Last Class: Fault Tolerance

• Basic concepts and failure models

• Failure masking using redundancy

• Agreement in presence of faults
  – Two army problem
  – Byzantine generals problem

Today: More on Fault Tolerance

• Reliable communication
  – One-one communication
  – One-many communication

• Distributed commit
  – Two phase commit
  – Three phase commit

• Failure recovery
  – Checkpointing
  – Message logging
Reliable One-One Communication

- Issues were discussed in Lecture 3
  - Use reliable transport protocols (TCP) or handle at the application layer
- RPC semantics in the presence of failures
- Possibilities
  - Client unable to locate server
  - Lost request messages
  - Server crashes after receiving request
  - Lost reply messages
  - Client crashes after sending request

![Diagram showing communication scenarios](image)

Reliable One-Many Communication

- Reliable multicast
  - Lost messages => need to retransmit
- Possibilities
  - ACK-based schemes
    - Sender can become bottleneck
  - NACK-based schemes

![Diagram showing multicast communication](image)
Atomic Multicast

• Atomic multicast: a guarantee that all process received the message or none at all
  – Replicated database example

• Problem: how to handle process crashes?

• Solution: group view
  – Each message is uniquely associated with a group of processes
    • View of the process group when message was sent
    • All processes in the group should have the same view (and agree on it)

Implementing Virtual Synchrony in Isis

a) Process 4 notices that process 7 has crashed, sends a view change
b) Process 6 sends out all its unstable messages, followed by a flush message
   c) Process 6 installs the new view when it has received a flush message from everyone else
Distributed Commit

• Atomic multicast example of a more general problem
  – All processes in a group perform an operation or not at all
  – Examples:
    • Reliable multicast: Operation = delivery of a message
    • Distributed transaction: Operation = commit transaction

• Problem of distributed commit
  – All or nothing operations in a group of processes

• Possible approaches
  – Two phase commit (2PC) [Gray 1978 ]
  – Three phase commit

Two Phase Commit

• Coordinator process coordinates the operation
• Involves two phases
  – Voting phase: processes vote on whether to commit
  – Decision phase: actually commit or abort
Implementing Two-Phase Commit

actions by coordinator:
while START_2PC to local log;
multicast VOTE_REQUEST to all participants;
while not all votes have been collected {
  wait for any incoming vote;
  if timeout {
    while GLOBAL_ABORT to local log;
multicast GLOBAL_ABORT to all participants;
  exit;
  }
  record vote;
}
if all participants sent VOTE_COMMIT and coordinator votes COMMIT{
  write GLOBAL_COMMIT to local log;
multicast GLOBAL_COMMIT to all participants;
} else {
  write GLOBAL_ABORT to local log;
multicast GLOBAL_ABORT to all participants;
}

* Outline of the steps taken by the coordinator in a two phase commit protocol

Implementing 2PC

actions by participant:
write INIT to local log;
wait for VOTE_REQUEST from coordinator;
if timeout {
  write VOTE_ABORT to local log;
  exit;
}
if participant votes COMMIT {
  write VOTE_COMMIT to local log;
send VOTE_COMMIT to coordinator;
wait for DECISION from coordinator;
if timeout {
  multicast DECISION_REQUEST to other participants;
  wait until DECISION is received; /* remain blocked */
write DECISION to local log;
}
if DECISION == GLOBAL_COMMIT
  write GLOBAL_COMMIT to local log;
else if DECISION == GLOBAL_ABORT
  write GLOBAL_ABORT to local log;
} else {
  write VOTE_ABORT to local log;
  send VOTE_ABORT to coordinator;
}

actions for handling decision requests:
/*executed by separate thread */
while true {
  wait until any incoming DECISION_REQUEST is received; /* remain blocked */
  read most recently recorded STATE from the local log;
  if STATE == GLOBAL_COMMIT
    send GLOBAL_COMMIT to requesting participant;
  else if STATE == INIT or STATE == GLOBAL_ABORT
    send GLOBAL_ABORT to requesting participant;
  else
    skip; /* participant remains blocked */
Three-Phase Commit

Two phase commit: problem if coordinator crashes (processes block)
Three phase commit: variant of 2PC that avoids blocking

Recovery

- Techniques thus far allow failure handling
- Recovery: operations that must be performed after a failure to recover to a correct state
- Techniques:
  - Checkpointing:
    - Periodically checkpoint state
    - Upon a crash roll back to a previous checkpoint with a consistent state
Independent Checkpointing

• Each process periodically checkpoints independently of other processes
• Upon a failure, work backwards to locate a consistent cut
• Problem: if most recent checkpoints form inconsistent cut, will need to keep rolling back until a consistent cut is found
• Cascading rollbacks can lead to a domino effect.

Coordinated Checkpointing

• Take a distributed snapshot [discussed in Lec 11]

• Upon a failure, roll back to the latest snapshot
  – All process restart from the latest snapshot
Message Logging

• Checkpointing is expensive
  – All processes restart from previous consistent cut
  – Taking a snapshot is expensive
  – Infrequent snapshots => all computations after previous snapshot will need to be redone [wasteful]

• Combine checkpointing (expensive) with message logging (cheap)
  – Take infrequent checkpoints
  – Log all messages between checkpoints to local stable storage
  – To recover: simply replay messages from previous checkpoint
    • Avoids recomputations from previous checkpoint