

Distributed Systems (ICE 601)

Distributed Transactions

Dongman Lee
ICU

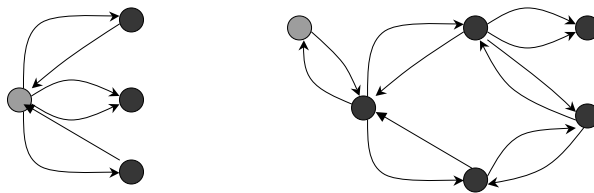
Class Overview

- Distributed Transactions
- Atomic Commit Protocol
- Distributed Deadlock

Distributed Transactions

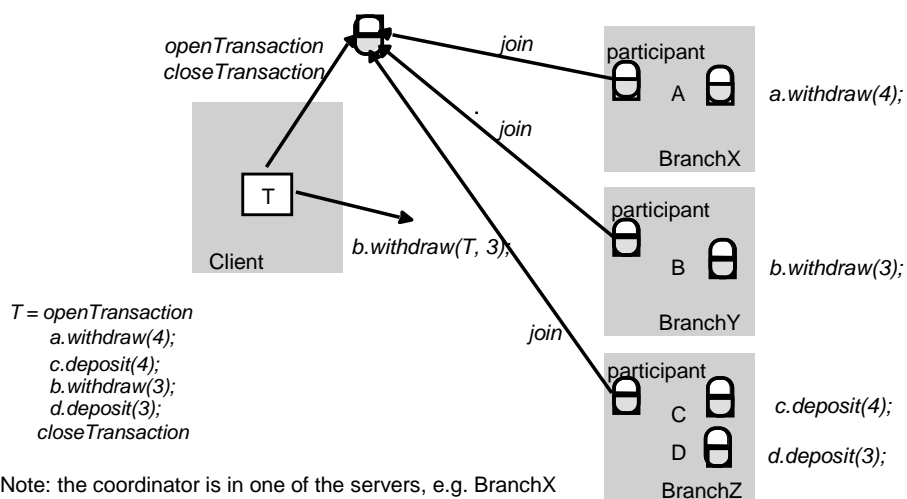
- Definition

- a transaction in which more than one server is involved
 - ♦ multiple servers are called by a client (simple distributed transaction)
 - ♦ a server calls another servers (nested transaction)
- execution of program accessing shared data at multiple sites [Lamport]



Distributed Systems - Distributed Transactions

Distributed Transactions - example



Distributed Systems - Distributed Transactions

Atomicity in Distributed Transaction

- Requirement
 - a client requires to get congruent commitment from involved servers due to atomic property of a transaction
- Resolution
 - Coordination
 - Atomic commitment protocol

Distributed Systems - Distributed Transactions

Coordination in Distributed Transaction

- How it works
 - one of servers become a coordinator and the others workers
 - ♦ who becomes a coordinator
 - simple transaction: first server
 - nested transaction: top-level server
 - each transaction should be globally identifiable (server id + unique #)
 - coordinator
 - ♦ maintains a list of participating servers
 - ♦ collects results from workers and makes a decision to guarantee congruent commitment of transaction
 - workers
 - ♦ aware of coordinator's existence
 - ♦ reports its result to the coordinator and follows a decision from it

Distributed Systems - Distributed Transactions

Atomic Commit Protocol

- Atomic commitment problem [Babaoglu & Toueg]
 - bring a transaction to a globally consistent conclusion despite failures
 - ♦ commit: all participants will make the transaction's update permanent
 - decision is based on unilateral agreement among all participants
 - ♦ abort: none will
- ⇒ *atomic commit protocol* that should satisfy these properties
- ♦ all participants that decide reach the same decision
 - ♦ if any participant decides commit, then all participants must have voted yes
 - ♦ if all participants vote yes and no failure occur, the all participants decide commit
 - ♦ each participant decides at most once (i.e. decision is not reversible)

Distributed Systems - Distributed Transactions

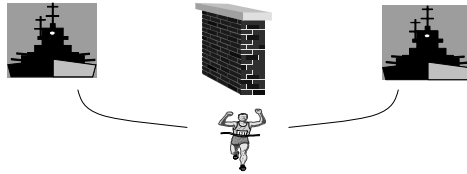
Atomic Commit Protocol (cont.)

- Broadcast property
 - (validity) if a coordinator broadcasts a message m , the all participants eventually receive m
 - (integrity) for any message m , each participant receives m at most once and only if a coordinator actually broadcasts m
 - (timeliness) there exists a known constant d such that broadcast of m is initiated at real-time t , no participant receives m after real-time $t + d$

Distributed Systems - Distributed Transactions

Atomic Commit Protocol (cont.)

- Generals Paradox



- There is no fixed-length protocol that will allow the generals to agree on a common time to attack

Distributed Systems - Distributed Transactions

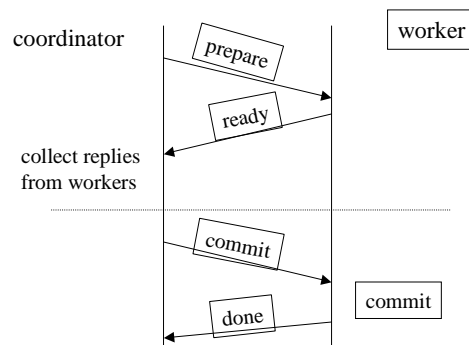
Why Multiple Phase Atomic Commit Protocol?

- Example: one phase atomic commit
 - mechanism
 - ♦ coordinator keeps sending workers a commit or abort request until all of them acknowledged that they had carried it out
 - does not allow a coordinator to make a unilateral decision to abort a transaction when a client requests a commit
 - ♦ there's no room for servers to have decision consensus process among themselves
 - ♦ it is caused mainly by concurrency control
- ➔ allow one or more preparation phases before making a final decision
 - two phase commit protocol is most widely used
 - ♦ general and inexpensive
 - ♦ window of time during which servers are not allowed to abort the transaction is small

Distributed Systems - Distributed Transactions

Two Phase Commit (2PC) Protocol

- Mechanism
 - commit process consists of two message passing phases
 - ♦ phase 1: voting
 - ♦ phase 2: completion of voting result



Distributed Systems - Distributed Transactions

2PC Protocol (cont.)

- Phase 1
 - coordinator
 - ♦ send "prepare (CanCommit?)" message to each worker
 - ♦ wait until
 - a response ("ready" or "no" is received from each worker, or
 - timeout occurs
 - workers
 - ♦ wait until "prepare" message is received from coordinator
 - ♦ if transaction is ready to commit
 - then, send "ready" message to coordinator
 - otherwise, send "no" message to coordinator and abort
- Phase 2
 - coordinator
 - ♦ if "ready" message was received from every worker
 - send "commit" message to each worker
 - otherwise, send "abort" message to each worker
 - ♦ wait until
 - acknowledgement is received from each worker
 - workers
 - ♦ wait until "commit" or "abort" message is received from coordinator
 - ♦ do appropriate work according to the message
 - ♦ send acknowledgement

Distributed Systems - Distributed Transactions

2PC Protocol for Nested Transactions

- Why extra care?
 - sub-transactions can make an independent decision to commit provisionally or to abort
 - transaction can commit only if all of its provisionally committed child transactions can commit
- Extra steps
 - assumption
 - ♦ servers for sub-transactions record information regarding what sub-transactions have committed provisionally or aborted => top-level will get a list of all sub-transactions with their status
 - phase 1
 - ♦ if worker has any provisionally committed sub-transactions
 - then, check whether they do not have aborted ancestors
 - » if yes, send “no” and abort
 - » otherwise, send “yes”
 - otherwise, send “no”

Distributed Systems - Distributed Transactions

Timeout in 2PC Protocol

- Objective
 - make 2PC protocol non-blocking in the presence of
 - ♦ coordinator failure
 - ♦ worker failure
- Additional properties
 - atomic commit protocol properties
 - ♦ every correct participant that executes atomic commit protocol eventually decides
 - broadcast properties
 - ♦ (uniform agreement) if any participant (correct or not) receives a message m, then all correct participants eventually receive m

Distributed Systems - Distributed Transactions

Timeout in 2PC Protocol (cont.)

- Worker timeout
 - coordinator failed to send “ready” message
 - ♦ workers unilaterally abort
 - coordinator failed to send decision
 - ♦ workers send a coordinator a probing message (GetDecision) or
 - sub-transaction can ask its parent in case of nested transaction
 - ♦ workers cooperatively obtain a decision
- Coordinator timeout
 - workers failed to send “yes” messages
 - ♦ coordinator decides to abort transaction

Distributed Systems - Distributed Transactions

Concurrency Control in Distributed Transactions

- Locking
 - distributed deadlock may occur
- Timestamp ordering concurrency control
 - if two transactions access the same data items on various servers, they must commit them in the same order
 - ♦ to achieve this, servers should agree on the ordering of their timestamp using synchronized physical clock
- Optimistic concurrency control
 - parallel validation
 - ♦ resolve commitment deadlock

Distributed Systems - Distributed Transactions

Distributed Deadlock

- Centralized deadlock detection
 - each server sends its local wait-for graph and the central deadlock detector checks a cycle by global wait-for graphs
 - phantom deadlocks
 - ♦ happens when one of transactions that holds a lock (and creates deadlock) will have aborted during deadlock detection phase

Distributed Systems - Distributed Transactions

Distributed Deadlock (cont.)

- Distributed deadlock detection
 - called edge chasing or path pushing
 - no global wait-for graph
 - mechanism
 - ♦ lock manager informs the coordinator when transactions start waiting and when they become active again
 - ♦ three phases
 - initiation
 - » if transaction A starts waiting for transaction B waiting to access a data item at another server, transaction B's server sends a probe containing the wait-for relationship to the server of data item where transaction B is blocked and all the servers in which transactions share lock with transaction B
 - detection
 - » if the data item is hold by another transaction (by consulting with coordinator), add this relationship to the probe and forward the probe in the same manner as above
 - resolution
 - » when cycle is detected, a transaction in a cycle is aborted to break the deadlock

Distributed Systems - Distributed Transactions

Distributed Deadlock (cont.)

