Research Issues in Databases for ARCS: Active Rapidly Changing data Systems

Anindya Datta
Dept. of MIS
University of Arizona
Tucson, AZ 85721
adatta@loochi.bpa.arizona.edu

April 10, 1994

Abstract

We identify an emergent class of database systems that has not been dealt with extensively in the literature that we call ARCS (Active, Rapidly Changing data Systems) databases. These systems impose certain unique requirements on databases that monitor and control them. These requirements are such that traditional data and transaction management models appear inadequate. We present an analysis of data and transaction characteristics in ARCS systems and identify relevant research issues.

1 Introduction

In this paper we analyze the characteristics of an emergent class of database systems that we call ARCS (Active Rapidly Changing Systems). Examples of such systems include telecommunications network management systems, manufacturing systems, process control, air traffic control. Typically in such systems, the database models a real world environment where the state changes rapidly (i.e., link parameters in a network) and it is required to take corrective/restorative actions when unacceptable events occur in the environment (e.g., link utilization going beyond unacceptable bounds). The general system model that has been proposed for such systems include the following features:

- The database is the repository of all system information that need to be accessed or manipulated.
- Monitoring tools (commonly known as sensors) are distributed throughout the real system being modeled. These sensors monitor the state of the system and report to the ARCS database. Such state reports arrive at the database in very high frequency (e.g., each sensor reports every 60 seconds)
- The correct operation of the system requires the application of controls i.e., in the event of semantically incorrect operation of the system, certain actions need to be taken. We call such actions control actions. Such actions are taken from within the ARCS database (where active objects signal events) by automatic control mechanisms and are communicated to the real system using database *executors*.
- Typically, control actions need to be performed under temporal constraints, e.g., it the temperature in a reactor has reached 80 deg, certain actions need to be taken within 60 seconds.

In order for a database to monitor and control such a system, it needs *active, temporal* and real time capabilities. In this paper we identify the data and transaction characteristics of ARCS databases, and discuss relevant research issues.

2 Data Characteristics in ARCS

We classify ARCS data into two main categories:

2.1 Sensor Data

Sensor (or measurement) data is the raw information that is received from the system monitoring processes. Since ARCS systems tend to be extremely dynamic, the value of sensor data items usually change very rapidly. Typically sensor data represents dynamic performance characteristics of the system. For example, in the case of network management databases, the sensor data may include variables such as node queue lengths, retransmission rates, link status and call statistics. One assumption we make is that sensors are distributed and work in their own private data partition – thus a particular instance variable is always reported by one particular sensor. Usually each sensor's data arrives in the database at a regular frequency under normal system operation. However, under extraordinary conditions (e.g., fault conditions in a network) sensors may generate data at a higher rate than normal. Sensor data, though in many respects similar to notions of data in real time systems, has certain differences. For example, it is required, for certain data to be persistent e.g., security violations, customer billing information etc. In typical ARCS databases (such as one for network management) several gigabytes of sensor data need to be processed per day [3].

2.2 Non Sensor Data

These are data items that are more "permanent", in the sense that non-sensor data items retain their value for longer durations. In that sense these data items resemble the notions of data in classical databases. We classify non-sensor data into two further subclasses.

2.2.1 Structural Data:

In contrast to sensor data, structural data is composed of "static" (slowly changing) information. In the case of network management databases, such information may include network topology, the configuration of network switches and trunks, the data encryption keys etc. Most of the structural data is stored at system initiation time and is changed at a moderate rate consistent with classical database applications such as banking. An important point to note is that unlike sensor data, structural data is valid even when the system is not in operation.

2.2.2 Control Data:

The final and very critical category of data is control data which captures the setting of the system tuning parameters. It is this class of data that characterizes ARCS databases as not only repositories of data but also as control systems. In network management databases control data are information like maximum flows in individual trunks, the traffic split ratios on output links of switches, the routing table etc. Control data is changed from within the system and such changes affect the operational dynamics of the network. For example, changing the maximum flow rate on a particular link may increase link utilization on that link while decreasing delay for the overall system. The process of changing an existing set of control settings may be initiated in two ways:

- By human operators;
- By automatic triggering of control processes as a function of the information contained in the sensor data;

In addition to the above classification, we also view ARCS data from another perspective. There is a class of data items, the updating of which may invoke control actions. These are called *reactive* items. An example of a reactive attribute for a network management database is the *link-utilization* data item for any link object. Updating of the *link-utilization* data item invokes rules such as **if** *link-utilization* **is greater than 80%, reduce** *transmission-rate* **on attached nodes**. As we will see in the next section, the updating of reactive attributes leads to the generation of control transactions in the system.

3 Transaction Characteristics

Examination of transaction characteristics yields three different types of transactions:

1. State Reporting Transactions (SRT): SRTs are transactions that are generated by sensors dispersed throughout the entity being modeled (e.g., a heterogeneous communications network). These transactions report the current states of real world objects and thus consist purely of $SET(X)$ operations. In most dynamic systems these transactions are generated at a high frequency (e.g., every 15 sec.). These transactions do not conflict with each other as the network monitoring tools (i.e., sensors) work in private data partitions. Also these updates are different for conventional database updates in that the updated value is independent of the current value of the data objects. These are referred to as blind writes [2]. Thus, due to the absence of write-write conflicts between these transactions no explicit concurrency control is necessary to prevent such conflicts. Adopting a data centric view, it may be stated that since these transactions work on dynamic performance data no explicit concurrency schemes need to be defined for these data items. However, since it is important to ensure that the database project a coherent state of the real world, SRTs need to be atomic, i.e., we cannot allow these transactions to commit their operations on some objects and not on others.

2. Querying Transactions (QT): QTs are generated by readers of MIB data – typically human users, and consist of $GET(X)$ operations. QTs may want to read both sensor as well as non-sensor information. Certain interesting scenarios arise when QTs want to read sensor data, i.e., conflicts arise between operations of QTs and SRTs. As we show in section 4.1, enforcing serializability to resolve these conflicts appear to be too restrictive. Typically, QTs are expected to fulfill two conditions:

- The Coherency Condition: The values returned by QTs (of sensor data) must reflect a coherent state of the database. However, this notion of coherency is somewhat different from the standard definition of consistency. Notions like *snapshot consistency* [5] may be used.
- The Recency Condition: The values returned by QTs must reflect as recent a state of the real system as possible. This is a problem in the case of sensor data that changes very fast – we show later that standard concurrency control mechanisms do not provide adequate recency.

3. Control Action Transactions (CAT): CATs are transactions that are generated due the invocation of automatic control actions inside the database. Please note that control transactions may also be generated due to operator intervention. The goal of of active systems is to reduce the need of such interventions so that the system itself may take care of situations. In such situations it is of critical importance to ensure the correct behavior of the system.

CATs are generated by the updating of reactive attributes (usually by SRTs but also potentially by other CATs). Such updates generate events and rules and the CATs are generated through the ACTION part of rules. In such situations there is a potential of *transaction spreading* i.e., where the effect of one CAT invokes one or more other CATs.

It is clear from the above discussion that CATs exhibit characteristics that fit the definition of nested transaction models [12]. This is so because, the generation of a CAT may lead to the updating of other reactive data items which may lead to the generation of other CATs and so on.

4 Relevant Issues in ARCS Database Research

4.1 Why are Conventional Transaction Models Not Suitable for ARCS Transactions?

Conventional transaction models (i.e. commutativity based models) appear too restrictive for ARCS because of several reasons, especially with respect to the execution of SRTs and QTs. We provide two reasons below in sections 4.1.1 and 4.1.2.

4.1.1 Irrelevancy of Cascading Aborts

The issue of *cascading aborts* (or the avoidance of this phenomenon) has been one of the fundamental guiding forces behind the development of conventional transaction models of concurrency [7]. Consider for example, the following execution subhistory of three transactions[1]:

$$read_1(x,1), write_1(x,4), read_2(x,4),$$
$$write_2(y,7), read_3(y,7), write_3(z,8)$$

In the above history, according to conventional transaction models, if transaction 1 (t_1) aborts, we must remove its effects from the database. However, since t_2 read (through the operation $read_2(x,4)$) the value of a data item written by t_1, effectively it read a value that does not really exist (as the

[1]Please note that in this execution sequence we assume *immediate effect* of operations, i.e., the effects of an operation are visible immediately. A similar analysis may be provided assuming *deferred effect*

effect of t_1 is eliminated). Thus t_2 would need to be aborted. Following the same logic, t_3 would now need to abort as it read uncommitted update of t_2. This phenomenon is known as cascading aborts and it helps shape the notion of *failure atomicity* in conventional transaction models. As shown in the following example, because of particular semantics of ARCS data and transactions, cascading aborts is not an issue when dealing with sensor data.

Example: Consider an object K, with sensor data item X and non-sensor data item Y. Further assume that X is a reactive attribute, i.e., when X is updated, there is potential for rule activation and consequent generation of CATs. Now consider the following interleaving of two transactions t_i and t_j, that both access object K.

$$SET_i(X, v_1), SET_j(Y, v_2), abort_i$$

We provide the following semantic interpretation for the above interleaved sequence of operations: t_i is a SRT, which sets the value of data item X to v_1. X being a reactive attribute, this update leads to rule activation and generation of the CAT t_j. This CAT takes the control action of setting control data item Y to v_2. Subsequently t_i aborts. According to the theory of cascading abort, t_j should also abort now, as a dependency exists between t_j and t_i. However the semantics of sensor data are such that t_j need not abort. This is because:

- t_i reports the state of the system. Thus, in spite of the abortion of t_i, the fact that the state of X is v_1 in the real system, is true. Thus the reason for the invocation of t_j still holds good (remember t_j is a CAT invoked as a result of X being set to v_1). To consider a real life example, in the network management scenario, assume X refers to the data item *link-utilization*. Needless to say, the values of *link-utilization* for all links is rapidly changing, and is being reported by SRTs. *link-utilization* is also a reactive attribute, as there exists a rule of the form **if link utilization on any link exceeds 90%, reduce the split ratio on that link by 10%**. Thus every time *link-utilization* is updated this rule is checked. Now, it is easy to see the validity of the general example given above, by substituting $Y = split - ratio$, and $v_1 = .92$.

4.1.2 The Notion Correctness of Execution of a Set of Transactions

In traditional transaction models, the notion of correctness for the execution of a set of transactions is *serializability*. Serializability requires that an execution be equivalent to one that would have occurred without any concurrency in the system. Several variations of the serializability paradigm have been proposed [1, 6]. However, most transaction models require some sort of serialized execution. In other words, users (transactions) must not see the effects of concurrent execution. However, the semantics of ARCS transactions validate the correctness of *non-serialized* execution, as shown in the example below.

Example: Assume the following interleaved subhistory of the concurrent execution of two transactions t_i and t_j. Further assume that t_i is a SRT while t_j is a QT.

$$set_i(X, v_1), get_j(X, v_1), set_i(Y, v_2),$$
$$get_j(Y, v_2), commit_j, ...$$

The above execution is clearly not a serializable execution as the effect of t_i is visible to t_j while they are executing concurrently. However, from the ARCS perspective it is an allowable schedule because t_j *is* reporting a coherent state of the systems as it returns values that occur together in the system. Actually, enforcing serializability in this context would only lead to decreased flexibility as it would stipulate unnecessary restrictions on scheduling. A corollary to the above argument is that in ARCS scenarios we have a different notion of *consistency* than traditional systems. This issue is explored in the next section.

4.2 Notions of Consistency in ARCS Databases

Apropos the above discussion, ARCS consistency is different from consistency in classical database application, where satisfaction of the ACID properties ensures consistency []. Clearly, one measure of ARCS database consistency is how closely it represents the real environment being monitored and controlled. This is, in many respects similar, to notions of *temporal consistency* [17], where two components of consistency are defined:

- **Absolute Consistency:** Keeping state of the database objects within certain temporal limits of the state of the corresponding environment objects. This limit is called the *absolute validity interval.*

- **Relative Consistency:** A *Relative Consistency Set* R, is defined to a set of data items used to derive a new data item, e.g., *connectivity* and *current link utilization* information may be used to derive *routes* for network traffic. Relative Consistency is involved in keeping the base data items (i.e., members of R) within certain temporal limits of each other. This limit is termed the *relative validity interval.*

These ideas, though important to consider in ARCS databases, need to be significantly expanded as they appear not take into account the active and real-time dimensions of ARCS databases. For instance, taking the active nature of ARCS into account would change the notion of absolute temporal consistency criterion which is defined as [13]:

$$(current\_time \ - \ d_{timestamp}) \leq d_{avi}$$

where d is any data item, and d_{avi} denotes d's absolute validity interval. This definition implies that the state of environment objects *always* changes before that of database objects, i.e., the database objects always lag the real objects in the outside world. This is true in a passive monitoring systems. However, in ARCS databases, events in the outside world may trigger actions in the database which may change parameters in the database *first*, which would subsequently be propagated to the environment.

Also, absolute and relative consistency definitions do not explicitly consider real time characteristics [13] of ARCS, i.e., they do not explicitly consider the time constraints on query and transaction processing. Thus, in temporal databases, when environment state changes, it does not cause an inplace update of data base object, but rather involves creating a new version. Real time systems however, have viewed an environment change as causing a modification in corresponding database entities. Also, transactions are usually allowed to access only current, valid information in the database. In ARCS however, it is required to store historical as well as current data, while allowing modifications to both data classes.

5 Conclusion: Concrete Research Questions in ARCS Databases

5.1 An Active, Temporal, Real Time Data Model

Clearly, an expressive data model is the first order of business. Modeling activity for ARCS can benefit a great deal from the recent upsurge in efforts to provide database support for complex new applications such as CAD and scientific databases. Also, there exists several data models for active databases [8, 11] as well as temporal databases [14, 15, 16]. Recently, there also has been some attempts to incorporate real time characteristics in temporal databases [13], integrating active and temporal databases [9, 10] as well as integrating real time and active systems [4]. There appears to be a consensus among researchers, that complex data models should be specified in the context of simpler, well established ones such as relational and object-oriented models. Object-Oriented (OO) models appear to be a good starting point for ARCS databases, as they naturally model complex application domains such as network management or manufacturing. Thus, an approach with promise would be to take the basic OO model and enhance its functionalities. Certain specific questions that need to be answered are:

1. How to model versions of active, real time objects?

2. How do we represent complex triggering constraints (i.e., the violation of which will trigger CATs) and temporal constraints? Possible ways range from representing constraints as rules and storing them as first class objects that may be manipulated like environmental objects (weak coupling) or modeling constraints as database consistency constraints (strong coupling) (i.e., if a constraint is violated database is inconsistent) and letting the concurrency control mechanism handle restorative action. The first approach has been explored by researchers [4, 11] while we have postulated the second approach. The second approach appears to be promising for reasons beyond the scope of this paper.

5.2 An Active, Temporal, Real Time Transaction Model

This is yet a wide open research area. In section 3 we briefly identified classes of ARCS transactions and in section 4.1 we showed some examples of the unsuitability of serializability based models. Some specific questions that need to be answered are:

1. Under ARCS conditions do transactions preserve global consistency? In other words, does the traditional definition of transaction hold for ARCS databases? Assume that an event happens in the environment that violates some consistency constraint. The SRT that reports this to the database is going to render the database inconsistent, even though this inconsistency is simply a reflection of the "true" state of the environment. In order to handle situations like these, it may be necessary to rethink the definition of a "correct" transaction in a ARCS database.
2. How to formulate scheduling algorithms that take into account timing constraints as well as resource constraints?
3. How to do contingency planning, i.e., in the face of temporal constraints how does the system choose the "optimal" control action? This implies the need for a mechanism to perform a priori time estimates of actions, i.e., a mechanism to estimate upper bounds of transaction execution times..

References

[1] F. Bancilhon, W. Kim, and H.F. Korth. A Model of CAD Transactions. In *Proceedings of VLDB, Stockholm*, pages 25–33, September 1985.

[2] P. Bernstein, V. Hadzilakos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, Reading, MA, 1987.

[3] Sprint Network Management Center. Site Visit, April 1992.

[4] S. Chakravarthy and D. Mishra. Snoop: An Expressive Event Specification Language for Active Databases. Technical Report UF-CIS-TR-93-007, Dept. of CIS, University of Florida, 1993.

[5] A. Datta. A Semantic Transaction Management Model for ARCS - Active, Rapidly Changing data Systems. Working Paper, MIS Dept., University of Arizona, 1993.

[6] W. Du and A.K. Elmagarmid. Quasi Serializability: a Correctness for Global Concurrency Control in Interbase. In *Proceedings of the Fifteenth International Conference on Very Large Data Bases*, pages 347–355, 1989.

[7] A.K. Elmagarmid, Y. Leu, J.G. Mullen, and O. Bukhres. Introduction to Advanced Transaction Models. In A.K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, pages 34–52. Morgan Kaufman, 1989.

[8] O. Etzion. PARDES - A Data Driven Oriented Active Database Model. *SIGMOD-RECORD*, 22(1):7–14, 1993.

[9] O. Etzion, A. Gal, and A. Segev. Temporal Support in Active Databases. In *Proceedings of the Second Workshop on Information Technologies and Systems (WITS)*, December 1992.

[10] O. Etzion, A. Gal, and A. Segev. Temporal Active Databases. In *Proceedings of the International Workshop on an Infrastructure for Temporal Databases*, June 1993.

[11] N.H. Gehani and H.V. Jagadish. Ode as an Active Database: Constraints and Triggers. In *Proceedings of VLDB, Barcelona*, September 1991.

[12] E. Moss. *Nested Transactions*. MIT Press, Cambridge, Massachusetts, 1985.

[13] K. Ramamritham. Time for Real-Time Temporal Databases. In *Proceedings of the International Workshop on an Infrastructure for Temporal Databases*, June 1993.

[14] E. Rose and A. Segev. TOODM - A Temporal Object-Oriented Data Model with Temporal Constraints. In *Proceedings of the 10th International Conference on the Entity-Relationship Approach*, pages 205–229, 1991.

[15] E. Rose and A. Segev. A Temporal Object-Oriented Algebra and Data Model . Technical Report LBL-32013, Lawrence Berkeley Laboratories, 1992.

[16] R. Snodgrass and I. Ahn. A Taxonomy of Time in Databases. In *Proceedings of ACM SIGMOD*, June 1985.

[17] X. Song and J.W.S. Liu. How well Can Data Temporal Consistency be Maintained. In *Proceedings of the IEEE Symposium on Computer-Aided Control Systems Design*, 1992.