Te[j,i,1]) < \mu_1(Te[j,i,2]) < \mu_1(Te[j,i,3]) < \mu_1(Te[j,i,4]) < \mu_1((e[j,i,1])$.

In the following, we prove that either TRUE(j,i) must hold for all literal components $[j,i]$, or FALSE(j,i) must hold for all literal components $[j,i]$, which correspond to a literal $u[i]$ or $\bar{u}[i]$. Suppose the contrary: for two literal components $[j_1,i]$ and $[j_2,i]$, $j_1 \neq j_2$, either (TRUE(j_1,i) and FALSE(j_2,i)) or (TRUE(j_2,i) and FALSE(j_1,i)) is satisfied in s_1. But TRUE(j_1,i) and FALSE(j_2,i) lead to a contradiction, since they generate the following cycle: $\mu_1(Tt[i,2]) < \mu_1(Te[j_2,i,2]) < \mu_1(Tt[i,1]) < \mu_1(Te[j_2,i,4]) < \mu_1(Tt[i,2])$. Similarly, TRUE$(j_2,i)$ and FALSE(j_1,i) also lead to a contradiction, since they generate the following cycle: $\mu_1(Tt[i,1]) < \mu_1(Te[j_1,i,4]) < \mu_1(Tt[i,2]) < \mu_1(Te[j_2,i,2]) < \mu_1(Tt[i,1])$.

Below, we define a function "VAL": VAL is a mapping from the set of

all literals $z[j,k]$, $k=1, 2, 3$, in all clauses $c[j]=(z[j,1], z[j,2], z[j,3])$ in C, $j=1,2,\ldots,M$, to the set {true, false}: VAL$(z[j,k])$=true iff $((z[j,k]=u[i]$ and TRUE$(j,i))$ or $(z[j,k]=\bar{u}[i]$ and FALSE$(j,i))$; VAL$(z[j,k])$=false iff $((z[j,k]=u[i]$ and FALSE$(j,i))$ or $(z[j,k]=\bar{u}[i]$ and TRUE$(j,i))$.

Now we prove that in any valid database system state $Q_1=(PE_1, NE_1, TT_1, s_1)$ such that s_1 is a serial schedule of $(\{Tr\}\cup PE\cup TT)$ and Q_1 is a valid P-execution of Q: for each clause $c[j]=(z[j,1], z[j,2], z[j,3])$ in C, $j=1,2,\ldots,M$, VAL$(z[j,k])$=true must be satisfied for at least one literal $z[j,k]$ in $c[j]$, $1\leqq k\leqq 3$.

In each clause component j, for the three literal components $[j,i_1]$, $[j,i_2]$ and $[j,i_3]$ which correspond to the same clause $c[j]=(z[j,1], z[j,2], z[j,3])$, where for $k=1,2,3$, $z[j,k]=u[i_k]$ or $z[j,k]=\bar{u}[i_k]$, the following formula must be satisfied:

$CL(j)$: $\mu_1(Tr)<\mu_1(T\text{out}[j,i_1])$ and $\mu_1(T\text{in}[j,i_1])<\mu_1(T\text{out}[j,i_2])$ and $\mu_1(T\text{in}[j,i_2])<\mu_1(T\text{out}[j,i_3])$ and $\mu_1(T\text{in}[j,i_3])<\mu_1(Tr)$ in s_1.

This is because if $CL(j)$ is not satisfied, then either Tr will read $w[j,i_1,0]$ from $T\text{out}[j,i_1]$, or $T\text{in}[j,i_1]$ will read $w[j,i_2,i_1]$ from $T\text{out}[j,i_2]$, or $T\text{in}[j,i_2]$ will read $w[j,i_3,i_2]$ from $T\text{out}[j,i_3]$, or $T\text{in}[j,i_3]$ will read $w[j,0,i_3]$ from Tr in s_1.

For every literal component $[j,i_k]$ in each clause component j, $TRUE(j,i_k)$ implies that $\mu_1(Te[j,i_k,3])<\mu_1(Te[j,i_k,4])$ must be satisfied in s_1; FALSE(j,i_k) implies that $\mu_1(Te[j,i_k,1])<\mu_1(Te[j,i_k,2])$ must be satisfied in s_1. Earlier, we defined $T\text{out}[j,i_k]$ to be $Te[j,i_k,1]$ and $T\text{in}[j,i_k]$ to be $Te[j,i_k,2]$ iff $z[j,k]=u[i_k]$; and we defined $T\text{out}[j,i_k]$ to be $Te[j,i_k,3]$ and $T\text{in}[j,i_k]$ to be $Te[j,i_k,4]$ iff $z[j,k]=\bar{u}[i_k]$; This and the definition of the function VAL implies that whenever VAL$(z[j,k])=$ *false*, $\mu_1(T\text{out}[j,i_k])<\mu_1(T\text{in}[j,i_k])$ must be satisfied in s_1. Suppose that for some clause $c[j]$, VAL$(z[j,1])$=false and VAL$(z[j,2])$=false and VAL$(z[j,3])$=false. This would imply that the following must be satisfied: $\mu_1(T\text{out}[j,i_1])<\mu_1(T\text{in}[j,i_1])$ and $\mu_1(T\text{out}[j,i_2])<\mu_1(T\text{in}[j,i_2])$ and $\mu_1(T\text{out}[j,i_3])<\mu_1(T\text{in}[j,i_3])$ in s_1. But this and CL(j) would lead to the following cycle: $\mu_1(Tr)<\mu_1(T\text{out}[j,i_1])<\mu_1(T\text{in}[j,i_1])<\mu_1(T\text{out}[j,i_2])<\mu_1(T\text{in}[j,i_2])<\mu_1(T\text{out}[j,i_3])<\mu_1(T\text{in}[j,i_3])<\mu_1(Tr)$. Hence VAL$(z[j,k])$=true must be satisfied for at least one literal $z[j,k]$, $1\leqq k\leqq 3$ in $c[j]$.

Suppose there exists $Q_1=(PE_1, NE_1, TT_1, s_1)$ such that s_1 is a serial schedule of $(\{Tr\}\cup PE\cup TT)$ and Q_1 is a valid P-execution of $PR=\{Tr\}$ and Q. We assign the value true to each variable $u[i]\in U$ iff TRUE(j,i) is satisfied for all transactions belonging to any literal component $[j,i]$, and assign the value false to each variable $u[i]\in U$ iff FALSE(j,i) is satisfied for all transactions belonging to any literal component $[j,i]$. From what we have proved above and the definition of the truth setting function VAL, at least one literal $z[j,k]$, $1\leqq k\leqq 3$, will be set true in each clause $c[j]=(z[j,1], z[j,2,], z[j,3])$, $j=1,2,\ldots,M$. Thus, we obtain a truth setting assignment to all variables $u[i]\in U$ which satisfies every clause $c[j]$ in C.

Conversely, we show that if there exists a satisfying truth assignment for C, then for valid database system state $Q=(PE, NE, TT, s)$ and the set of P-requesting transactions $PR=\{Tr\}$, there exists a valid database system state $Q_1=(PE_1, NE_1, TT_1, s_1)$, such that s_1 is a serial schedule of $(PR\cup PE\cup TT)$ and Q_1 is a valid P-execution of PR and Q under the *FTWP*, *LTRD*, *UPDW* and $2V$ constraints. Let $t: U\to\{T,F\}$ be a satisfying truth assignment for C.

We can construct a serial schedule s_1 of $(\{Tr\} \cup PE \cup TT)$ for the database system state $Q_1 = (PE_1, NE_1, TT_1, s_1)$ where $PE1 = (PE \cup \{Tr\})$ and $TT1 = TT$ and Q_1 is a valid P-execution of Q by topological sorting the following acyclic directed graph $G = (X, Z)$ where X is the set of nodes and Z is the set of arcs in G: $X = \{T_v \mid T_v \in (PE \cup TT \cup \{Tr\})\}$; $Z = \{(T_u, T_v) \mid$ for all u, v, j, i, $1 \leqq j \leqq M$, $1 \leqq i \leqq N$, $(t(u[i]) = T$ and $(\mathrm{TRUE}(j, i) \to \mu_1(T_v) < \mu_1(T_u)$ in $s_1))$ or $(t(u[i]) = F$ and $(\mathrm{FALSE}(j, i) \to \mu_1(T_v) < \mu_1(T_u)$ in $s_1))$ or $(CL(j) \to \mu_1(T_v) < \mu_1(T_u)$ in $s_1)\}$. Then for all i, if $t(u[i]) = T$ then s_1 would have the same ordering of transactions as in the formula $\mathrm{TRUE}(j, i)$: $T1(j, i)$ and $T2(j, i)$, i.e., $\mu_1(Te[j, i, 2]) < \mu_1(Tt[i, 1]) < \mu_1(Te[j, i, 1])$, and $\mu_1(Tt[i, 2]) < \mu_1(Te[j, i, 3]) < \mu_1(Te[j, i, 4])$ in s_1; and for all i, if $t(u[i]) = F$ then s_1 would have the same ordering of transactions as in the formula $\mathrm{FALSE}(j, i)$: $F1(j, i)$ and $F2(j, i)$, i.e., $\mu_1(Tt[i, 1]) < \mu_1(Te[j, i, 1]) < \mu_1(Te[j, i, 2])$, and $\mu_1(Te[j, i, 4]) < \mu_1(Tt[i, 2]) < \mu_1(Te[j, i, 3])$ in s_1. Also, s_1 would have the same ordering of transactions as in the formula $CL(j)$, i.e., $\mu_1(Tr) < \mu_1(T\mathrm{out}[j, i_1])$ and $\mu_1(T\mathrm{in}[j, i_1]) < \mu_1(T\mathrm{out}[j, i_2])$ and $\mu_1(T\mathrm{in}[j, i_2]) < \mu_1(T\mathrm{out}[j, i_3])$ and $\mu_1(T\mathrm{in}[j, i_3]) < \mu_1(Tr)$ in s_1.

One can verify that in s_1, as in s, for all i, j, $Te[j, i, 1]$ reads $a[j, i]$ from $Tt[i, 1]$ and $Te[j, i, 3]$ reads $b[j, i]$ from $Tt[i, 2]$. Also, in s_1, as in s, no transaction reads any of the variable names $c[j, i]$, $d[j, i]$, $e[j, i]$, $f[j, i]$, $w[j, i_1, i_2]$, $1 \leqq j \leqq M$, $1 \leqq i, i_1, i_2 \leqq N$ from any other transaction; and T_r reads the initial version for each variable name in its read set. Thus both the consistency condition and existence condition of Definition 3.2 are satisfied, and Q_1 is a valid P-execution of PR and Q. Since the write sets of all terminated transactions in TT do not share any variable names with each other, the $FTWP$ and $2V$ constraints are satisfied in Q_1. Since none of the terminated transactions in TT have written a version for any variable name that is in either the read set or the write set of the P-requesting transaction T_r, the $LTRD$ and $UPDW$ constraints are also satisfied in Q_1. □