CONCURRENT EXECUTION OF SYNTACTICALLY IDENTICAL TRANSACTIONS

by

Ouri Wolfsohn

Technical Report #387
November 1985
CONCURRENT EXECUTION OF SYNTACTICALLY IDENTICAL TRANSACTIONS

Ouri Wolfson

The Technion - Israel Institute of Technology.
Computer Science Dept., Haifa 32000, Israel.

ABSTRACT

Syntactically identical transactions, or copies of a transaction, execute concurrently when a task has to be performed at the same time by multiple database users (e.g., airline seat reservation). We assume a distributed database which controls concurrency by locking, and present the following two results. In order to determine whether an arbitrarily large set of copies is deadlock free or not at least \( m \) copies must be tested for deadlock freedom, where \( m \) is the number of referenced database entities. On the other hand, to determine whether such a set ensures serializability it suffices to test only two copies.

Keywords: Concurrency control, Deadlock, Locking, Serializability.
1. INTRODUCTION

When transactions accessing a database execute concurrently, their actions interleave. Concurrency control has to ensure that the interleaving (or schedule) is correct. The accepted definition of correctness is serializability ([1]), and the simplest, and thus most popular, mechanism to ensure it is locking of database entities. However, locking has two drawbacks. First, it does not ensure serializable schedules for any set of transactions which lock the entities they access. Second, it can produce a deadlock. Considerable amount of work in concurrency control dealt with the issues of deadlock freedom and serializability assurance of locked transaction systems ([2,4,6,7,8,9]). The desirability of serializable schedules is well established by now ([1,5]). Deadlock freedom is desirable because if it can be established then the deadlock-detection-and-elimination mechanism of the database-management-system can be turned off, reducing overhead.

In this paper we address these issues as they pertain to syntactically identical transactions, or copies. In a centralized database, multiple copies of a transaction cannot deadlock (they are modeled by straight line programs). It has been shown in [7] that in a distributed database they can. The question we address here is: what is the minimal number of copies of a transaction that must be tested for deadlock freedom to determine whether any finite number of copies of the transaction are deadlock-free. We prove that this number is equal to the number of
entities referenced by the transaction. Then we consider the serializability assurance issue. A set of transactions ensures serializability if every possible schedule is serializable. In [8] it has been demonstrated that multiple copies of a transaction can produce a nonserializable schedule even in a centralized database. We show that if a set of two copies ensures serializability then an arbitrarily large set does so. This is true for every transaction, regardless of the number of entities it references.

2. THE MODEL

A distributed database $D$ is a finite set of entities partitioned into pairwise disjoint subsets called sites. A transaction $T$ over $D$ is a partial order (directed acyclic graph) of Lock-Unlock steps referencing the entities of the database [2]. The lock of an entity $x$ is denoted by $Lx$ and the unlock by $Ux$. The partial order has to satisfy the following conditions: 1) $Lx$ is in $T$ if and only if $Ux$ is in $T$, and $Lx$ precedes $Ux$. 2) All the steps referencing entities which reside at the same site in the database are totally ordered.

To be completely precise read and write steps have to be included in the transaction definition. However, it is clear that they can be disregarded when analyzing deadlock. The same is true for serializability ([2,6]). The reason for condition 2) is that steps referencing entities residing at the same site are sequentially executed by one computer. Thus, if the database has only one site (is centralized) the transaction is reduced to the popular representation of a centralized transaction as a sequence of steps.

An extension $e$ of a partial order $P = (V,A)$ is a linear ordering of the members of $V$ such that if $a$ precedes $b$ in $P$ it does so in $e$. Given a set of transactions $\tau = \{T_1 = (V_1,A_1), ..., T_n = (V_n,A_n)\}$ we assume that the steps in $V_i$ are superscripted by $i$ (to distinguish between two $Lx$ steps from different transactions). $\tau$ can be
regarded as a partial order \((X, A)\) where \(V = \bigcup_{i=1}^{n} X_i\) and \(A = \bigcup_{i=1}^{n} A_i\). \(S\) is a schedule of \(\tau\) if it is an extension of the partial order in which, between every two Lock steps of an entity there is an Unlock step of the entity. Intuitively, a schedule is a possible sequence-of-execution of the transaction steps (a locked entity cannot be locked by another transaction before it is unlocked). Subgraph \(P'\) of partial order \(P\) is a prefix of \(P\) if in \(P\) there are no arcs from a node which is not in \(P'\) to a node in \(P'\). This definition subsumes the regular prefix definition if \(P\) is a total order. \(S\) is a partial schedule of \(\tau\) if it is a schedule of some prefix of \(\tau\); namely an extension of some prefix of \(\tau\) in which the locks are respected. Intuitively, a partial schedule is an incomplete execution of the transactions. \(\tau\) is deadlock-free if any partial schedule of \(\tau\) can be extended to a complete schedule. \(\tau\) is a set of copies of a transaction \(T\) if each \(T_i\) in \(\tau\) can be obtained by superscripting the steps of \(T\) by \(i\). It means that all \(T_i\)'s represent the same transaction.

3. DEADLOCK

3.1 Some Remarks about Distributed Deadlock

First we formalize the deadlock situation among arbitrary transactions (not necessarily copies) in graph-theoretic terms. This formalization is necessary since deadlock of distributed transactions, modeled by partial orders, is more complicated than the well-studied case of deadlock in a centralized environment, where transactions are modeled by total orders. We use the results of this subsection in proving Lemma 3.

Assume that after executing a partial schedule \(S\) of a set of transactions \(\tau = \{T_1, ..., T_q\}\) a deadlock occurs, namely no additional step can be executed. Then for each transaction \(T_i\) there exist among the unexecuted steps locks \(L_{y_1}, ..., L_{y_k}\) on which the transaction waits indefinitely. They have the following properties: (1)
Each \( y_i \) is locked but not unlocked in \( S \) by another transaction (2) Each step which precedes some \( L y_i \) in \( T_i \), is executed in \( S \). (3) Any other step \( s \) of \( T_i \) which is not executed in \( S \) succeeds some \( L y_i \) (otherwise \( S \) can be continued by executing \( s \)). We define \( S \) to be a deadlock partial schedule of \( \tau \) and \( L y_1, \ldots, L y_k \) the critical steps defined by \( S \) in \( T_i \). Two vertices of a directed acyclic graph (dag) are incomparable if there is no path from one to the other.

**Proposition 1:** Let \( \tau \) be a set of transactions and \( T \in \tau \). If \( L x \) and \( L y \) are two critical steps defined in \( T \) by a deadlock partial schedule of \( \tau \) then \( L x \) and \( L y \) are incomparable in \( T \).

**Proof:** If \( L x \) precedes \( L y \) then by property (2) in critical steps definition it is executed in the deadlock partial schedule. Thus it cannot be a critical step. []

In a deadlock situation a transaction may wait indefinitely on one or more entities. Proposition 1 indicates that all these entities must reside at different sites (since the lock steps referencing entities residing at the same site are totally ordered). Therefore, each transaction deadlocked in a centralized database waits indefinitely on exactly one entity. This is the heart of the difference between distributed and centralized deadlock.

Next, assume that \( \tau \) is a set of transactions. The **deadlock graph** of \( \tau \), denoted \( DG(\tau) \), is a directed graph (digraph) having all entities referenced in \( \tau \) as nodes and an arc \( x \rightarrow y \) if and only if there is a transaction in \( \tau \) where \( L x \) does not precede \( L y \) and \( L x \) precedes \( L y \).

**Lemma 1:** Let \( \tau \) be a set of transactions and \( S \) a deadlock partial schedule of it. Then \( Z = \{ x | x \text{ is referenced by a critical step defined by } S \text{ in some transaction} \} \) induces a cyclic digraph in \( DG(\tau) \).

**Proof:** Denote the transactions of \( \tau \) by \( T_1, \ldots, T_\tau \). Let \( y \) be some entity in \( Z \) and assume that \( y \) is locked by a critical step of some transaction \( T_j \). Then there is
another transaction, \( T_h \), such that \( y \) is locked by \( T_h \) in \( S \) and unlocked by \( T_h \) after one of its critical steps. (Otherwise, \( Ly \) is not a critical step of \( T_h \).) Assume that the critical step of \( T_h \) which precedes \( Lj \) is \( Lx \). Since \( T_h \) locks \( y \) in \( S \) but not \( z \), it means that \( Lz \) does not precede \( Ly \) in \( T_h \). Then, by definition, there is an arc \( x \rightarrow y \) in \( DG(\tau) \). Note also that \( z \) is in \( Z \) since \( Lz \) is a critical step. Therefore, for an arbitrary entity \( y \) in \( Z \) we found an entity \( z \) in \( Z \) and an arc from \( x \) to \( y \) in \( DG(\tau) \). If \( Z \) induces a directed acyclic graph in \( DG(\tau) \), then for a root of this dag there is no incoming arc from an entity of \( Z \). []

**Theorem 1**: A set of transactions is deadlock-free if its deadlock graph is acyclic.

**Proof**: Follows from Lemma 1. []

The condition of Theorem 1 can be tested in polynomial time and is sufficient for deadlock freedom. However it is not necessary and to realize it assume that \( \tau \) consists of two copies of the transaction in Fig. 1(a). \( DG(\tau) \) given in Fig. 1(b) is cyclic, but it is clear that the copy that seizes \( z \) first will prevent all others from beginning until it completes; so deadlock cannot occur.

![Diagram](a) \quad \text{Figure 1} \quad \begin{tikzpicture}
\node (x) at (0,0) {$x$};
\node (y) at (0,-1) {$y$};
\node (z) at (1,0) {$z$};
\draw[->] (x) -- (y);
\draw[->] (y) -- (x);
\end{tikzpicture}

(b)

3.2 Main Result

Denote by \( F(m) \) the directed graph of Fig. 2. We will show in this subsection that \( m \) copies of \( F(m) \) are necessary and sufficient to produce a deadlock partial schedule.
Lemma 2: $F(m)$ is a transaction over any database $D$ distributed among two or more sites and consisting of $2m-2$ or more entities.

Proof: We have to show that we can select entities $x_1, \ldots, x_m$ from $D$ such that $x_m$ and $x_i$ reside at different sites for $1 \leq i \leq m-1$ (since $Lx_m$ and $Lx_i$ are incomparable). To do so, assume that $D$ is distributed among $r$ sites and the number of entities at some site $i$ is $N_i$. Let $b$ be the site for which $\min\{N_i | 1 \leq i \leq r\}$. There exist at least $m-1$ entities which do not reside at site $b$. Define $x_m$ to be some entity at site $b$ and $x_1, \ldots, x_{m-1}$ some entities at a site other than $b$. 

Lemma 3: Let $\tau$ be a set of copies of $F(m)$. $\tau$ is deadlock free if and only if the number of copies in $\tau$ is less than $m$.

Proof: (only if) We will prove that $m$ copies of $F(m)$ can deadlock. Consider the partial schedule $S$ of copies $T_1, \ldots, T_m$ of $F(m)$. $S$ is defined as follows. Each $T_i$ executes the sequence of steps $t_i$ where: $t_1 = Lx_1$, and $t_i = Lx_1, Lx_2, Ux_1, Lx_3, \ldots, Lx_i, Ux_{i-1}$ for $2 \leq i \leq m-1$, and $t_m = Lx_m$. $S$ consists of $t_{m-1}, t_{m-2}, \ldots, t_1, t_m$ executed in this order. $S$ is a legal partial schedule because when each $T_i$ completes the sequence $t_i$, it holds a lock on one entity alone, $x_i$. Also, $t_j$ does not reference any entity with an index higher than $j$ (these are the only entities which are held locked by copies executing before $t_j$ in $S$). $S$ is a deadlock partial schedule of copies $T_1, \ldots, T_m$. Figure 3 illustrates the steps executed by each $T_i$ in $S$, (to the left of the vertical line), the entity it holds a lock on (underlined), and its critical steps (to the right of the vertical line).
(If) We will prove that less than $m$ copies of $F(m)$ cannot deadlock. Assume that $S$ is a deadlock partial schedule of a set of copies $T$. Note that $DG(T)$ is exactly the cycle $x_m, x_{m-1}, \ldots, x_1, x_m$. Thus, by Lemma 1 after execution of $S$ the critical steps must lock all entities of the set $\{x_1, \ldots, x_m\}$. By Proposition 1, if a copy has more than one critical step then they are pairwise incomparable in $F(m)$. Steps $\ell x_1, \ldots, \ell x_{m-1}$ are totally ordered in $F(m)$, thus two of them cannot be critical steps of the same copy. Therefore, each one is a critical step of a different copy, which implies that in the deadlock are involved, at least $m-1$ copies. Denote the copy which has $\ell x_1$ as a critical step by $T_i$, for $1 \leq i \leq m-1$. Some copy involved in the deadlock locks $x_{m-1}$ in $S$. That copy must also lock (and possibly unlock) $x_1, \ldots, x_{m-2}$ in $S$, therefore its only critical step must be $\ell x_m$. Thus the copy cannot be any of the defined $T_i$'s and there must be an additional copy, $T_m$. 

For a transaction $T$ let $d(T) = \{ n \mid \text{if a set of } n \text{ copies of } T \text{ is deadlock-free then an arbitrarily large set of copies of } T \text{ is deadlock-free} \}$. We are interested in the minimal number in $d(T)$. 

\begin{figure}[h]
\begin{center}
\begin{tabular}{c|c}
$T_{m-1}$: $\ell x_1, \ell x_2, \ell x_3, \ell x_m-1, \ell x_m-2$ & $\ell x_m$ \\
$T_{m-2}$: $\ell x_1, \ell x_2, \ldots, \ell x_{m-2}, \ell x_{m-3}$ & $\ell x_{m-1}$ \\
$T_2$: $\ell x_1, \ell x_2, \ell x_3$ & $\ell x_m$ \\
$T_1$: $\ell x_1$ & $\ell x_2$ \\
$T_m$: $\ell x_m$ & $\ell x_1$ \\
\end{tabular}
\end{center}
\caption{Figure 3}
\end{figure}
Theorem 2: For every positive integer $m$ there exists a transaction $T$ referencing $m$ entities such that $m = \min\{d(T)\}$.

Proof: By Lemma 3 $\min\{d(F(m))\} \geq m$. Left to prove is that $m \in d(F(m))$, namely that if a set of $m$ copies of $F(m)$ is deadlock free then an arbitrarily large set is deadlock free. Assume that a deadlock of more than $m$ copies of $F(m)$ occurs. In the deadlock partial schedule $S$ there must be copies which do not hold locks on any entities. They can be eliminated from $S$ with $S$ remaining a deadlock schedule. 

Consider now the important case in which a transaction $T$ is two phase locked, namely all locks precede all unlocks (e.g. the transactions of Figures 1(a) and 4).

Proposition 2: Two copies of a two-phase-locked transaction $T$ are deadlock free if and only if the transaction has a lock step which precedes all other steps.

Proof: (Only if) Assume that $T$ does not have a lock step which precedes all other steps. Then there exist at least two locks $L_x, L_y$ without predecessors in $T$. A partial schedule $S$ in which copy $T_1$ locks $x$ as the first step and copy $T_2$ locks $y$ as the first step, must end in a deadlock; in $S$, $T_1$ and $T_2$ may be able to execute some more lock steps but copy $T_1$ will not be able to unlock any entity until it executes all locks (including $L_x$), and the symmetric situation exists for $T_2$.

(If) Suppose that $L_x$ precedes all other steps. Then a copy $T_i$ which begins execution (seizes $x$) prevents any other copy from starting until it unlocks $x$, namely until $T_i$ obtains all locks. Thus $T_i$ will not deadlock. 

Corollary 1: A set consisting of an arbitrary number of copies of a two phase locked transaction is deadlock free if the two copies of the transaction are deadlock free.
3.4 Extended Havender Scheme

For transactions whose execution is modeled by a total order of Lock-Unlock steps Havender proposed the following deadlock-prevention scheme [10]. Impose a total order $\mathcal{L}$ on the entities and request that when a transaction issues a $Lx$ step it does not hold a lock on an entity which succeeds $x$ in $\mathcal{L}$. In other words, if a transaction $T$ accesses entities $x$ and $y$ and in $T$

1) $Lx$ succeeds $Ly$, and

2) $Lx$ precedes $Uy$

then $y$ precedes $x$ in $\mathcal{L}$. What if the transaction executes simultaneously at several computers and is modeled by a partial order? Then the scheme does not work any more. To realize that consider the transaction $T$ of Fig. 4. Any two copies of $T$ obey the scheme, but by Proposition 2 they are not deadlock free.

Now consider the following extended Havender scheme for deadlock prevention of arbitrary transactions. Impose a total order $\mathcal{L}$ on the entities and require the following. If a transaction $T$ references entities $x$ and $y$, and in $T$

1) $Lx$ does not precede $Ly$ and

2) $Lx$ precedes $Uy$

then $y$ precedes $x$ in $\mathcal{L}$. The difference between the restricted (or traditional) Havender scheme and the extended one is that condition 1) changes. In the extended version it holds if $Lx$ succeeds $Ly$ or $Lx$ is incomparable to $Ly$. 

Figure 4
Theorem 3: If each transaction in a set of transactions $\tau$ obeys the extended Havender scheme then $\tau$ is deadlock-free.

Proof: $DG(\tau)$ is a partial order (having $I$ as an extension). By Lemma 1 $\tau$ is deadlock free. [ ]

Observation 1: The extended Havender scheme reduces to the restricted one for transactions modeled by total orders.

Observation 2: It has been pointed out in [7] that a set of transactions $\tau$ is deadlock free if the set $\tau' = \{t \mid t$ is an extension of some $T \in \tau\}$, regarded as a set of centralized transactions, is deadlock free. The extended Havender scheme does not ensure deadlock freedom by ensuring that all extensions obey the restricted Havender scheme; it is weaker. To realize that assume that $T$ consists of the nodes $\{Lx, Ux, Ly, Uy\}$ and the arcs $\{Lx \rightarrow Ux, Ly \rightarrow Uy\}$. Then two copies of $T$ obey the extended Havender scheme and are indeed deadlock free. However, the extensions $Lx, Ly, Ux, Uy$ and $Ly, Lx, Ux, Uy$ can deadlock.

Observation 3: In two-phase-locked transactions all locks precede all unlocks. Thus, if such transactions are to obey the extended Havender scheme, two locks cannot be incomparable and therefore all locks must be totally ordered. (Unlocks at different sites can still be incomparable.)

4. SERIALIZABILITY

We will prove that two copies of a transaction can produce a nonserializable schedule if multiple copies can do so. First we will define serializability of a schedule $S$ of transactions $T_1, ..., T_k$ as follows. Given $S$ build a digraph $P(S)$ which has $T_1, ..., T_k$ as its nodes and an arc $(T_i, T_j)$ if $T_i$ references an entity in $S$ before $T_j$ references the same entity. Then $S$ is serializable if $P(S)$ is acyclic. In [1] this definition is justified. A set of transactions ensures serializability if every schedule of the set is serializable.
Theorem 4: If a set of two copies of a transaction $T$ ensures serializability then an arbitrarily large set of copies of $T$ does so.

Proof: Assume that there exists a schedule $S$ of multiple copies of a transaction $T$ which is nonserializable. Let $T_1, T_m, T_1$ be a cycle in $P(S)$ and assume w.l.o.g. that $T_1$ is the copy which executes the first lock of an entity in $S$. Denote that entity by $x$. $T_m$ locks some entity in $S$ before $T_1$ locks the same entity (the arc $T_m \rightarrow T_1$ in $P(S)$). Let this entity be $y$. Consider the subsequence $S'$ of $S$ consisting of the steps of $T_1$ and $T_m$ only. $S'$ is obviously a schedule and in it $T_1$ locks $x$ before $T_m$ and $T_m$ locks $y$ before $T_1$. []

This result holds even if we allow a transaction to have Read (Shared) and Write (Exclusive) locks, (see [5; Sec. 11.3] for definition of the model and the precedence graph $P(S)$). The only difference in the proof above is that $T_1$ would be the first copy to write - lock an entity.

5. CONCLUSION

In this paper we addressed the issues of deadlock freedom and serializability assurance of transaction copies. They arise when a transaction is started simultaneously by multiple users. The main result indicates that for every $m$ there exists a transaction referencing $m$ entities such that $m$ copies are necessary and sufficient to produce a deadlock. Thus if a transaction references $m$ entities and we want to determine whether multiple copies of it are deadlock free or not, we must test at least $m$ copies for deadlock freedom. Consider this result opposite the one we obtain about serializability assurance. A transaction for which three copies can produce a nonserializable schedule but two cannot, does not exist.

There are two interesting side-effects of the analysis leading to the main result. First is the extension of Havender's scheme of deadlock prevention (accessing resources in a fixed order) to processes, or transactions, modeled by partial ord-
ers. Second, is the establishment of an efficiently testable condition which is sufficient for deadlock freedom of multiple transactions.

The complexity of determining deadlock freedom of two transaction copies remains open for future work. The same applies to serializability assurance. Presently, both problems can be solved only by some exhaustive search method ([7], [2]) and we conjecture that they are coNP-complete. However, this may be tolerable because the determination is not performed at run time.
REFERENCES


