Locking based on a pairwise decomposition of the transaction system

Georg Lausen

Fakultät für Mathematik und Informatik, Universität Mannheim, W-6800 Mannheim, Germany

Received 15 January 1990
Revised 12 May 1991

Abstract


Locking is a synchronization primitive used in database systems to guarantee correctness of a concurrent execution of a set of user transactions accessing the database. The way locking is used should not restrict potential concurrency more than necessary. Thus, there is the problem how to insert lock and unlock operations into the transactions. This paper surveys work on locking, which is based on a pairwise decomposition of the transaction system. Such a decomposition supports the application of two-dimensional geometric techniques, which have turned out to be very helpful in designing locking algorithms.

1. Introduction

Locking is a synchronization primitive used in database systems to guarantee correctness of a concurrent execution of a set of user transactions accessing the database. Each transaction consists of a sequence of read and write actions to the entities stored in the database. One of the simplest examples to see that correctness indeed is affected by concurrency is a situation in which there are two transactions both incrementing the same counter. Denote the counter by \( x \), and let \( x \) be stored in a database. Then the action sequence of the two transactions can be written as:
$T_1: R_1 x increment_1(x) W_1 x$, respectively, $T_2: R_2 x increment_2(x) W_2 x$, where $R_1 x$, respectively, $W_1 x$, denotes a read, respectively, write action of transaction $T_1$ to database entity $x$. Now assume, due to concurrency, the transactions are interleaved as follows: $R_1 x increment_1(x) R_2 x increment_2(x) W_2 x W_1 x$. Assume, at the beginning, we have $x = 1$. Then after having executed both transactions, we have $x = 2$, and not $x = 3$, as we would like to expect from the two increment operations. We have observed a “lost update” due to conflicting actions; both transactions first read the same initial value from the database into their local memory, the increment is performed on the respective local memory and then is written back into the (global) database. Clearly, there is no way for transaction $T_2$ to recognize that $x$ in the meanwhile has already been incremented by $T_1$.

By locking we can disallow incorrect transaction interleavings. Let us denote by $L_i x$ a lock operation of transaction $T_i$ to entity $x$, and by $U_i x$ the corresponding unlock. Now what we have to do to disallow the above incorrect interleaving is to surround the actions to $x$ in both transactions by a corresponding pair of lock and unlock: $LT_i : L_i x R_i x increment_i(x) W_i x U_i x$, $i \in \{1, 2\}$. After a transaction has executed its lock operation, no other transaction can execute a lock operation with respect to the same entity before the first one has unlocked the corresponding lock. A situation in which both transactions increment in a way that the second increment is performed before the first has been written in the database, now becomes impossible. This is the idea to guarantee correct interleaved transactions on a database by locking. The problem of locking is to insert lock and unlock operations into the transactions, such that, on the one side, no incorrect interleavings may occur, while on the other side, there still are allowed as much interleavings as possible to guarantee a high degree of potential concurrency.

Locking has been investigated continuously over the recent years. In Papadimitriou [10] we can find a theoretical exposition, while Bernstein et al. [1] focus on locking from a more application oriented viewpoint. In this paper we survey techniques based on a pre-analysis of a set of transactions with the aim to insert lock and unlock operations into transactions in a way to allow as much correct interleavings as possible [3, 4, 6, 7]. However, it is well known, that the set of all correct schedules of a given set of transactions cannot be achieved by locking [7, 8, 12]. As a characteristic feature, our techniques are based on a pairwise decomposition of the set of transactions. Such a decomposition allows the application of two-dimensional geometric techniques, which have turned out to be very helpful in designing locking algorithms.

The structure of the paper is as follows. In Section 2 we introduce our necessary definitions and in Section 3 we introduce the geometry of concurrency control. Section 4 is devoted to locking and in Section 5 we discuss the problem of maximal concurrency by locking. The topics of Sections 6 and 7 are pre-analysis locking and two-phase locking. Two-phase locking is the most commonly used locking strategy [2]. In Section 8 we show how pre-analysis algorithms can be designed to dominate two-phase locking. Section 9 concludes the paper.
2. Basic definitions

A transaction system $\tau = \{T_1, \ldots, T_d\}$ is a set of transactions, where each transaction $T_i = (A_{i1}, \ldots, A_{im_i})$, $m_i \geq 1$, is a sequence of actions. Each action $A_{ij}$ is associated with an entity $x_{ij} \in E$, where $E$ is a set of entities forming the database. We distinguish read and write actions, $A_{ij} = RX_{ij}$ meaning read $x_{ij}$ and $A_{ij} = WX_{ij}$ meaning write $x_{ij}$. We will write $R_i x (W_i x)$, respectively $RX (WX)$, if the position of the action, respectively also the corresponding transaction, is given by the context. Two actions of two different transactions conflict, if they involve the same entity and at least one of them is a write action.

A schedule $s$ of $\tau$ is a permutation of all actions of $\tau$, with the actions within each transaction in the prescribed order. A schedule models a concurrent, i.e., interleaved, execution of the transactions in a given system. If $A_{ij}$ is an action in $s$, then $s(A_{ij})$ denotes the position of $A_{ij}$ in $s$. If $s$ is a schedule of $\tau$ and $T_i, T_j \in \tau$, then $s_{ij}$ is the projection of $s$ onto the actions of transactions $T_i$ and $T_j$. A schedule $s$ is serial if the transactions are not interleaved, i.e., $T_i = (A_{i1}, \ldots, A_{im_i})$, $m_i \geq 1$, implies $s(A_{ij+1}) = s(A_{ij}) + 1$, $1 \leq j < m_i$. For schedule $s$, the conflict graph $D(s)$ has as vertices the transactions of $\tau$, and an arc $T_i \rightarrow T_j$ whenever an action of $T_i$ precedes in $s$ a conflicting action of $T_j$. A schedule is called serializable if $D(s) = \bar{D}(s')$ for some serial schedule $s'$ of $\tau$, or, equivalently, if $D(s)$ is acyclic.

Serializability is the correctness criterion used for concurrent transactions in database systems. A detailed discussion can be found in [10]. Serializability guarantees correctness for the following reasons. It is a basic assumption, that each single transaction is programmed correctly. Thus any serial execution of a set of transactions always guarantees correctness. Observe further, that we have edges in the conflict graph whenever we have conflicting actions in the corresponding schedule. If a conflict graph is acyclic, then any topological sorting of its nodes may be used to construct a serial schedule, in which conflicting actions are in the same order as they are in the original schedule. The same order of the conflicts guarantees the same effects of the transactions (cf. [10]); thus, indeed, we can use acyclicity of the conflict graph as a sufficient condition for correctness of a given schedule.

3. The geometry of a concurrency control

We now introduce the geometric interpretation for concurrency control [9, 10, 13]. Let $\tau = \{T_1, \ldots, T_d\}$ be a transaction system. Consider each transaction to be an axis in a $d$-dimensional coordinate system, with the actions being the coordinate values on the axes. Each pair of transactions corresponds to a plane with a grid imposed by the actions. Any nondecreasing curve from point $(0,0)$ to point $(m_i+1, m_j+1)$ in a plane $(T_i, T_j)$ not passing through any other grid point is the geometric image of the projection $s_{ij}$ of a schedule $s$ of $\tau$ on $T_i$ and $T_j$. The order of the actions in the schedule is the order in which the curve intersects the corresponding grid lines.
From the class of curves representing a schedule we shall choose the uniquely defined shortest one, which passes every grid square through its middle point. In the sequel we shall identify a schedule with such a curve, called the curve of the schedule. Figure 1 shows the curve of the schedule

$$R_1a \ W_1a \ R_2a \ R_2b \ R_1b \ W_1b \ R_1c \ W_1c \ R_2c.$$  

In [6] we introduced direct and indirect conflict points to represent conflicting actions in the geometric setting.

A point $P = (A_{ip}, A_{jq})$ in plane $(T_i, T_j)$ is a direct conflict point between $T_i$ and $T_j$, $i \neq j$, if $A_{ip}$ and $A_{jq}$ conflict. Point $P = (A_{ip}, A_{jq})$ is called an indirect conflict point between $T_i$ and $T_j$, $i \neq j$, if there exists a set of transactions

$$\{T_{ki} \mid 1 \leq l \leq n, n \geq 1\} \subseteq (\tau - \{T_i, T_j\}),$$

such that $(A_{ki,p}, A_{ki+1,q})$, for some $p_l$ and $q_{l+1}$, is a direct conflict point between $T_{ki}$ and $T_{ki+1}$ for all $l$, $1 \leq l \leq n-1$, and $(A_{ip}, A_{ki,q})$, for some $q_1$, and $(A_{kn,p}, A_{jq})$, for some $p_n$, are direct conflict points between $T_i$ and $T_{ki}$, and $T_{kn}$ and $T_j$, respectively.

Direct and indirect conflict points are also called conflict points. Figure 2 shows the conflict points of a transaction system of three transactions. For example, indirect conflict point $(W_1b, R_2d)$ is implied by the direct conflict points $(W_1b, R_1b)$, $(W_2d, R_2d)$. If a conflict point is direct and indirect, then it is represented as direct one, only.

For a two-transaction system $\tau = \{T_1, T_2\}$, it is straightforward to see that a schedule is serializable iff its curve doesn't separate two direct conflict points (cf. [10]). By separation we mean that the schedule’s curve goes between two conflict points. For example, let $T_1, T_2$ be the corresponding transactions, and let $(W_1a, R_2a)$, $(W_1b, R_2b)$ be two conflict points. Assume further a schedule $s$ with a curve as shown in Fig. 1. The curve goes below $(W_1a, R_2a)$ and above $(W_1b, R_2b)$. It passes the $W_1a$ grid line before the $R_2a$ grid line, and the $R_2b$ grid line before the $W_1b$ grid line. Thus, $s(W_1a) < s(R_2a)$, $s(R_2b) < s(W_1b)$ and therefore $\bar{D}(s)$ contains a cycle, i.e., $s$ is not serializable.

Next we introduce an interesting property of nonserializable schedules for ar-
Locking

T2

I A

I

 arbitrary transaction systems. Based on this property we will later design concrete locking algorithms. Consider a nonserializable schedule $s$ of a system $T$ of more than two transactions. Then $\bar{D}(s)$ contains a cycle, say $T_1 \rightarrow T_2 \rightarrow \cdots \rightarrow T_{k+1} = T_1$, where $k \geq 3$. Let $s(A_{i_{p_i}}) < s(A_{j_{q_j}})$, where $A_{i_{p_i}}, A_{j_{q_j}}$ are conflicting actions of $T_i, T_j$, respectively, which imply $T_i \rightarrow T_j$, where $1 \leq i \leq k, j = i + 1$. Then $(A_{i_{p_i}}, A_{j_{q_j}})$ is a direct conflict point, and, as $T_i, T_j$ are contained in the cycle, $(A_{i_{p_i}}, A_{j_{q_j}})$ is an indirect conflict point. As we are interested in locking based on a pairwise decomposition of the transaction system, in analogy to the two-transaction case, we would like to expect that $s$ has a plane projection, i.e., a projection on the actions of the transactions defining the plane, such that the corresponding projected curve separates some conflict points in that plane. Assume the contrary. Then for all $i, j$ adjacent in the cycle, we have $s(A_{i_{p_i}}) < s(A_{j_{p_j}})$. Direct and indirect conflict points imply the ordering $s(A_{1_{p_1}}) < s(A_{2_{p_2}}) < s(A_{3_{p_3}}) < \cdots < s(A_{1_{p_1}})$, which is a contradiction to the linear order of the actions within each transaction. As each of the planes we have considered contains at least one direct conflict point, we have thus shown, that in a system of more than two transactions, each nonserializable schedule has a plane projection such that the curve of the corresponding schedule separates some conflict points, where at least one of the separated conflict points is a direct one [6]. However, some serializable schedules also separate conflict points. Figure 2 provides examples. The solid schedule

$$R_1 a W_1 a R_2 a W_2 a R_2 d W_2 d R_1 b W_1 b R_1 c W_1 c R_3 c W_3 c T_3$$

![Fig. 2. Direct (●) and indirect (○) conflict points, and the curves of two schedules.](image-url)
is serializable although conflict points are separated, the dashed schedule

\[ R_1 a \ W_1 a \ R_2 a \ W_2 a \ R_2 d \ W_2 d \ T_3 \ R_1 b \ W_1 b \ R_1 c \ W_1 c \ R_2 c \ W_2 c \]

is not serializable.

As each nonserializable schedule has a plane projection which separates two conflict points, one of which is a direct one, there is an obvious way to guarantee serializable schedules by forbidding all schedules which have a plane projection of the described way. In geometric terms, we have to introduce so-called *forbidden regions* which contain all conflict points in those planes, in which a critical separation might occur. Forbidden region means, that it is not allowed for a schedule to pass through it. We shall see in the next section how such regions can be implemented by locking. We require that forbidden regions are connected and that the contour of each forbidden region is composed of two nondecreasing (staircase) curves, which pass the grid squares through the middle points. A schedule is forbidden, if one of its plane projections passes through a forbidden region; tangents of a forbidden region are allowed. In Fig. 3 we see an example of a forbidden region and an allowed schedule; the curve depicted in Fig. 1 would be forbidden.

4. Locking in the geometric setting

Forbidden regions can be implemented by locking (cf. [10]). Locking is a synchronization primitive used in database systems to guarantee serializable schedules. To this end, lock and unlock operations are inserted into the transactions. These operations may act on the entities forming the database, or on special locking variables. Whenever a transaction has locked such an object, no other lock to that object is allowed before the transaction has executed the corresponding unlock. In this way, locking may disallow certain schedules. We are interested in the question how to use locking to maximize the set of allowed schedules by disallowing all nonserializable schedules.

---

Fig. 3. A forbidden region and the curve of an allowed schedule.
Let \( \tau = \{T_1, \ldots, T_d\} \) be a transaction system. A **locked transaction system** \( L\tau \) of \( \tau \) is a set of **locked transactions**, \( L\tau = \{LT_1, \ldots, LT_d\} \), where each locked transaction is a transaction that contains, besides the actions, pairs of **lock** \( u (Lv) \) and **unlock** \( v (Uv) \) operations, where \( v \) is an element of \( LV \), the set of **locking variables**, \( E \cap LV = \emptyset \), and \( Lv \) always precedes \( Uv \). A locked schedule \( Ls \) of \( L\tau \) is **legal**, if there is an \( Uv \) operation between any two \( Lv \) operations in \( Ls \). The set of schedules accepted by a locked transaction system \( L\tau \) is defined as \( \text{Acc}(L\tau) = \{s \mid Ls \text{ is a legal schedule of } L\tau, \text{ and } s \text{ is obtained from } Ls \text{ by deleting all lock and unlock operations}\} \). \( L\tau \) is safe if \( \text{Acc}(L\tau) \) contains only serializable schedules. Let \( \text{SR}(\tau) \) be the set of all serializable schedules of \( \tau \). We say a locked transaction system \( L\tau \) **realizes** \( \text{SR}(\tau) \), whenever \( \text{Acc}(L\tau) = \text{SR}(\tau) \). Let \( L\tau \) and \( L'(\tau) \) be two locked versions of transaction system \( \tau \). We say \( L\tau \) **dominates** \( L'(\tau) \) if \( \text{Acc}(L\tau) \supseteq \text{Acc}(L'(\tau)) \).

In our setting locking variables are uninterpreted, they are not associated with entities. Instead, each locking variable \( v \) is associated with at most one pair of transactions, where only one \( Lv \) and one \( Uv \) operation is in each of these transactions. More than one locking variable may be associated with the same two transactions. We use locking variables instead of the usual entity locks, as our goal is to study locking based on a pairwise decomposition of the transaction system. In such a context locking variables have the appealing property than they only act in one plane, while an entity lock in one transaction would act in all planes containing this transaction and another transaction accessing the corresponding entity.

In the geometric representation \( Lv \) and \( Uv \) operations are coordinate values on the axis of the coordinate system, too. The geometric image of a pair of \( Lv \) and \( Uv \) operations in a transaction plane is a rectangle which cannot be entered by the curve of any legal locked schedule [10] (cf. Fig. 4). We call such a rectangle a **forbidden rectangle**.

![Fig. 4. The geometry of lock and unlock.](image)
A forbidden region in a plane now can be implemented by locking in the following way. We first have to compute a set of rectangles which covers the forbidden region. Then we have to define the rectangles by inserting appropriate lock and unlock operations into the transactions. Figure 4 shows an implementation of the forbidden region of Fig. 3; algorithms are described in detail in [6].

Forbidden regions, which are implemented by locking, are a geometric means to guarantee serializable schedules by enforcing an appropriate order on the actions appearing in the transactions. Let $L_T$ be a locked transaction system, $T_i, T_j \in \tau$, $i \neq j$. Let $F_{ij}$ be a forbidden region in plane $(T_i, T_j)$, which is implemented by lock and unlock operations in $LT_i$ and $LT_j$. Then for any legal locked schedule $L_s$ of $L_T$, as $F_{ij}$ is connected, $F_{ij}$ enforces an order on the actions of transactions $T_i, T_j$:

$$(A_{ip}, A_{jq}), (A_{ip}, A_{jq}) \in F_{ij}, \quad s(A_{ip}) < s(A_{jq}) \Rightarrow s(A_{ip}) < s(A_{jq}),$$

where $s$ is $L_s$ with all lock and unlock operations deleted.

5. Maximal concurrency by locking

Here we first restrict our considerations to two-transaction systems $\tau = \{T_1, T_2\}$. We already know, that for two-transaction systems a schedule is serializable iff its curve does not separate two (direct) conflict points. Therefore, any safe locked transaction system has a forbidden region which contains all conflict points.

The interesting question now is, whether for any such $\tau$ there exists a locked version $L_T$ which realizes $SR(\tau)$. In general, the answer is no. Observe first, that any such safe $L_T$ must imply a smallest forbidden region $F$, which contains all conflict points, i.e., there does not exist a safe $L_T$, such that for the corresponding forbidden region $F'$ there holds $F' \subset F$, $F' \neq F$. Observe then, that such smallest forbidden regions may not be unique. In Fig. 5 we given an example. Two different smallest forbidden regions are shown, which contain all conflict points. Assume that the

Fig. 5. A situation in which a smallest forbidden region is not uniquely defined.
solid one corresponds to $L^1_r$, and the dashed one to $L^2_r$. Now consider the following two serializable schedules:

$$s_1 = R_1 a R_1 b W_2 a W_2 b R_2 c W_1 e R_1 d W_1 c W_2 c W_2 d,$$

$$s_2 = W_2 a W_2 b R_1 a R_1 b W_1 e R_2 c W_2 c W_2 d R_1 d W_1 c.$$

We have $s_1 \in \text{Acc}(L^1_r)$, $s_1 \notin \text{Acc}(L^2_r)$, $s_2 \in \text{Acc}(L^2_r)$, $s_2 \notin \text{Acc}(L^1_r)$. Obviously, there cannot exist a forbidden region such that $s_1$ and $s_2$ are accepted by a corresponding safe transaction system. Therefore, the set of all serializable schedules cannot be realized by a safe locked transaction system, in general. However, if there exists a unique smallest forbidden region, then the corresponding locked transaction system realizes $\text{SR}(r)$ [7].

In cases where we cannot accept the set of all serializable schedules, there is the problem to find a safe locked transaction system which maximizes the number of accepted schedules; the corresponding forbidden region is called optimal. We only have to consider cases in which there exists more than one smallest forbidden region. It is shown in detail in [7], if we intersect all smallest forbidden regions, the results are two or more connected subregions, which are strictly ordered from left to right and bottom to top. Observe that every conflict point is contained in one such subregion. Figure 5 provides an example. Regions $A_1$ and $A_2$ are the result of the intersection of the two existing smallest forbidden regions, which are indicated in the figure. In general, to achieve an optimal forbidden area, any two components adjacent in the order have to be connected by a rectilinear curve, which passes each grid square through the middle point. Now the computational problem is to determine which of the (exponentially many) rectilinear curves should be selected. Figure 6 de-
picts an example with one rectilinear curve, connecting $A_1$ with $A_2$. In the example, this curve is optimal among all ten possible curves. In general, each rectilinear curve bridging a gap of $g_i$ actions in transaction $T_i$, $i = 1, 2$, corresponds to an interleaving of the $g_i$ actions of $T_1$ with the $g_2$ actions of $T_2$. The corresponding partial schedule of $g_1 + g_2$ actions can be completely described by specifying the positions within the sequence of $g_1 + g_2$ actions at which the $g_1$ actions of $T_1$ occur (or, equivalently, at which the $g_2$ actions of $T_2$ occur). Therefore we get $(g_1 + g_2) = (g_1 + g_2)$ possible curves for a gap of $g_i$ actions in $T_i$, $i = 1, 2$. One of these curves has to be selected for each gap with respect to the global optimum over all rectilinear curves bridging all gaps.

We conjecture that the decision problem corresponding to this optimization problem is NP-complete [7].

How do these results relate to the case of arbitrary transaction systems? For a locked transaction system $L_T = \{L_{T_1}, \ldots, L_{T_d}\}$, $d \geq 3$, to realize the full set of serializable schedules, at first any projected two-transaction system $L_{T_{ij}} = \{L_{T_i}, L_{T_j}\}$, $1 \leq i < j \leq d$, has to realize SR(\{T_i, T_j\}). One might expect, whenever a locked transaction system $L_T$ has this pairwise property and additionally is safe, that in this case it also realizes SR($T$). It is shown in [7] that this conjecture does not hold, in general. However, if we require a certain locking technique called LU-free locking, i.e., between any lock and unlock there is at least one read or write action, then, whenever for every pair $T_i, T_j$ the full set of all serializable schedules can be realized by locking, we can merge all locked two-transaction systems to a locked transaction system $L_T$ such that $L_T$ realizes SR($T$).

6. A framework for pre-analysis algorithms

In the preceding section we have seen, that even in the case of two transactions, the full set of all serializable schedules cannot be accepted by a safe locked transaction system, in general. Moreover, any algorithm which performs optimal locking to maximize the number of accepted schedules (with respect to a certain plane) seems to be computationally prohibitive. In this section we present a general framework for pre-analysis algorithms, which take a given transaction system $T$ as input and derive a safe locked transaction system $L_T$. We make no assumptions for the strategies in finding forbidden regions in the respective planes.

Let $T = \{T_1, \ldots, T_d\}$ be a given transaction system. For each plane $(T_i, T_j)$, we will be interested in a forbidden region which contains some given set $C(T_i, T_j)$ of points. For each such plane, the construction of such a forbidden region can be done by the following steps (see [6] for details):

1 Think here of conflict points; however, we will later introduce a locking algorithm, in which we will also consider points which are not conflict points.
(1) Construct a minimal connected rectilinear region which contains the respective set of points in the plane \((T_i, T_j)\).

(2) Compute a set of rectangles that covers the constructed rectilinear region.

(3) Make the region forbidden, i.e., insert corresponding lock/unlock pairs into the transactions which define the covering rectangles. The resulting locked transactions are denoted \(LT^i_i\) and \(LT^j_j\), where \(LT^i_i\) is the locked version of \(T_i\), and \(LT^j_j\) is the locked version of \(T_j\) with respect to the plane \((T_i, T_j)\).

A corresponding safe locked transaction system then is derived by merging all versions \(LT^i_i\), \(1 \leq j \leq d\), \(i \neq j\) for each transaction \(T_i\), such that the order of the actions and lock and unlock operations for each version is preserved. The order of lock and unlock operations stemming from different versions is not necessarily unique. For example, consider \(LT^i_i = \cdots Lw \cdots A_{(j+1)} \cdots\) and \(LT^k_k = \cdots Lw \cdots A_{(l+1)} \cdots\), where \(j \neq k\) and \(v \neq w\). Thus, after merging \(LT^i_i\) and \(LT^k_k\), \(Lw\) either precedes \(Uw\), or \(Uw\) precedes \(Lv\). In general, in such situations the order of lock/unlock operations may affect serializability. However, for the policies we shall propose it is sufficient that for each plane \((T_i, T_j)\) the respective set \(C(T_i, T_j)\) is contained in the forbidden region. If during the merging there exists more than one possible order for lock or unlock operations stemming from different versions \(LT^i_i, LT^k_k, j \neq k\), then any order may be chosen. Thus, starting with an unlocked transaction system we can derive a locked version of this system by implementing forbidden regions and afterwards merging the resulting locked versions of each transaction.

7. Pre-analysis locking and two-phase locking

In this section we discuss locking based on a pre-analysis in comparison to two-phase locking, which is the commonly used technique to guarantee safety. Pre-analysis and two-phase locking are general strategies and not fixed algorithms. Therefore, a comparison cannot be based on two specific implementations of the general principle. Here, the notion of a locking policy is helpful. A locking policy \(P\) is a function that maps a transaction system \(\tau\) to a set of locked versions of \(\tau\). If \(L\tau \in P(\tau)\), we say that \(L\tau\) is locked according to \(P\). Locking policy \(P\) is called safe, if any \(L\tau\) locked according to \(P\) is safe. Locking policy \(P_1\) dominates locking policy \(P_2\), if for every transaction system \(\tau\) there holds:

\[
L\tau \in P_2(\tau) \implies \text{there exists } L'\tau \in P_1(\tau) \text{ such that } L'\tau \text{ dominates } L\tau.
\]

Basic pre-analysis locking (bPAL) is a locking policy that maps any transaction system \(\tau\) to a set of locked transaction systems, such that for each corresponding \(L\tau\) a forbidden region is created for each pair of transactions \(T_i, T_j\), whose plane contains at least two conflict points, one of which is a direct one, such that all conflict points in that plane, direct and indirect ones, are included. (In general, for any plane, there may exist many forbidden regions which contain the relevant set of con-
flict points and therefore yield different locked transaction systems.) This construction of forbidden regions guarantees that no nonserializable schedule is legal, but in general also forbids serializable schedules. In [6] we have described in detail a certain bPAL implementation which is free from deadlock. Figure 7 shows forbidden regions derived according to this bPAL implementation. Neither schedule in Fig. 2 is accepted.

The locking policy two-phase locking (2PL) maps any transaction system \( \tau \) to a set of locked transaction systems such that the following two conditions are fulfilled [2]: in each locked transaction each action is surrounded by a lock and unlock operation, and every lock operation precedes all unlock operations. (There exist several possibilities to lock according to 2PL. We may insert every lock operation at the beginning and every unlock operation at the end of the transaction, or, every lock at the beginning and every unlock immediately after the corresponding read or write action, etc.) Usually entity locks are used such that each action accessing entity \( e \) is surrounded by \( Le \) and \( Ue \). In the sequel, whenever we refer to 2PL, we will assume entity locks. As 2PL does not use a conflict point based pre-analysis, a lock-

![Fig. 7. Forbidden regions implied by bPAL.](image-url)
Locking 229

ing according to 2PL cannot be derived by considering each plane in a pairwise decomposition of the respective transaction system independently from the others, in general.

The geometric representation of a locked transaction system according to 2PL can be characterized as follows (cf. [9]). First, for any plane \((T_i, T_j)\) the forbidden region is composed of overlapping rectangles having at least one point in common. Second, for any transaction triple \(T_i, T_j, T_k\) each forbidden rectangle in \((T_i, T_j)\) overlaps all forbidden rectangles in \((T_j, T_k)\) on their projections on the \(T_j\)-axis. Figure 8 shows a locked transaction system according to 2PL.

There is no dominance relationship between bPAL and 2PL, in general. To see that there exist schedules which are accepted by a locked transaction system according to bPAL and cannot be accepted by any locked transaction system according to 2PL, we may compare Figs. 7 and 8. The locked transaction system according to bPAL accepts the schedules

\[
R_1 a W_1 a R_2 a R_1 b W_2 a W_1 b R_2 d R_1 c W_2 d W_1 c R_2 c W_2 c T_3,
\]

\[
R_2 a W_2 a R_1 a W_1 a R_2 d W_2 d R_1 b W_1 b R_2 c W_2 c R_1 c W_1 c T_3,
\]

---

[Image of Figure 8 showing forbidden regions implied by 2PL.]
which could not be accepted simultaneously by any locked version according to 2PL. On the other hand, from the geometric pattern of the forbidden regions in plane \((T_1, T_3)\) or plane \((T_2, T_3)\) in Figs. 7 and 8, we can conclude that there are schedules accepted by a 2PL version, and which could not be accepted by any bPAL version.

The differences between bPAL and 2PL are further illustrated in Fig. 9. An example of a transaction system is given, for which a locked transaction system according to bPAL strictly dominates any locked transaction system according to 2PL. Note that bPAL does not introduce a forbidden region in plane \((T_2, T_3)\) as there is no direct conflict point. In this case bPAL recognizes that the cycles in the directed conflict graph could be of length 2 only, while 2PL also cares for cycles of length 3, i.e., the forbidden rectangles in different planes are forced to overlap.

Fig. 9. Forbidden regions implied by (a) bPAL and (b) 2PL.
Continuing the discussion of Figs. 7 and 8 we can observe that bPAL needs eleven lock operations, while for any 2PL version four operations are sufficient. Whether bPAL or 2PL needs more lock operations, in general, depends on the geometric pattern of the conflict points. Figure 10 shows an example where bPAL needs only one lock operation, while any locked transaction system according to 2PL will require \( n \) lock operations.

Can we find a policy, which, based on a pre-analysis, dominates 2PL for any transaction system? The next section will show, that this indeed is possible.

8. Dominating two-phase locking

We introduce a sufficient geometric condition for safe locked transaction systems, which, similar to 2PL, guarantees safety by overlapping forbidden regions. However, this will not imply that in adjacent planes, i.e., planes which have one transaction in common, all forbidden rectangles have to overlap. Thus the restrictive lock/unlock rule of 2PL is weakened.

Let \( L_\tau \) be a locked transaction system. \( D(\tau) \) is an undirected graph which has as nodes the transactions in \( \tau \) and an edge \( T_i \rightarrow T_j \) whenever \( T_i \) and \( T_j \) have conflicting actions. \( L_\tau \) is called overlap locked (OL locked) [3,4], if:

1. In each plane, in which there is at least one direct conflict point, there is a forbidden region which contains all direct conflict points.
2. Let \( S_1 - S_2 - \cdots - S_n - S_1, \ n > 2 \) be a minimal cycle in \( D(\tau) \). Then there exist

2 A cycle \( n_1 - \cdots - n_n - n_1 \) is called minimal, if \( n_i \neq n_j \) for \( 1 \leq i < j \leq n \), and there are no other edges in the underlying graph between any two nodes in the cycle.
coordinates $P_1, \ldots, P_m$ (not necessarily grid coordinates), where $m = k \cdot n$, $k \geq 1$, such that

$$(P_1, P_2) \in F_{12}, (P_2, P_3) \in F_{23}, \ldots, (P_m, P_1) \in F_{m1}.$$ 

$F_{12}, F_{23}, \ldots, F_{m1}$ are the forbidden regions of planes $(S'_1, S'_2, (S'_2, S'_3), \ldots, (S'_m, S'_1)$, respectively, $\{S'_1, \ldots, S'_m\} = \{S_1, \ldots, S_n\}$ and

$$S'_1 - \cdots - S'_m - S'_i = k(S_1 - \cdots - S_n) - S_i.$$ 

The points $(P_1, P_2), \ldots, (P_m, P_1)$ are called the overlap points of the corresponding cycle.

It is easy to see that any OL locked transaction system is safe. Let $s$ be a schedule which is not serializable, i.e., $D(s)$ contains a minimal cycle $S_1 \rightarrow S_2 \rightarrow \cdots \rightarrow S_n \rightarrow S_1$. As all direct conflict points in a plane are contained in the forbidden region, we can assume $n > 2$. Let $(A_{iq}, A_{jq})$ be a direct conflict point of plane $(T_i, T_j)$, where $T_i \rightarrow T_j$ is contained in the cycle. Thus $s(A_{iq}) < s(A_{jq})$. As all direct conflict points and all overlap points in one plane are contained in the same forbidden region, we can conclude $s(P_i) < s(P_j)$ for all overlap points $(P_i, P_j)$ of the corresponding cycle. Since this holds for any pair of neighboring transactions in the cycle, $s(P_i) < s(P_j)$ would be implied if $Lt$ accepted $s$, a contradiction.

We now show, that two-phase locking and bPAL are variants of OL locking.

**Fact 1.** Any two-phase locked transaction system $Lt$ is also OL locked.

**Proof.** In any transaction all lock operations precede all unlock operations, and each action is surrounded by a lock/unlock pair. Therefore, condition (OL1) is fulfilled. For any locked transaction $LT \in Lt$ let $Q_i$ be the coordinate of the first unlock operation in $LT$. Let $T_i - T_j - T_k = T_l$ be a path in $D(\tau)$. Then $F_{ki}, F_{ij}$ and $F_{jl}$ all contain at least one rectangle implied by lock/unlock operations, say $R_{ki}, R_{ij}$ and $R_{jl}$. Since unlock operations always succeed all lock operations, we have $(Q_k, Q_i) \in R_{ki}, (Q_i, Q_j) \in R_{ij} \subseteq F_{ij}$ and $(Q_j, Q_l) \in R_{jl} \subseteq F_{jl}$. The same arguments hold for any edge in $D(\tau)$. Thus $(Q_i, Q_j)$ is an overlap point for $(T_i, T_j)$ with respect to any minimal cycle in $D(\tau)$. Condition (OL2) is fulfilled and $Lt$ is OL locked.

**Fact 2.** Any locked transaction system $Lt$ according to bPAL is also OL locked.

**Proof.** Clearly condition (OL1) is fulfilled. Let $S_1 - \cdots - S_{n+1} = S_i$ be a minimal cycle in $D(\tau)$, $n > 2$. Assume first $n$ is even. Consider the respective direct conflict points $(A_{1p_1}, A_{2q_1}, (A_{2p_2}, A_{3q_1}), \ldots, (A_{np_n}, A_{1q_1})$. Then $(A_{1q_1}, A_{2p_1}), (A_{2q_2}, A_{3p_1}), (A_{3q_3}, A_{4p_1}), \ldots, (A_{n-1q_{n-1}}, A_{np_n}), (A_{np_n}, A_{1q_1})$ are indirect conflict points. As all

For $k \geq 1$ we denote by $k(n_1 - \cdots - n_n)$ the path in which $n_1 - \cdots - n_n$ is repeated $k$ times.
direct and indirect conflict points lie in forbidden regions determined by locks of 
$L_\tau$, we conclude that $(A_{1p_1}, A_{2q_2}), (A_{2p_2}, A_{3p_3}), (A_{3p_3}, A_{4q_4}), \ldots, (A_{n-1p_{n-1}}, A_{nq_n}),$
$(A_{nq_n}, A_{1p_1})$ are contained in forbidden regions and therefore are overlap points
for the cycle. Now assume $n$ is odd. Similar to the above, we conclude that
$(A_{1p_1}, A_{2p_2}), (A_{2p_2}, A_{3p_3}), (A_{3p_3}, A_{4q_4}), \ldots, (A_{n-1p_{n-1}}, A_{nq_n}),$
$(A_{nq_n}, A_{1p_1})$ are contained in forbidden regions. Further, $(A_{1q_1}, A_{2p_2}), (A_{2p_2}, A_{3q_3}), (A_{3q_3}, A_{4p_4}), \ldots, (A_{n-1q_{n-1}}, A_{nq_n}),$
$(A_{nq_n}, A_{1q_1})$ are contained in forbidden regions. Thus, all these points together are
appropriate overlap points for the cycle. (Note, that to fulfill condition (OL2) we
have to run through the cycle twice.) Therefore, condition (OL2) is fulfilled and
bPAL is OL locked. \(\square\)

We shall now present an efficient locking policy based on OL locking which
dominates 2PL. The policy overlap point locking (OL) selects for each transaction
one unique overlap point coordinate. For any given transaction system the corres-
ponding set of locked transaction systems is derived by the following nondeter-
ministic algorithm OL [3, 4]:

**Algorithm OL.** Let $\tau = \{T_1, \ldots, T_d\}$, $d \geq 2$, be a transaction system. Algorithm OL
considers each transaction pair and may realize forbidden regions in the corre-
ponding planes in order to construct a locked transaction system $L_\tau$.

*Step 1.* For each transaction $T_i$ select one coordinate on the $T_i$-axis as overlap
point coordinate; denote this coordinate $Q_i$.

*Step 2.* For each pair of transactions $T_i$, $T_j$, $i \neq j$, initialize the set $C(T_i, T_j)$ to
contain the direct conflict points of plane $(T_i, T_j)$. If $C(T_i, T_j)$ is not empty, then
add overlap point $(Q_i, Q_j)$ to $C(T_i, T_j)$.

*Step 3.* For each pair of transactions $T_i, T_j$, $i \neq j$ such that $C(T_i, T_j)$ is not empty,
implement a forbidden region which contains all points in $C(T_i, T_j)$. The resulting
locked transactions are denoted $LT_i^j$ and $LT_j^i$, respectively.

*Step 4.* For each transaction $T_i$ and each pair of transactions $T_i$, $T_j$, $i \neq j$, merge
the locked transactions $LT_i^j$ to $LT_i$. $L_\tau$ is then the set of all such $LT_i$, $1 \leq i \leq d$.

Policy OL is safe and dominates policy 2PL. Safety follows since OL is based on
OL locking. Then let $L_\tau$ be any locked transaction system according to 2PL. We
will show that Algorithm OL can construct a locked transaction system $L'\tau$ which
dominates $L_\tau$. To this end choose in Step 1 as coordinates of the overlap point the
first unlock operation in each locked transaction in $L_\tau$. It follows that the overlap
points are contained in the forbidden regions of $L_\tau$. Then, in Step 3, realize forbid-
den regions in such a way, that each forbidden region of $L'\tau$ is contained in the cor-
responding forbidden region of $L_\tau$. Since the forbidden regions in $L_\tau$ contain all
direct conflict points and the respective overlap point, such forbidden regions
always exist for $L'\tau$.

Figure 11 shows a locked transaction system according to OL which dominates
the locked version according to 2PL in Fig. 8.
9. Conclusion

In this paper we have discussed locking policies which are based on a pairwise decomposition of the transaction system. The benefit of a pairwise decomposition is the possibility to apply two-dimensional geometric algorithms to find appropriate positions for the needed lock and unlock operations. According to the presented policy OL, a pre-analysis can be performed in a way such that any given two-phase locked transaction system is dominated. In [5] it is shown, that any safe locked transaction system using entity locks is also OL locked. Thus, OL locking is as least as powerful as safe entity locking.

We have conjectured that optimal locking with respect to the number of accepted schedules is an NP-complete problem, even in the case of two transactions. Therefore, optimal locking seems to be computationally prohibitive. Thus, there is the need to design good heuristics which can be implemented efficiently. We have sketched the topic of maximal concurrency for systems containing many transac-
Locking

If we have performed some kind of optimal locking for each transaction pair, what are the implications for the merged transaction system? There exist results with respect to the question whether or not the set of all serializable schedules can be realized by a certain locked transaction system [7]. But if we cannot, how should we apply locking to maximize the number of accepted schedules?

In this paper we have assumed that the set of all transactions is a priori known such that a pre-analysis can be applied. This assumption does not always hold in practice. Algorithms for the case where new transactions may enter the system dynamically while other transactions are already executed are proposed in [3,4]. However, the principle problem of optimal locking remains the same; once we have computed the relevant set of points to be included in a forbidden region, we have to find appropriate positions for lock and unlock operations which allow a high degree of potential concurrency as possible.

Acknowledgement

The current paper is based on joint work with Eljas Soisalon-Soininen and Peter Widmayer. The material concerning maximal concurrency control is taken from [7], the first pre-analysis algorithm has been presented in [6], and the conditions which can be used to derive locked transaction systems which dominate two-phase locked transaction systems are taken from [3,4].

References