CSCI 251: Concepts of Parallel and Distributed Systems

Alvin Lin

November 1st, 2017

Transactions

Transactions are created and managed by a coordinator interface. It involves a client program, an object, and a coordinator. Example operations:

- `openTransaction()`: starts a new transaction and delivers a unique transaction ID. This identifier is used by other operations in the transaction.
- `closeTransaction(identifier)`: ends a transaction by committing it.
- `abortTransaction(identifier)`: ends a transaction by aborting it.

A transaction may be aborted by the client or the server, in which case nothing about that transaction is stored in permanent memory.

Failure model

Server actions:

- Server crash: new process aborts uncommitted transactions and recovers values produced by committed transactions.
- Client crash: uses expiry time to abort transactions.

Client actions:

- Server crash: operations return an exception after timeout. If they are replaced by another process, uncommitted transactions will not be valid.
- Client crash: decision at the client's end.

Concurrency Control

- Locks
- Optimistic concurrency
- Timestamp ordering

Serial Equivalence

Serially Equivalent Interleaving: the combined effect is the same as if the transactions had been performed one at a time in some order. Serial equivalence prevents lost updates and inconsistent retrievals.

Conflicting Operations

A conflict when the combined effect of a pair of operations depends on the order in which they are executed. Serial equivalence is a necessary and sufficient condition where all pairs of conflicting operations are executed in the same order at all objects they both access.
Isolation Property: The uncommitted state of one transaction should not be visible to another transaction.

Nested Transactions

A top level transaction may open child transactions which can run concurrently along with their descendants. They can commit or abort independently of one another. A transaction can commit or abort only after its child transactions have completed. When a parent transaction aborts, all of its child transactions must abort. When a parent transaction commits, all the child transactions can commit, provided none of their ancestors have aborted.

Locks

Serialization is achieved by the use of exclusive locks, which allow only one transaction to access an object until it closes. When an operation accesses an object within a transaction:

- If the object is not already locked, the operations acquires a lock and the operation proceeds.

- If the object has a conflicting lock set by another transaction, the current transaction must wait until it is unlocked.
- If the object has a non-conflicting lock set by another transaction, the lock is shared and the operation proceeds.
- If the lock has already been locked in the same transaction, the lock will be promoted if necessary and the operation proceeds. When promotion is prevented by a conflicting lock, the second rule applies.

When a transaction closes, it unlocks all objects it locked for the transaction. Locks reduce concurrency in general because they are not released until the end of the transaction. Lock maintenance is also an overhead; deadlock prevention and detection is expensive and reduces concurrency.

Optimistic Concurrency Control

The likelihood of two transactions occurring at the same time is very low. Transactions are allowed to proceed as if there are no conflicts until it is closed. If a conflict is detected, the transaction may be aborted. Transactions occur in three phases:

1. **Working Phase**
2. **Validation Phase**
3. **Update Phase**

Reminders

Check MyCourses for details on Project 2.
Professor Mohan Kumar:
`mjkvcs@rit.edu`
`https://cs.rit.edu/~mjk`

Rahul Dashora (TA):
`rd5476@mail.rit.edu`

You can find all my notes at `http://omgimanerd.tech/notes`. If you have any questions, comments, or concerns, please contact me at alvin@omgimanerd.tech