Integrating Temporal, Real-Time, and Active Databases

Krithi Ramamritham, Raju Sivasankaran, John A. Stankovic, Don T. Towsley, Ming Xiong

Department of Computer Science
University of Massachusetts
Amherst MA 01003-4610

e-mail: {krithi,sivasank,stankovic,towsley,xiong}@cs.umass.edu

Abstract

To meet the needs of many real-world control applications, concepts from Temporal, Real-Time, and Active Databases must be integrated:

- Since the system's data is supposed to reflect the environment being controlled, they must be updated frequently to maintain temporal validity;
- Many activities, including those that perform the updates, work under time constraints;
- The occurrence of events, for example, emergency events, trigger actions.

In these systems, meeting *timeliness, predictability*, and *QoS guarantee* requirements – through appropriate resource and overload management – become very important. So, algorithms and protocols for concurrency control, recovery, and scheduling are needed. These algorithms must exploit semantics of the *data* and the *transactions* to be responsive and efficient. Whereas time cognizant scheduling, concurrency control and conflict resolution have been studied in the literature, recovery issues have not. We have developed strategies for data placement at the appropriate level of memory hierarchy, for avoiding undoing/redoing by exploiting data/transaction characteristics, and for placing logs at the appropriate level in the memory hierarchy. Another issue that we have studied deals with the assignment of priority to transactions in active real-time database systems. We are also studying concurrency control for temporal and multi-media data. We have built RADEx, a simulation environment to evaluate our solutions.

1 Introduction

In real-time applications, the state of the environment *as perceived by the controlling system* must be consistent with the actual state of the environment being controlled. Otherwise, the decisions of the controlling system may be wrong and their effects disastrous. Hence the timely monitoring of the environment, the timely processing of the sensed information, and the timely derivation of needed data are essential. Data maintained by the controlling systems and utilized by its actions must be up-to-date and temporally correlated. This *temporal consistency* must be maintained through the timely scheduling of the actions that refresh the data. (Some) of the transaction deadlines result from these temporal[1] requirements imposed on the data.

In real-time applications, actions are triggered by the occurrence of events, that is, associated with each action is an event occurrence. An event triggers an action only if certain conditions hold. For instance, the occurrence of the event corresponding to a temperature reading would trigger an emergency reaction only if the temperature value is over a threshold. The Event-Condition-Action (ECA) paradigm of active databases is very convenient to enforce these constraints and also to trigger the actions. In fact, an active real-time database can serve as a repository of the data about the environment being controlled, and as a repository of the meta-control data and algorithms. For instance, the rules can be designed to trigger entry into specific modes; to trigger the necessary adaptive responses to time constraint violations – to effect recovery, to trigger actions if temporal data is (about to become) invalid; and to shed loads as well as adjust deadlines and other parameters, e.g., importance levels and QoS, of actions, when overloads occur [12].

Extant work on real-time databases focuses mainly on the meeting of time constraints. But it is clear that there is a need to integrate time-constraint handling with active and temporal requirements. This obviously calls for time cognizant priority assignment, concurrency control and conflict resolution mechanisms. Whereas past work in real-time databases has examined deadline-based priority assignment policies for atomic transactions, here we have transactions that trigger other subtransactions during their execution. As a result, the priority assignment problem becomes more involved. Another complexity arises from the need to run transactions with temporally valid data. This demands tailored data placement, logging, and recovery policies that exploit the data temporal semantics, specifically the fact that some of the data is non-persistent and may have short lifetimes that do not allow for storage into and access from disks.

In the rest of this paper, we first show how data and transaction characteristics can be exploited to determine the polices for data placement, logging, and recovery so as to achieve efficient transaction processing. These policies are intended for transactions

This research is funded in part by the National Science Foundation under grant NSF IRI-9208920.

[1] The semantics of the term temporal is different from the notion used in the temporal database literature.

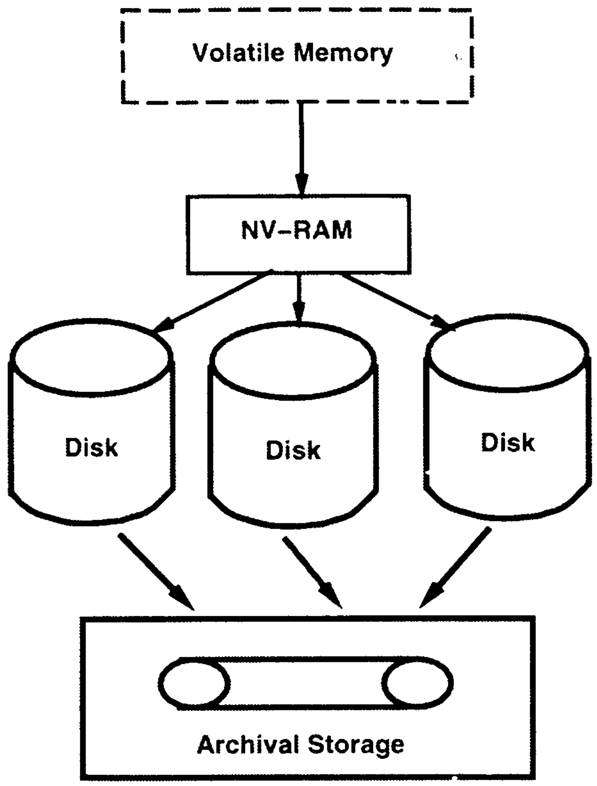

Figure 1: **Four Level Storage Hierarchy**

in active database systems which have temporal data and where transactions have deadlines. Next we discuss transaction processing issues. To set the stage, we first present a four-level memory hierarchy that is more suited to meet the needs of temporal data. At the end of the paper, we summarize our work on priority assignment when transactions trigger other transactions and when data has temporal properties.

2 Memory Hierarchy in Real-Time Active Databases (RTADBs)

We assume a 4 level memory hierarchy, as shown in Figure 1. The first level consists of main memory that is volatile. At the second level is RAM that is non-volatile (NV-RAM). The third level consists of the persistent disk storage subsystem and at the fourth level is the archival tape storage.

The motivation for the use of NV-RAM stems from the fact that maintaining large amounts of data in main memory can be very expensive while disk I/O times might be unacceptable in certain situations in a RTADB. For instance, writing to disk for the purpose of logging the data touched by critical transactions and reading from disk to undo critical transactions might be too expensive. It is not difficult to conceive of situations where writing to disk may result in missing their deadlines, but by writing to NV-RAM, deadlines can be met. Also, maintaining large amounts of data in main memory can be very expensive while disk access can be very slow. NV-RAM can be used purely as a disk cache where the data moved to NV-RAM later migrates to the disk or it can be used as a temporary stable storage where the data is stored for performance reasons and later may or may not not migrate to the disk depending on the

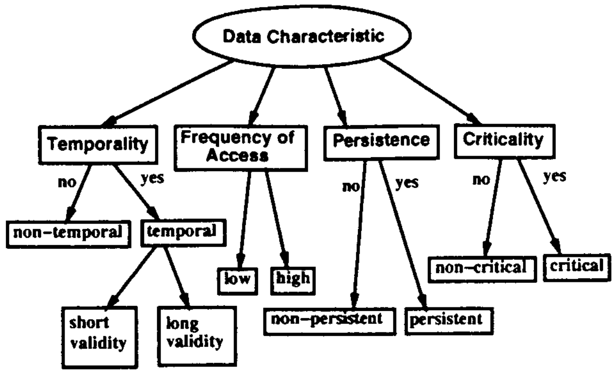

Figure 2: **Dimensions of Data Characteristics**

(durability) characteristics of the data.

Persistence of system data structures such as global lock tables, and rule bases that represent the dependencies between transactions could become potential bottlenecks in RTADBs. Making these system data structures persistent in NV-RAM results in better performance.

3 Consideration of Data Characteristics

Figure 2 shows the different dimensions of the characteristics of data in real-time applications. The characteristics of a particular type of data will determine where data is placed in the four-level memory hierarchy, where the logs are maintained and how the system recovers from transaction aborts [13].

- *Temporality* of data refers to the temporal validity properties of the data.
  - Non-temporal Data
    Data found in traditional database systems belong to this category. Passage of time does not affect the validity of such data, i.e., no temporal validity intervals are associated with this kind of data. So, the placement, logging and recovery and concurrency control technique would depend on the other attributes of the data.
  - Temporal Data
    Data with this attribute has temporal validity intervals attached to it. Data is valid only within this interval. Data with short validity must be kept in main memory since by the time the data is stored on disk or is fetched from disk, the data is likely to become invalid. It will not be necessary to write conventional undo/redo log for such data because undoing would imply throwing away the data and redoing would imply freshly acquiring (or calculating) the data. This is because by the time the disk is accessed and undo is performed, it could become invalid. By the time redo is performed, that version could have become invalid. Data with long validity may be stored on disk as the transactions will be able to access, modify and commit the data from the disk before the validity expires.

An important issue arises if temporal data can be moved to persistent storage for space reasons because of the Steal buffer management policy. In this case, the data has to be valid for the duration of the time it is flushed to the persistent store and retrieved again. It is possible that stolen data with not very short validity can be moved to NV-RAM and data with long validity can be moved to disk.

Note that when transactions access temporal data, the data must be up-to-date, as specified by its absolute consistency requirements, and correlated, as specified by their relative consistency requirements [11].

- *Frequency of Access* of data refers to the frequency with which the data is read or written, that is, it indirectly reflects the need for the data to be kept in main memory.
  - High frequency
    This kind of data corresponds to what is usually called *hotspots*. For performance reasons this data could always be in main memory, that is, it is never allowed to be stolen, and if persistence is necessary, it can moved to NV-RAM from where it can migrate to disk at regular intervals and the persistent copy in NV-RAM makes recovery faster. (For even faster recovery, the before image can be kept in the main memory itself.) Undo logging is not necessary if this data is not ***stolen*** to disk. Data on hot selling stocks in the stock trading database is an example of this kind of data.
  - Low frequency
    This is the regular disk resident data for which conventional undo/redo logging is applicable. Most of the data in the database is of this kind.
- *Persistence* refers to the durability properties of the data.
  - Persistent data
    The data could be made persistent either on the NV-RAM or disk depending on other data characteristics (e.g., frequency of access).
  - Non-persistent data
    This data need not be persistent and hence could just reside in main memory until its temporal validity expires. But for space reasons data might migrate to the persistent storage. Again, where it migrates to might depend on the other characteristics of the data. Since some of the data is temporal, such data need not be durable. However, for record keeping purposes, some of the sensor readings and actuator control settings may be archived, that is, made durable.
- *Criticality* refers to the importance of the data for processing a transaction.
  - Critical Data
    The validity and consistency of critical data must be ensured. Critical data ought to be recovered quickly and efficiently but other related data touched by the same transaction that touched the critical data can be recovered later.
  - Non-critical data
    Data whose availability is not crucial for completing a transaction.

  This attribute of data implies that it is possible to do partial recovery in time and resource constrained situations. With traditional (sequential) logs partial recovery implies traversing the log more number of times than would be the case with traditional recovery. One way to achieve partial recovery with low overheads is to cluster logs and chain them such that sets of data can be recovered in order of their importance. This kind of logging and recovery scheme is also amenable to parallelism. For faster recovery, one could conceive of multiple disks to store the separate logs.

  In addition, separate checkpoints can be created, each corresponding to a different criticality level of data. Having separate log for each category helps in quick recovery for the critical data.

  Note that criticality of a data item could be an inherent property of the data or could arise from the criticality of the transactions accessing the data

Let us look at a few combinations of data characteristics and examine the data placement and logging and recovery techniques that they need.

- short validity, high frequency of access, non-persistent, critical data:
  Positions of flying aircraft is an example of this data. This kind of data will be placed in main memory. For space reasons, the data might temporarily migrate to NV-RAM if the validity is long enough for a NV-RAM write and read. No-steal buffer policy will be used and so there is no need to undo. In addition, given the short validity, redos will also not be needed or feasible.
- Long validity, high frequency of access, persistent, critical data:
  Reactor temperatures in a chemical plant have this property. Temperatures typical change slowly, but the values are used for many other critical calculations. Also, temperature records are maintained on disk for future trend computations. This kind of data will be placed in main memory for performance reasons. No-steal buffer policy along with the force policy will be used where data is forced to NV-RAM and subsequently to disk.
- Long validity, low frequency of access, persistent, noncritical:
  This kind of data can be disk resident and will be brought into main memory when necessary.

The frequency of access attribute dictates where one should place the data such that I/O costs are minimized. In traditional databases, disk pre-fetching is a technique that is used to minimize the I/O delay. In our context, an analog of this technique can be used, also to ensure that valid data is available when needed. Specifically, a similar technique, namely, *pre-triggering*, can be used to acquire temporal data that is going to be accessed, but is invalid or will become invalid by the time the data is needed. Instead of triggering a transaction to acquire the data just before it is needed, the transaction can be triggered earlier, at some opportune time.

It is important to point out that data characteristics can change during the course of a mission. For instance, the space shuttle goes through mode changes, from ascent, to orbiting, to descent. With mode changes, the characteristics of the data associated with the mission also change. Thus, the type of techniques adopted for processing the data also need to change.

For example, it is possible to dynamically change the logging model depending on the changing characteristics of the data. At some point in time, if a certain data item becomes a hotspot then it is preferable to keep the data in main memory. Let us assume that undo-redo logging was being done for this data item. Once it becomes a hotspot, redo only logging can be done, saving space and transaction recovery time.

4 Consideration of Transaction Characteristics

The previous section considered how data characteristics affect data placement, loggin and recovery. Here we study how transaction characteristics affect the same issues.

Transaction priority may decide the placement of data, i.e., data of high priority transactions may always reside in main memory and may never migrate to disk and data of low priority transactions may migrate to disk. One could also conceive of transactions whose data migrate only to the NV-RAM. Stealing can be disallowed. This leads to redo only logging for high priority transactions and undo-redo logging for low priority transactions. This will not only improve the running time of high priority transactions, but also make the undo-restart time of high priority transactions low.

- For instance, say, there are two critical transactions t_1 and t_2, and t_1 is aborted by t_2. If no-undo and redo only logging is done for t_1 then the restart time for t_1 will be low (low undo time) and hence the probability of meeting the deadline despite a restart will be high.

It should be noted that priority based buffer allocation and replacement might have the same result, i.e., the pages of a critical transaction are not flushed. But if make explicit choices are made in logging then logging overheads can also be saved.

Scheduling/aborting transactions should take into consideration the temporal validity of the data.

- For example, let transaction t_1 read data item x and transaction t_2 read data item y. If one of the transactions must be aborted, it can be the one that touches a data item that will be valid for a longer duration, in this case long enough to abort the transaction and rerun it to meet the deadline. By doing that both the transactions could meet their deadlines.

In making decisions about which transaction to abort, the cost of abortions and restarts must be considered.

- Let there be two transactions t_1 and t_2, and one of them has to be aborted. Transaction t_1 has touched data that has migrated to disk and t_2 has touched data that has not. The cost of undoing t_1 is higher than the cost of undoing t_2. Because of this, aborting t_2 might result in better performance.

Aborts because of the expiration of deadlines must take recovery costs into consideration. It might be less expensive to complete a transaction whose deadline has expired than to undo it.

5 Priority Assignment

In this section, we consider the problem of assigning priorities to transactions having deadlines – in active and temporal databases.

Consider a transaction in an active database system. The transaction has a deadline from which its own priority has been previously assigned. When this transaction triggers another transaction – to execute and complete before the triggering transaction, what should be the priority of the triggering and triggered transactions? To answer this question, we have examined several candidate policies:

- PD: Deadline of triggered transaction = Deadline of triggering transaction.

  This is a very simple policy and can serve as a baseline.

- DIV: Divide a transaction's slack among triggering and triggered transactions.

  Slack refers to the length of time a transaction can afford to wait and yet complete before its deadline. Computing the slack necessitates knowledge of transaction execution times.

- SL: Adjust slack of triggering transaction whenever it triggers.

  This policy initially assigns the triggering transaction's priority based on assumptions about its probability of triggering a transaction when events occur. The priorities are subsequently adjusted based on whether or not transactions are triggered at these events.

Our performance results [9] indicate that DIV and SL provide better performance to transactions that trigger than PD; SL's performance depends on the relative slacks of transactions; Extra information about triggering characteristics improves performance; With information about I/O, SL performs very well for disk resident databases. This study shows that new policies are necessary to deal with the priority assignment problem when transactions trigger other transactions during their execution and that wherever possible, we must attempt to exploit the knowledge about computation times, and triggering patterns to improve performance.

Now we consider additional issues that arise when transactions access temporal data. A transaction is considered to be correct if

1. it leaves the database in a logically consistent state and produces results that are logically consistent,
2. it meets its deadline, and
3. it reads temporally consistent data, and the data is still valid when it commits.

The third requirement means that we may have to finish a transaction's execution sooner than is allowed by its deadline. Specifically, if the data read by a transaction will expire before the deadline expires then we must try to finish before the former, called, the *data_deadline*. If the transaction does not complete by its *data_deadline*, we must abort it and restart it, provided there is still time to complete it before its deadline using temporally valid data. Because of these reasons, the assignment of priorities must be transaction-deadline-cognizant as well data-validity-cognizant. Clearly, if the transaction deadline is smaller than the *data_deadline*, then the transaction deadline will prevail. Otherwise, we have several options captured by the following priority assignment formula:

$$\text{priority}_T = \alpha * \text{data\_deadline}_T + (1 - \alpha) * \text{deadline}_T;$$

where $(0 \leq \alpha \leq 1)$. Depending on the value of α several policies emerge:

1. *Earliest Deadline First* (EDF): $\alpha = 0$.
   This uses only the transaction's deadline and ignores temporal consistency constraints. Therefore, a transaction might be aborted due to temporal inconsistency of data and restarted again.
2. *Earliest Data Deadline First* (EDDF): $\alpha = 1$.
   To take into account temporal consistency constraints, transactions are scheduled according to their *data_deadlines*. The risk is that a transaction T_1 with a long deadline, but a tight *data_deadline*, is given a higher priority than another transaction T_2 with a tight deadline. Therefore, transaction T_2 might not be able to complete before its deadline whereas it might be preferable to abort T_1 and restart it thereby allowing both to complete before their deadlines.

3. *Hybrid Approach* (Hybrid): α is the fraction of work that has been done.
   This is a compromise between the first two. The more work a transaction has done, the larger the weight assigned to its *data_deadline*.

4. *Half-Half Approach* (HH):

$$\alpha = \begin{cases} 0 & \text{if the fraction of work done} \leq 0.5 \\ 1 & \text{Otherwise} \end{cases}$$

   This algorithm is another attempt at a compromise between the first two. It assigns the priority of a transaction according to its deadline if it has not completed at least half of its work. Otherwise, the priority is assigned according to the *data_deadline* of the transaction.

Notice that the Hybrid and HH approaches can only be used if the total amount of work of a transaction is known a priori. They do require that the system keep track of the amount of work a transaction has done. The motivation behind those two approaches is to reduce the amount of wasted work and to avoid recovery overheads.

When we have transactions that have both temporal and active data, the two types of priority assignment policies discussed in this section need to be combined. We are currently exploring this issue.

6 Conclusions

Transaction scheduling and concurrency control aspects of real-time databases have been studied in detail [1, 5, 6, 7]. Active database researchers have been involved in the development of system models that incorporate the ECA paradigm, examining issues related to the specification and recognition of different types of events and conditions and the relationships of triggering and triggered actions [4, 2]. Transaction scheduling in active databases [3] and recently in real-time active databases [9] has been studied.

Little, if any attention has been paid to transaction scheduling when transactions dynamically trigger other transactions or access temporal data, to data-characteristic-dependent transaction processing, to the placement of data and logs, and to logging and recovery techniques. These are crucial to meet the performance requirements of RTADBs. Specifically, logging and recovery is a complex, unexplored issue in the context of RTADBs and has great impact on the predictability and performance of the system. For instance, in the undo/redo recovery model, undoing the effects of a transaction consumes resources that can interfere with ongoing transactions. RTADB recovery techniques must consider resource availability to determine the most opportune time to do recovery.

In this paper we outlined the approaches developed to address these problems. Since most situations offer multiple options, we have developed a simulator, called RADEx, that allows us to study the tradeoffs and then develop the policies. This simulator supports object-oriented data and has the following components:

- DBManager - specification of the data model.
- Transaction Generator - simulates the application or the environment.
- Transaction Manager - schedules the transactions.
- Object Manager - responsible for concurrency control.
- Rule Manager - triggers rules depending on the database events.
- Resource Manager - simulates CPU and disk usage.
- Log and Recovery Manager - responsible for logging and recovery.

The experiments dealing with different priority assignment policies were conducted using this simulator.

In addition to our work on the algorithms for data placement, logging, and recovery, we are currently pursuing transaction processing with multi-media QoS considerations, concurrency control for temporal data, and finally distribution and heterogeneity issues.

References

[1] R. Abbott and H. Garcia-Molina, "Scheduling Real-Time Transactions with Disk Resident Data," *Proceedings of the 15th International Conference on Very Large Databases*, Aug. 1989.

[2] U. Dayal et. al., "The HIPAC Project: Combining Active Databases and Timing Constraints," *SIGMOD Record*, 17, 1, March 1988.

[3] U. Dayal, M. Hsu, and R. Ladin, "Organizing Long-Running Activities with Triggers and Transactions," *Proceedings of ACM SIGMOD*, 1990.

[4] Special Issue on Active Databases, *Data Engineering*, Dec. 1992.

[5] J.R. Haritsa, M.J. Carey, and M. Livny, " Earliest Deadline Scheduling for Real-Time Database Systems," *Proceedings of the Real-Time Systems Symposium*, Dec. 1991.

[6] J. Huang, J.A. Stankovic, K. Ramamritham, and D. Towsley, "Experimental Evaluation of Real-Time Optimistic Concurrency Control Schemes", *Proceedings of the 17th Conference on Very Large Databases*, Sept. 1991.

[7] Y. Lin and S.H. Son, "Concurrency Control in Real-Time Databases by Dynamic Adjustment of Serialization Order," *Proceedings of the Real-Time Systems Symposium*, Dec. 1990.

[8] B. Purimetla, R. M. Sivasankaran and J.Stankovic, "A Study of Distributed Real-Time Active Database Applications," *IEEE Workshop on Parallel and Distributed Real-time Systems*. April 1993.

[9] B. Purimetla, R. Sivasankaran, J.A. Stankovic, K. Ramamritham and D. Towsley, "Priority Assignment in Real-Time Active Databases," *Conference on Parallel and Distributed Information Systems*, Oct 1994. (selected to appear in) *VLDB (Very Large Data Bases) Journal*, Jan. 1996.

[10] E. Rahm "Use of Global Extended Memory for Distributed Transaction Processing," *Proc. 4th Int'l Workshop on High Performance Transaction Systems*, Sept. 1991.

[11] K. Ramamritham, "Real-Time Databases," *International Journal of Distributed and Parallel Databases*, 1993.

[12] K. Ramamritham, "Where Do Deadlines Come from and Where Do They Go?," *Journal of Database Management*. (to appear) 1996.

[13] R. M. Sivasankaran, K. Ramamritham, J. A. Stankovic, and D. Towsley. "Data Placement, Logging and Recovery in Real-Time Active Databases," *Workshop on Active Real-Time Database Systems*, Sweden, June 1995.