Dynamic Timestamps Ordering

A. Farrag
Faculty of Computer Science
Dalhousie University
Halifax, N.S., Canada, B3H 3J5
zizo@dal.ca

Extended Abstract
This paper proposes several timestamp ordering mechanisms in which the timestamps assigned to transactions can be changed dynamically during execution. These timestamps are not stored with the entities (of the database) and the process of modifying them is simple. The proposed mechanisms achieve a higher level of concurrency (than traditional timestamp ordering mechanisms) for the following reasons. First, the operations of (certain classes of) read-only transactions can always be accepted. Second, the mechanisms avoid rejecting an operation of an update transaction by modifying its current timestamp. The proposed mechanisms do not require multiversion of each entity to be stored, but it maintains a digraph that represents the "conflicts" among transactions. The complexity is reduced by searching the digraph only when the acceptance of an operation may violate consistency; otherwise, the operation is accepted without search.

1. Introduction
In any multiuser database system, a concurrency control mechanism is needed to resolve any conflicts that might arise among transactions, and to ensure that their overall execution is correct, i.e., cannot violate consistency (Papadimitriou1979). Many such mechanisms have been previously proposed, e.g., (Agrawal1985, Bayer1980, Conway2012, Esran1976, Farrag1982, Garcia-Molina1982, Gray1978, Kung1981, Lamport1976, Lessner2011, Reed1978, Rosenkrantz1978); and most of them utilize some form of locking, e.g., (Esran1976, Farrag1987, Gray1976, Kedem1979, Rosenkrantz1978, Usui2010). That is, before a transaction can access any entity, it must first obtain an (appropriate) lock on it, and if this lock cannot be granted, the transaction will be delayed. This can potentially lead to a deadlock, which can be resolved by aborting one or more transactions.

Other approaches to controlling concurrency avoid delaying the transactions by assigning each of them a unique timestamp, and then requiring all conflicting operations to be processed in a timestamp order, e.g., (Farrag1987, Lamport1976, Reed1978, Thomas1978). In this case, when an operation arrives out of order, it will be rejected, that is, the transaction that issued it will be aborted. Abortion is a serious drawback, and can degrade performance if it occurs frequently.

To avoid unnecessary abortions, we propose some dynamic timestamps ordering methods in which the timestamps assigned previously to transactions can be dynamically modified during execution. The timestamps are not stored with the entities in our methods, and the process of modifying them is a simple operation. As will be shown, separating timestamps from the entities will help to solve some issues that arise in all timestamp ordering mechanisms.

References