Concurrency Control in Real-Time E-Collaboration Systems

Wenbing Zhao
Cleveland State University, USA

INTRODUCTION

E-collaboration refers to the collaboration of a group of people sharing the same task, using electronic technologies (Kock & Nosek, 2005). In the Internet age, the interactions and communications among the collaborators, the management of related information, the recording of the progress, and the outcome of the task are primarily facilitated by modern software systems, often called e-collaboration systems or groupware. Such systems range from those enabling loosely coupled, asynchronous collaborations, such as e-mail and software source version control systems, to those supporting tightly coupled, synchronous (also termed as *real-time*) collaborations, such as group editors, e-classroom, and group-decision systems.

For all e-collaboration systems, some degree of concurrency control is needed so that two people do not step on each other's foot. The demand for good concurrency control is especially high for the tightly coupled, real-time e-collaboration systems. Such systems require quick responses to user's actions, and typically require a WYSIWIS (what you see is what I see) graphical user interface (Ellis, Gibbs, & Rein, 1991). This requirement, together with the fact that users are often separated geographically across wide-area networks, favors a decentralized system design where the system state is replicated at each user's site. This places further challenges on the design of concurrency control for these systems.

Furthermore, the users of e-collaboration systems often follow a social protocol during their work on the common task (Ellis et al., 1991). For example, if a number of users are collaborating on a shared document, one would not edit a paragraph if it is clear that another user is editing it. This is very different from other type of concurrent systems, such as database systems, where users' actions are completely independent and isolated. Therefore, real-time e-collaboration systems often favor an optimistic concurrency control approach.

There has been intense research on this issue in the past two decades or so. There are primarily two approaches: (1) locking-based, which uses locks to synchronize different users on access to a shared document (Greeberg & Marwood, 1994); (2) serialization based, which ensures a consistent order of operations on a shared document for all users. Both are initially derived from the concurrency control practices in database systems and have been significantly extended over the years. The most dominant approach appears to be the optimistic serialization based on operational transformation (Ellis & Gibbs, 1989).

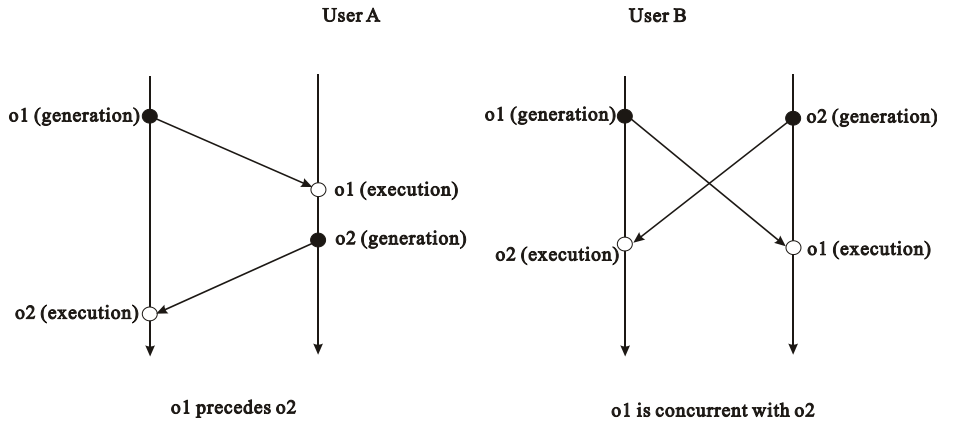
BACKGROUND

Event Ordering

Assuming that several people are working together on a shared document using an e-collaboration system, a user might decide to insert or delete a character in a particular position in the shared document. Such an insertion or deletion will need to be propagated from the user who performed the operation to all other users. Consequently, we distinguish the *operation generation* (from a particular site by a user) from *operation execution* (locally or remotely). Depending on the concurrency control used, the local execution might not be identical to the remote execution.

To perform concurrency control, we often need to establish the order of the operations in the system. To determine such an order, a logical lock (Lamport, 1978) can be used to assign a logical timestamp for each operation. Following the notion of the *happened-before* relationship (see the terminology section for definition), a partial ordering, termed as *precedence property*, can be established for all operations in e-collaboration systems. Given two operations o1 and o2, o1 is said to precede o2 if and only if the execution of o1 at site S happened before the generation of o2 at S (Ellis & Gibbs, 1989).

Figure 1. Relationship between different operations



As shown in Figure 1, some operations have the precedence relationship, while some do not. The events that cannot be associated with a precedence relationship are called concurrent operations. Two concurrent operations are said to *conflict* with each other, if both operate on the same object. To further distinguish the order of concurrent operations, we might need to determine a *total order* of all operations.

CONSISTENCY AND CONCURRENCY CONTROL

Concurrency control is the activity of coordinating concurrent operations that might be conflicting with each other. Concurrency control determines whether to allow, postpone, or deny the execution of a user's operation, depending on the relative ordering of the operation under consideration with respect to other operations, and the object on which the operation applies. If an optimistic concurrency control is used, it may also include the tasks such as undo (roll back a user's change) and operation transformation if a conflict of two or more operations occur.

The purpose of concurrency control is to maintain the *consistency* of the system state replicated at different sites, at the end of each operation execution, or after each conflict resolution. It is easy to see why some form of control is needed to preserve the consistency of the replicas in the presence of concurrent activities. If two conflicting operations are applied at user A and user B in different order, the replica at user A might be different from the replica at user B. If this happens, the state is said to have *diverged*.

In addition to the consistency requirement, concurrency control should also preserve users' intent. This requirement is unique in e-collaboration systems. To understand this subject better, consider the following example. Two users A and B are working on a shared document using a group editor. Assume that the initial document (i.e., the system state) contains a string "collaboration system." User A wants to insert a character "e" and a hyphen "-" right before the word "collaboration". Concurrently, user B wants to insert a character "s" at the end of the word "system." If both users' intent is preserved, the concurrency control should ensure the final state at both user A's and B's sites to be "e-collaboration systems".

Locking-Based Concurrency Control

Locking-based concurrency control (Greeberg & Marwood, 1994), as the name suggests, uses a lock to control the order in which a shared object is updated. A lock is a software construct that guards the access of a shared object. To access the object, one must *acquire* the lock first. After the update is completed, the lock must be *released*. Subsequently, the lock can be granted to the next user waiting in line. Since a lock can be granted to only one user at a time, it ensures sequential access of the shared object.

Pessimistic Locking

In early e-collaboration systems, pessimistic locking is often used as a way of concurrency control. In pessimistic locking, the request-for-lock operation is