Data Placement, Logging and Recovery in Real-Time Active Databases[1]

Rajendran M. Sivasankaran, Krithi Ramamritham, John A. Stankovic & Don Towsley

Department of Computer Science

University of Massachusetts, Amherst, MA 01003

stankovic@cs.umass.edu

413-545-0720 FAX 413-545-1249

Abstract

In the past, real-time transaction processing systems have largely considered transaction characteristics such as deadlines and criticality to schedule/abort transactions, and to perform concurrency control and buffer management. Active database researchers have been involved in the development of system models that incorporate the Event-Condition-Action paradigm, examining issues related to the specification and recognition of different types of events and conditions and the relationships of triggering and triggered actions. Given the need to provide for both active and real-time capabilities, novel techniques, combining the two areas must be developed. Also, existing techniques have several deficiencies. Data characteristics have not been explicitly considered to do transaction processing. The impact of data characteristics on data placement, logging and recovery issues have not been studied. We show that exploiting the characteristics of data for transaction processing, placing the data at the appropriate level of the memory hierarchy, and performing logging and recovery of data appropriate for each type of data is crucial to achieve high performance in real-time active database systems.

1 Introduction

An Active Real-Time DataBase (ARTDB) is a system where transactions have timing constraints such as deadlines, where data may become invalid with the passage of time and where transactions may trigger other transactions. The applicability of ARTDB in applications such as cooperative distributed navigation systems and Network Services databases was studied in detail in [12, 17]. There is a number of other applications such as air-traffic control, stock-trading, banking, office work-flow management sysems, and nuclear plant management systems where ARTDB might be applicable. The kind of ARTDB system that we consider consists of

[1]This work was supported, in part, by NSF under grant IRI-9208920.

a *controlling system* (the database system) and a *controlled system* (environment) where the state of the controlled system is modeled by data items in the controlling system. Thus, the actions of the controlling system can be modeled as transactions that can change the state of the environment. In this paper, we

- discuss data and transactions characteristics in such ARTDB systems, and
- show that exploiting the characteristics of data for transaction processing, placing the data at the appropriate level of the memory hierarchy, and performing logging and recovery of data appropriate for each type of data is crucial to achieve high performance in real-time active database systems.

Section 2 provides motivation for this work. Section 3 discusses the memory hierarchy suitable for the performance requirements of ARTDBs. Section 4 presents the different data characteristics and shows how they affect placement, logging, and recovery strategies. Section 5 does the same with respect to transaction characteristics. The paper concludes with directions for future work (Section 6).

2 Motivation

In conventional (disk-resident) databases data resides on disk and when a transaction wants to read or write the data it is brought into main memory. Conventional transactions exhibit ACID[2] properties which are achieved through a choice of concurrency control mechanism (usually, two-phase locking), and recovery mechanism (usually, No-Force/Steal buffer management policy along with Undo/Redo logging). In ARTDBs data and transactions have different properties than those solely implied by the ACID properties and, therefore, require a redesign of the strategy to do concurrency control, buffer management, and logging and recovery.

In this paper we examine the characteristics of data and transactions so as to achieve the required performance and to cater to the new performance metrics of meeting deadlines. This requires

1. Proper scheduling of transactions.
2. Proper concurrency control and conflict resolution.
3. Placement of data in the appropriate level of memory hierarchy to reduce disk reads and writes.

[2]ACID stands for Atomicity, Consistency, Isolation and Durability.

4. Proper choice of (No)Steal/(No)Force policies – to reduce undoing work of aborted transactions that has to be discarded, but has migrated to disk and redoing work of committed transactions that has been lost due to crashes.

5. Proper logging techniques and the proper placement of logs in the appropriate level of storage.

The transaction scheduling and concurrency control aspects of real-time databases have been studied in detail in [1, 9, 10, 11]. Most investigations have focused on the processing of transactions with soft deadlines by adopting priority-assignment policies and conflict resolution mechanisms that explicitly take time into account. The most common results have been the development of various *time-cognizant* extensions of two phase locking, optimistic, and timestamp based protocols.

Active database researchers have been involved in the development of system models that incorporate the Event-Condition-Action paradigm, examining issues related to the specification and recognition of different types of events and conditions and the relationships of triggering and triggered actions [6, 5]. Transaction scheduling in active databases [7] and recently in real-time active databases [13] has been studied.

Little, if any attention has been paid to data-characteristic-dependent transaction processing, placement of data and logs, and logging and recovery techniques. These are crucial to meet the performance requirements of ARTDBs. Specifically, logging and recovery is a complex, unexplored issue in the context of ARTDBs 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. ARTDB recovery techniques must consider resource availability to determine the most opportune time to do recovery.

3 Memory Hierarchy in an ARTDB

One of our main goals is to discuss the issues regarding data placement in a ARTDB arising from its high performance and predictability requirements. Hence, the design of the system memory hierarchy is very important. We do not assume any particular database architecture; our discussion is applicable to shared disk, shared nothing, shared everything environments, and client-server and heterogeneous database systems. 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 a RAM that is non-volatile (NV-RAM). Basically, the NV-RAM acts as a cheaper RAM and

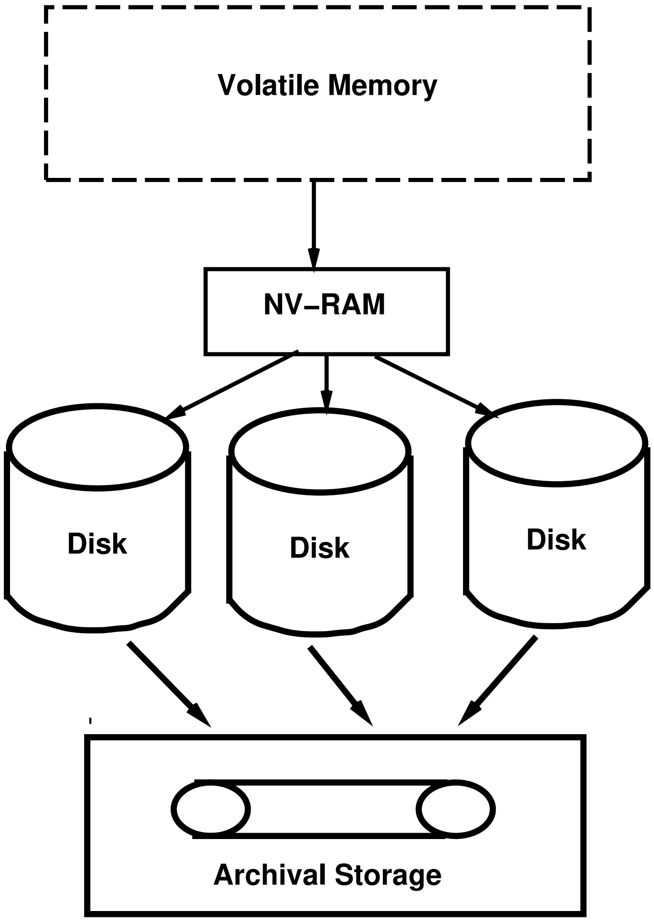

Figure 1: **Four Level Storage Hierarchy**

a fast persistent store. The third level consists of the persistent disk storage subsystem and at the fourth level is the archival tape storage.

It is obvious that a main memory database system is ideal for a real-time active database. But, the size of the database of the real-time applications that we consider might be too large to fit in main memory. For instance, in the automated co-operative navigation system [12], large maps and images have to be stored in the database, making it either impractical or too expensive to maintain the whole database in main memory. The second level NV-RAM is similar to the extended storage discussed in [14] which could be extended memory, solid-state disks or disk-caches. In spite of the differences in the access speeds between different extended storages they have one thing in common, i.e., they all provide much better I/O speeds than disks. In this paper, we use the term NV-RAM to indicate that we are talking about extended memories with battery backup.

3.1 Why NV-RAM in an ARTDB?

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 whereas disk I/O times may be unacceptable in certain situations in an ARTDB. 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. One could conceive of situations where writing to disk may result in missing deadlines, but by writing to NV-RAM we could meet the deadlines. 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 migrate to the disk depending on the (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 ARTDBs. Making these system data structures persistent in NV-RAM results in better performance.

4 Data Characteristics and their Impact

Figure 2 shows the different dimensions of the characteristics of data in a real-time active database. The time taken to read, modify, and commit (write) data resident at different levels of the hierarchy is shown in the figure. The relationship of hign and low *avi* to read, write and modify times is also shown. The characteristics of a particular type of data determines where we place the data and how we process it.

- *Temporality* of data refers to the temporal validity properties of the data[3]. Since (some of) the data in an ARTDB represents the state of the controlled environment and the environment changes with time, the data becomes outdated with the passage of time. Hence, *absolute validity intervals* or *avi*s are associated with data such that as long as the current time falls within the *avi* of a data item, the data is considered valid.

  - 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. Hence, the placement, logging and recovery and concurrency control technique would depend on the other attributes of the data. Customer records in a Network Service Database application [12] are examples of non-temporal data.

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

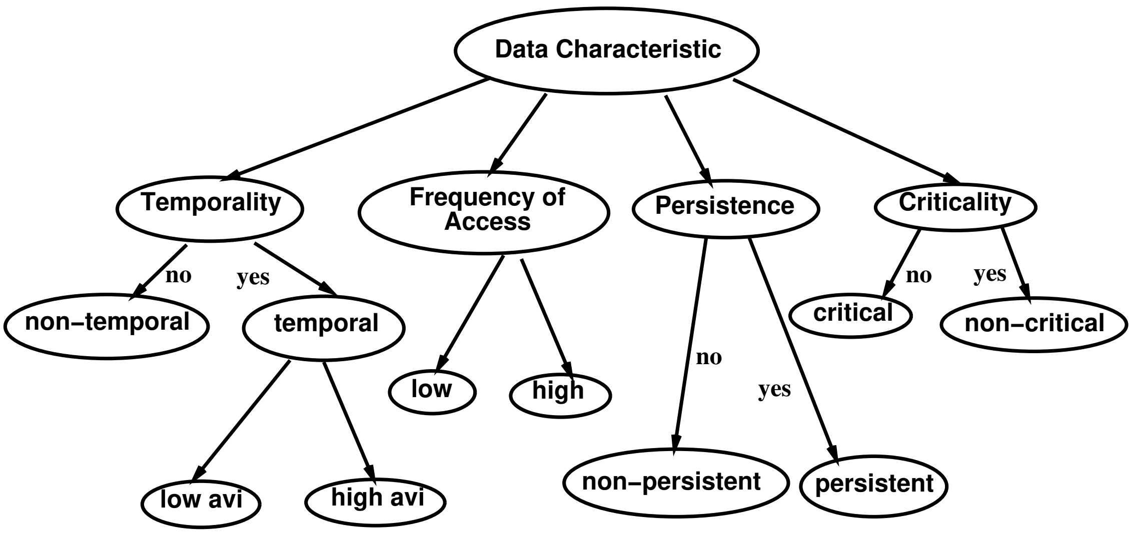

Figure 2: **Dimensions of Data Characteristics**

– Temporal Data

Data with this attribute has temporal validity intervals attached to it. Data is valid only within this interval. The sampled (or calculated) value of an attribute of the controlled system that is represented as data in the controlling system's database should not lag behind the *current* value of that attribute by more than a specified time. This specified time is the *absolute validity interval* (*avi*). There are other dimensions of temporality such as the data being *primary* or *derived* [15] and *periodic* or *aperiodic*. But these attributes may not affect the data placement, logging and recovery *as much as* the *avi* associated with the data.

We divide the temporal data into two categories, low *avi*, and high *avi* depending on the *avi* associated with the data[4]. Since we are examining data characteristics with respect to the processing of transactions that operate on this data, we characterize the *avi* of a data item as low (a small interval) or high (a large interval) relative to the time taken by a transaction to read, modify, and commit if the data is disk-resident. We use figure 3 to illustrate this point. It should be noted that it is possible to come up with more categories of data depending on the relationship between the *avi* and where the data can reside, but even this coarse categorization is sufficient

[4]The categorization of this attribute as well as for the others that follow is kept coarse to keep the discussions simple. Obviously, the range of *avi* values can be large and a spectrum of possible approaches are needed in general.

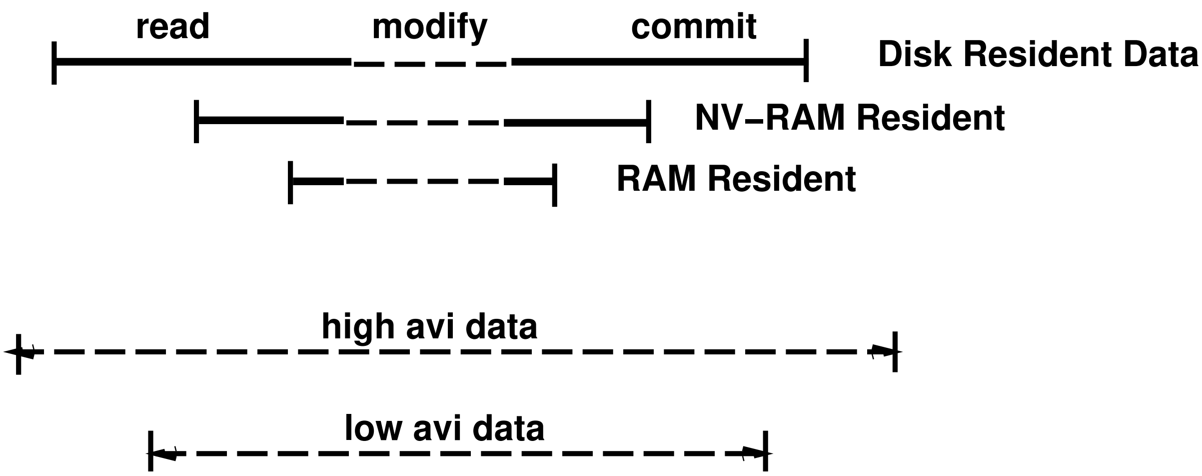

Figure 3: **Relationship between *avi* and Storage Speeds**

to point out the implications of *avi* values on ARTDB techniques. The Network Traffic Management data in the network Service Database is an example of data with temporal validity.

* Low *avi*

  The *avi* of the data item is much lower than the time taken to read-modify-commit disk resident data. Such data 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 is not necessary to write a 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, perform undo operations, and get the before image that version would have become invalid, and, by the time we redo and get a version, the chances are likely that the version would have become invalid.

* High *avi*

  The *avi* of this kind of data is much greater than the time to read-modify-commit disk resident data. Data with high *avi* may be stored on disk as the transactions are generally able to access, modify and commit the data from the disk before the validity expires.

An important question arises in this context if the 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 moved to the persistent store and retrieved again. It is possible that stolen data with not very low *avi* can be moved to NV-RAM and data with high *avi* can be moved to disk.

- *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 we usually call *hotspots*. For performance reasons one could conceive of having this data always in the main memory, that is, never allowing it to be stolen, and if persistence is necessary then it could be made persistent in 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 needed if we do not let this data be *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 we do conventional undo/redo logging. We can bring this data into main memory when needed. 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 in 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 could just reside in main memory until its temporal validity expires. However, 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; other related data touched by the same transaction that touched the critical data can be recovered later. To ensure that transactions that access critical data meet their deadlines, such data is best placed in main memory.
  - 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 (e.g., of the critical data only) in time and resource constrained situations. With traditional (sequential) logs partial recovery implies traversing the log a greater number of times than would be necessary by doing traditional recovery. One way to achieve partial recovery with low overheads is to cluster logs and chain them such that we can recover sets of data in the order of their importance. This kind of logging and recovery scheme is also amenable to parallelism. For faster recovery, one could store the separate logs on multiple disks.

  In addition, separate checkpoints can be created, each corresponding to a different criticality level of data. Having a separate log for each category helps to speed up 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 some combination of data characteristics and the data placement and logging and recovery techniques that they require.

- Low *avi*, high frequency of access, non-persistent, critical data
  Critical sensor data updated at frequent intervals exemplifies this data. This kind of data is placed in main memory. If the storage capacity of main memory is limited, the data might temporarily migrate to NV-RAM if the *avi*, while low, is loose enough for at least an NV-RAM write and read. A no-steal buffer policy is to be used. There is no need to undo. In addition, given the low *avi*, redos are not needed or feasible.

- High *avi*, high frequency of access, persistent, critical data
  Reactor temperatures in a chemical plant have this property. Temperatures typically

change slowly, but the values are used in many other critical calculations. Also, temperature records are maintained on disk for future trend computations. This kind of data is placed in main memory for performance reasons. A no-steal buffer policy along with the force policy is used where data is forced to NV-RAM and subsequently to disk.

- High *avi*, low frequency of access, persistent, noncritical
  This kind of data can be disk resident and is 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 we want the data. Instead of triggering a transaction to acquire the data just before it is needed, the transaction is triggered earlier, at some opportune time.

Before we end this section, 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 we come to know that a certain data item becomes a hotspot then it is preferable to keep the data in main memory. Let us assume we were doing undo-redo logging for this data item. Now, since we know that it is a hotspot we could change the kind of logging we were doing, i.e., we could start doing redo only logging, saving space and transaction recovery time.

5 Transaction Characteristics and their Impact

A deadline is probably the most often considered transaction attribute for real-time transaction processing. The other attributes that have been considered are criticality and length of transactions. Most real-time transaction processing is priority based and the priority of a transaction is usually a function of its deadline, criticality and length. Priority based scheduling and concurrency control is a well researched area. However, explicit data placement and logging and recovery decisions based on the priority of transactions have not been studied. Also, the kind of data touched, the kind of logging done and the recovery cost of transactions have not been

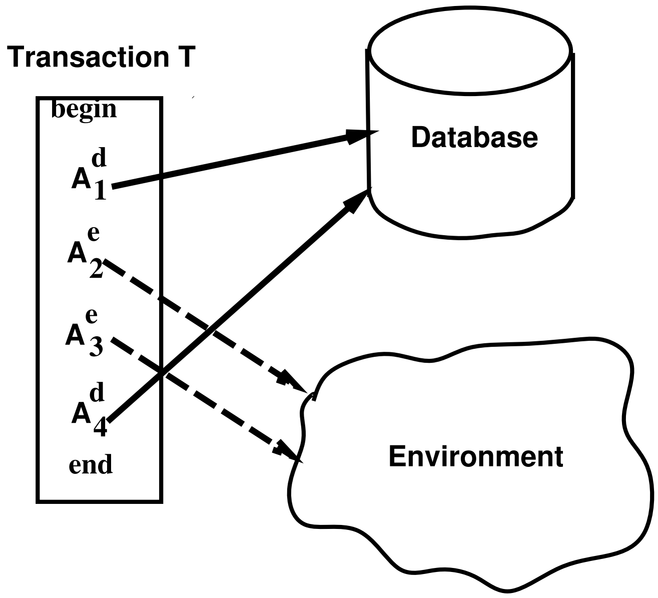

Figure 4: **An Integrated Transaction**

explicitly considered in transaction processing. In this section, we first consider the characteristics of data, in particular, environmental data versus archival data accessed by transactions to show how this data introduces special considerations. Next we consider scenarios where taking recovery costs into consideration during transaction aborts may result in better performance. We also briefly discuss the impact of transaction priorities on the placement of data.

5.1 Processing Conventional and Controlling Transactions

As mentioned earlier, the transactions in ARTDBs not only change the database, but can *physically* affect the environment. In other words, the transactions can perform actions for which there is no undo in a conventional transactional sense, but can only be compensated by running compensating transactions. For instance, let us assume we added some chemical in the nuclear reactor as a part of a transaction. Undoing the transaction would not only involve *undoing* the data that models the fact that the chemical was added, but also taking compensating actions such as removing the chemical, perhaps by adding a nullifying agent. We call such transactions that update the database as well as perform actions that affect the environment as *integrated* transactions to distinguish them from conventional transactions that only modify archival data that can be undone and controlling transactions that only affect the environment. When we say that these integrated transactions perform actions that affect the environment, we mean that the action of the transaction *could* affect the environment even before the transaction commits.

Figure 4 shows an example of such a transaction $\mathbf{T}$, which consists of four actions A_1^d, A_2^e, A_3^e and A_4^d. Actions A_1^d and A_4^d affect the database, and actions A_2^e and A_3^e affect the environment. In an active database the actions of integrated transaction can be modeled as an immediate transaction or an independent transaction depending on the semantics of the action being taken. But there is a problem with this approach. Modeling it as an immediate transaction implies that action will commit (in a traditional sense) if and only if the parent commits. Modeling it as an independent transaction implies that the logging and recovery of both the transactions is going to be independent. What we need is a new kind of semantics where the actions affecting the environment have to be made persistent (committed), but will be compensated (if necessary) if the parent transaction aborts. Extended transaction models such as Sagas can be helpful here [8].

Distinguishing between the two kinds of actions, i.e., one kind that affects the database and the other that affects the controlled system (environment), can be helpful in exploring the logging and recovery semantics that are associated with such actions.

Let us look at some examples.

- For action A_1^d that affects the database, logging/recovery can be based on the conventional undo/redo model. Optimizations such as not doing the undo logging or the redo logging depending on data characteristics and buffer management policies is possible for such actions.

- In the case of an action that affects the environment such as A_2^e, it is necessary to log the action in persistent storage, unless it is possible to come back after a crash recovery and detect in a reasonable amount of time from the environment that such an action has been taken. Even in the case where detection is possible, it might be necessary to undo actions that affected the database and compensate for the actions that affected the environment together because they belong to the same transaction. In such a situation it might be necessary to know which transaction performed the action that affected the environment. Hence, it might be necessary to log the action to perform the proper undos and compensations.

- Let us assume that transaction $\mathbf{T}$ is critical. Performing disk operations to log A_2^e and A_3^e might be too expensive. In such a case, we can use NV-RAM to log such actions.

Logging the actions of controlling transactions poses new problems. For instance, a transaction could send a signal to the environment or add a chemical to the reactor. The log has to be made stable before an action is taken so that in case of a failure the state of the system is known. We could lose state if the crashes just after making the log stable. We need some kind of fault tolerance model and techniques to support it.

5.2 Impact of Transaction Priority on Data Placement

Transaction criticality (priority) may determine 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. We could disallow stealing and thus do redo only logging for high priority transactions and do undo-redo logging for low priority transactions. This not only improves 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 we do no-undo and redo only logging for t_1 then the restart time for t_1 is low (low undo time) and, hence, the probability of meeting the deadline despite a restart is 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 we make explicit choices in logging then we could save time in logging as well.

5.3 Considerations in Aborting Transactions

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 we have to abort one of the transactions then we could abort a transaction that touches a data item that is 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 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. 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.

6 Conclusion

In this paper we looked at the characteristics of data that are present in real-time active database systems and discussed how to do data placement, logging and recovery to meet the performance requirements. We also discussed transaction characteristics that can influence the data placement, logging and recovery in ARTDBs. We briefly saw that clustering of logs and data is an important issue that can have great impact on the performance. There are other issues such as media recovery and checkpointing that have to be looked at in future.

Another important property of ARTDB transactions that is worth exploiting relates to the *consistency* of data. For instance, for many applications, it is possible to relax the isolation[5] property [16]. If we know that some inconsistency can be tolerated or that a certain data needs to be consistent only at specific points in time then we could allow more concurrency and also avoid expensive logging procedures.

The unit of consistency is yet another dimension of the consistency attribute and it can be used for the placement of data. For instance, say, there are two sets of data, critical and the non-critical set and the data belonging to either set has to be consistent only with other data items in the same set. The logs of the sets can be placed (say, on different stable storage media) in such a way that we recover the critical set first and we recover the non-critical set later.

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*, 1989.

[2] S. Chakravarthy, "A Comparative Evaluation of Active Relational Databases", Tech. Report UF-CIS-TR-93-002, Computer and Information Sciences, University of Florida, Gainesville, Jan., 1993.

[3] P. K. Chrysanthis, "ACTA, A Framework for Modeling and Reasoning About Extended Transactions", *COINS Technical Report 91-90*, UMass, Amherst, Sept., 1991.

[4] P. K. Chrysanthis and K. Ramamritham, "Synthesis of Extended Transaction Models using ACTA", *ACM Transactions on Database Systems*, Sept. 1994, pp. 450-191.

[5] Isolation of the Atomicity, Consistency, Isolation and Durability (ACID) properties of transactions.

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

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

[7] U. Dayal, M. Hsu, and R. Ladin, "Organizing Long-Running Activities with Triggers and Transactions", ACM 1990.

[8] A. K. Elmagarmid, Editor, "Database Transaction Models for Advanced Applications", Morgan Kaffman Publishers, 1993.

[9] 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.

[10] 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.

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

[12] 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.

[13] 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.

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

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

[16] K. Ramamritham and Chrysanthis, P. K. "A Taxonomy of Correctness Criteria in Database Applications", *VLDB (Very Large Data Bases) Journal*, (to appear) 1995.

[17] R. M. Sivasankaran, B. Purimetla, J. Stankovic, and K. Ramamritham, "Network Services Databases - A Distributed Active Real-Time Database (DARTDB) Application", *IEEE Workshop on Real-Time Applications*, May 1993.