Triggered Real-Time Databases with Consistency Constraints *

Henry F. Korth Nandit Soparkar Abraham Silberschatz

Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712-1188 USA

Abstract

Real-time database systems incorporate the notion of a *deadline* into the database system model. Usually, deadlines are associated with transactions, and the system attempts to execute a given set of transactions so as to both meet the deadlines and ensure the database consistency. This paper presents an alternative model of real-time database processing in which deadlines are associated with consistency constraints rather than directly with transactions. This model leads to a predicate-based approach to transaction management that allows greater concurrency and more flexibility in modeling real-world systems.

1 Introduction

Real-time database systems (RTDBs) incorporate timing considerations into a database system. Not only must the transactions execute correctly, but also, they must complete execution within some time limit called a *deadline*. Systems that incorporate strict deadlines are called *hard* real-time systems while those that do not are called *soft* real-time systems.

Real-time systems are usually applied for process-control which often require a large database of information. Hence, recent efforts have aimed at integrating the real-time systems with database systems to facilitate the efficient and correct management of the resulting *real-time database* systems [18]. There are several difficulties in accomplishing such an integration. A database operation (read or write) takes a highly variable amount of time depending on whether disk I/O, logging, etc. are required. Furthermore, if concurrent transactions are allowed, the concurrency control may cause aborts or delays of indeterminate length.

Most previous work on real-time transactions assumes a set of transactions and associated deadlines. It is the responsibility of the transaction manager to find a correct schedule for the transactions that will ensure that the deadlines are met.

There has been extensive study of real-time systems [19]. Formal aspects of such systems have been examined from the standpoints of scheduling (e.g., [8]) and verification [9]. In the context of real-time databases, [1, 2] consider alternative queuing disciplines with lock-based concurrency control of real-time transactions, and use simulation results to compare these techniques. [16] proposes concurrency control techniques for distributed real-time systems based on a partitioning of data. [15] discusses the specific time-dependent application of stock-market trading.

The RTDB models outlined above apply time-constraints directly to transactions, but they do not model situations where the time-constraints apply directly to states of the systems. Time-constraints on the states of the system enforce similar time-constraints on transactions that are triggered by those states. As an example, consider an RTDB application in a manufacturing environment. Suppose that the state of the information maintained in the database indicates that the temperature in a furnace has fallen below a particular threshold value. This state of the system may necessitate the triggering of some actions that restore the temperature to a value above the threshold. The application may enforce a maximum period of time for which the temperature is permitted to remain below the threshold — and that enforces a deadline on the actions triggered by the low value. Furthermore, it may be the case that several actions may be candidates for the restoration of the temperature. For instance,

*Research partially supported by TARP grant 4355, NSF grant IRI-8805215, and a grant from the IBM Corporation.

Proceedings of the 16th VLDB Conference
Brisbane, Australia 1990

there may be actions that initiate more fuel getting pumped-in, or actions that increase the oxygen supply, etc. Thus, a choice may be available, and depending on the time constraints (and other factors such as the cost of the actions), one particular action may be initiated to restore the temperature value. These actions are reflected as triggered transactions within the database.

In this paper, we propose a new approach to the modeling of an RTDB. Our approach is based on a set of explicitly defined consistency constraints for the database. Each transaction ensures that upon completion, the database remains in a state that satisfies these consistency constraints. However, in addition to such transactions that maintain correct database states, transactions may be invoked to record the effects of some external event that is generated outside the system. The ensuing change in the database state may render a consistency constraint invalid, and that constraint may need to be restored within a specific deadline. The system restores constraints by choosing one or more transactions from a pre-defined library of transactions. These transactions restore certain constraints but may invalidate other constraints. In the absence of further external events, the system must eventually return the entire database to a consistent state. In a dynamic real-time system, external events may occur with sufficient frequency to prevent global consistency, but the system must seek to ensure that no constraint remains invalid for an interval longer than a specified limit, the *deadline* of the constraint.

2 Transaction Model

In this section, we give an informal characterization of real-time transactions and relate this to other work on extended transaction models. In Section 3, we present a formal model for reasoning about transactions and constraints.

A real-time transaction system interacts with the external world in several ways. Events in the external world are recorded in the database. Transactions in the transaction system initiate external actions. This leads us to partition the set of transactions in a real-time system into three categories:

1. **External-input transactions.** Such a transaction records in the database some event that has occurred in the external world. Often, such a transaction is a write-only transaction, and is usually of short-duration.

2. **Internal transactions.** Such a transaction accesses the database in a similar manner as any standard database transaction except that it may be of long-duration. The purpose of this type of transaction is the restoration of "consistency" that may have been violated as a result of some external-input transaction.

3. **External-output transactions.** Such a transaction causes some event to occur in the world external to the system. These transactions are of short-duration from a system perspective, although the external actions they trigger may take a longer time to complete. We do not permit external-output transactions to wait for the acknowledgement of completion of the external activity. Instead, we treat transactions of this type as performing only the initiation. If further action is to be taken as a result of completion of the external activity, another transaction (an external-input transaction) must record in the database the completion of the external activity, which then triggers the execution of further internal or external-output transactions.

These three types of transactions differ in their atomicity and concurrency requirements. A write-only external-input transaction should never wait. Its writes should succeed immediately unless a "newer" value has already been recorded in the database. These requirements are justified since the external-input transactions are used to record the outside world within the system. In a real-time application, such events need to be recorded in the database as soon as possible so that any resulting inconsistency may be corrected. A consequence of this is that it may not be desirable to ensure serializable executions even if multiple versions of data are retained.

The notion of transactions violating the database consistency and other transactions reading possibly inconsistent database states is a major deviation from the standard transaction model. We represent such actions using the NT/PV model of [12] by defining input and output conditions for each transaction. These conditions are predicates on the database state. The input condition is a pre-condition of transaction execution and must hold on the state that the transaction "observes." The output condition is a post-condition which the transaction guarantees on the database state at the end of the transaction provided that there is no concurrency and the database state initially seen by the transaction satisfies the input condition. Thus, in the NT/PV model, as in the standard model, transactions are assumed to be correct programs, and responsibility for correct concurrent execution lies with the transaction manager.

The actions required to restore consistency may involve more than direct database access. Internal trans-

actions spawn external-output transactions as *subtransactions* to modify the outside world as part of a process of restoring consistency. Other subtransactions may be required to test the results of external-output transactions. The potential long-duration of internal transactions make a requirement of serializability impractical [10]. Furthermore, the nested nature of these transactions requires an extension of the transaction model to support *nested transactions* [14]. A serializability-based approach to nested transactions is discussed in [14] while correctness of nested transactions without the requirement of serializability is presented in [3, 7, 12].

The use of multiple versions of data is often indicated in real-time database applications. An obvious utility is in situations which require the monitoring of data as it assumes different values in time; that is, the "trends" exhibited by the values of the data are used to trigger actions. Examples include rising temperature of a furnace in a nuclear application and the change in the distance of an approaching aircraft in radar tracking systems.

The above considerations lead us to suggest that our real-time transaction model may include (1) nesting, (2) versions, and (3) correct concurrent execution without the requirement of traditional serializability. We use the NT/PV model of [12] as the basis for our work since this model supports the above features.

Transactions in real-time systems may be submitted either by users, or by external devices. In addition, transactions may also be triggered by the state of the system. If an external-input transaction changes the database state to an inconsistent state, an internal transaction must be run to restore consistency. These transactions are not necessarily triggered by an external-input transaction. Rather they may depend on both the external-input transaction and the database state. For a given inconsistent state, there may be several transactions that are enabled for triggering. The transaction system is free to choose a subset of those transactions that are enabled provided that subset is sufficient to restore consistency. This choice is, in its most general form, computationally complex. We explore this idea further in Section 4. Our model of triggered transactions is related to that used in *active* databases [13]. However, for the purposes of this paper, the manner in which transactions are selected for execution differs from active databases in that we base the selection on the goal of consistency restoration.

The concept of triggered transactions, along with our characterization of real-time transactions above, provides five types of transactions: (1) external-input transactions (non-triggered by definition), (2) triggered internal transactions, (3) non-triggered internal transactions, (4) triggered external-output transactions, and (5) non-triggered external-output transactions.

The system model we consider in this paper may be regarded as comprising of a set $T = \{t_1, t_2, \ldots, t_n\}$ of predefined *transaction-types*, and a finite set $C = \{c_1, c_2, \ldots, c_m\}$ of predefined consistency constraints in the form of *conjuncts*. Conjuncts are formulae consisting of a disjunction of possibly negated terms. The consistency constraint for the entire database may be represented by $P \equiv \bigwedge_{i=1}^{m} c_i$. As far as the consistency constraints are concerned, for the purposes of this paper, we restrict our attention to predicate calculus rather than first-order logic since all quantifiers will be over a finite set (the database). Some instances of the transaction-types are triggered by the falsehood of a conjunct, and may function to restore the truth of the conjunct. Certain instances of the transaction-types, upon execution, may render inconsistent some conjuncts. Thus, the system may be regarded as consisting of transaction-types and conjuncts that interact with each other.

3 Predicate-Priority Graph

To facilitate the description of our model, and to make the algorithmic analyses easier, we define a *predicate-priority graph* (PPG). The PPG captures the relationships between the transaction-types and the conjuncts, and its annotations are used to incorporate various timing constraints. A PPG is a directed bipartite graph with a set of vertices $V = T \cup C$, where T denotes the set of transaction-types, and C denotes the set of conjunct vertices.

The edges in a PPG represent the triggering of transaction-types by the falsehoods of the conjuncts, and the invalidation of conjuncts by the transaction-types. If an instance of a transaction-type t_i may invalidate a conjunct c_j, then the directed edge (t_i, c_j) appears in the graph. If a transaction-type t_k ensures the truth of a conjunct c_l upon completion, then the directed edge (c_l, t_k) appears in the graph. Thus, the PPG represents the transaction-types available to the system for restoring consistency.

We exclude non-triggered transactions in order that the PPG may be a static structure. The only dynamic aspect to this graph will be the *markings* introduced below. The term *transaction-type* was used above to emphasize that we are creating a vertex for each type of triggered transaction, not a vertex for each execution of a specific transaction. If we had a vertex for each actual execution, then the PPG would become a dynamic structure. In this paper, we restrict attention to only static PPGs. The reason is that the static sit-

uation is a subcase of the dynamic one, and hence, it indicates some problems that may be encountered in the analyses of the more general situation. As we shall see, the analysis of the static PPG itself reveals several computationally intractable problems that indicate the need for heuristic approaches — and these results also apply to the dynamic PPG.

An example of a PPG is shown in Figure 1. Transaction-types are represented by square vertices, and the round vertices correspond to conjunct vertices. In the example, an inconsistency in conjunct c_1 may be resolved by executing an instance of either one of the transaction-types t_1 or t_2. Furthermore, the execution of a transaction of type t_1 may result in the invalidation of the conjuncts c_5, c_6 and c_7.

Let us now examine how the PPG is used. If the database is inconsistent, the vertices corresponding to the false conjuncts are *marked*. To restore consistency, it is necessary to run an instance of the transaction-type associated with the head of at least one out-edge of each marked vertex. However, running these transactions may lead to side-effects beyond restoring the truth of certain previously-false conjuncts. Possibly, these side effects will result in other conjuncts becoming false. This results in further marked vertices. It is important to note that, given a graph and a set of marked vertices, there may exist many ways to resolve the inconsistencies. For each marked vertex, the out-degree indicates the number of potential options for restoring the truth of the corresponding conjunct.

If a vertex corresponding to a conjunct is a sink (has no out-edges), then there is no way to restore the truth of this conjunct within the system. A non-triggered transaction (either an external-input or a non-triggered internal transaction) is required to restore the truth of this conjunct. Such a situation requires either human intervention or a "lucky" turn of events external to the system. Thus, we require that all sinks correspond to transaction-types.

A cycle in the PPG represents a potentially unstable situation. The situation is only *potentially* unstable, since an edge from a transaction-type vertex to a conjunct vertex means only that an instance of the transaction-type *may* make the conjunct false. Also, if in restoring the truth of a conjunct, a transaction-type vertex that is not within the cycle is chosen, the situation may not be unstable.

A safe strategy (i.e., one that is not potentially unstable) for resolving an inconsistent database state can be represented by an acyclic subgraph of the PPG such that the subgraph contains all the marked conjunct vertices of the PPG, retains all outedges in the PPG of transaction-type vertices in the subgraph, and retains at least one outedge of each conjunct vertex in the subgraph. We shall restrict attention to strategies that are not potentially unstable.

As an example, consider the PPG of Figure 1 again. The subgraph shown within the dotted outline in the figure provides a strategy to resolve the inconsistencies if c_1 and c_2 (and possibly any or all of c_5, c_6, and c_7) are the only marked vertices.

We consider sub-DAGs of the PPG that resolve an inconsistent database to have roots at all marked vertices and sinks that are transaction-types. As before, all outedges in the PPG from transaction-types in such a DAG must be included in the DAG. The partial order on transaction-types induced by the DAG must be observed if the execution will, in fact, restore consistency (without requiring the execution of multiple instances of a transaction-type).

As described above, a DAG subgraph may be identified in a marked PPG so as to resolve the inconsistencies. The subgraph should include all the marked vertices, and all the sinks should correspond to transaction-types with no outgoing edges. We call such a subgraph an *inconsistency-resolution subgraph* (IRS) for a given marked PPG. An IRS provides a strategy by which the inconsistencies in the PPG may be resolved: Executions of the transactions within an IRS that obey the partial order imposed by the IRS will resolve the inconsistencies.

A more formal definition a PPG and an IRS is now provided.

Definition 1. *A* predicate-priority graph *is a 3-tuple* (C, T, E) *representing a bipartite graph with vertex set* $C \cup T$ *and edge set* $E \subseteq ((C \times T) \cup (T \times C))$. *A* marked PPG *is a PPG in which a nonempty set of vertices* $X \subseteq C$ *is identified as being "marked".* □

The inconsistency-resolution subgraph (IRS) defined below represents a strategy for restoring consistency to the database given that the marked set of conjuncts are false.

Definition 2. *Let* $G = (C, T, E)$ *be a PPG in which the vertices in* $X \subseteq C$ *are marked. An* inconsistency-resolution subgraph *of* G *is a 3-tuple* $G' = (C', T', E')$ *such that,*
(1) $X \subseteq C' \subseteq C$, $T' \subseteq T$, *and* $E' \subseteq E$,
(2) For all edges $(t_j, c_i) \in E$ *such that* $t_j \in T'$, *we have* $c_i \in C'$ *and* $(t_j, c_i) \in E'$,
(3) For all $c_i \in C'$, *there exists a path in* G' *from* c_i *to* t_k, *where* t_k *is a sink in* G, *and*
(4) G' *is acyclic.* □

A natural question arises as to whether an IRS exists for a particular marked PPG. The following result implies that the question is easily settled.

Theorem 1. *Let G be a marked PPG. The problem of deciding whether there is an IRS G' for G is solvable in polynomial-time.*

Proof Sketch. We provide a sketch of a requisite polynomial-time algorithm that manipulates the PPG, G. For the ease of presentation, introduce a (pseudo) transaction-type vertex, $t' \in T$, with out-edges (t', c_i) for every $c_i \in X$, and a (pseudo) conjunct vertex, $c' \in C$, with an out-edge (c', t').

1. **while** $c' \in C$ **do**
   (a) Choose a sink transaction-type vertex, t_j. If none exists, **print** "No IRS exists", and **stop**.
   (b) For each conjunct vertex c_i such that $(c_i, t_j) \in E$, delete all edges that are adjacent to c_i. Hence, delete c_i.
   (c) Delete t_j.
   endwhile
2. **print** "IRS exists", and **stop**.

With appropriate data structures, the algorithm takes $O(|C| + |T| + |E|)$ time, and since it finds a way to resolve every vertex in X, it places the problem in polynomial-time. □

4 Timing Considerations

Timing constraints are represented in the PPG by associating a *time interval* with each conjunct and a *time cost* with each transaction-type. The value associated with each conjunct represents the maximum duration of a time interval during which the corresponding conjunct may be false. The time cost represents an estimate of the execution time of the transaction-type. Typically, real-time analysis is based upon worst-case assumptions about execution time so as to ensure the correctness of a schedule. If we took that approach to real-time database management, we would be forced to make drastic assumptions about page-fault frequency, delays due to concurrency control requirements, and other resource-contention factors. For example, unless detailed information about the physical-level schema is made available to the real-time system, it is necessary to assume that every data item reference incurs a page fault, consisting of the write of a page frame back to disk, the reading of the data page, plus requisite disk access to support write-ahead logging and index page access. The difference between the worst case and the expected case is so large that a worst-case analysis for real-time database transactions would find a solution only for systems that have an economically unjustifiable amount of redundant computing power. Therefore, although the *formal* model developed in this paper is independent of the determination of the execution times, for the purposes of this paper, we consider the expected-case estimates of the transaction-type execution times. Indeed, if the database is entirely memory-resident (see, e.g., [17]), the differences between the worst-case and expected time estimates are likely to be negligible. In fact, most real-time applications have memory-resident data.

The incorporation of time into our model is achieved by the use the functions W_κ and W_τ which denote mappings from the conjuncts C and the transaction-types T, respectively, to the set of non-negative integers. This requires the redefinition of the PPG to incorporate the timing constraints. We term this new PPG as a *weighted* PPG, while the original PPG is termed an *unweighted* PPG. These terms will be used in case of ambiguity in referring to the different types of the PPGs.

Definition 3. *A* (weighted) predicate-priority graph *(PPG) is a 5-tuple $(C, T, E, W_\kappa, W_\tau)$ representing a bipartite graph with vertex set $C \cup T$ and edge set $E \subseteq ((C \times T) \cup (T \times C))$. W_κ and W_τ are the time interval and time cost functions, respectively.*

Notice that an unweighted PPG can be represented by a PPG in which W_τ maps all elements of T to 1, and W_κ maps all elements of C to l (where l is suitably chosen). Also, we can extend the notion of a marked unweighted PPG to a marked weighted PPG in a natural manner. Note that the case where $W_\kappa(c_i) < W_\tau(t_j)$ for a conjunct c_i and a transaction-type t_j, it is not worthwhile to include an edge (c_i, t_j) in the PPG. Hence, we shall always assume that for an edge (c_i, t_j) in a PPG, it is always the case that $W_\kappa(c_i) \geq W_\tau(t_j)$.

We also need to redefine the inconsistency-resolution subgraph for a weighted PPG. Again, the IRS represents a strategy for restoring consistency to the database given that the marked set of conjuncts are false.

Definition 4. *Let $G = (C, T, E, W_\kappa, W_\tau)$ be a weighted PPG in which the vertices $X \subseteq C$ are marked. An* inconsistency-resolution subgraph *of G is a 5-tuple $G' = (C', T', E', W'_\kappa, W'_\tau)$ such that,*
(1) $X \subseteq C' \subseteq C$, $T' \subseteq T$, and $E' \subseteq E$,
(2) For all edges $(t_j, c_i) \in E$ such that $t_j \in T'$, we have $c_i \in C'$ and $(t_j, c_i) \in E'$,
(3) For all $c_i \in C'$, there exists a path in G' from c_i to t_k, where t_k is a sink in G,
(4) G' is acyclic,
(5) W'_κ is the restriction of W_κ to C', and W'_τ is the

restriction of W_T to T', and
(6) For all $c_i \in C'$, there is an edge $(c_i, t_j) \in E'$ such that $W_T(t_j) \leq W_\kappa(c_i)$. □

An IRS can be used to decide how to resolve the inconsistencies, and it must ensure that each conjunct is false for a period no longer than its time interval, on the assumption that time costs for transaction-types are accurate. There may exist several DAGs that may be used for a particular marked PPG and each represents an IRS as defined above. In this case, a decision needs to be made as to which particular one is to be chosen. Intuitively, the IRS that represents the best strategy to resolve the inconsistencies should be the one that is selected. Although the precise definition of a good IRS is dependent on the application, it is possible to identify certain important traits that an IRS should possess. For example, an IRS that provides a method to restore consistency promptly should be regarded as being better than one that implies a slower method. Concurrency aspects for running the restoring transaction-types need to be considered to achieve this. Another measure of goodness could be the choice of an IRS that renders the least number of consistency constraints false. A third measure of goodness arises from the potential inaccuracy in time costs for transaction-types. This measure is related to the scheduling of transactions with regard to the available *slack time* which, in the case of a transaction-type t_j that is chosen to resolve the inconsistency in a conjunct c_i, is $W_\kappa(c_i) - W_T(t_j)$. Sufficiently large slack times "absorb" the inaccuracies of the time estimates for transactions that are scheduled sufficiently early, and this is further discussed in Section 5. Therefore, we suggest that the goodness of an IRS may also be measured as a function of the amount of slack time left for the restoration of the truth of conjuncts. The nature of this function is application-dependent. Example functions include the total slack time, the geometric mean of slack times, and the minimum of the slack times for each conjunct.

The conjunct-based model of real-time transactions represented by the PPG provides the system with additional degrees of freedom in managing a real-time database. Not only can the concurrency and recovery managers take into account the conjunct deadlines and time costs associated with the transaction-types, but also the system has some choice among the set of transaction-types to use in response to a particular collection of violated conjuncts that arise due to external events. Below, we consider the computational complexity of taking optimal advantage of these degrees of freedom.

5 Selecting an IRS

The PPG and the IRS defined above allow us to pose several important questions regarding the algorithms that will use them. The issues related to a PPG and an IRS are two-fold. First, an efficient selection procedure is needed to identify a good IRS, where goodness is related to how profitably the IRS can be used to resolve the inconsistencies within the deadlines imposed. Second, once the IRS has been identified, efficient approaches are needed to execute the actions of the transaction-types specified by the IRS. For the time being, let us disregard the effects of the PPG-imposed partial ordering among the transaction-types and concurrency control issues.

5.1 Selection Based on Weights

Consider an unweighted, acyclic PPG, G. We may assume that the selection criterion for an IRS is obtaining one that includes the fewest number of transaction-type vertices.

Problem 1. *(TUAP) The Transaction-weight Problem for a marked, unweighted, acyclic PPG is: Given a marked, unweighted, acyclic PPG, G, and an integer K, is there an IRS, G', such that the number of elements in T' is at most K?* □

Theorem 2. *The TUAP problem is NP-complete.*

Proof Sketch. The proof of NP-easiness is as follows. We demonstrate how to verify in polynomial-time that a non-deterministically selected graph G' is an IRS with $|T'| \leq K$. Verifying that G' represents an IRS is accomplished by checking that $X \subseteq C'$, and that for every $c_i \in C'$, there exists an edge $(c_i, t_j) \in E'$. Checking that $|T'| \leq K$ completes the verification.

We now prove NP-hardness. An instance of the NP-complete Satisfiability problem (L01 in [5]) is reduced to the TUAP problem. Let P represent the conjunction of m clauses in L01, i.e., $P \equiv \bigwedge_{i=1}^{m} C_i$ where the clauses are formed over n boolean variables $x_1, x_2, \ldots, x_n$. As shown in Figure 2, form an instance of a PPG, G, with $C = \{p, c, c_1, c_2, \ldots, c_m, x_1, x_2, \ldots, x_n\}$, $T = \{p', c', Fx_1, Fx_2, \ldots, Fx_n, Tx_1, Tx_2, \ldots, Tx_n\}$, and $X = \{p\}$. Besides the edges explicitly shown in Figure 2, G includes an out-edge from a vertex c_i to either Tx_j or to Fx_j for every positive or negative literal, respectively, formed using an x_j occurring in the clause C_i of the Satisfiability problem instance. We prove that P is a satisfiable instance of L01 if and only if G contains an IRS, G', with $|T'| \leq (n+2)$. Note that the construction guarantees the existence of an IRS.

Assume that a requisite IRS, G', exists. $|T'| \geq (n+2)$ since included in G' are p', c', and at least one of Tx_i or

Fx_i for every x_i. Since G' is a requisite IRS, we have $|T'| = (n+2)$. This implies that exactly one of the vertices reachable from a vertex x_i is included in T'. Assign a boolean value of **T** or **F** to the corresponding variable x_i in L01 according as Tx_i or Fx_i is included, respectively, in T'. It is clear that every clause of L01 will have one satisfied literal by this assignment.

If there is a truth assignment for every x_i in the problem instance of L01 that satisfies P, consider a subgraph G' as described next. The set T' consists of p', c', and Tx_i or Fx_i according as x_i is assigned **T** or **F**, and the set $C' = C$. The subgraph G' contains all possible edges of G. It is easy to see that G' is an IRS with $|T'| \leq (n+2)$. □

If we introduce the timing constraints in terms of the functions W_κ and W_τ, a selection criterion for an IRS could be the minimization of the sum of the time costs of the transaction type vertices included in the IRS. This criterion is suggested by the need for the "fastest" inconsistency-resolution strategy.

Problem 2. *(TWP) The Transaction-weight Problem for a marked, weighted, acyclic PPG is: Given a marked, weighted PPG, G, and an integer K, is there an IRS, G', such that the sum of the weights of the elements in T' is at most K?* □

Theorem 3. *The TWP problem is NP-complete.*

Proof Sketch. The TUAP problem is the TWP problem with unit weight assignments to the elements of T. □

We consider now a different selection criterion that is based on the number of conjuncts that may be rendered false. In the case of an marked, unweighted, acyclic PPG, a related measure of goodness would be to find an IRS which minimizes the number of consistency conjuncts that it renders false.

Problem 3. *(PUAP) The Predicate-weight Problem for a marked, unweighted, acyclic PPG is: Given a marked, unweighted, acyclic PPG, G, and an integer K, is there an IRS, G', such that the number of elements in C' is at most K?* □

Theorem 4. *The PUAP problem is NP-complete.*

Proof Sketch. The proof of NP-easiness is the same as that for the TUAP problem with a verification of $|C'| \leq K$ replacing $|T'| \leq K$.

To prove NP-hardness, we exhibit a similar reduction from the problem L01 as we did for the TUAP problem. The instance of the PPG constructed is modified to have the additional subgraphs at the nodes Tx_i and Fx_i as shown in Figure 3. Set $K = (2n+m+2)$. The proof is now clearly similar to the NP-hardness proof of the TUAP problem. □

The above theorems indicate that the selection procedures to find optimal IRS graphs for the PPG graphs is difficult. We conjecture that there exist interesting cases of the PPG problems that are both of practical interest and of polynomial complexity. Furthermore, we begin to anticipate the need for heuristic approaches to find good IRS graphs in place of the "best" IRS graph.

5.2 Selection Based on Slack Times

Large slack times allow a greater flexibility in scheduling transactions, and in time-constrained systems, this flexibility is valuable. To analyze the PPG in terms of slack times and scheduling, we first formalize some of these notions.

Definition 5. *The* potential slack time *for a conjunct vertex c_i in a PPG, $G = (C, T, E, W_\kappa, W_\tau)$, is given by $S_\kappa(c_i) = W_\kappa(c_i) - \min_{(c_i,t_j)\in E}(W_\tau(t_j))$.* □

The slack time $S_\kappa(c_i)$ does not provide a precise value for a conjunct c_i since there is an inherent inaccuracy associated with the W_τ values. Furthermore, unless the transaction-type vertex t_j that corresponds to the minimum weight is chosen to resolve the inconsistency, the potential slack time may not be realized. However, S_κ does serve the purposes of approximation, especially if the transaction-types can be assumed to take unit time — in which case the potential slack time is always realized subject to accurate estimates for the transaction-type time costs.

5.2.1 Total Slack Time

The sum of the slack times associated with the conjunct vertices of an IRS, G', is called the total slack time of the IRS, and is denoted by $slack(G')$. As mentioned earlier, assume that some application indicates that a selection criterion may be based on the maximization of the the total slack time. With the S_κ values as provided, the IRS chosen directly would be the PPG itself — clearly an unacceptable choice. Hence, we use the method described below to limit the number of vertices chosen while retaining the criterion of total slack time maximization.

Definition 6. *The* inverse slack time *associated with a conjunct vertex c_i is given by $S'_\kappa(c_i) = \eta - S_\kappa(c_i)$ where $\eta \geq (1 + \max_{c_j\in C}(S_\kappa(c_j)))$.* □

The constraint on the value of η is to ensure that $S'_\kappa(c_i) \geq 1$ for all $c_i \in C$. The reason why η is left unspecified in the definition is explained below.

Suppose that an IRS, G'_{min}, is chosen such that the sum of the S'_κ values associated with its conjunct vertices is the smallest among all the IRSs, G', that are possible. Using the above definition, we have $\eta|C'_{min}| - slack(G'_{min}) \leq \eta|C'| - slack(G')$. Notice that $|C'_{min}| = |C'|$ implies that $slack(G'_{min}) \geq slack(G')$, and that $slack(G'_{min}) = slack(G')$ implies that $|C'_{min}| \leq |C'|$. Thus, for two IRSs, if the number of conjunct vertices in each is the same, the one with a larger total slack time is preferred by this minimization criterion. If the total slack times of the two IRSs are equal, then this criterion chooses the one with fewer conjunct vertices.

As mentioned above, attempting to maximize the total slack time *without* using a notion such as the inverse slack time leads to the selection of an unnecessarily large IRS with too many conjunct vertices. This is undesirable since the inclusion of a conjunct vertex in an IRS implies that the inconsistency-resolution process may cause that conjunct to become inconsistent. Thus, there exists a trade-off between increasing the total slack time, $slack(G')$, and decreasing the number of conjunct vertices, $|C'|$, in the IRS. It is the value of η that determines the importance attached to each. A small value of η gives more importance to maximizing $slack(G')$, whereas a large value of η gives more importance to minimizing $|C'|$. This is clear by examining the expression $\eta|C'| - slack(G')$ which is the sum of the inverse slack times of the vertices in C'.

Consider a modified PPG, G, in which for all $c_i \in C$ and $t_j \in T$, we set $W_\kappa(c_i) = S'_\kappa(c_i)$ and $W_\tau(t_j) = 1$. By introducing inverse slack times in this manner, and choosing a desired value for η, the question of maximizing the total slack time for an IRS reduces to the following problem.

Problem 4. *(PWP) The Predicate-weight Problem for a marked, weighted PPG is: Given a marked, weighted, acyclic PPG, G, and an integer K, is there an IRS, G', such that the sum of the weights of the elements in C' is at most K?* □

Theorem 5. *The PWP Problem is NP-complete.*

Proof Sketch. The PUAP problem is the PWP problem with unit weight assignments to the elements in C. □

5.2.2 Large Individual Slack Times

It may be argued that it is more germane to use a selection criterion for an IRS based on the largeness of the slack times associated with the conjuncts. That is, the cost of an IRS $G' = (C', T', E', W'_\kappa, W'_\tau)$ is $\max_{c_i \in C'}(S'_\kappa(c_i))$. Large slack times provide the flexibility in scheduling the inconsistency-resolving instances of transaction-types which may be necessitated by concurrency control considerations. As mentioned earlier, if a transaction is scheduled early, the inaccuracies in the transaction execution time estimates are less likely to affect the deadline requirements on the conjunct inconsistencies. We examine slack times in more detail in Section 6. In the discussion to follow, we assume for simplicity that an IRS exists.

Problem 5. *(IST) The Individual Slack Time Problem for a PPG is: For a given marked, weighted, acyclic PPG, G, and an integer K, is there an IRS, G', such that* $\max_{c_i \in C'}(S'_\kappa(c_i))$ *is at most K?* □

Theorem 6. *The IST problem is in polynomial-time.*

Proof Sketch. Add a (pseudo) transaction-type vertex t' to T with outedges (t', c_i) to every $c_i \in X$. With each vertex $v \in C \cup T$, associate two values, $V(v)$ and $tag(v)$. Set $tag(t_j) = 1$ for each sink transaction-type vertex t_j, and set all the remaining V and tag values to 0.

1. **while** $tag(t') = 0$ **do**
   (a) Choose vertex v with $tag(v) = 0$ and all successor vertices u with $tag(u) = 1$.
   (b) Set the value of $V(v)$ to $\max_{(v,u)\in E}(V(u))$ or $\max(S'_\kappa(v), \min_{(v,u)\in E}(V(u)))$ according as $v \in T$ or $v \in C$, respectively.
   (c) $tag(v) := 1$.
2. **if** $V(t') \leq K$ **then print** "Yes" **else print** "No", and **stop**.

At the end of loop statement, a tagged vertex, v, has the value $V(v)$ that provides the cost of the subgraph of the best IRS (in the IST sense) that is rooted at that vertex. With the use of suitable data structures, the algorithm runs in $O(|C| + |T| + |E|)$ time — thereby placing IST in polynomial-time. □

5.3 Discussion

The significance of the intractable results is only that the optimal solutions are computationally very costly to obtain. However, as in many other situations, near optimal solutions would serve almost as well. By sacrificing optimality, we can make use of several approximation methods available in the literature (e.g., from [5]). Such heuristic methods are well-studied and provide computationally inexpensive means to obtain near-optimal solutions. The fact that formal analysis of this nature is possible in our formulation is a very encouraging indication.

The PPG that we have dealt with so far may be regarded as "static", since the only "dynamic" aspect of the PPG are the markings. It is possible to consider a more complex "dynamic" version of a PPG where the weights may change dynamically, or the transaction-types are replaced by transaction instances. However, the intractability of the problems encountered in the static case indicate that the dynamic version would definitely pose problems that are at least as difficult. Thus, the study of a simpler model provides a basis for directly seeking heuristics in the more complicated models.

6 Using the IRS

Once an IRS is chosen, the question arises as to how the actions that it implies should be scheduled. It may be argued that since the transactions are likely to interact, concurrency control requirements may render the selection criteria for the IRS untenable. However, note that the intractability of the problems encountered indicate that additional criteria will not make the problems any easier, and heuristic methods must be used. Therefore, we separate the two issues of selection and scheduling for an IRS. The detailed analysis of the use of an IRS is beyond the scope of this paper, and we restrict ourselves to indicating the important issues involved in such analyses.

6.1 Scheduling, Slack Times, and Nested Transactions

Consider a subgraph of an IRS in Figure 4. The parenthesized numbers give the values of W_κ and W_τ for the conjunct vertices and the transaction-type vertices, respectively. Assume that c_2 and c_3 become inconsistent immediately after the completion of t_1. The IRS chosen does not allow any slack time for the resolution of the inconsistency in either of these conjuncts, and hence, t_2 and t_3 are scheduled immediately. The conjuncts c_4 and c_5 may become inconsistent immediately after t_2 and t_3 complete, respectively. Notice that since neither c_4 nor c_5 have any slack time, and hence, as soon as either of them becomes inconsistent, t_4 must be scheduled. However, in this example, c_4 and c_5 become inconsistent within $W_\tau(t_4) = 3$ time units of each other (in fact, within 1 time unit) — but not simultaneously. Thus, if the same transaction from the transaction-type t_4 is used to resolve the inconsistencies, irrespective of when it is scheduled, one of the two conjuncts will remain inconsistent for a period greater than its deadline. Furthermore, assuming that c_4 and c_5 do not become inconsistent within $W_\tau(t_4)$ time units of each other, it is the case that a single execution of t_4 will not suffice to resolve *both* the inconsistencies.

In a similar situation, the example in Figure 5 shows a conjunct vertex, c_1, that may become inconsistent due to the execution of either t_1 or t_2. Suppose that t_1 makes c_1 inconsistent, and t_2 does the same within the next $W_\kappa(c_1) = 3$ units of time. In this situation, no matter when t_3 is scheduled, the time period for which c_1 will remain inconsistent will exceed $W_\kappa(c_1)$.

The occurrence of problems such as those illustrated in the two examples above is not peculiar to our particular formulation. They will occur in general in systems with timing constraints, and the problems must be addressed if real-time databases are to be realized. Our model serves to exhibit these problems as well as to serve as a tool by which they may be analyzed.

In the examples just discussed, notice that if the conjuncts have larger slack times due to larger deadlines, the problems may be alleviated. For example, if we changed Figure 4 to have $W_\kappa(c_5) = 4$, and changed Figure 5 to have $W_\kappa(c_1) >> 3$, then the scheduling of the inconsistency-resolution transactions may be successfully accomplished. These examples show how large slack times permit the transaction-types to exceed their inherently inaccurate estimates of execution-times so long as their instances are scheduled sufficiently early.

Large slack times are useful in other contexts as well. Before transactions begin executing, it is often the case that the resources they need must be obtained — and this could be time-consuming. Furthermore, the duration of this resource-gathering phase is indeterminate and it depends on the other transactions that are executing concurrently in the system. If the transactions are triggered by conjuncts with large slack times, the initial phase of the transactions could be safely accommodated by scheduling the transactions early. One way to accomplish this to a certain extent is to identify the conjuncts with large slack times, and to use the notion of nested transactions as follows. The conjuncts with small slack times that occur in the IRS are embodied within the nested transaction-types. The conjuncts that have been identified with large slack times serve as triggering conjuncts for the nested transactions. Thus, the IRS is regarded as a collection of partially ordered nested transaction-types — most of which are triggered by conjuncts with large slack times. Details regarding nested transactions are available in [12, 14].

As an example, consider the PPG shown in Figure 6. We represent conjuncts that have been identified to have large slack times by triangular vertices. In the manner explained above, some vertices of the PPG are shown to be grouped together by the dotted outlines to form nested transaction-types that are denoted by nt_1, nt_2, nt_3, and nt_4. The conjunct vertex c_1 may trigger

instances of either one of the nested transaction-types nt_1 or nt_2. In nt_1, the parent transaction of type t_1 may spawn the child transactions of type t_5, t_6, and t_7 by making the conjuncts c_5, c_6, and c_7 inconsistent. Similarly, nt_2 has a parent transaction-type t_2, an instance of which may spawn child transactions of type t_6 and t_7. Notice that an instance of nt_2 could make c_8 inconsistent, and this would trigger an instance of nt_4 which consists of the single transaction-type t_8. The nested transaction-type nt_3 has a parent transaction-type t_4, an instance of which may spawn just a single child transaction of type t_8.

6.2 Concurrency Control Issues

In the execution of the transactions indicated by an IRS, besides correctness, the issue of the timing constraints is also of importance. We have noted above that an inconsistency-resolution subgraph induces a partial order on the set T' of transaction-types. Given that we seek the prompt restoration of consistency in a real-time system, the need for a significant amount of concurrency among instances of the transaction-types in T' is required. In this section, we briefly examine aspects of concurrency control protocols germane to our model.

From the model of transactions described earlier, the use of methods that deal with nested transactions is indicated clearly. The subgraphs of an IRS are best described as nested transaction-types with added timing constraints. Existing work on nested transactions should be modified to handle the timing considerations to be used in this context.

Obviously, the presence of timing constraints will affect the concurrency control. The increased needs for concurrency may be achieved using less restrictive correctness criteria as compared to the traditional serializability — for example, the correct concurrent execution criteria of [12]. Our model allows for external-output transactions and transactions with stringent timing requirements. Both of these suggest that the facility for undoing the effects of a transaction may be unavailable. Also, situations that may result in cascading aborts must be avoided — which does not necessarily preclude other transactions from "observing" data produced by uncommitted transactions since our model is not the traditional one. These factors suggest that it may be necessary to introduce the notion of compensating transactions [6, 11].

Transactions that run concurrently in our system interact due to the shared data that they may access. The timing constraints imply that the delays arising as a result of these interactions should be minimized. For example, deadlock or livelock situations should be avoided. The use of versions of data in this context also helps to alleviate the problem. Clearly, it is important to identify where the transactions may interact so as to reduce the interactions to limit the delays. Thus, our model plays the dual role of describing the transactions as well as prescribing their design to control the contention. We highlight some of the immediate facets of transaction interaction next.

Let S be an IRS of a PPG G. Let $c_{k_1}, t_{j_1}, c_{k_2}, t_{j_2}, \ldots, c_{k_m}, t_{j_m}$ be a path in S. We assume the NT/PV model of [12] with input and output conditions (pre- and post-conditions) for each transaction-type. Then, we expect the following to hold in many cases. For an edge $e = (c_{k_i}, t_{j_i})$ in S, since t_{j_i} is triggered by the falsehood of c_{k_i}, the input condition of t_{j_i} mentions all the data items occurring in c_{k_i}. Also, since t_{j_i} makes c_{k_i} true, the output condition of t_{j_i} mentions potentially all the data items occurring in c_{k_i}. For an edge $e = (t_{j_i}, c_{k_{i+1}})$ in S, since t_{j_i} may invalidate $c_{k_{i+1}}$, the output condition of t_{j_i} mentions potentially all the data items occurring in $c_{k_{i+1}}$. It is unlikely that all the data items of a conjunct will be affected by one transaction-type. These observations can be used to identify bounds on the read and write sets of transaction-types. Such information can be used to advantage in concurrency control. Concurrency along paths in S could be managed by the preemptive protocol described in [KS88], perhaps simplified to its single-version variant.

It is valuable to identify the potential for concurrency among the transaction-types that do not lie on the same path in the IRS. Although two transaction-types may not have any common conjuncts in their pre- and post-condition sets, it may happen that they access common data items. This is because different conjuncts may mention common data items. Also, consider an example of a PPG in which there are two out-edges (c_1, t_1) and (c_1, t_2) from the same conjunct vertex c_1. It may happen that the IRS for the PPG contains both the transaction-types t_1 and t_2, but only one of the two edges, say (c_1, t_1). This could happen if t_2 is chosen to resolve the inconsistency for some conjunct other than c_1. In such a situation, the analysis to find common data items should include consideration for both the edges (c_1, t_1) and (c_1, t_2). After this analysis is done, it becomes necessary to ensure that the instances of the transaction-types that access the common data are correctly controlled by the concurrency protocol.

Even after reducing the extent to which the transaction-types interact, any reasonable real-time database system will have transactions competing for the resources. In such situations, the study of preemptive protocols to manage the timing and prior-

ity constraints is important. Thus, the satisfaction transaction-type timing constraints may result in the sacrifice of the best overall throughput of the system. Research along these lines is desirable for our model, and in this context, work such as [1] may be extendible.

7 Conclusions

We have proposed a model of real-time transaction processing based upon deadlines associated with consistency constraints. We have demonstrated that, in general, finding a strategy for restoring database consistency is computationally intractable. This negative result does not preclude the practical use of our model. Rather, it indicates that heuristics or suitable "protocols" are required for transaction processing. An analogous situation exists for standard transaction processing, where the set of two-phase locked schedules is usually accepted as a suitable subset of the set of serializable schedules whose recognition problem is NP-complete.

We have suggested some approaches toward the development of practical transaction management algorithms for our real-time model, but many issues remain to be addressed. For example, heuristics for the selection of an acceptable inconsistency-resolution subgraph are needed as is the development of a complete concurrency protocol that exploits the semantics of the inconsistency-resolution subgraph. The introduction of dynamic violations of consistency constraints in real-time database systems requires the system to modify its consistency restoration strategy as external events occur. Rather than recomputing a complete strategy, an incremental algorithm is desirable. Techniques of this nature are already in use in expert database systems [4].

Acknowledgements

The authors wish to thank Robert Abbott, Hector Garcia-Molina, and Eliezer Levy for helpful discussions.

References

[1] R. Abbott and H. Garcia-Molina. Scheduling real-time transactions. *SIGMOD Record*, 17(1):71–81, March 1988.

[2] R. Abbott and H. Garcia-Molina. Scheduling real time transactions: A performance evaluation. In *Proceedings of the Fourteenth International Conference on Very Large Databases, Los Angeles*, pages 1–12, 1988.

[3] C. Beeri, P. A. Bernstein, and N. Goodman. A model for concurrency in nested transaction systems. *Journal of the ACM*, 36(2):230–269, April 1989.

[4] C. Forgy. RETE: A fast match algorithm for the many pattern/ many object pattern match problem. *Artificial Intelligence*, (19):17–37, 1982.

[5] M. R. Garey and D. S. Johnson. *Computers and Intractability.* W. H. Freeman and Company, New York, 1979.

[6] J. N. Gray. The transaction concept: Virtues and limitations. In *Proceedings of the Seventh International Conference on Very Large Databases, Cannes*, pages 144–154, 1981.

[7] T. Hadzilacos and V. Hadzilacos. Transaction synchronisation in object bases. In *Proceedings of the Seventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Austin*, pages 193–200, March 1988.

[8] R. Holte, A. K.-L. Mok, L. Rosier, I. Tulchinsky, and D. Varvel. The pinwheel: A real-time scheduling problem. In *Proceedings of the 22nd Hawaii International Conference on System Sciences, Kailua-Kona*, pages 693–702, January 1989.

[9] F. Jahanian and A. K.-L. Mok. Safety anaylsis of timing properties in real-time systems. *IEEE Transactions on Software Engineering*, SE-12(9):890–904, September 1986.

[10] H. F. Korth, W. Kim, and F. Bancilhon. On long duration CAD transactions. *Information Sciences*, 46:73–107, October 1988.

[11] H. F. Korth, E. Levy, and A. Silberschatz. A formal approach to recovery by compensating transactions. In *Proceedings of the Sixteenth International Conference on Very Large Databases, Brisbane*, pages ? –?, August 1990.

[12] H. F. Korth and G. Speegle. Formal model of correctness without serializability. In *Proceedings of ACM-SIGMOD 1988 International Conference on Management of Data, Chicago*, pages 379–388, June 1988.

[13] D. R. McCarthy and U. Dayal. The architecture of an active data base management system.

In *Proceedings of ACM-SIGMOD 1989 International Conference on Management of Data, Portland, Oregon*, pages 215–224, June 1989.

[14] J. E. B. Moss. Nested transactions: An introduction. In B. Bhargava, editor, *Concurrency Control and Reliability in Distributed Systems*, pages 395–425. Van Nostrand Reinhold, 1987.

[15] P. Peinl and A. Reuter. High contention in a stock trading database: A case study. In *Proceedings of ACM-SIGMOD 1988 International Conference on Management of Data, Chicago*, pages 260–268, June 1988.

[16] L. Sha, R. Rajkumar, and J. P. Lehoczky. Concurrency control for distributed real-time databases. *SIGMOD Record*, 17(1):82–98, March 1988.

[17] M. Singhal. Issues and approaches to design of real-time database systems. *SIGMOD Record*, 17(1):19–33, March 1988.

[18] S. H. Son, editor. *SIGMOD Record: Special Issue on Real-Time Databases.* ACM, March 1988.

[19] J. A. Stankovic. Misconceptions about real-time computing. *IEEE Computer*, pages 10–19, October 1988.

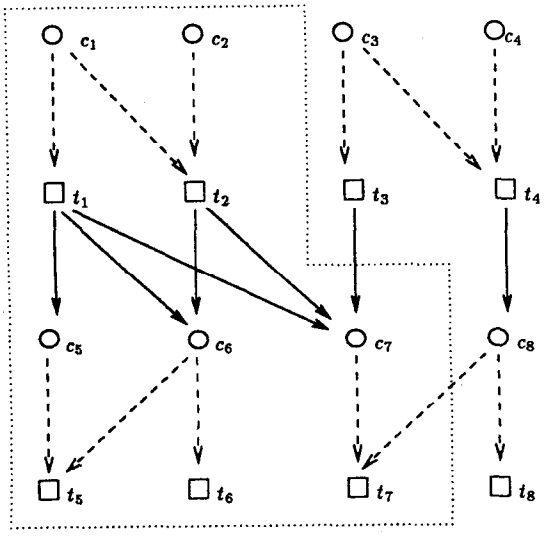

Figure 1: An example PPG

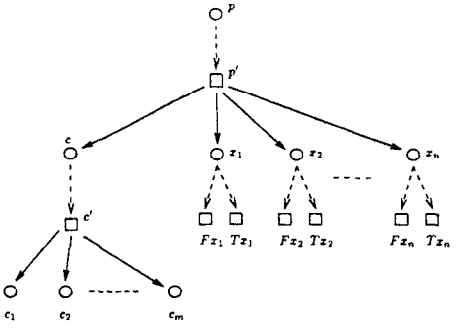

Figure 2: Construction of the PPG for the TUAP problem

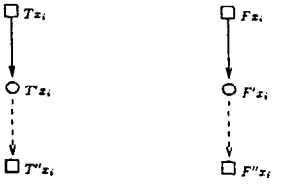

Figure 3: Construction of the PPG for the PUAP problem

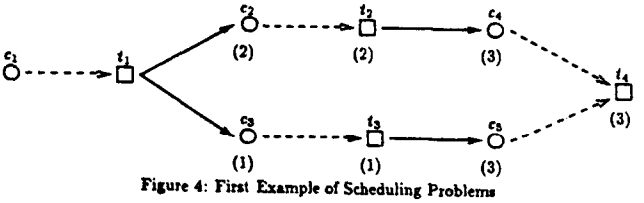

Figure 4: First Example of Scheduling Problems

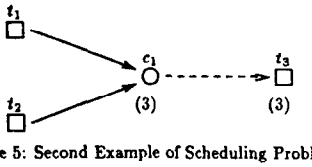

Figure 5: Second Example of Scheduling Problems

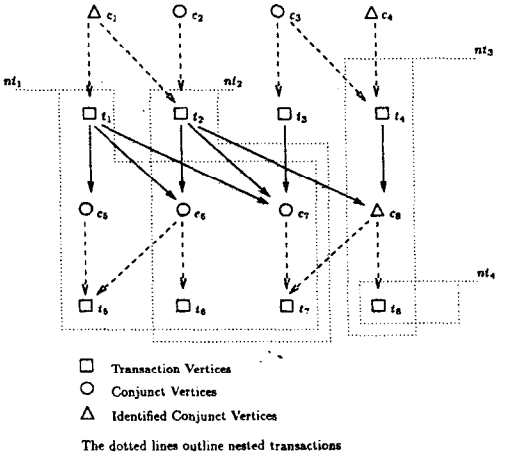

Figure 6: Large Slack Times and Nested Transaction-types