This Lecture

- More 2PC
  - [BN97] Ch 9

- Starting MOM and Message Brokers
  - [ACKM04] Section 2.5
  - [BN97] Ch 4
2-Phase Commit

• Phase 1:
  • Coordinator sends PREPARE to all participants and waits for responses
  • Participants reply YES or NO, or fail to reply
2-Phase Commit

• Phase 2:
  • Coordinator decides YES iff received YES votes from all participants
  • Coordinator sends decision to all participants
  • Participants reply DONE
  • Coordinator frees resources after receiving DONE from all participants
2-Phase Commit - Blocking

- Correctness:
  - After voting NO participant may abort
  - After voting YES participant may not commit or abort until receiving the coordinator decision -- *in doubt*
  - What if coordinator fails while some participants are in doubt? *Blocked!*
2-Phase Commit - Theorems

- For every possible distributed commit protocol, a communication failure can cause a participant to become blocked.

- No distributed commit protocol can guarantee independent recovery (recovery without cooperation from coordinator) of failed participants.
Logging in 2PC

• Coordinator and participants must log enough information to enable recovery if a failure occurs during execution of the 2PC protocol

• Participant P is not prepared to commit txn T unless all after-images for T are in stable storage at P

• No participant may commit unless all are prepared
Logging in 2PC

1. Log a START record
2. Log a PREPARE record
3. Log a COMMIT record
4. Log a COMMITTED record
5. Log a DONE record
Logging in 2PC

- Log a START record
- Log an ABORT record
- Log a DONE record

**PREPARE**
- Log a NO record
- Log an ABORTED record
- Log a DONE record
Error Handling - Coordinator

- Broadcast PREPARE
  - No error possible
- Receive replies from all participants
  - Any replies timeout => assume NO
- Decide to commit or abort
  - No error possible
- Broadcast decision (COMMIT or ABORT)
  - No error possible
Error Handling - Coordinator

- Receive DONE from all participants
- Timeout => re-solicit DONE messages from all participants, infinite loop
- Free all resources associated with transaction
- No error possible
Error Handling - Participant

- Receive PREPARE from coordinator
- Timeout without PREPARE request => abort the transaction unilaterally
- Txn mentioned in PREPARE does not exist => just ignore the request
- Prepare the transaction for commit
- No error possible
- Result is commit vote (YES or NO)
Error Handling - Participant

• Send vote (YES or NO)
  • No error possible
• Receive decision
  • Timeout => *blocked*!
• Implement the decision (commit or abort) and send DONE
  • No error possible
Recovery

Log a START record

PREPARE

YES

Log a COMMIT record

YES

DONE

Log a PREPARED record

Log a COMMITTED record

Log a DONE record
Recovery

Log a START record

PREPARE

Log a NO record

NO

Log an ABORT record

NO

Log an ABORTED record

DONE

Log a DONE record
Recovery - Participant

- No START in log
  - Participants will eventually abort
- No COMMIT/ABORT in log
  - Broadcast NO
- No DONE in log
  - Broadcast decision from log (again?)
- Done in log
  - No action required
Recovery - Coordinator

- No PREPARED record in log
- Abort unilaterally
- No COMMITED/ABORTED in log
- Execute “Termination Protocol”
- COMMITED/ABORTED in log
- Send DONE (again?)
Simple Termination Protocol

- Reestablish communication with coordinator (wait indefinitely for this)
- Resend vote
- Coordinator will resend decision
  - Cannot have forgotten the txn as it has not received a DONE message!
Message Oriented Systems

- MOM
- Message Brokers
Async RPC as two messages

Message : quoteRequest {
  QuoteReferenceNumber: 325
  Customer: Acme,INC
  Item: #115 (Ball-point pen, blue)
  Quantity: 1200
  RequestedDeliveryDate: Mar 16, 2003
  DeliveryAddress: Palo Alto, CA
}

Message: quote {
  QuoteReferenceNumber: 325
  ExpectedDeliveryDate: Mar 12, 2003
  Price: 1200$
}
MOM Core

client application

queued messages

quotatião tool

Message-Oriented Middleware (MOM)

MOM routes messages and maintains input queues
Multiple Servers

Queue acts as worklist for multiple processes
Multiple Clients and Servers

- Load balancing on both sides of queue

Clients

Servers
Multiple Clients and Servers

Txn 1:
get input
construct request
enqueue
commit

Txn 2:
dequeue
process request
enqueue
commit

Txn 3:
dequeue
decode reply
process output
commit

Client

Server