CISC 434: Fault Tolerance - III

- Introduction to fault tolerance
- Process resilience
- Reliable client-server communication
- Reliable group communication
- Distributed commit
- Recovery
Distributed Commit

- **Atomic Multicast Problem**
  - Messages are delivered either to all processes or to none
  - All messages are delivered to all the processes in the same order
  - An example of a more general problem - distributed commit

- **Distributed commit**
  - Ensures that an operation is performed by each member of a process group, or none at all
  - Often established by a coordinator
Distributed Commit

Protocols

- One phase commit protocol (1PC)
  - The coordinator tells all other processes whether or not to perform an operation
  - If one of the processes cannot perform the operation, there is no way to inform the coordinator

- Two phase commit protocol (2PC)
  - A coordinator first checks whether all processes agree to perform the same operation and then multicasts the outcome of that poll

- Three phase commit protocol (3PC)
  - Handle the crash of the coordinator without having to block all processes to reach agreement until the coordinator recovers
Two-Phase Commit (2PC)

a) Coordinator in 2PC
b) A participant

- The coordinator/participants have states (WAIT/READY) in which they block waiting for incoming messages – any process crash may lead to deadlock. Two solutions to this problem:
  - Time out mechanisms
  - Participants communicate with each other
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;
}
Two-Phase Commit – 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;
}
Two-Phase Commit – 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 */
}

- Steps taken for handling incoming decision requests
- If all participants have received and processed the VOTE_REQUEST from the coordinator and then the coordinator crashes, participants cannot cooperatively decide on the final action to take
- A participant will need to block until the coordinator recovers – 2PC is also known as blocking commit protocol – solution 3PC
Three-Phase Commit (3PC)

- Avoids blocking problem in the event of fail-stop crashes
- Not usually used since the conditions under which the 2PC blocks rarely occur
- Two conditions
  - There is no single state from which it is possible to make a transition directly to either a COMMIT or an ABORT state
  - There is no state in which it is not possible to make a final decision, and from which a transition to a COMMIT state can be made
Three-Phase Commit – contd.

(a) Coordinator in 3PC

b) A participant
Recovery

- **Recovery**
  - Once a failure is occurred, the process must reach to a correct state from the failure state
  - Major issue – when and how the states can be recorded and recovered to a correct state

- **Recovery Mechanisms** – store the process states or the communications between processes to replay the execution of the system after a crash
  - Checkpointing
  - Message logging
Error Recovery

- Backward Recovery
  - The state of the system is periodically check-pointed on some stable storage that is not affected by failure
  - When an error is detected, the system is rolled back to its previously check-pointed (assumed error-free) state
  - Overhead: check-pointing and rollback
Error Recovery

**Forward Recovery**
- No previous state is available – go forward for making the state error-free by taking appropriate corrective actions
- Less overhead – requires an accurate assessment of the damage for correcting the error
- Less common than backward recovery
Checkpointing

- Checkpointing sets a recovery line
  - Most recent distributed snapshot - most recent consistent cut
Checkpointing – contd.

- **Independent Checkpointing**
  - Processes take local checkpoints independent of each other
  - The process of cascaded rollback – the domino effect

- **Coordinated Checkpointing**
  - All processes synchronize to jointly write their states to local stable storage
  - Solutions – Distributed snapshot algorithm, Two phase blocking protocol
Message Logging

- To improve performance, many distributed systems combine checkpointing with message logging.
- A checkpointed state is taken as a starting state, and all messages that have been sent since are simply retransmitted and handled accordingly (replayed).
- A globally consistent state can still be reached but without having to restore that state from stable storage – reduces the number of costly checkpoints.
- Message logging involves decisions about which messages to log or when – incorrect replay of messages after recovery lead to an orphan process.
Message Logging – contd.

- **Orphan process** – a process that survives the crash of another process, but whose state is inconsistent with the crashed process after its recovery.

- **Stable message** – it can no longer be lost, for example, it has been written in a stable storage and can be replayed for recovery.
Message Logging – contd.

Types of Message Logging

- Pessimistic logging protocol – ensures that each non-stable message is delivered to at most one process
- Optimistic logging protocol – Once a crash occurs, for a message $m$, each process is rolled back to a state in which it no longer belongs to the set of processes to whom $m$ has been delivered
Summary

- Distributed commit - an operation is performed by each member of a process group, or none at all
  - One phase commit protocol (1PC)
  - Two phase commit protocol (2PC)
  - Three phase commit protocol (3PC)
- Recovery
  - Forward and backward recovery
  - Checkpointing
  - Message Logging