Atomic Transactions in Distributed Systems

Definition – *Transaction*

- A sequence of operations that perform a single logical function
- Examples
  - Withdrawing money from your account
  - Making an airline reservation
  - Making a credit-card purchase
- Usually used in context of databases

Definition – *Atomic Transaction*

- A transaction that happens completely or not at all
  - No partial results
- Example:
  - Cash machine hands you cash and deducts amount from your account
  - Airline confirms your reservation and
    - Reduces number of free seats
    - Charges your credit card
    - (Sometimes) increases number of meals loaded on flight
  - ...

Atomic Transaction Review

- Fundamental principles – *A C I D*
  - *Atomicity* – to outside world, transaction happens indivisibly
  - *Consistency* – transaction preserves system invariants
  - *Isolated* – transactions do not interfere with each other
  - *Durable* – once a transaction "commits," the changes are permanent

Programming in a Transaction System

- *Begin_transaction*
  - Mark the start of a transaction
- *End_transaction*
  - Mark the end of a transaction and try to “commit”
- *Abort_transaction*
  - Terminate the transaction and restore old values
- *Read*
  - Read data from a file, table, etc., on behalf of the transaction
- *Write*
  - Write data to file, table, etc., on behalf of the transaction

Programming in a Transaction System (continued)

- As a matter of practice, separate transactions are handled in separate threads or processes
- *Isolated* property means that two concurrent transactions are *serialized*
  - I.e., they run in some indeterminate order with respect to each other

- **Commit**- Changes are saved , resources are released; State is consistent

- **Abort** -For an outsider, nothing happened

Programming in a Transaction System (continued)

- *Nested Transactions*
  - One or more transactions inside another transaction
  - May individually commit, *but* may need to be undone
- Example
  - Planning a trip involving three flights
  - Reservation for each flight “commits” individually
  - Must be undone if entire trip cannot commit

Tools for Implementing Atomic Transactions (single system)

- Stable storage
  - i.e., write to disk “atomically” (ppt, html)
- Log file
  - i.e., record actions in a log before “committing” them (ppt, html)
  - Log in stable storage
- Locking protocols
  - Serialize *Read* and *Write* operations of same data by separate transactions
- ...

Tools for Implementing Atomic Transactions (continued)

- *Begin_transaction*
  - Place a *begin* entry in log
- *Write*
  - Write updated data to log
- *Abort_transaction*
  - Place *abort* entry in log
- *End_transaction* (i.e., *commit*)
  - Place *commit* entry in log
  - Copy logged data to files
  - Place *done* entry in log

Tools for Implementing Atomic Transactions (continued)

- Crash recovery – search log
  - If *begin* entry, look for matching entries
  - If *done*, do nothing (all files have been updated)
  - If *abort*, undo any permanent changes that transaction may have made
  - If *commit* but not *done*, copy updated blocks from log to files, then add *done* entry

Distributed Atomic Transactions

- Atomic transactions that span multiple sites and/or systems
- Same semantics as atomic transactions on single system
  - *A C I D*
- Failure modes
  - Crash or other failure of one site or system
  - Network failure or partition
  - Byzantine failures

General Solution – Two-phase Commit

- One site is elected *coordinator* of the transaction *T*
  - See *Election* algorithms (ppt, html)
- *Phase 1:* When coordinator is ready to commit the transaction
  - Place *Prepare*(*T*) state in log on stable storage
  - Send *Vote_request*(*T*) message to all other participants
  - Wait for replies

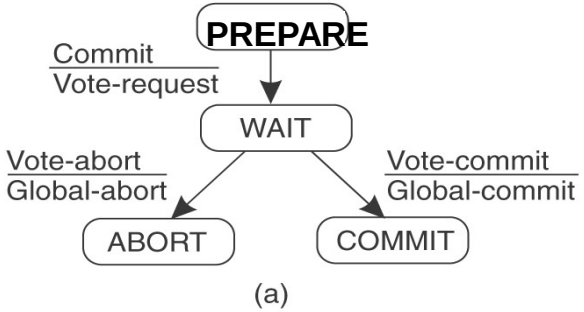
Two-Phase Commit (continued)

- *Phase 2:* Coordinator
  - If any participant replies *Abort*(*T*)
    - Place *Abort*(*T*) state in log on stable storage
    - Send *Global_Abort*(*T*) message to all participants
    - Locally abort transaction *T*
  - If *all* participants reply *Ready_to_commit*(*T*)
    - Place *Commit*(*T*) state in log on stable storage
    - Send *Global_Commit*(*T*) message to all participants
    - Proceed to commit transaction locally

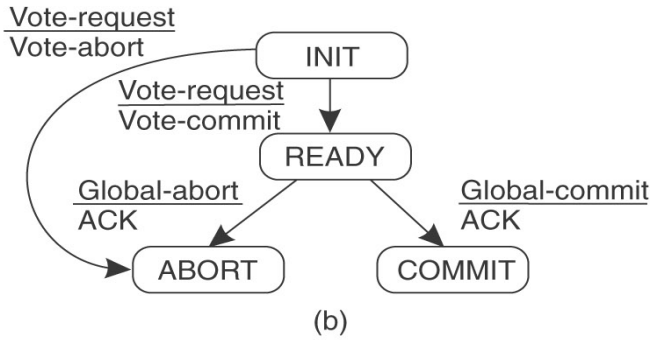
Two-Phase Commit (continued)

- Phase I: Participant gets *Vote_request*(*T*) from coordinator
  - Place *Abort*(*T*) or *Ready*(T) state in local log
  - Reply with *Abort*(*T*) or *Ready_to_commit*(*T*) message to coordinator
  - If *Abort*(*T*) state, locally abort transaction
- Phase II: Participant
  - Wait for *Global_Abort*(*T*) or *Global_Commit*(*T*) message from coordinator
  - Place *Abort*(*T*) or *Commit*(T) state in local log
  - Abort or commit locally per message

Two-Phase Commit States



coordinator



participant

Failure Recovery – Two-Phase Commit

- Failure modes (from coordinator's point of view)
  - Own crash
  - *Wait* state: No response from some participant to *Vote_request* message
- Failure modes (from participant's point of view)
  - Own crash
  - *Ready* state: No message from coordinator to *Global_Abort*(*T*) or *Global_Commit*(*T*)

Lack of Response to Coordinator *Vote_Request*(*T*) message

- E.g.,
  - participant crash
  - Network failure
- Timeout is considered equivalent to *Abort*
  - Place *Abort*(*T*) state in log on stable storage
  - Send *Global_Abort*(*T*) message to all participants
  - Locally abort transaction *T*

Coordinator Crash

- Inspect Log
- If *Abort* or *Commit* state
  - Resend corresponding message
  - Take corresponding local action
- If *Prepare* state, either
  - Resend *Vote_request*(*T*) to all other participants and wait for their responses; *or*
  - Unilaterally abort transaction
    - I.e., put *Abort*(*T*) in own log on stable store
    - Send *Global_Abort*(*T*) message to all participants
- If nothing in log, abort transaction as above

No Response to Participant's *Ready_to_commit*(*T*) message

- Re-contact coordinator, ask what to do
- If unable to contact coordinator, contact other participants, ask if they know
- If any other participant is in *Abort* or *Commit* state
  - Take equivalent action
- Otherwise, wait for coordinator to restart!
  - Participants are blocked, unable to go forward or back
  - Frozen in *Ready* state!

Participant Crash

- Inspect local log
  - *Commit* state:
    - Redo/replay the transaction
  - *Abort* state:
    - Undo/abort the transaction
  - No records about *T*:
    - Same as *local_abort*(*T*)
  - *Ready* State:
    - Same as no response to *Ready_to_commit*(*T*) message

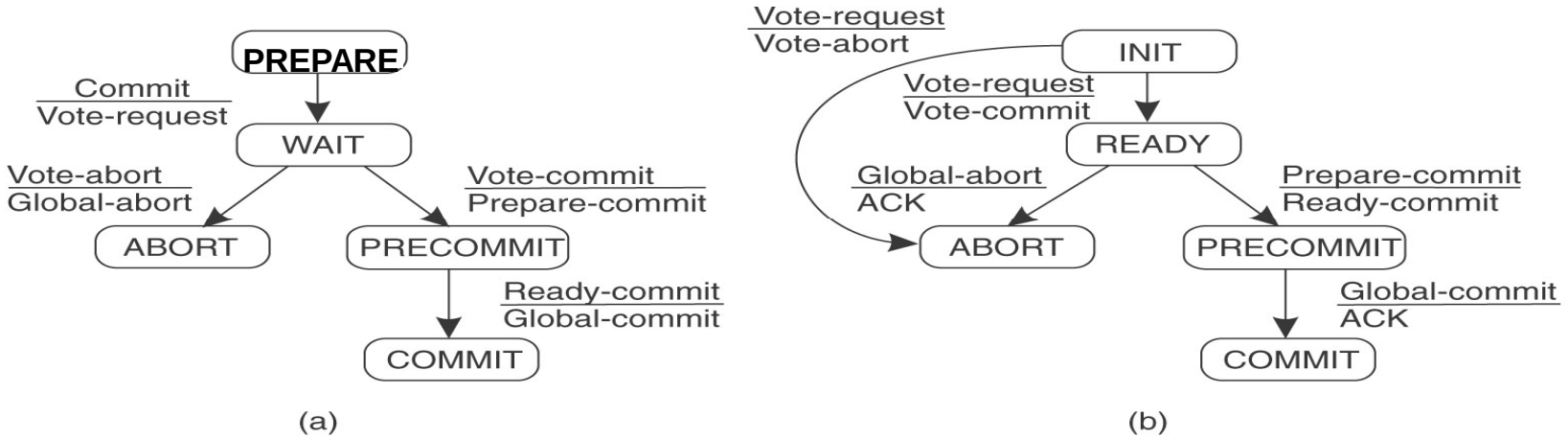
Two-Phase Commit Summary

- Widely used in distributed transaction and database systems
- Generally works well
  - When coordinators are likely to reboot quickly
  - When network partition is likely to end quickly
- Still subject to participant blocking

Three-Phase Commit

- Minor variation
- Widely quoted in literature
- Rarely implemented
  - Because indefinite blocking due to coordinator failures doesn’t happen very often in real life!

Three-Phase Commit (continued)



- There is no state from which a transition can be made to either *Commit* or *Abort*

- There is no state where it is not possible to make a final decision and from which transition can be made to *Commit*.

Three-Phase Commit (continued)

- Coordinator sends *Vote_Request* (as before)
- If all participants respond affirmatively,
  - Put *Precommit* state into log on stable storage
  - Send out *Prepare_to_Commit* message to all
- After all participants acknowledge,
  - Put *Commit* state in log
  - Send out *Global_Commit*

Three-Phase Commit Failures

- Coordinator blocked in *Ready* state
  - Safe to abort transaction
- Coordinator blocked in *Precommit* state
  - Safe to issue *Global_Commit*
  - Any crashed or partitioned participants will commit when recovered
- …

Three-Phase Commit Failures (continued)

- Participant blocked in *Precommit* state
  - Contact others
  - Collectively decide to commit
- Participant blocked in *Ready* state
  - Contact others
  - If any in *Abort*, then abort transaction
  - If any in *Precommit*, the move to *Precommit* state
  - …

Three-Phase Commit Summary

- If any processes are in *Precommit* state, then all crashed processes will recover to
  - *Ready*, *Precommit*, or *Committed* states
- If any process is in *Ready* state, then all other crashed processes will recover to
  - *Init*, *Abort*, or *Precommit*
  - Surviving processes can make collective decision

Application

- managing atomic transactions between distributed applications, transaction managers and resource managers.