Proceedings of 24th Conference
on Decision and Control
Ft. Lauderdale, FL • December 1985

A STATE MODEL FOR THE CONCURRENCY CONTROL PROBLEM IN DATABASE MANAGEMENT SYSTEMS*

Stéphane Lafortune and Eugene Wong

Department of Electrical Engineering and Computer Sciences
University of California, Berkeley, CA 94720.

1. INTRODUCTION

This paper considers the concurrency control problem in database management systems from a control-theoretic approach. Its main purpose is to propose a dynamical system model for the concurrent actions of many users on a database. Concurrency control is the task of scheduling these users in order to satisfy a criterion specified below.

Despite the extensive amount of work that has been done on concurrency control, this problem had never been formulated in the framework of dynamical systems. This was the motivation for our work, and we have used such an approach to characterize what can and what cannot be achieved by existing concurrency control techniques, and, in addition, to suggest ways of improving these techniques. We have analyzed in detail the well-known *Two-Phase Locking Protocol* introduced in [1] and the recently proposed *Declare-Before-Unlock Protocol* [4]. In particular, our model shows explicitly how the two features *declare* and *must-precede graph* introduced in [4] are useful in improving the performance of the widely used two-phase locking protocol.

This summary is confined to a description of the main concepts involved in this work. Section 2 introduces some background necessary for the description of our model given in section 3. Control by locking, an important concurrency control technique, is presented in section 4. Finally, section 5 is an outline of the results we have derived using our model. The complete details can be found in [5].

2. THE CONCURRENCY CONTROL PROBLEM

We begin with a simple model of concurrency control. Suppose that $a, b, c, \ldots$ denote atomic units of data that we call *objects*. A *transaction* is a description of the actions of one user on the database. It consists of a finite sequence of reads and writes, each action touching a single object. We denote a transaction T_i by $T_i = \tau_i(o_1)\tau_i(o_2)\cdots\tau_i(o_{n_i})$ where $\tau_i(o_j)$ means that T_i acts on object o_j. For simplicity, we do not distinguish between actions of different types, nor do we assume that the objects for the same transaction are distinct.

A *complete execution* of a (finite) set of transactions is an interleaved sequence of all the actions from the transactions, i.e., it respects the ordering within each transaction. An *execution*, denoted E, is a prefix of a complete execution. An execution is said to be *serial* if there is no interleaving between the transactions.

We say that transaction T_i *precedes* T_j in E if there exist some object b and actions $\tau_i(b)$ and $\tau_j(b)$ such that $\tau_i(b)$ comes before $\tau_j(b)$ in E. In such a case, $(\tau_i(b), \tau_j(b))$ is called a *conflicting pair*. An execution E is is said to be (conflict) *serializable* if and only if *precedes in* E is a partial ordering, i.e., if and only if all the conflicting pairs in E are consistently ordered ([1-3]). A serializable execution produces the same effect on the database as some serial execution.

The dynamics of this problem correspond to the generation of a complete execution from the actions of all the transactions. The concurrency control problem consists in scheduling the transactions to produce only serializable complete executions. Serializability is widely accepted as the appropriate criterion for concurrency control. Non-serializable executions are not acceptable, because they result in possible violations of the consistency of the database; the lost-update problem ([2]) is an example of such undesirable situations. Observe that, in general, the set of objects touched by each transaction is not known beforehand.

3. STATE MODEL FOR CONCURRENCY CONTROL

In this section, we give a brief description of the model that we propose for the analysis of concurrency control. Consider the situation where N transactions are to be executed concurrently. Given an (incomplete) execution, we can determine if it is serializable by examining the ordering of its conflicting pairs. But we cannot determine if this execution has a serializable completion unless we know for each transaction the objects that remain to be acted on.

This justifies the following construction for the state of this system. The initial state, denoted q_0, is a graph with N nodes representing the N transactions and one undirected dashed arc (between the appropriate nodes) for each conflicting pair among the N transactions. The transition function is as follows. In response to an input action $\tau_j(o)$, all the conflicting pairs in which this action is involved are determined. In those pairs for which the other action in the pair has not occurred yet, a direction is put out of node j on the corresponding dashed arc. Otherwise, the corresponding dashed arc into node j is replaced by a solid arc with the same direction.

The state space consists of all such *state graphs* (SG) that can be generated by all possible executions of the N transactions. Let Σ be the set of all the actions from the N transactions. We define the following (strictly nested) sets of input strings w from the elements of Σ (without repetition):

$\Sigma_e = \{w : w \text{ is an execution}\}$

$\Sigma_{se} = \{w : w \text{ is a serializable execution}\}$

$\Sigma_{cse} = \{w : w \text{ is a prefix of a complete serializable execution}\}$

and the corresponding (also strictly nested) sets of states Q_e, Q_{se} and Q_{cse} generated by the elements of these sets. The state space is the set Q_e. Finally, we define the (partial) transition function $\phi_e : \Sigma \times Q_e \to Q_e$ according to the above description. Then, given an input action from Σ, the current SG and ϕ_e completely and uniquely determine the new SG.

Example: As a simple example, consider the non-serializable complete execution:

$$E^c = \tau_1(a)\tau_2(b)\tau_2(a)\tau_1(b).$$

E^c has two conflicting pairs: $(\tau_1(a), \tau_2(a))$ and $(\tau_1(b), \tau_2(b))$. The initial SG and the SG after each action of E^c are given in the figure below.

* Research sponsored by A.R.O. contract DAAG29-82-K-0091, N.S.F. grant ECS-8300463, and in the case of the first author, a scholarship from N.S.E.R.C. (Canada).

CH2245-9/85/0000-0441 $1.00 © 1985 IEEE

□

Our input-state dynamical model can be interpreted as a deterministic automaton or generator (cf. [6]): $G = (Q_e, \Sigma, \phi_e, q_0, Q_m)$, where Q_m can be taken as the set of states corresponding to the complete serializable executions. The above sets Σ_e, Σ_{se} and Σ_{cse} are *languages*, and in particular, Σ_e is the language generated by the uncontrolled generator.

The control problem is, ideally, the construction of a controller, such that the language generated by the controlled generator will be Σ_{cse}. Equivalently, we want the state not only to remain within Q_{cse} at all times, but also to reach all of Q_{cse}, since this means achieving maximum concurrency. Clearly, for this to be possible, the controller must know q_0 beforehand. In such a case, since the language Σ_{cse} is controllable (as defined in [6]), the desired controller will exist. Since this condition is rarely satisfied in practice, the concurrency control problem is one of control with *partial state information*. A necessary requirement is that the state remains within Q_{se} at all times, because any execution must be serializable. The objective is to bring the set of states reachable by the controlled generator as close as possible to Q_{cse}, given the information that is available.

The state space can be characterized as follows (with obvious terminology): (i) Q_{cse} contains all cycle free state graphs; (ii) $Q_{se} - Q_{cse}$ consists of those state graphs with dashed or mixed cycles, but with no solid cycles; (iii) $Q_e - Q_{se}$ consists of those state graphs with solid cycles.

4. CONTROL BY LOCKING

Locking is a widely used concurrency control technique ([1-4]). In addition to the two basic locking actions "lock" and "unlock," a third action "declare" was introduced in [4]. Locking actions are added to a transaction to produce an augmented transaction. Each object in an augmented transaction must be: (i) declared before it is locked, (ii) locked before it is used, and (iii) unlocked before the end of the transaction. Each upgrading voids the previous state of lock. The interleaving of augmented transactions is subject to the constraint that locks are exclusive, i.e., there cannot be more than one lock on an object at any given time. However, "declares" do not conflict with each other, nor do they conflict with "lock." Hence, there can be more than one transaction holding "declare" on an object, even if this object is locked.

Locking does not by itself prevent non-serializable executions, since all executions in Σ_e have an augmentation that satisfies the above locking constraints. But a locking protocol, by imposing constraints on when locks can be obtained and released, will reduce the set of admissible input strings, and in some cases, e.g., when "declare" is used, will enable the controller to construct a better estimate of the state of the system.

5. APPLICATION OF STATE MODEL

We now summarize the main results we have derived using our model. The reader is referred to [5] for the complete details.

1) The various graphs used in concurrency control by locking, namely the wait-for graph (WFG), the precedence graph (PG), and the must-precede graph (MPG), are subgraphs of the SG, i.e., they are partial states.

2) Any controller using the PG as state estimate can do no better than maintaining the state within Q_{se}. On the other hand, the locking action "declare" permits constructing the MPG, a better approximation of the SG, by acting as a predictive feature in the reconstruction of the unknown initial state q_0. Using this MPG in conjunction with a locking protocol makes it possible for the controller to reduce the state transitions to a set strictly smaller than Q_{se}. In the case of complete state information, the *Prior Declaration Protocol* of [4] is an example of an "ideal" controller attaining Q_{cse} exactly.

3) The Declare-Before-Unlock protocol of [4] is an interesting compromise of a protocol which can reach all of Q_{cse}, i.e., all serializable complete executions, but strictly less than Q_{se}, with relative ease of implementation.

4) The state graph model permits a precise description of the effects of the important *two-phase condition* of the two-phase locking protocol ([1]). In particular, it explains why this protocol guarantees serializability even though it only uses the WFG as state estimate.

5) In the case of distributed databases, it is possible to break the complete SG into local sub-SGs at each sub-database. This decentralization effectively corresponds to the idea of model aggregation in decentralized control theory ([7]).

6) Other concurrency control techniques, such as timestamping ([2]), can also be analyzed with our model.

REFERENCES

[1] K. P. Eswaran, J. N. Gray, R. A. Lorie and I. L. Traiger, "The Notions of Consistency and Predicate Locks in a Database System," *Communications of the ACM*, Vol. 19, No. 11, November 1976, pp. 624-633.

[2] C. J. Date, *An Introduction to Database Systems - Volume II*, Reading, MA: Addison-Wesley, 1983.

[3] C. H. Papadimitriou, "Serializability of Concurrent Updates," *Journal of the ACM*, Vol. 26, No. 4, October 1979, pp. 631-653.

[4] S. Lafortune and E. Wong, "A New Locking Protocol That Achieves All Serializable Executions," Memorandum No. UCB/ERL M84/77, Electronics Research Laboratory, University of California, Berkeley, September 1984.

[5] S. Lafortune and E. Wong, "A State Model for the Concurrency Control Problem in Database Management Systems," Memorandum No. UCB/ERL M85/27, Electronics Research Laboratory, University of California, Berkeley, April 1985.

[6] P. J. Ramadge and W. M. Wonham, "Supervisory Control of a Class of Discrete Event Processes," Systems Control Group Report # 8311, Department of Electrical Engineering, University of Toronto, Canada, October 1983.

[7] N. R. Sandell, Jr., P. Varaiya, M. Athans and M. G. Safonov, "Survey of Decentralized Control Methods for Large Scale Systems," *IEEE Trans. on Automatic Control*, Vol. AC-23, No. 2, April 1978, pp. 108-128.