Achieving Bounded and Predictable Recovery using Real-Time Logging

LihChyun Shu
Dept. of Information Management
Chang Jung University
Tainan County, Taiwan 711, ROC
shulc@mail.cju.edu.tw

John A. Stankovic and Sang H. Son
Dept. of Computer Science
University of Virginia
Charlottesville, VA 22904
{stankovic,son}@cs.virginia.edu

Abstract

Real-time databases are increasingly being used as an integral part of many computer systems. During normal operation, transactions in real-time databases must be executed in such a way that transaction timing and data time validity constraints can be met. Real-time databases must also prepare for possible failures and provide fault tolerance capability. Principles for fault tolerance in real-time databases must take timing requirements into consideration and are distinct from those for conventional databases. We discuss these issues in this paper and describe a logging and recovery technique that is time-cognizant and is suitable for an important class of real-time database applications. The technique minimizes normal runtime overhead caused by logging and has a predictable impact on transaction timing constraints. Upon a failure, the system can recover critical data to a consistent and temporally valid state within predictable time bounds. The system can then resume its major functioning, while non-critical data is being recovered in the background. As a result, the recovery time is bounded and shortened. Our performance evaluation via simulation shows that logging overhead has a small effect on missing transaction deadlines while adding recovery capability. Experiments also show that recovery using our approach is 3 to 6 times faster than traditional recovery.

1 Introduction

In recent years, with the advances in hardware and networking technologies, more and more real-time database (RTDB) applications are emerging. Many of these applications, such as air traffic control, network management, internet programmed trading, and command and control systems demand predictable low-latency access to data, coupled with stringent durability and availability requirements. Data in such applications often has distinguishing characteristics [9]. For instance, some data has temporal validity intervals associated with them. During a validity interval, a particular data value is deemed useful as far as the application's semantics is concerned. Data gets out of date by the simple passage of time. On the other hand, some real-time data is more critical to the operation of a real-time application than other data. As an example, consider an internet programmed trading application where transactions are submitted from around the world 24 hours a day. In such a system, customer balances constitute critical data and the state of each customer's balance will remain valid until a transaction is issued to change it. This data does not become invalid simply with the passage of time. On the other hand, stock market prices are also critical, but have finite validity intervals; a stock price that is too old is worthless. Following the terminology used by Ramamritham in [5], we call those data whose values will not change with time or whose validity intervals are infinite *invariant* data. Other data is called *variant* data.

When a failure occurs in these applications, it is important that the system can recover critical data to a consistent and temporally valid state within predictable time bounds. The system can then resume its major functioning, while non-critical data is being recovered in the background. Principles that are commonly addressed in conventional database failure recovery do not take real-time requirements into consideration. In this paper, we first define principles which are appropriate for logging and recovery in RTDB ap-

This work was supported, in part, by NSF grant EIA-9900895 and by NSC grant 88-2213-E-309-002.

Proceedings of the Eighth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'02)
0-7695-1739-0/02 $17.00 © 2002 IEEE

plications. We then present a logging and recovery technique that is time-cognizant, supports these principles, and is suitable for a class of real-time database applications, such as internet programmed trading. The key features of our scheme are as follows:

- We allow a system designer to specify which data is critical to the system's major operation. The types of data remain unchanged during system's operation.
- We partition the log across critical and non-critical data segments. Partitioning the log this way allows critical data to be recovered independently. Transactions that only access critical data can start executing before non-critical data has been recovered.
- In order to reduce and bound logging overheads and post-crash recovery time, we store critical data in non-volatile RAM.
- We employ different logging strategies for different data types. For example, the temporal property of variant data is exploited so that we can design a no-undo/with-redo logging algorithm for variant data. The benefit is that log records with invalid data can be reclaimed with minimum effort (checkpointing is done for free). For data without the temporal property, we design a no-redo/with-undo algorithm in such a way that system-wide checkpointing is avoided.
- We clearly characterize the impact of logging, commit processing, and recovery on satisfying transaction timing constraints and post-crash performance requirements.

We organize this paper as follows. Section 2 discusses related work. Section 3 describes basic concepts and introduces some terms used in the remainder of the paper. Section 4 discusses principles that are appropriate for logging and recovery in RTDB applications. Section 5 details our logging and failure recovery algorithm, its characteristics, and advantages. Section 6 presents the performance evaluation of our approach for various system parameters. Section 7 concludes the paper.

2 Related Work

Real-time logging and failure recovery addresses the problem of restoring a RTDBS upon a memory failure or crash, using logs created during normal operation, to a consistent state so that a recovery time requirement can be met. During normal runtime, one also wants to minimize runtime effort due to logging and have a predictable impact on transaction timing constraints. Research in this domain distinguishes itself from that in traditional database recovery mainly in the following two requirements: first, upon a crash, it is important for a RTDBS to predictably recover within a pre-determined time bound and be responsive to the environment again; second, when the system resumes its operation, data in a RTDBS must be temporally valid or be made so. Hence, while some data may be recovered from existing data in the logs, other data must be refreshed by reading environment states, e.g., reading the current stock prices. When these requirements are satisfied, one can then be dealing with controlling and/or monitoring real-time environments.

Considerable research has been done in traditional database recovery. Two recent books by Kumar and Hsu [2] and Kumar and Son [3] discuss recovery in detail and contain descriptions of recovery methods used in a number of existing relational database products. Research in traditional database recovery generally places emphasis on performance issues. Timing predictability is seldom a concern in this field. As an example, consider the ARIES recovery algorithm [4] which has been quite successful in practice. While ARIES has employed several novel techniques to optimize its recovery-time performance, it cannot ensure that restored data is temporally valid, nor can it ensure that the system will come back on line in time. In short, what ARIES lacks for a RTDB are time cognizant protocols for logging and recovery.

Compared to research done in traditional database recovery, relatively little research has been done for recovery in RTDB. Sivasankaran et. al.[8] look at the characteristics of data that are present in real-time active database systems and discuss how to do data placement, logging and recovery to meet the performance requirements. They also discuss transaction characteristics that can influence the data placement, logging and recovery in real-time active databases. Sivasankaran et. al.[7] show the need to design novel logging and recovery algorithms by observing the "priority diversion" problem[1] where conventional logging and recovery algorithms are not suitable in a priority oriented RTDB setting. They present a taxonomy of data characteristics and propose two data classes that are derived from data types and transaction types. They also develop a

[1] Priority diversion occurs when high priority requests do work for low priority requests, e.g., committing a transaction T by flushing log records belonging to both T and other lower priority transactions.

Proceedings of the Eighth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'02)
0-7695-1739-0/02 $17.00 © 2002 IEEE

suite of algorithms targeted at RTDB. The major differences between our work presented in this paper and that presented in [7] are the following: first, we propose principles underlying real-time logging and failure recovery, independent of technologies used; second, one of our major concerns for logging and recovery is maintaining temporal consistency of data; third, we employ different logging strategies for different data types; fourth, we exploit application properties to improve the performance of our recovery algorithm; finally, we clearly characterize the impact of logging, commit processing, and recovery on satisfying transaction timing constraints and post-crash performance requirements.

3 Background and Assumptions

Real-time systems (RTS) must react to stimuli from the environment within time intervals dictated by the environment. Hence, the state of the operating environment is constantly monitored by a RTS. We assume the database $D = \{x_1, x_2, \ldots, x_n\}$ and a subset of the data items in D be a representation of the operating environment. Suppose $x_i, 1 \leq i \leq l$, represents x_i^e in the external environment. We term each x_i an *internal* variable and x_i^e an *external* variable.

Because the external environment changes state from time to time, we can expect that a particular state may remain valid for a limited period of time. The *temporal validity interval* of a state of an external variable is defined as the time span from the time the state is generated until the time the external variable changes to a new state. We define the temporal validity interval of an external variable x_j^e, denoted as $TVI(x_j^e)$, as the minimum of the temporal validity intervals of all possible states of x_j^e. The temporal validity interval of an internal variable x_j, denoted as $TVI(x_j)$, is defined analogously.

The temporal validity interval is a distinguishing attribute of real-time data. Real-time data has other attributes. For example, some data in D are more critical for the normal operation of real-time systems than others. We denote critical data in D as $D_{critical}$. Transactions that access real-time data can also be classified into different types. For a detailed taxonomy of attributes associated with real-time data and transactions, interested readers are referred to [7].

In this paper, we assume transactions pre-declare their data needs. We assume transactions that update critical data are also critical. Critical transactions are assumed to update only critical data. Further, we assume the number of critical transactions is known at design time. Critical transactions that update variant data are periodic. We assume variant data updated by each transaction becomes temporally valid when the updating transaction commits. Other transactions that update critical invariant data or non-critical data are aperiodic. Consider the internet trading example again. Stock prices are critical variant data. They are refreshed periodically by critical sensor transactions. Customer balances are critical invariant data and are updated by on-demand trading transactions. Other data such as the number of transactions being processed today is an example of non-critical data.

The failure model that we assume in this paper includes all kinds of failures that can bring down the system and cause all data in volatile memory to be lost, but leaves all data on stable secondary storage intact. We assume that consecutive failures are separated by at least the time needed to recover the major function of the system. Hence, no failure will occur during recovery from a failure.

4 Principles Underlying Real-Time Logging and Failure Recovery

Conventional database failure recovery is primarily concerned with keeping the database in a consistent state. Principles such as atomicity and durability commonly addressed in database failure recovery do not take real-time requirements into consideration. In this section, we describe principles which are appropriate for fault tolerance in real-time database applications.

To maintain the consistency of data despite system failures, real-time databases must perform bookkeeping activities, e.g., logging, during the normal operation of the system. Logging writes all updates to stable storage. These activities imply extra run-time overheads, which must be amenable to pre-run-time timing analysis for real-time databases. In particular, one must bound and minimize the size of each log. Further, the overheads in maintaining each log structure must be predictably accounted for. Typically, these involve both memory and I/O operations. In a nutshell, logging activities must not jeopardize the timeliness of transactions.

Most real-time data has limited temporal validity intervals. As a consequence, when a system failure occurs, we should only restore persistent data which will remain temporally valid when the system resumes execution. As in conventional database recovery, the restored value of each data item x_j must be created by the last committed writer of x_j prior to the crash in the execution history. To determine whether x_j's restored value is temporally valid, we make use of x_j's last valid time instant. However, testing temporal validity for

Proceedings of the Eighth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'02)
0-7695-1739-0/02 $17.00 © 2002 IEEE