Chapter LXXXII
A Novel Crash Recovery Scheme for Distributed Real-Time Databases

Yingyuan Xiao
Tianjin University of Technology, China

INTRODUCTION

Recently, the demand for real-time data services has been increasing (Aslinger & Son, 2005). Many applications such as online stock trading, agile manufacturing, traffic control, target tracking, network management, and so forth, require the support of a distributed real-time database system (DRTDBS). Typically, these applications need predictable response time, and they often have to process various kinds of queries in a timely fashion. A DRTDBS is defined as a distributed database system within which transactions and data have timing characteristics or explicit timing constraints and system correctness that depend not only on the logic results but also on the time at which the logic results are produced. Similar to conventional real-time systems, transactions in DRTDBSs are usually associated with timing constraints. On the other hand, a DRTDBS must maintain databases for useful information, support the manipulation of the databases, and process transactions. Timing constraints of transactions in a DRTDBS are typically specified in the form of deadlines that require a transaction to be completed by a specified time. For soft real-time transactions, failure to meet a deadline can cause the results to lose their value, and for firm or hard real-time transactions, a result produced too late may be useless or harmful. DRTDBSs often process both temporal data that lose validity after their period of validity and persistent data that remain valid regardless of time. In order to meet the timing constraints of transactions and

data, DRTDBSs usually adopt main memory database (MMDB) as their ground support. In an MMDB, "working copy" of a database is placed in the main memory, and a "secondary copy" of the database on disks serves as backup. Data I/O can be eliminated during a transaction execution by adopting an MMDB so that a substantial performance improvement can be achieved. We define a DRTDBS integrating MMDB as a distributed real-time main memory database system (DRTMMDBS).

The existing researches on DRTDBS focus mainly on concurrency control (Gustafsson, Hallqvist & Hansson, 2005; Lam, Kuo, Tsang & Law, 2000; Ulusoy, 1993), replication (Aslinger & Son, 2005; Son & Kouloumbis, 1993; Ulusoy, 1994), and commitment (Haritsa, Ramamritham & Gupta, 2000; Qin & Liu, 2003; Xiao, Liu, Deng & Liao, 2006). The studies of failure recovery for a DRTMMDBS are relatively scarce. However, due to the complexity of distributed environments together with volatility and vulnerability of the main memory, the possibility of failure in a DRTMMDBS becomes much larger than in centralized disk resident database systems. When a site crash occurs, as the databases are not available for transactions, many transactions may miss deadlines, and a large amount of temporal data may lose their validity before they can be used, so a DRTMMDBS should have the ability of high fault-tolerance and can resume services again as quickly as possible after crashes.

BACKGROUND

The traditional sequential logging and disk-based recovery techniques cannot meet the high performance requirement of a DRTMMDBS. To aim at real-time databases, some failure recovery methods have been put forward. Choi, et al. (2000) presented a parallel processing architecture adopting double-CPU. In this architecture, one CPU, which is called a DP, is responsible for usual transaction processing, while another CPU, called an RP, is only responsible for recovery processing such as logging, checkpoint, and failure recovery. This architecture has better performances than the traditional single CPU, but the utilization of RP is not high. Sivasankaran, Ramamritham, Stankovic, and Towsley (1995) analyzed the characteristics of data in real-time databases and discussed the strategies of logging and recovery in real-time active databases. To solve the problem of low efficiency of the sequential permanent logging, partitioned logging and ephemeral logging have been proposed, respectively (Agrawal & Jagadish, 1989; Lam & Kuo, 2001). Partitioned logging stores log records according to transaction class (data class); that is, the log records belonging to a different transaction class (data class) are stored in different partitions. Partitioned logging can avoid the performance bottleneck caused by severe contention for single logging store partition. However, the problems such as how to classify transactions (data), how to standardize logging partitions, and how to recover database systems to the correct and consistent state after failures must be solved for partitioned logging. Ephemeral logging does not have to keep log records permanently. For ephemeral logging, when a transaction commits, the data buffers modified by the transaction need to be flushed into stable storage devices. So the log records may be deleted as soon as the corresponding transactions commit. The advantage of ephemeral logging is that the log processing time after failures is reduced prominently, while the disadvantages lie with the lack of permanent logs and inconvenience in auditing and tracing. Liu, Amman, and Jajodia (2000) gave the technique of accelerating recovery speed, but the technique still requires stopping the system services in entire recovery process. For MMDBs, shadow paging recovery schemes have been proposed (Kim & Park, 1996). The shadow paging recovery scheme does not require writing up a log, but it requires a large amount of main memory space. Woo, Kim, and Lee (1997)