WORKING PAPER
ALFRED P. SLOAN SCHOOL OF MANAGEMENT

TIME SLICE MANIPULATION
IN INFORMATION HIERARCHY

Meichun Hsu
Stuart E. Madnick

April 1987 #WP 1875-87

MASSACHUSETTS
INSTITUTE OF TECHNOLOGY
50 MEMORIAL DRIVE
CAMBRIDGE, MASSACHUSETTS 02139
TIME SLICE MANIPULATION
IN INFORMATION HIERARCHY

Meichun Hsu
Stuart E. Madnick

April 1987 #WP 1875-87
Time Slice Manipulation in Information Hierarchy

By

Meichun Hsu
Harvard University
Cambridge MA 02138

and

Stuart E. Madnick
Massachusetts Institute of Technology
Cambridge, MA 02139

Acknowledgement: Work reported herein has been supported, in part, by the Naval Electronic Systems Command through contracts N0039-83-c-0463 and N0039-85-c-0571.

ABSTRACT

In this paper we present theoretical results on a generalized form of partial order, called hierarchical partial order, for enforcing serializability that takes advantage of transaction analysis in database systems. Transaction analysis partitions the database into data partitions that may assume a hierarchy of priorities, such that transactions primarily updating less critical data partitions will not interfere with transactions primarily updating the more critical data partitions, or will do so to a lesser extent than those in conventional systems. This results from the ability of transactions in the system to access different data partitions using different synchronization protocols. The rules governing the different protocols are presented and their correctness with respect to serializability is proven.
1. Introduction

Conventional algorithms for database concurrency control ensure transaction serializability by forcing transaction dependencies to obey certain partial order. In the case of two-phase locking [Eswaran76], the partial order coincides with the order of transaction lock points. In the case of basic timestamping algorithm [Bernstein80, Reed78], the partial order coincides with the order of transaction timestamps, typically the transaction initiation times. Therefore, for example, if the basic timestamping algorithm is used, a transaction \( t \) with a timestamp \( TS_t \) is allowed to write a data element \( d \) only if \( d \) has not been read by any other transaction whose timestamp is greater than \( TS_t \). Conventional algorithms rigidly obey this chosen partial order which is assigned to transactions without much consideration for other factors, such as a priori knowledge of potential interferences among classes of transactions.

In this paper we present theoretical results on a more generalized form of partial order for enforcing serializability that takes advantage of transaction analysis. Consider a database application system with a database \( D \) partitioned into \( D_1 \) and \( D_2 \). Transactions are partitioned accordingly into classes \( T_1 \) and \( T_2 \), where transactions in \( T_1 \) primarily read and update data elements in \( D_1 \), and those in \( T_2 \) primarily update data elements in \( D_2 \), but also make frequent read accesses to data elements in \( D_1 \). Using conventional concurrency control algorithm, transactions in \( T_2 \), due to their read accesses to \( D_1 \), would interfere with concurrent transactions in \( T_1 \). However, if \( T_1 \) is considered a class of higher priority than \( T_2 \), this interference may be reduced or eliminated if we allow transactions in \( T_2 \) to use a slightly older "time slice" of \( D_1 \). In particular, assume that the system uses timestamps. If a transaction \( t \) in \( T_2 \), which has been assigned a timestamp \( TS_t \), uses a pseudo-timestamp \( TS_t- \) which is smaller than the timestamp of the oldest active transaction in \( T_1 \) at the time of access, it would not interfere at all with transactions in \( T_1 \). In addition, \( t \) may still be using \( TS_t \) to coordinate accesses to data elements in \( D_2 \).

In this example, data elements in \( D_1 \) can be considered raw data, while data elements in \( D_2 \) derived data. It appears that transactions responsible primarily for writing derived data (e.g.,
transactions in class $T_2$), or, in general, transactions of lower priorities, can be prevented from interfering with those processing the raw data, or, in general, those of higher priorities. The above example may be generalized to an application composed of more than one data partitions (and, therefore, more than 2 transaction classes), forming an information hierarchy. An information hierarchy can be represented as a directed acyclic graph (DAG), where nodes are data partitions and arcs represent the derivation path or priority ordering.

We examine the following question in this paper: given a transaction class $T_j$ which is primarily responsible for writing to data partition $D_j$, what are the rules it must follow in computing the time slice it uses for each data partition without compromising overall serializability of the system? Results are presented for the case in which the information hierarchy is a semi-tree, a restricted form of DAG.

Relevant work: Conflict analysis among transactions has been proposed in the research of SDD-1 [Bernstein80] as a vehicle to discover certain (static) conflict patterns among transaction classes that may enable a more flexible timestamp protocol (e.g., Protocol 1 in SDD-1's terminology) to be used. However the SDD-1 approach stops short of developing a generalized theory. Multi-version databases, conducive to the implementation of the notion of database time slices, has been shown to provide a higher level of concurrency than the conventional single-version ones [Bernstein83, Papadimitriou84]. A relevant multi-version algorithm has been presented in [Chan82, Chan85]. The results in this paper offer a fundamental proof method for verifying correctness of algorithms designed specifically for information hierarchies (e.g. methods proposed in [Hsu86] is a special case of scenarios supported by results in this paper). In comparison, the tree locking protocol [Silberschatz80, Kedem83] is a non-two-phase locking protocol which aims at reducing the amount of time the locks on the "high-level" nodes of a tree must be held by each transaction. The hierarchy used in their tree protocol is entirely different from the kind of information hierarchy the current paper is concerned about.
We now present the definition of the generalized partial order, called *hierarchical partial order*, followed by the interpretation and the proof of the *acyclicity theorem*, the major result of the paper.

2. The Hierarchical Partial Order of Transactions

The hierarchical partial order among transactions requires the decomposition of a database into a number of data partitions. We construct a data partition hierarchy which is basically a partial order of the data partitions subject to certain constraints.

2.1. Data Partition Hierarchy and Transaction Classification

*Definition.* Given a data decomposition $P$ of a database $D$ into data partitions $D_1, D_2, \ldots, D_n$, and a transaction analysis which partitions all potential update transactions in the database into a set $T_s$ of update transaction types, $TP_1, \ldots, TP_m$, a *data partition hierarchy*, denoted as $DPH(P, T_s)$, is any acyclic graph with nodes corresponding to $D_1, D_2, \ldots, D_n$, such that

[C1] it is a semi-tree

(a semi-tree is an acyclic digraph where there exists one and only one undirected path between any pair of nodes), and

[C2] if there exists a *type* of update transactions in the system which write in $D_i$ and read from or writes in $D_j$, then there exists a directed path between $D_i$ and $D_j$ in $DPH(P, T_s)$.

It is noted that only update transactions need participate in constraining the data partition hierarchy. *There is no need for read-only transactions to participate in the transaction analysis*, eliminating the difficulties of pinning down, a priori, the nature of all ad hoc queries.

There may be multiple data partition hierarchies that satisfy the above definition given a database decomposition. In particular, any total order of partitions in $P$ satisfies the definition. The actual choice of the data partition hierarchy will reflect the perceived priorities of the write-processing in each of the data partitions.
Given a data partition hierarchy, each transaction is assigned to one of the data partitions it writes into. Typically, the data partition chosen is the one in which the transaction performs all or most of its writes. This data partition is called the home data partition of the transaction, and all transactions with the same home data partition are grouped into a transaction class. From [C2] of the definition of data partition hierarchy, if a transaction's home data partition is $D_i$, denoted as $t \in D_i$, then there is a directed path between $D_i$ and any other data partition in DPH that the transaction accesses (read or write).

In the remainder of the paper, the notation $DPH$ refers to a particular data partition hierarchy chosen to base our hierarchical partial order of transactions. We say that data partition $D_i$ is higher than data partition $D_j$, denoted as $D_i > D_j$, if there exists a directed path in $DPH$ from $D_i$ to $D_j$. Intuitively, in our notation, higher level data partitions are most likely raw data from which the lower data partitions derive their contents. We say that $D_i$ and $D_j$ are related if either $D_i = D_j$ or $D_i$ and $D_j$ are connected by a directed path. We also say that $D_i$ and $D_j$ are neighbors if they reside on directed paths in $DPH$ that intersect. By definition, if $D_i$ and $D_j$ are related then they must be neighbors.

2.2. The $=>$ Relation

**Definition.** A relation "$=>", (pronounced as "L-follows"), is defined for a pair of transactions $t_1$, $t_2$ where $t_1 \in D_i$, $t_2 \in D_j$, and $D_i$ and $D_j$ are neighbors in DPH. Given two functions $TS$ and $L$, we say that $t_2 => t_1$ with respect to $TS$ and $L$, if there exists a $D_k$ in DPH such that $D_k$ is related to both $D_i$ and $D_j$, and

$$L_{i,k}(TS_{t_2}) > L_{i,k}(TS_{t_1}),$$

where $TS$ is a function which maps a transaction to a time value such that no two transactions have the same time value; the function $L_{i,j}$, which stands for link function, is defined for any pair of related data partitions $D_i$ and $D_j$ and maps one time value to another time value as follows:

1. If $D_i = D_j$ then $L_{i,j}(m) = m$;
(2) if \( D_i > D_j \) then \( L_{i,j}(m) = DN_{i,j}(m) \);

(3) if \( D_i < D_j \) then \( L_{i,j}(m) = UP_{i,j}(m) \);

where the function \( UP_{i,j} \), standing for \textit{upward function}, is defined for any pair of data partitions \( D_i \) and \( D_j \) where \( D_i < D_j \), and maps a time value to another time value; the function \( DN_{i,j} \), standing for \textit{downward function}, is defined for any pair of data partitions \( D_i \) and \( D_j \) where \( D_i > D_j \), and maps a time value to another time value; for any pair of \( D_i \) and \( D_j \) where \( D_i < D_j \), functions \( UP_{i,j} \) and \( DN_{j,i} \) must satisfy the following three properties:

\begin{enumerate}
\item \textbf{Composable:} for all \( D_i, D_k \) and \( D_j \) where \( D_j > D_k > D_i \), for all times \( m \),
\[ UP_{k,j}(UP_{i,k}(m)) = UP_{i,j}(m), \]
\[ DN_{k,i}(DN_{j,k}(m)) = DN_{j,i}(m). \]
\item \textbf{Non-decreasing:} for all \( D_i \) and \( D_j \) where \( D_j > D_i \) and for all times \( m, m' \) where \( m > m' \),
\[ UP_{i,j}(m) \geq UP_{i,j}(m'), \quad \text{and} \quad DN_{j,i}(m) \geq DN_{j,i}(m'). \]
\item \textbf{Value-interlocked:} for all \( D_i \) and \( D_j \) where \( D_j > D_i \) and for all time \( m \),
\[ UP_{i,j}(DN_{j,i}(m)) \leq m, \quad \text{and} \quad DN_{j,i}(UP_{i,j}(m)) \geq m. \]
\end{enumerate}

Intuitively, \( \Rightarrow \) is a relation between transactions based on both the timing of the transactions and the hierarchical levels in the DPH of the transaction classes that the transactions belong to. To be more specific, \( t_1 \Rightarrow t_2 \) always means that \( t_1 \) is "later" than \( t_2 \). However, this "later" is not only based on when the two transactions are physically active, but also on the relative levels of the data partitions in which \( t_1 \) and \( t_2 \) are assigned to. Clearly, \( \Rightarrow \) is defined only between transactions that belong to neighboring data partitions, and the \textit{UP} and \textit{DN} functions are defined only between related data partitions.

Note also that the functions \textit{TS}, \textit{UP} and \textit{DN} are not completely specified; only their necessary properties have been specified. This means that, by manipulating the instantiation of these functions, different instances of the relation \( \Rightarrow \) may be defined. For example, both the initiation timestamp function, which maps a transaction to its initiation time, and the commit timestamp function, which maps a transaction to its commit time, are acceptable instantiations of the \textit{TS} function. An example of a construction of \( UP_{i,j} \) and \( DN_{j,i} \) which satisfies the non-decreasing and
value-interlocking properties is $UP_{i,j}(m) = m - C_{ij}$ and $DN_{j,i}(m) = m + C_{ij}$ for an appropriate constant $C_{ij}$.

2.3. The Acyclicity Theorem

The key result of the paper is the following theorem:

*The Acyclicity Theorem.* A digraph where nodes are transactions and every arc $t_2 \rightarrow t_1$ implies $t_2 \Rightarrow t_1$ has no cycle.

The theorem states that if a concurrency control algorithm allows a transaction dependency $t_2 \rightarrow t_1$ to occur only when $t_2 \Rightarrow t_1$ holds, then serializability is guaranteed. Given the structure of an information hierarchy, and the desire to reduce or eliminate interferences from lower-priority transactions to higher-priority transactions, one needs to:

1. Define a TS function;
2. Construct a function UP which maps time values to successively smaller values along the hierarchical path upwards in DPH, and a DN which maps time values to successively larger values downwards, where UP and DN also satisfy the compositibility, non-decreasing and value-interlocking requirements;
3. Allow lower-level transactions ($t \in D_i$) to access an older time slice before $UP_{i,j}(TS_i)$ in a higher data partition $D_j$; and
4. Allow higher-level transactions ($t \in D_j$) to access and timestamp a lower-level data partition ($D_i$) with the time value $DN_{j,i}(TS_i)$.

It is noted that when DPH consists of a single data partition, or when $UP$ and $DN$ are assigned identity functions, $\Rightarrow$ degenerates to the partial order typically enforced in conventional algorithms.
2.4. Proof

To prove this theorem, we will first define a weaker relation \( \approx > \), (pronounced as "weakly L-follows"), such that \( = > \) implies \( \approx > \) and \( t_2 \approx > t_1 \) implies \( \neg (t_1 = > t_2) \). We prove that \( \approx > \) is locally transitive, i.e., if \( t_3 \approx > t_2 \) and \( t_2 \approx > t_1 \) and \( t_1, t_2 \) and \( t_3 \) are neighbors of one another then \( t_3 \approx > t_1 \). Local transitivity completes the proof of the Acyclicity Theorem for a data partition hierarchy in which all data partitions are neighbors of one another. Finally we extend the transitivity result to show global transitivity. (In the following proofs, for notational convenience, we denote the home data partition of a transaction \( t_i \) as \( D_i \).)

Definition. Given the definition of \( = > \), we say that \( t_2 \) weakly \( = > \) \( t_1 \), denoted as \( t_2 \approx > t_1 \), if (a) if \( D_1 \) and \( D_2 \) are related then for all \( D_m \) on the shortest path in DPH between \( D_1 \) and \( D_2 \) inclusive, \( L_{2,m}(TS_{t_2}) \geq L_{1,m}(TS_{t_1}) \) \( [0.1] \), and (b) if \( D_1 \) and \( D_2 \) are not related but the shortest path between them turns direction at \( D_m \), then for all \( D_p \) on the shortest path in DPH between \( D_2 \) and \( D_m \) inclusive, \( L_{2,p}(TS_{t_2}) \geq L_{1,m}(TS_{t_1}) \) \( [0.2] \), and for all \( D_i \) on the shortest path in DPH between \( D_1 \) and \( D_m \) inclusive, \( L_{2,m}(LS_{t_2}) \geq L_{1,i}(LS_{t_1}) \) \( [0.3] \). (Note that by the fact that DPH is a semi-tree there exists one and only one shortest path between any pair of data partitions and by definition of neighbors the shortest path between any neighboring data partitions can turn direction at most once, i.e., the path between any pair of neighboring data partitions is either one-phase or two-phase.)

Lemma 1. \( t_2 = > t_1 \) implies \( t_2 \approx > t_1 \).

Proof. We want to show that (a) \( [0.1] \) in the above definition of \( \approx > \) is true, and (b) \( [0.2] \) and \( [0.3] \) in the definition of \( \approx > \) is true. Since \( t_2 = > t_1 \), let \( L_{2,i}(TS_{t_2}) > L_{1,i}(TS_{t_1}) \) for some \( D_i \) related to both \( D_1 \) and \( D_2 \) \( [1.1] \).

(a) Suppose \( [0.1] \) is not true. Then there exists \( D_k \) on the shortest path between \( D_1 \) and \( D_2 \) inclusive such that \( L_{2,k}(TS_{t_2}) < L_{1,k}(TS_{t_1}) \) \( [2.1] \). (For brevity, "inclusive" is always implied from now on.) Consider the following two cases: (a.1) \( D_1 \geq D_2 \). Let \( D_p \) be any data partition on the
shortest path in DPH between \( D_1 \) and \( D_k \). Then applying \( UP_{k,p} \) to both sides of [2.1] and making use of the properties of \( UP \) and \( DN \) functions we have \( UP_{2,p}(TS_{t_2}) \leq DN_{1,p}(TS_{t_1}) \). Similarly let \( D_q \) be any data partition on the shortest path in DPH between \( D_2 \) and \( D_k \). By applying \( DN_{k,q} \) on both sides of [2.1] we have \( UP_{2,q}(TS_{t_2}) \leq DN_{1,q}(TS_{t_1}) \). Therefore if [2.1] were true then for all \( D_m \) in between \( D_1 \) and \( D_2 \) we have \( UP_{2,m}(TS_{t_2}) \leq DN_{1,m}(TS_{t_1}) \). However, if this were true then there cannot exist any \( D_i \) satisfying [1.1], contradictory. Therefore [2.1] cannot be true. (a.2) \( D_2 \geq D_1 \).

Then for all \( D_p \) s.t. \( D_2 \geq D_p \geq D_k \) we have \( DN_{2,p}(TS_{t_2}) \leq UP_{1,p}(TS_{t_1}) \), since if this were not true then we get \( DN_{2,q}(TS_{t_2}) \geq UP_{1,q}(TS_{t_1}) \), contradictory to [2.1]. Similarly for all \( D_q \) s.t. \( D_k \geq D_q \geq D_1 \) we have \( DN_{2,q}(TS_{t_2}) \leq UP_{1,q}(TS_{t_1}) \). Using same argument in (a.1) we obtain contradiction to [1.1] and therefore [2.1] cannot be true. Combining (a.1) and (a.2) we conclude [0.1] is true.

(b) We have \( L_{2,m}(TS_{t_2}) > L_{1,m}(TS_{t_1}) \) [3.1] since if this were not true there cannot exist \( D_i \) such that [1.1] is true. Consider two cases: (b.1) \( D_m > D_1, D_2 \). Applying \( DN_{m,q} \) to both sides of [3.1] we have \( DN_{m,q}(L_{2,m}(TS_{t_2})) \geq UP_{1,q}(TS_{t_1}) \), i.e., \( L_{m,q}(L_{2,m}(TS_{t_2})) \geq L_{1,q}(TS_{t_1}) \), therefore [0.3] is true. Suppose [0.2] were not true. Then we have some \( D_p \) such that \( L_{2,p}(TS_{t_2}) < L_{m,p}(L_{1,m}(TS_{t_1})) \) [3.2]. Applying \( UP_{p,m} \) to both sides of [3.2] we have \( UP_{2,m}(TS_{t_2}) \leq L_{1,m}(TS_{t_1}) \), contradictory to [3.1]. Therefore [0.2] must be true. So we have both [0.2] and [0.3] hold. (b.2) \( D_1, D_2 \geq D_m \).

Using similar arguments as presented in (b.1) one can show [0.2] and [0.3] hold. Combining (b.1) and (b.2) we conclude [0.2] and [0.3] true.

**Lemma 2.** \( t_2 \approx > t_1 \) implies \( \neg(t_1 = > t_2) \).

**Proof.** From Lemma 1, it is clear that if \( t_2 \approx > t_1 \) then for all \( D_k \) related to both \( D_1 \) and \( D_2 \)

\[ L_{2,k}(TS_{t_2}) \geq L_{1,k}(TS_{t_1}) \]

Therefore \( \neg(t_1 = > t_2) \).

**Lemma 3.** (Local Transitivity) The relation \( \approx > \) is transitive, i.e., if there exists \( t_1 \in D_1, t_2 \in D_2, t_3 \in D_3 \), such that \( t_2 \approx > t_1, t_3 \approx > t_2 \) and \( D_1, D_2 \) and \( D_3 \) are neighbors of one another, then \( t_3 \approx > t_1 \).
Proof. Since $D_1$, $D_2$, and $D_3$ are neighbors of one another, let the shortest path between $D_1$ and $D_3$ contains $D_m$, where $D_m$ is on the path between $D_1$ and $D_2$ and $D_2$ and $D_3$. Then every $D_p$ between $D_3$ and $D_m$ is also on that between $D_3$ and $D_2$, and every $D_q$ between $D_1$ and $D_m$ is also on that between $D_2$ and $D_1$. For $D_i$ and $D_j$ neighbors, we denote as $LL_{i,j}(z)$ either $L_{i,j}(z)$ if $D_i$ and $D_j$ are related, or $L_{i,j}(L_{i,k}(z))$ if $D_i$ and $D_j$ are not related but the shortest path between them turns at $D_k$. Then by $t_2 > t_1$, $t_3 > t_2$ we have $LL_{3,m}(TS_{i,y}) > LL_{2,m}(TS_{i,y}) > LL_{1,m}(TS_{i,y})$. Therefore $LL_{3,p}(TS_{i,y}) > LL_{2,p}(TS_{i,y}) = LL_{m,p}(LL_{2,m}(TS_{i,y})) > LL_{m,p}(LL_{1,m}(TS_{i,y})) = L_{1,p}(TS_{i,y})$. Similarly we derive $LL_{3,4}(TS_{i,y}) > LL_{1,4}(TS_{i,y})$. Therefore for all $D_1$ in between $D_1$ and $D_3$ we have $LL_{3,4}(TS_{i,y}) > LL_{1,4}(TS_{i,y})$. Therefore $t_3 > t_1$.

Next we extend local transitivity to allow for more general data partition hierarchy.

**Lemma 4.** Given a DPH, if $t_i = \cdots = t_e$, denoted as $LF(t_i,t_e)$, and $D_i$ and $D_j$ are neighbors, then $t_i > t_e$.

**Proof.** We prove by induction in the length $l$ (i.e., number of arcs) in $LF(t_i,t_e)$. (a) If $l = 2$, then $t_i > t_e$ by local transitivity. (b) Show that if $t_i > t_j$ for any $LF(t_i,t_j)$ whose length is less than $g$ and $D_i$ and $D_j$ are neighbors. then $t_i > t_{i+1}$ for any $LF(t_i,t_{i+1}) = \cdots > t_{i+1}$ whose length is $g$ and $D_i$ and $D_{i+1}$ are neighbors. Consider two subcases: (b.1) If $D_i$ and $D_j$ are neighbors, then $t_i > t_j$. Therefore $t_i > t_{i+1}$. (b.2) If $D_i$ and $D_j$ are not neighbors, then since they have a common neighbor $D_{i+1}$, there exists $t_k$ such that $LF(t_k,t_j) = t_k \cdots > t_{i+1}$ where $D_k$ are neighbors of both $D_i$ and $D_{i+1}$. Since $LF(t_k,t_j)$ has length less than $g$, we have $t_k > t_{i+1}$. Likewise we have $t_k > t_{i+1}$. By local transitivity, we have $t_i > t_{i+1}$. Q.E.D.

**Proof of Acyclicity Theorem.** Suppose there is a transaction dependency cycle $t_1 \cdots \rightarrow t_n \rightarrow t_1$. Then from Lemma 4, we have $t_i > t_n$, and therefore, from Lemma 2, we have $-(t_n > t_1)$. This means that there cannot be a transaction dependency $t_n \rightarrow t_1$, contradictory with the given. Therefore there cannot be a cycle. Q.E.D.
3. References


