Distributed Object Transactions

Outline
- Transaction Principles
- Concurrency Control
- Two-Phase Commit Protocol
- Services for Distributed Object Transactions
  - CORBA Transaction Service
  - Microsoft Transaction Service
  - Java Transaction API

Transaction Principles
Motivation

- What happens if a failure occurs during modification of resources?
- Which operations have been completed?
- Which operations have not (and have to be done again)?
- In which states will the resources be?

Transaction Concepts

1 ACID Properties
  - Atomicity
  - Consistency
  - Isolation
  - Durability
2 Transaction Commit vs. Abort
3 Flat vs. Nested Transactions
4 Central vs. Distributed Transactions

Atomicity

- Transactions are either performed completely or no modification is done.
- Start of a transaction is a continuation point to which it can roll back.
- End of transaction is next continuation point.
Consistency

- Shared resources should always be consistent.
- Inconsistent states occur during transactions:
  - hidden for concurrent transactions
  - to be resolved before end of transaction.
- Application defines consistency and is responsible for ensuring it is maintained.
- Transactions can be aborted if they cannot resolve inconsistencies.

Isolation

- Each transaction accesses resources as if there were no other concurrent transactions.
- Modifications of the transaction are not visible to other transactions before it finishes.
- Modifications of other transactions are not visible during the transaction at all.
- Implemented through:
  - two-phase locking or
  - optimistic concurrency control.

Durability

- A completed transaction is always persistent (though values may be changed by later transactions).
- Modified resources must be held on persistent storage before transaction can complete.
- May not just be disk but can include battery-backed RAM or Flash RAM.
Transaction Commands

- **Begin:**
  - Start a new transaction.
- **Commit:**
  - End a transaction.
  - Store changes made during transaction.
  - Make changes accessible to other transactions.
- **Abort:**
  - End a transaction.
  - Undo all changes made during the transaction.

Flat Transactions

![Flat Transaction Diagram]

Nested Transactions

![Nested Transaction Diagram]
Central vs. Distributed Transactions

- Transactions in a Database
  - Centralized
  - DBMS controls transaction execution
  - DBMS implements concurrency control
  - Transaction processing transparent to application developers

- Problem occurs if:
  - Data kept in different databases or
  - Distributed objects do not use a database
  - Transaction processing not transparent to application developers

The Two-Phase Commit Protocol

Roles of Components

- Distributed system components involved in transactions can take role of:
  - Transactional Client
  - Transactional Server
  - Coordinator
Coordinator

- Coordinator plays key role in managing transaction.
- Coordinator is the component that handles begin / commit / abort transaction calls.
- Coordinator allocates system-wide unique transaction identifier.
- Different transactions may have different coordinators.

Transactional Server

- Every component with a resource accessed or modified under transaction control.
- Transactional server has to know coordinator.
- Transactional server registers its participation in a transaction with the coordinator.
- Transactional server has to implement a transaction protocol (two-phase commit).

Transactional Client

- Only sees transactions through the transaction coordinator.
- Invokes services from the coordinator to begin, commit and abort transactions.
- Implementation of transactions are transparent for the client.
- Cannot tell difference between server and transactional server.
Two-Phase Commit

- Multiple autonomous distributed servers:
  - For a commit, all transactional servers have to be able to commit.
  - If a single transactional server cannot commit its changes every server has to abort.
- Single phase protocol is insufficient.
- Two phases are needed:
  - Phase one: Voting
  - Phase two: Completion.

Phase One

- Called the voting phase.
- Coordinator asks all servers if they are able (and willing) to commit.
- Servers reply:
  - Yes: it will commit if asked, but does not yet know if it is actually going to commit.
  - No: it immediately aborts its operations.
- Hence, servers can unilaterally abort but not unilaterally commit a transaction.

Phase Two

- Called the completion phase.
- Co-ordinator collates all votes, including its own, and decides to
  - commit if everyone voted ‘Yes’.
  - abort if anyone voted ‘No’.
- All voters that voted ‘Yes’ are sent
  - ‘DoCommit’ if transaction is to be committed.
  - Otherwise ‘Abort’.
- Servers acknowledge DoCommit once they have committed.
**Example: Funds Transfer**

```
begin()  
debit()   register_resource()  
credit() register_resource()  
commit() prepare()  
        prepare()  
        commit()  
```

**Server Uncertainty (1)**
- Period when a server must be able to commit, but does not yet know if has to.
- This period is known as server uncertainty.
- Usually short (time needed for co-ordinator to receive and process votes).
- However, failures can lengthen this process, which may cause problems.

**Complexity**
- Assuming N participating servers:
  - N registration requests from servers to coordinator
  - N Voting requests from coordinator to servers.
  - N Completion requests from coordinator to servers.
- Hence, complexity of requests is linear in the number of participating servers.
Recovery in Two-Phase Commit

- Failures prior to start of 2PC results in abort.
- Coordinator failure prior to transmitting commit messages results in abort.
- After this point, co-ordinator will retransmit all Commit messages on restart.
- If server fails prior to voting, it aborts.
- If it fails after voting, it sends GetDecision.
- If it fails after committing it (re)sends HaveCommitted message.

Committing Nested Transactions

- Cannot use same mechanism to commit nested transactions as:
  - subtransactions can abort independently of parent.
  - subtransactions must have made decision to commit or abort before parent transaction.
- Top level transaction needs to be able to communicate its decision down to all subtransactions so they may react accordingly.

Key Points

- A distributed object transaction is an atomic, consistency-preserving, isolated durable sequence of object requests
- Objects participating in transactions can be transactional clients, transactional servers and transaction co-ordinators
- Isolation is achieved by two-phase locking that can either be delegated to a database or be done explicitly by the server designer
**Key Points**

- Atomicity is achieved by two-phase commit, which consists of a voting and a completion phase
- Object-oriented middleware supports distributed transaction through transaction services
  - CORBA Transaction Service
  - Microsoft Transaction Server
  - Java Transaction Service