11

SCHEDULING ACCESS TO TEMPORAL DATA IN REAL-TIME DATABASES

Ming Xiong, Rajendran Sivasankaran, John A. Stankovic, Krithi Ramamritham and Don Towsley

Department of Computer Science, University of Massachusetts, Amherst, MA 01003, USA

1 INTRODUCTION

A real-time database system is a transaction processing system designed to handle workloads in which transactions have deadlines. However, many real-world applications involve not only transactions with time constraints, but also data with time constraints. Such data, typically obtained from sensors, become inaccurate with the passage of time. Examples of such applications include autopilot systems, robot navigation, and program stock trading [14]. While considerable work has been done on real-time databases, most of it assumes that only transactions have deadlines [1, 7]. New solutions that consider data time constraints are required for both concurrency control and cpu scheduling. In this chapter, new solutions are presented based on data-deadlines and time cognizant forced wait policies. Informally, data-deadline can be viewed as the deadline that a transaction implicitly gets due to the temporal constraints of the data accessed by the transaction. When a transaction attempts to access data that is about to expire, the forced wait policy delays it until the data is updated using new sensor inputs. Each of these is integrated with both earliest deadline first (EDF) and least slack first (LSF) cpu scheduling policies. It is shown that, while the data-deadline policy improves performance, the forced wait policy exhibits a more significant performance improvement. Under the conditions tested, it is also shown that LSF policies work better than EDF based policies.

The remainder of this chapter is organized as follows. Related work is discussed in Section 2. Section 3 describes the overall real-time database architecture. Section 4 discusses the main issues and solutions developed to maintain temporal consistency in a real-time database. Section 5 outlines the transaction

A. Bestavros et al. (eds.), Real-Time Database Systems
scheduling policies employed in the solutions. Section 6 discusses the results of the experimental study and section 7 summarizes and concludes the study.

2 RELATED WORK

Over the past few years real-time databases have become an important area of research. Experimental studies reported in [1, 6, 7, 8, 12, 15, 16] cover many aspects of real-time transaction processing, but have not considered time constraints associated with data.

Database systems in which time validity intervals are associated with the data were discussed in [17, 9, 10]. Such systems introduce the need to maintain data temporal consistency in addition to logical consistency. In these systems, transactions are aborted (and possibly restarted) when any of the temporal data objects they read become invalid before the transaction commits. Scheduling policies that are cognizant of transactions’ deadlines but not the validity of data may not work well because of such data related considerations. Also, it is conceivable that a transaction will not be able to commit before the validity of a temporal data object that it accesses expires. In such a case, observations from our research indicate that waiting for the next valid version of the data object can save unnecessary aborts. The solutions presented here exploit this concept. In addition, sometimes, the version of the data object read by a transaction could be very close to the current version of the same data object (i.e., the two versions are similar [9, 10]) obviating the need to abort the transaction.

In the model introduced in [17], a real-time system consists of periodic tasks which are either read-only, write-only or update (read-write) transactions. Data objects are temporally inconsistent when their ages or dispersions [17] are greater than the absolute or relative thresholds allowed by the application. Transactions access temporally inconsistent data can commit if they can meet their deadlines. Two-phase locking and optimistic concurrency control algorithms, as well as rate-monotonic and earliest deadline first scheduling algorithms are studied in [17]. These studies show that the performances of the rate-monotonic and earliest deadline first algorithms are similar when the load is low. At higher loads, earliest deadline first outperforms rate-monotonic when maintaining temporal consistency. They also observed that optimistic concurrency control is generally worse at maintaining temporal consistency of data than lock based concurrency control, even though the former allows more trans-