A multiversion cautious scheduler with dynamic serialization constraints for database concurrency control

Naoki Katoh
Department of Management Science, Kobe University of Commerce, Kobe 651-21, Japan

Toshihide Ibaraki
Department of Applied Mathematics and Physics, Kyoto University, Kyoto 606, Japan

Tiko Kameda
School of Computing Science, Simon Fraser University, Burnaby, B.C., Canada V5A 1S6

Received 15 February 1990
Revised 1 March 1991

Abstract

Let $MC$ stand for a class of logs (i.e., sequences of read/write steps) that are serializable when multiple versions of the data items are maintained in the database. In this paper we propose a new type of multiversion cautious scheduler for database concurrency control, which dynamically imposes serialization constraints, consisting of all $rw$ (read-write)-constraints and a subset of other serialization constraints that is dynamically determined. We shall show that (i) the key step of our scheduler is carried out by checking the acyclicity of a certain directed graph, and hence can be done in polynomial time, (ii) this scheduler achieves a higher degree of concurrency than any existing cautious schedulers such as $MCS(MWW)$ and $MCS(MWRW)$, if concurrency is measured in terms of their fixed point sets, and (iii) it exhibits neither cancellation nor augmentation anomaly. It is also shown that our scheduler immediately grants all write requests.
1. Introduction

A transaction scheduler for database concurrency control must decide if each arriving read/write request can immediately be granted without violating serializability. A series of our papers [5, 6, 8] proposed cautious schedulers which have a nice property that they cause neither deadlocks nor rollbacks of transactions for the purpose of preserving serializability, under the assumption that each transaction, upon arrival, predeclares its read and write sets (i.e., the set of data items to be read and written respectively). The crucial part of cautious schedulers is the completion test which examines whether the future requests can be arranged so that the partial schedule already output followed by such a sequence yields a serializable schedule. This test can be done through examining certain combinatorial properties of the so-called active TIO graph (see Section 3) constructed to represent the current situation of the schedule.

In this paper we propose a new type of multiversion cautious scheduler. In a single-version schedule, a read operation on a data item \( X \) reads the most recent value of \( X \). In a multiversion schedule, on the other hand, a read operation can read either the current version or any past version of \( X \). This may increase the concurrency of schedules. Our recent paper [6] proposed two types of multiversion cautious schedulers which are important in practice in the sense that (i) both schedulers can be executed in polynomial time, (ii) they achieve higher degree of concurrency than their single-version counterparts, if concurrency is measured in terms of their fixed point sets (see Section 7), and (iii) they do not exhibit cancellation anomaly (see Section 5).

Our new scheduler in this paper has all these properties, and enjoys even higher degree of concurrency than the above two types of schedulers.

The idea behind the new scheduler is as follows. The completion test for the previous two schedulers is carried out by testing the acyclicity of the corresponding active TIO graph, which incorporates the so-called reads-from arcs as well as certain constraint arcs determined by the schedulers (see Section 3). The scheduler we propose introduces less constraint arcs than previous ones to the active TIO graph. More specifically, it dynamically imposes constraints, only when necessary, to guarantee that the acyclicity of the active TIO graph implies the success of the completion test. Since the resulting active TIO graph is less constrained, it can be shown that this new scheduler has higher degree of concurrency than the previous ones. It also has another desirable property that all write requests can be granted immediately, though read requests may be delayed as in other schedulers.

This paper is organized as follows. Section 2 describes the database model used in this paper. Section 3 reviews the multiversion cautious scheduler. Section 4 proposes a new multiversion cautious scheduler, and Section 5 proves its correctness. Section 6 shows that our scheduler is free of cancellation and read-augmentation anomalies, and Section 7 investigates its fixed point set. Finally Section 8 reports some simulation results indicating that the proposed scheduler
attains a significantly higher concurrency over the existing multiversion cautious schedulers.

2. Database system model

We describe the database system model of this paper, which is based on [5, 6, 8]. A database consists of a set \( D \) of data items, and a set \( \mathcal{S} = \{ T_0, T_1, \ldots, T_f \} \) of transactions. A write operation \( W_i[X] \) of transaction \( T_i \) creates a new version of data item \( X \), and a read operation \( R_j[X] \) of transaction \( T_j \) returns the value of a version of \( X \). Let \( S \) be a subset of \( D \). A read step, \( R_j[S] \) is an indivisible set of read operations of transaction \( T_j \), i.e., \( R_j[S] = \{ R_j[X] \mid X \in S \} \). We similarly define a write step \( W_i[S] = \{ W_i[X] \mid X \in S \} \).

The operations in a single step may be executed in any order. A transaction is a sequence of read and write steps. The set of data items read (written) by a transaction \( T_i \) is called its read (write) set and is denoted \( R_i \) (\( W_i \)). \( T_0 \) and \( T_f \) are fictitious transactions, called the initial transaction and the final transaction, respectively [16].

A sequence \( h \) of all steps in all \( T_i \in \mathcal{S} \) is called a log if (i) its first step is \( W_0[D] \) of \( T_0 \) and the last step is \( R_f[D] \) of \( T_f \), and (ii) all steps of each transaction \( T_i \in \mathcal{S} \) appear in \( h \) in the order given by \( T_i \). Given a log \( h \), a schedule is a pair \( s = (h, \mathcal{I}) \), where \( \mathcal{I} \) is a mapping, called an interpretation [17] or version assignment, from all read operations into the set of write operations such that \( \mathcal{I}(R_j[X]) \) precedes \( R_j[X] \) in \( h \). If \( \mathcal{I}(R_j[X]) = W_i[X] \), we say that \( T_j \) reads \( X \) from \( T_i \) in \( s \).

Example 2.1. Consider the following schedule \( s = (a, \mathcal{I}) \), where \( a = W_0[X, Y] R_1[X] R_2[X] W_2[X, Y] R_3[X] W_3[Y] W_f[Y] R_f[X, Y] \), and \( \mathcal{I} \) is given by \( \mathcal{I}(R_1[X]) = W_0[X] \), \( \mathcal{I}(R_3[X]) = W_2[X] \) and \( \mathcal{I}(R_f[Y]) = W_3[Y] \). We compactly represent this as follows by changing \( W_i[X] \) to \( W_i[X_i] \) and \( R_j[X] \) with \( \mathcal{I}(R_j[X]) = W_i[X] \) to \( R_j[X_i] \).

\[
s = W_0[x_0, y_0] R_1[x_0] R_2[x_0] W_2[x_2, y_2] R_3[x_2] W_3[y_1] W_f[y_3] R_f[x_2, y_3].
\]

In order to preserve the data consistency in a database, only serializable schedules are allowed. For brevity, we do not go into the formal definition of serializability, but give below an equivalent condition stated in graph theoretic terminology.

The transaction IO graph or TIO graph [7] TIO\((s)\) for a schedule \( s = (h, \mathcal{I}) \) is a labeled directed multigraph with the node set \( \mathcal{S} \cup \mathcal{S}' \) and the arc set \( \mathcal{A} \), where \( \mathcal{S}' \) is given below. If \( T_j \) reads \( X \) from \( T_i \), it has a reads-from arc \((T_i, T_j) \in \mathcal{A}\) labeled by \( X \) (denoted by \((T_i, T_j) : X\)). If \( T_i \) performs \( W_i[Y] \) but no other transaction reads \( Y \) from \( T_i \), then we introduce a dummy node \( T'_i \in \mathcal{S}' \) together with a dummy arc \( 1 \) We shall use abbreviations, such as \( R_j[X, Y] \) for \( R_j[[X, Y]] \) and \( W_i[Y, Z] \) for \( W_i[[Y, Z]] \).
(T_i, T_i'): Y. Dummy nodes will be represented by small circles without labels.

For a schedule s in Example 2.1, $TIO(s)$ is illustrated in Fig. 1.

Let $s = \langle h, I \rangle$ be a schedule. A total order $\preceq$ on the set of nodes of $TIO(s)$ is a disjoint-interval topological sort (DITS, for short [7]), if it satisfies the following two conditions for any nodes $T_g, T_i, T_j, \text{and } T_k$:

(a) [topological sort] if $T_i \preceq T_j$ then there is no path from $T_j$ to $T_i$ in $TIO(s)$, and

(b) [exclusion rule] for any two arcs with the same label $X$, $(T_g, T_i): X$ and $(T_j, T_k): X$, such that $g \neq j$, either $T_i \preceq T_j$ or $T_k \preceq T_g$ holds. (If $T_i \preceq T_j$ in a DITS, we say that $T_i$ is serialized before $T_j$.)

Theorem 2.2 [7]. A schedule $s$ is serializable if and only if $TIO(s)$ has a DITS which orders $T_0$ first and $T_f$ last.

In Fig. 1, we have placed the nodes in a DITS order $\preceq$. Unfortunately, it is in general NP-complete [7] to decide whether a given $TIO(s)$ has a DITS or not. To make this polynomially solvable, some types of constraint arcs, such as write-write (ww), write-read (wr), read-write (rw) arcs and combinations of them, have been introduced [7], e.g., $(T_i, T_j)$ is a ww-constraint arc, if $W_i[X]$ precedes $W_j[X]$ in a log $h$ (such ww-constraint forces that $T_i$ be serialized before $T_j$). Let $c$ be a set of constraint arcs (e.g., ww, wr, rw and their combinations), and let $MC$ (e.g., $MWW, MWR, MRW$) be the corresponding class of logs. A log $h$ belongs to $MC$ if there exists an interpretation $I$ such that the TIO graph of schedule $s = \langle h, I \rangle$ augmented by the constraint arcs of $c$, denoted $TIO_c(s)$, has a DITS that orders $T_0$ first and $T_f$ last. Because of its importance, set $c$ of wr- and rw-constraints will be denoted by wrw, and the corresponding class by $MWRW$.

3. Review of cautious scheduler

Before presenting a new scheduler, we review the general framework of a multiversion cautious scheduler $MCS(MC)$ [6]. Let $\langle P, I \rangle$ be a partial schedule, where $P$ denotes the log that has so far been generated by the scheduler and $I$ is its interpretation, and let $q$ denote the current request, i.e., the step being examined for granting or delaying. Let

$$PEND = \{\text{steps in } T_i \in \mathcal{I} \text{ which are known to the scheduler}\} - \{\text{steps in } Pq\}.$$
We note that \( PEND \) consists of two kinds of steps: those which have already arrived but have not yet been granted, and those which have been predeclared but have not yet arrived. The steps of the first type are separately stored in list \( DEL \). The following is crucial in cautious scheduling.

**Definition 3.1** [MC-completion test]. Given \( \langle P, I \rangle, q, \) and \( PEND \), the MC-completion test determines if it is possible to complete the partial schedule \( \langle P, I \rangle \) by appending to it a sequence \( qQR_f[\mathcal{D}] \), and interpretations \( I' \) and \( I_f \) for \( qQ \) and \( R_f[\mathcal{D}] \), respectively, such that

(i) \( Q \) is a sequence over \( PEND \),

(ii) the order of steps in \( Q \) is consistent with that among the steps of each transaction, and

(iii) \( TIO_s(s) \) of the resulting schedule \( s = \langle PQQR_f[\mathcal{D}], II'If \rangle \) has a DITS, where \( II'I_f \) is the union of interpretations \( I, I' \) and \( I_f \).

In response to each new request \( q \), cautious scheduler \( MCS(MC) \) performs the MC-completion test. If the test fails, then the scheduler delays \( q \) and appends it to \( DEL \). (We assume that \( DEL \) is organized as a FIFO queue.) If the MC-completion test succeeds, on the other hand, step \( q \) is granted, and the steps in \( DEL \) are re-examined one after another in order to see if they can now be granted. The formal description of \( MCS(MC) \) was given in [6] and hence is omitted in this paper.

All operations in \( MCS(MC) \), except for the MC-completion test, are simple and can be carried out efficiently. To describe the algorithms for the MC-completion test, [6] introduced the following useful concept.

**Definition 3.2.** The active TIO graph (or ATIO graph, for short), denoted by \( ATIO(\langle P, I \rangle, q, PEND) \), is a labeled multigraph with a node set \( \mathcal{T} \cup \mathcal{T}' \) and an arc set \( \mathcal{A} \cup \mathcal{A}' \), where \( \mathcal{T}, \mathcal{T}', \mathcal{A} and \mathcal{A}' \) are defined as follows. \( \mathcal{T} \) consists of the transactions whose steps are already in \( Pq \) and/or \( PEND \), and the final transaction \( T_f \). The arc set \( \mathcal{A} \) is defined for \( \langle P, I \rangle \) in the same manner as in the TIO graph of Section 2. In addition, \( \mathcal{A}' \) is defined to be the set of dummy arcs \((T_i, T'_j) : X \) such that \( W_i[X] \) are in \( \{q\} \cup PEND \). Arcs in \( \mathcal{A}' \) are called pending write arcs.

In what follows, we draw the arcs in \( \mathcal{A} \) thick and those in \( \mathcal{A}' \) thin. In addition, if \( q \) is a write step, the corresponding dummy arcs will be drawn thick. The dummy nodes are drawn as small circles. Also, as a reminder, we indicate a not-yet-granted read operation \( R_i[X] \) in \( \{q\} \cup PEND \) as a dangling arc to node \( T_i \) labeled by \( X \). But these “pending read arcs” are formally not part of the ATIO graph.

**Example 3.3.** Let

\[
\langle P, I \rangle = W_0[x_0, y_0] W_1[y_1] W_2[x_2] R_2[y_1],
\]

\[
q = R_3[X], \quad PEND = \{W_1[X]\}.
\]

The corresponding ATIO graph is illustrated in Fig. 2.
The concept of a DITS introduced in Section 2 can be carried over to ATIO graphs. Constraint arcs due to serialization constraints are also added to $ATIO((P, I), q, PEND)$. For two steps $A$ and $B$, we write $A < B$ if either $A$ precedes $B$ in $Pq$, or $A \in Pq$ and $B \in PEND$. For a given set $c$ of constraints, if a step $A$ of $T_i$ and a step $B$ of $T_j$ such that $A < B$ satisfies a constraint in $c$, then a constraint arc $(T_i, T_j)$, called a $c$-arc, is introduced. For a set $c$ of constraints, let $ATIO_c((P, I), q, PEND)$ stand for the active TIO graph augmented by the $c$-arcs.

Let $S \subseteq \emptyset$. A transaction sequence $\tau$ over $\mathcal{S}$ is said to be $S$-readable at $T_j$ with respect to a partial log $P$, if $W_i[X] \in P$ holds for each $X \in S$, where $T_i$ is the last transaction before $T_j$ in $\tau$, which has a write operation on $X$.

**Theorem 3.4** [6]. A partial schedule $(P, I)$, the current request $q$ issued by $T_j$, and a set of pending steps $PEND$ pass the MC-completion test if and only if (1) $ATIO_c((P, I), q, PEND)$ has a DITS which orders $T_0$ first and $T_f$ last, and (2) if $q = R_j[S]$, then the DITS is $S$-readable at $T_f$ with respect to $P$.

Exclusion closure $ATIO^*_c((P, I), q, PEND)$ defined below is very useful for the test of existence of DITS in $ATIO_c((P, I), q, PEND)$. Let $(T_g, T_i) : X$ and $(T_j, T_k) : X$ be two arcs in $ATIO_c((P, I), q, PEND)$, where $g \neq j$, such that there is a path from $T_g$ to $T_k$ (possibly through $T_i$ or $T_j$). Then we introduce an unlabeled exclusion arc $(T_i, T_j)$ unless a path already exists from $T_i$ to $T_j$. The exclusion closure $ATIO^*_c((P, I), q, PEND)$ is then obtained from $ATIO_c((P, I), q, PEND)$ by adding $(T_0, T_i)$ and $(T_j, T_f)$ for all $T_i \neq T_0, T_f$ and all the exclusion arcs. It can be shown that the exclusion closure is unique and does not depend on the order of adding exclusion arcs. It was shown in [6] that, for $MC = MWW$ and $MWRW$, the existence of a DITS can be determined by the acyclicity of the corresponding $ATIO^*_c((P, I), q, PEND)$. Note that acyclicity can be determined in polynomial time.

The $S$-readability requirement for $MC = MWW$ can easily be represented in $ATIO_{rw}(P, I), q, PEND)$ by the $rw$-constraints. For $MC = MWRW$, it is already met by the $rw$-arcs. Thus the MC-completion test for these classes can be done in polynomial time.
4. Description of new schedulers

We shall first explain in this section the basic idea behind our new scheduler. In view of Theorem 3.4, in order to execute the MC-completion test in polynomial time, it is necessary to check the following conditions in polynomial time.

1. If \( q = W_j[s] \), there exists a DITS in \( ATIO_c(s, q, PEND) \), where \( s = \langle P, I \rangle \).
2. If \( q = R_j[S] \), there exists a DITS in \( ATIO_c(s, q, PEND) \), which is \( S \)-readable at \( T_j \).

In [7], the existence of a DITS is reduced to the acyclicity of its exclusion closure through the following condition.

**Condition \( P \).** For any data item \( X \) and any pair of arcs \( (T_g, T_i) : X \) and \( (T_j, T_k) : X \) with \( g \neq j \) such that at most one of \( (T_g, T_i) : X \) and \( (T_j, T_k) : X \) is a dummy arc, there is a path in \( ATIO_c(s, q, PEND) \) either from \( T_g \) to \( T_i \) or from \( T_j \) to \( T_k \).

**Theorem 4.1** [7]. Suppose that \( ATIO_c(s, q, PEND) \) satisfies Condition \( P \) for a set \( c \) of constraints. Then \( ATIO_c(s, q, PEND) \) has a DITS if and only if \( ATIO_c(s, q, PEND) \) is acyclic.

An example of a constraint set \( c \) that guarantees Condition \( P \) is \( c = wrw \). However, all the \( wrw \)-constraints in \( ATIO_{wrw} \), may not be needed for this purpose as seen in the following example. Our new scheduler is based on the idea to add as few \( wrw \)-constraints as possible, so that the degree of concurrency is improved.

**Example 4.2.** Consider the following situation.

\[
\langle P, I \rangle = W_0[x_0, y_0] W_1[y_1] W_2[x_2] R_2[y_1],
q = R_3[X], \quad PEND = \{ W_3[X] \}.
\]

The corresponding \( ATIO_{wkw} \) graph is shown in Fig. 2. Due to \( wr \)-arc \( (T_2, T_3) \) and \( rw \)-arc \( (T_3, T_1) \), there is a cycle in the graph, and \( q = R_3[X] \) does not pass the \( MWRW \)-completion test. However, one of \( wr \)-arc \( (T_2, T_3) \) and \( rw \)-arc \( (T_3, T_1) \) is redundant, i.e., Condition \( P \) is satisfied even without one of them. With either of them being deleted, therefore, \( q \) passes the MC-completion test and is granted by the scheduler.

One possible approach to avoiding such redundant constraints is to impose all \( rw \)-constraints, but impose other serialization constraints only when they are needed to ensure Condition \( P \). Since \( S \)-readability in condition (2) is automatically satisfied by the \( rw \)-constraints, Theorem 4.1 implies that the resulting MC-completion test can be done in polynomial time. The resulting scheduler is denoted \( MCS(MRW^+) \). Here, constraint set \( rw^+ \) and its corresponding class \( MRW^+ \) stand for that the set of \( rw \)-constraints is augmented with some other constraints introduced by this scheduler.
Now we shall describe how to introduce additional constraint arcs in our scheduler. \(\text{MCS(MRW}^+\text{)}\) starts with an ATIO graph consisting only of the initial transaction \(T_0\) and the final transaction \(T_f\). Every time a request \(q\) of transaction \(T_j\) arrives at the scheduler, \(G = \text{ATIO}_r^w(\langle P, I \rangle, q, \text{PEND})\) is modified as follows. If \(q\) is the first step of \(T_j\), a new node \(T_j\) is added to \(G\). In general, some constraint arcs are introduced by rules (A) and (B) below. The set of constraints defined by rule (A) is undone if the corresponding request is not granted. If the current request is a read step and if it is granted, then version assignments whereby create-reads-from arcs and the set of constraints defined by rule (B); these changes persist thereafter.

(A) \(rw\)-arcs:

(i) If \(q\) is the first step of a transaction \(T_j\), introduce an \(rw\)-arc \((T_h, T_j)\) for each \(X \in WS_j\) and for each \(R_h[X] \in P\), where \(WS_j\) is the write set of \(T_j\).

(ii) If \(q = R_j[X]\), introduce an \(rw\)-arc \((T_h, T_j)\) for each \(W_h[X] \in \text{PEND}\) with \(X \in S\).

After introducing constraint arcs according to the above two rules, we examine the acyclicity of its exclusion closure \(G^* = \text{ATIO}_r^w(\langle P, I \rangle, q, \text{PEND})\). If it is acyclic, the current request \(q\) is granted. It should be remarked here that, the arcs of rule (A)(i) alone do not create a cycle in \(G\). Therefore \(\text{MCS(MRW}\}^+)\) immediately grants all write requests \(q = W_j[X]\) since no arcs of rule (A)(ii) are introduced in this case.

If the current request \(q = R_j[X]\) is granted, an appropriate version \(W_j[X]\) is assigned to \(R_j[X]\) for each \(X \in S\). This is done in the same fashion as in [6]; i.e., after obtaining a topological ordering of nodes in \(G^*\) (which is a DITS in \(G\) as we shall see later), assign \(W_j[X]\) to \(R_j[X]\), for each \(X \in S\), if \(T_i\) is the last transaction before \(T_j\) in this DITS, such that \(X \in WS_i\) \((W_i[X] \in P\) holds by \(rw\)-constraint arcs to write operations in \(\text{PEND}\)). Then, after introducing reads-from arc \((T_i, T_j) : X\), \(\text{MCS(MRW}\}^+)\) adds the following arcs in order to satisfy Condition \(P\).

(B) \(wr\)-arcs:

(i) For each \(X \in S\), introduce \(wr\)-arc \((T_k, T_j)\) for each \(W_k[X] \in P\) such that \(T_k \prec T_j\) in the above DITS order.

(ii) For each \(X \in S\), introduce a constraint arc \((T_j, T_i)\) for each \(W_j[X] \in P\) such that \(T_i \prec T_j\) in the above DITS order. (The arcs introduced by (B)(i)) are reverse \(wr\)-arcs because the direction is opposite.)

Example 4.3. Consider the partial schedule studied in Example 3.3, under \(\text{MCS(MRW}\}^+)\). For \(q = R_1[X]\), \(rw\)-arc \((T_3, T_1)\) is introduced according to rule (A)(ii). The \(\text{ATIO}_r^w\) graph at this moment is illustrated in Fig. 3. Since this is acyclic and \(T_0 T_3 T_1 T_2 T_j\) is a DITS order, \(R_3[X]\) is granted and \(W_6[X]\) is assigned to \(R_3[X]\) according to the DITS order (i.e., reads-from arc \((T_0, T_3) : X\) is added). After this, by rule (B)(ii), a constraint arc \((T_3, T_2)\) is introduced. Finally, when \(W_1[X]\) arrives, it is immediately granted.
Before closing this section, we remark that, if $q = R_j[S]$ is granted, the constraint arcs of types (A) and (B) added to $G^*$ never create a cycle. Therefore, the resulting $G^*$ is used for the next request, and cautious scheduling keeps on going.

5. Correctness of the new scheduler

We shall prove in this section that the MC-completion test for the new scheduler can be correctly done by checking the acyclicity of $ATIO_{r^+}((P, I), q, PEND)$.

Since the constraint arcs are dynamically introduced while executing the scheduling algorithm, the constraint set $c = rw^+$ of our scheduler is not explicitly defined in advance. This is different from the previous classes such as $MWW$ and $MWRW$. In order to make use of Theorem 3.4, we prove the following two lemmas.

**Lemma 5.1.** For $c = rw^+$, $G = ATIO_c((P, I), q, PEND)$ has a DITS which orders $T_0$ first and $T_j$ last if and only if $G^* = ATIO_{r^+}((P, I), q, PEND)$ is acyclic.

**Proof.** First assume that $G$ has a DITS. Then $G$ is acyclic by definition. $G^*$ is also acyclic since it is clear from the definition of exclusion arcs that all exclusion arcs are consistent with the obtained DITS order.

Conversely, assume that $G^*$ is acyclic. Find a topological ordering $\tau$ of $G^*$. We can assume without loss of generality that all the dummy nodes are placed immediately after their "parent" nodes. We shall show that $\tau$ is a DITS in $G$ by demonstrating that Condition $P$ as stated in Theorem 4.1 holds. Let us consider two arcs $(T_g, T_i) : X$ and $(T_j, T_k) : X$ with $g \neq j$ such that at most one of $(T_g, T_i) : X$ and $(T_j, T_k) : X$ is a dummy arc.

**Case 1:** None of $(T_g, T_i) : X$ and $(T_j, T_k) : X$ is dummy. This implies that all of $W_g[X], R_i[X], W_j[X]$ and $R_k[X]$ belongs to $Pq$. We assume without loss of generality that $R_i[X] < R_k[X]$, where relation $<$ was defined before Theorem 3.4.

**Subcase 1(A):** $(R_i[X] < W_j[X])$. Consider the instant when $R_i[X]$ is granted. If the first step of $T_j$ has already been granted by that time (i.e., $W_j[X] \in PEND$), $rw$-arc $(T_g, T_i)$ is introduced according to rule (A)(ii). If not, $rw$-arc $(T_i, T_j)$ is introduced according to rule (A)(i), when the first step of $T_j$ is granted. In any case, due to reads-from arc $(T_g, T_i) : X$ and $rw$-arc $(T_i, T_j)$, there is a path from $T_g$ to $T_j$.

**Subcase 1(B):** $W_j[X] < R_i[X]$. Consider the time instant when $W_j[X]$ was assigned to $R_i[X]$. Let $\tau'$ be the DITS used for version assignment at this time. If $T_j$. 

---

**Fig. 3. Illustration of $ATIO_{r^+}((P, I), q, PEND)$ for Example 4.3.**
precedes \(T_i\) in \(\tau'\), a \(wr\)-arc (\(T_j, T_i\)) is introduced according to rule (B)(i). Thus, an exclusion arc (\(T_k, T_g\)) is introduced when \(W_j[X]\) is assigned to \(R_k[X]\), due to (\(T_j, T_k\) : \(X\) and (\(T_i, T_j\)). If \(T_i\) precedes \(T_j\) in \(\tau'\), a reverse \(wr\)-arc (\(T_i, T_j\)) is introduced according to rule (B)(ii). In any case, \(T_g\) and \(T_j\) satisfy Condition \(P\).

Case 2: Either (\(T_g, T_i\)) : \(X\) or (\(T_j, T_k\)) : \(X\) is a dummy arc. Assume without loss of generality that (\(T_j, T_k\)) : \(X\) is a dummy arc. Thus, (\(T_g, T_i\)) : \(X\) is a nondummy arc.

Subcase 2(A): \(W_j[X] < W_g[X]\). Consider the time instant when \(W_g[X]\) is assigned to \(R_j[X]\). Let \(\tau'\) be the DITS used for version assignment at this time. The rest of the proof is similar to Subcase 1(B).

Subcase 2(B): \(W_g[X] < W_j[X]\). If \(R_i[X] < W_j[X]\), we can show in a manner similar to Subcase 1(A) that there is a path from \(T_g\) to \(T_j\). On the other hand, if \(W_j[X] < R_j[X]\), the case is analogous to Subcase 1(B). \(\square\)

**Lemma 5.2.** If \(q = R_j[S]\) and if \(G = ATIO_{rw}\langle\langle P, I \rangle, q, PEND\rangle\) has a DITS, \(G\) is \(S\)-readable at \(T_j\).

**Proof.** Since all nodes \(T_i\) with pending write operation \(W_i[X]\) for some \(X \in S\) follow \(T_j\) in the DITS of \(G\) due to \(rw\)-constraints (\(T_j, T_i\)) by rule (A)(ii), \(G\) is clearly \(S\)-readable at \(T_j\). \(\square\)

By Lemmas 5.1 and 5.2, we can restate Theorem 3.4 in the following manner.

**Theorem 5.3.** A partial schedule \(\langle P, I \rangle\), a current request \(q\) and a set \(PEND\) of pending steps pass the MRW\(^+\)-completion test if and only if \(ATIO_{rw}\langle\langle P, I \rangle, q, PEND\rangle\) is acyclic.

This theorem also implies that the MRW\(^+\)-completion test can be executed in polynomial time.

6. Cancellation and augmentation anomalies

In the cautious scheduling, it is assumed that each transaction upon arrival pre-declares its read set and write set. In the real situation, however, transactions may cancel some of their predeclared operations. It has been shown that some of the single-version cautious schedulers may block when some predeclared operations are cancelled. Namely, they exhibit cancellation anomaly, while most multiversion cautious schedulers, including our new scheduler, do not have such an undesirable feature.

**Theorem 6.1.** MCS(MRW\(^+\)) does not exhibit cancellation anomaly.

**Proof.** It is sufficient to show that, if \(G = ATIO_{rw}\langle\langle P, I \rangle, q, PEND\rangle\) has a DITS,
the deletion of the arcs representing the cancelled pending read or write operation (together with constraint arcs associated with it) does not destroy the DITS property. However, this is obvious in our ATIO graph, since no new constraint arcs are introduced by deleting the arcs of a pending operation.

Next, we consider the opposite situation, in which transactions want to expand their predeclared read/write set. As was shown in [5], for any \( \text{MCS(MC)} \) of interest, the addition of a new write step may cause scheduler blocking. Therefore we consider only the addition of new read steps, and say that a scheduler exhibits \textit{read augmentation anomaly}, if the addition of some unpredeclared read steps can cause scheduler blocking. It has been shown [5, 6] that cautious schedulers studied in [5, 6, 8, 10], except for \( \text{MCS(MWW)} \) in [6], exhibit augmentation anomaly.

In the following, we assume that a transaction is allowed to declare a new step only if it still has at least one pending step. This takes the form of either submission of an unpredeclared read step or expansion of the pending read set.

**Theorem 6.2.** \( \text{MCS(MRW}^+ \text{)} \) does not exhibit \textit{read augmentation anomaly}.

**Proof.** Obvious because the ATIO graph does not introduce any constraint arc with respect to a pending read operation.

Theorems 6.1 and 6.2 imply that each arriving transaction need not predeclare its read set to \( \text{MCS(MRW}^+ \text{)} \).

7. Fixed point set of the new scheduler

A log \( h \) belongs to the fixed point set \( MC^* \) of a certain multiversion cautious scheduler \( \text{MCS(MC)} \) if all steps in \( h \) are granted without delay under some interpretation. The degree of concurrency attained by \( \text{MCS(MC)} \) is usually measured by its fixed point set. We interpret that \( MC_1^* \supseteq MC_2^* \) is a mathematical statement of the fact that scheduler \( \text{MCS(MC}_2 \) has a higher degree of concurrency than scheduler \( \text{MCS(MC}_1 \).

In the next theorem, we compare \( \text{MCS(MRW}^+ \text{)} \) with existing \( \text{MCS(MWW)} \) and \( \text{MCS(MWRW)} \) [6] according to this definition.

**Theorem 7.1.** (i) \( (\text{MRW}^+)^* \supseteq \text{MWRW}^* \),
(ii) \( (\text{MRW}^+)^* \supset \text{MWW}^* \),
where \( \supset \) denotes proper inclusion.

**Proof.** (i) we first prove \( (\text{MRW}^+)^* \supset \text{MWRW}^* \) by showing that for any schedule \( (h, I) \) output by \( \text{MCS(MWRW)} \), \( \text{MCS(MRW}^+ \text{)} \) can grant all steps in \( h = s_1, s_2, \ldots, s_n \) without delay under the same interpretation \( I \). Let \( G_j \) (respectively \( H_j \)) be the
ATIO graph just before a request $q = s_i$ arrives at \textit{MCS(MWRW)} (respectively \textit{MCS(MRW+)}), $G_i^*$ (respectively $H_i^*$) be its exclusion closure and $G_i'$ (respectively $H_i'$) be the ATIO graph at the time of testing $q = s_i$ in \textit{MCS(MWRW)} (respectively \textit{MCS(MRW+)}). It suffices to show that a DITS $\tau$ for $G_i'$ is also a DITS for $H_i'$. This is proved by induction on $i$, by showing the following induction hypotheses.

(a) $G_i$ and $H_i$ have the same set of reads-from arcs, and
(b) the arc set of $H_i^*$ is a subset of that of $G_i^*$.

Initially, for $q = s_1$, where $s_1$ is the first step of transaction $T_1$, $G_1$ and $H_1$ consist only of two nodes $T_0$ and $T_f$. $G_1'$ and $H_1'$ are then obtained by adding node $T_1$ and constraint arcs $(T_0, T_1), (T_1, T_f)$. Therefore both schedulers grant $q$ and, if $q$ is a read step $R_1[S]$, assign $W_0[X]$ to $R_1[X]$ for each $X \in S$. Thus (a) and (b) hold.

Assuming (a) and (b) up to $i = k$, consider that $q = s_k$ has arrived. \textit{MCS(MWRW)} adds wrw-constraint arcs to $G_k$. The resulting graph $G_k' (= G_{k+1})$ has a DITS by definition. In this case, \textit{MCS(MRW+)} adds only rw-constraints of rule (A) to $H_k$. Therefore, by induction hypothesis, the exclusion closure of the resulting graph $H_k'$ is acyclic and $H_k'$ can have the same DITS as that of $G_k$ among others. Assume that \textit{MCS(MRW+)} chooses such DITS order. If $q = s_k$ is a read step $R_i[S]$, \textit{MCS(MRW+)} then adds the same reads-from arcs to $H_k'$ as those chosen by \textit{MCS(MWRW)} (since both employ the same DITS order), and also adds wr-arcs by rule (B)(i) and reverse wr-arcs by (B)(ii). The resulting graph is $H_{k+1}$. The wr-arcs of (B)(i) are already in $G_k'$. Since the DITS order in $G_k'$ implies that all $T_j$ with $W_j[X] \in P$ precede $T_i$ by wr-constraint arcs, \textit{MCS(MRW+)} does not introduce any constraint arc of rule (B)(ii). Thus the arc set of $H_{k+1}$ is a subset of that of $G_{k+1}$. This proves (a) and (b) for $i = k + 1$.

In order to prove proper inclusion, consider the following log.


As easily seen, before $R_3[X]$, \textit{MCS(MWRW)} grants each step immediately, and the interpretation up to this point is unique. The situation at $q = R_3[X]$ was studied in Example 4.2. Since the ATIO\textsubscript{wrw} graph at this moment has a cycle as shown in Fig. 2, $R_3[X]$ is delayed. Therefore, $h_1$ is not in \textit{MWRW*}. On the other hand, \textit{MCS(MRW+)} does not introduce wr-arc $(T_2, T_3)$, and ATIO\textsubscript{wwr} is acyclic. Hence $R_3[X]$ is granted by assigning $W_0[X]$ to $R_3[X]$. It is easy to see that $W_1[X]$ is also granted without delay. Thus $h_1 \in (MRW^+)^*$. (ii) Suppose $(h, l) \in MWW^*$. Similarly to (i), we show the following induction hypotheses.

(a) $G_i$ and $H_i$ have the same set of reads-from arcs, and
(b) the arc set of $H_i^*$ is a subset of that of $G_i^*$, where $G_i$ and $G_i^*$ are defined for \textit{MCS(MWW)} in a manner similar to those for \textit{MCS(MWRW)} in (i).

Induction basis for $q = s_1$ can be proved similarly to (i). Assuming (a) and (b) up to $i = k$, consider that $q = s_k$ has arrived. \textit{MCS(MWW)} adds ww-constraints to $G_k$ as well as rw-constraints of (A) (defined for \textit{MCS(MRW+)}). The resulting graph
A multiversion cautious scheduler

$G'_k (=G_{k+1})$ has a DITS by definition. In this case, $MCS(MRW')$ adds only $rw$-constraints of rule (A) to $H_k$. Therefore, by induction hypothesis, the exclusion closure of the resulting graph $H'_k$ is acyclic and has the same DITS as that of $G'_k$ among others. Assume that $MCS(MRW')$ chooses such DITS order. If $q=s_k$ is a read step $R_j[S]$, $MCS(MRW')$ then adds the same reads-from arcs to $H'_k$ as those chosen by $MCS(MWW)$. This proves (a) for $i=k+1$. $MCS(MRW')$ then adds $wr$-arcs by rule (B)(i) and reverse $wr$-arcs by rule (B)(ii), to obtain $H_{k+1}$ from $H'_k$. Let $W_i[X]$ be assigned to $R_j[X]$ for $X\in S$ by both schedulers. For $W_i[X] \in P$ preceding $W'_i[X]$ in the above DITS order, rule (B)(i) introduces wr-arc $(T_k, T_j)$ to $H'_k$. In $G_{k+1}$, since there is a $ww$-arc $(T_k, T_j)$ by $W_i[X] < W'_i[X]$, this $(T_k, T_j)$ and reads-from arc $(T_i, T_j)$ induce an exclusion arc $(T_k, T_j)$ in $G_{k+1}$*. Thus wr-arcs introduced by (B)(i) in $H'_k$ are not more restrictive than $ww$-arcs. Now for $W'_i[X]$ following $R_j[X]$ in the DITS order, $MCS(MRW')$ introduces a reverse wr-arc $(T_i, T_j)$ to $H'_k$. In this case, reads-from arc $(T_i, T_j)$ and $ww$-arc $(T_i, T_j)$ give rise to an exclusion arc $(T_i, T_j)$ in $G_{k+1}^*$. Thus constraint arcs introduced by (B)(ii) in $H'_k$ are not more restrictive than exclusion arcs in $G_{k+1}^*$. This proves (b) for $i=k+1$.

In order to prove proper inclusion, it can be shown that


can be output by $MCS(MRW')$ without delay, but cannot be output by $MCS(MWW)$ for any interpretation $I$.

8. Simulation experiments

Carey and Muhhmana [3] carried out simulation studies on three types of multiversion algorithms, namely those based on timestamp ordering [1], two-phase locking [2], and optimistic concurrency control [14]. They concluded that "the multiversion algorithms provide significant improvements over the single-version counterparts despite the additional disk accesses involved in accessing old versions of data". In addition, Nishio et al. [15] compared the performance of cautious schedulers with that of noncautious single-version schedulers (such as those based on two-phase locking and the serialization graph [2]), and concluded that caution schedulers outperform their noncautious counterparts. Furthermore, Sy [19] (see also [6]) compared the performance of the single-version cautious scheduler $CS(WW)$ [5] with its multiversion counterpart $MCS(MWW)$ [6], and concluded that the improvement due to the use of multiple versions was considerable. These results indicate that multiversion cautious schedulers are superior to other concurrency control algorithms from the viewpoint of attaining high concurrency.

In this section we will cite some results from our recent simulation results [12], which compare the relative performance of multiversion cautious schedulers $MCS(MWW)$, $MCS(MWRW)$ and $MCS(MRW')$. The major result was that the new scheduler $MCS(MRW')$ exhibits a significantly higher performance over other multiversion cautious schedulers.
8.1. Parameters

The basic scheme of our simulation experiments is the same as that of [19]. The mean inter-arrival time of transactions, \( T_{\text{Int\_Arr}} \), was varied in the range of 6 to 16, in order to see how conflicts among transactions affect performance. Table 1 shows the values of the other parameters used. \( \text{NumT} \) is the number of transactions that are generated in one simulation run. The number of data items is given by \( D\text{size} \). The size of the write set of a transaction, \( W\text{size} \), is a random variable having a uniform distribution over the range \([1, MX\text{WSIZE}] \). The size of the read set of a transaction is assumed to be, on average, 20\% larger than that of the write set. \( OV \) is the average percentage of the write set of a transaction that overlaps with its read set. More precisely, \((OV/100) \times (1 + MX\text{WSIZE})/2\) is the mean number of data items that are in both read set and write set of a transaction. \( D\text{size} = 45 \) should be contrasted to the average size of a write set size 4.5 and the maximum number of data items, 15, read or written by a transaction (which is possible when \( W\text{size} = MX\text{WSIZE} = 8 \), read set size = 7 and no overlap). \( MX\text{DPERSTEP} \) is the maximum number of data items that one step may access, which is uniformly distributed over \([1, MX\text{DPERSTEP}] \). The inter-arrival times of transactions and of the steps of a transaction are assumed to obey exponential distributions with means, \( T_{\text{Int\_Arr}} \) and \( S_{\text{Int\_Arr}} \), respectively. The ratio of these two will determine to what degree the steps of different transactions will conflict. Since \( S_{\text{Int\_Arr}} \) is fixed at 5, the conflicts among transactions will increase as \( T_{\text{Int\_Arr}} \) decreases.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Set values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{NumT} )</td>
<td>600</td>
<td>number of transactions</td>
</tr>
<tr>
<td>( MX\text{WSIZE} )</td>
<td>8</td>
<td>maximum size of a transaction's write set</td>
</tr>
<tr>
<td>( S_{\text{Int_Arr}} )</td>
<td>5</td>
<td>mean inter-arrival time of steps</td>
</tr>
<tr>
<td>( D\text{size} )</td>
<td>45</td>
<td>number of data items in database</td>
</tr>
<tr>
<td>( OV )</td>
<td>82%</td>
<td>percentage of write set that overlaps with read set</td>
</tr>
</tbody>
</table>

8.2. Results

The performance of the schedulers under comparison has been measured in terms of average response time, which is the delay that a step (request) encounters from the time it is submitted to the scheduler until it is granted. For simplicity, once a request is granted, it is assumed to be executed immediately. The average response time is normalized by dividing it by \( S_{\text{Int\_Arr}} \).

Figure 4 shows the average response time of cautious schedulers \( MCS(MWW) \), \( MCS(M\text{WRW}) \) and \( MCS(MRW^+) \), for values of various mean inter-arrival times of transactions, \( T_{\text{Int\_Arr}} \). As expected, the response time increases when \( T_{\text{Int\_Arr}} \) becomes small (relative to \( S_{\text{Int\_Arr}} \), because conflicts among transactions tend to
A multiversion cautious scheduler 393

![Diagram](image)

Fig. 4. Average response time versus mean transaction inter-arrival time.

increase. When $T_{\text{Int}\_\text{Arr}}$ is large, on the other hand, transactions are executed almost serially since the next transaction does not usually arrive until most, if not all, of the previous transactions have been completed. Figure 4 clearly indicates that the new scheduler $MCS(MRW^+)$ reduces the average response time by more than 20% for any choice of $T_{\text{Int}\_\text{Arr}}$ between 6 and 16, compared with other multiversion schedulers. This reduction is remarkable since, as observed in [19], the improvement of multiversion cautious schedulers, $MCS(MWW)$ and $MCS(MWRW)$, over the single-version cautious scheduler $CS(WW)$ was much less than 20%.

Some other simulation results are also presented in [12]. Since the size of the AT10 graph increases as new transactions arrive, the completion test becomes more and more time consuming. To prevent this, we incorporated a mechanism to erase some completed transactions (i.e., those which are not necessary for the future completion tests) from the AT10 graph. The detailed account of this mechanism is described in [11]. In our model, it is assumed that an unlimited number of versions are available. It turned out, however, that around 96% of read operations were assigned the most recent versions by $MCS(MRW^+)$. This result should be contrasted with the fact that around 98% of read operations were assigned the most
recent versions by \textit{MCS(MWW)} and \textit{MCS(MWRW)}. The difference between 98\% and 96\% leads to the above improvement in the average response time. Finally, we note that the CPU time and the space required for the completion test in the new scheduler are almost the same as those required for other multiversion schedulers, \textit{MCS(MWW)} and \textit{MCS(MWRW)}, because the number of active transactions observed in our new scheduler is almost the same as those in other schedulers.

9. Conclusion and discussion

A new multiversion cautious scheduler, which dynamically introduces serialization constraints, is proposed in this paper. We have shown theoretically and by simulation that (i) our scheduler can be executed in polynomial time, (ii) its degree of concurrency is higher than any of the existing cautious schedulers such as \textit{MCS(MWW)} and \textit{MCS(MWRW)}, if concurrency is measured in terms of their fixed point sets or their response time, and (iii) they do not exhibit cancellation or augmentation anomaly.

The basic idea of our new scheduler can be modified to obtain other types of schedulers. One such scheduler is to add partially rw-constraints as well as some wr (or ww-) constraints until Condition $P$ holds, while preserving the acyclicity of the ATIO graph. Since rw-constraints are only partially imposed in this case, we need to test the S-readability for the current request $q$ if $q = R_j[S]$. But we can show that such test can be done in polynomial time. The details of this scheme are presented in [9]. It is shown that this scheduler also has desirable features like those observed with \textit{MCS(MRW') introduced in this paper. This idea can also be applicable to the single-version cautious scheduler [13].

Acknowledgement

The authors wish to thank referees for helpful suggestions and comments.

References

A multiversion cautious scheduler