Distributed Systems
ECEG-6504

Fault Tolerance

Surafel Lemma Abebe (Ph. D.)

Topics

- Introduction
- Two-phase commit
- Three-phase commit

Introduction

- Partial failure
  - A characteristics of DS that distinguishes them from single-machine systems
  - Happens when a component in a DS fails
    - Could affect some other components while others are not affected
- Goal in DS design
  - To construct the system in such a way that it can *automatically recover from partial failures* without seriously affecting the overall performance

Introduction…

- Basic concepts
  - Fault tolerance is strongly related to what are called dependable systems
  - Dependability covers the following requirements for DS
    - Availability
      - Defined as the property that a system is ready to be used immediately
      - Refers to the probability that the system is available and operating correctly at any given moment
      - E.g., A system that goes down for one millisecond every hour, it has an availability of 99.99%
    - Reliability
      - Defined as the property that a system can run continuously without failure
      - Defined in terms of a time interval
      - E.g., a system that never crashes but is shut down for two weeks every Aug.

Introduction…

- Basic concepts
  - Dependability requirements for DS…
    - Safety
      - Refers to the situation that when a system temporarily fails to operate correctly, nothing catastrophic happens
      - E.g., process control systems for sending people to space
    - Maintainability
      - Refers to how easy a failed system can be repaired
      - A highly maintainable system may also show a high degree of availability

Introduction…

- Basic concepts…
  - A system is said to fail when it cannot meet its promises
  - Error
    - Part of a system's state that may lead to failure
  - Fault
    - Cause of an error
    - Classification
      - Transient faults
        - » Occur once and then disappear
      - Intermittent faults
        - » Occur then vanish of its own accord, then reappears, and so on
      - Permanent fault
        - » Is one that continues to exist until faulty component is replaced
  - Fault tolerance
    - A system can provide its services even in the presence of faults

Introduction…

- Failure masking by redundancy
  - Redundancy is the key to masking faults
  - 3 types of redundancy
    - Information redundancy
      - Extra bits are added to allow recovery from garbled bits
      - E.g., a Hamming code can be added to transmitted data to recover from noise on the transmission line
    - Time redundancy
      - An action is performed, and then if need be, it is performed again
      - Helpful when the faults are transient or intermittent
      - E.g., transaction
    - Physical redundancy
      - Extra equipment or processes are added to make it possible for the system as a whole to tolerate the loss or malfunctioning of some components
      - Could be either hardware or software

Two-phase commit

- Distributed commit
  - Involves having an operation being performed by each member or a process group, or none at all
    - E.g., message delivered to a group, commit of a transaction
  - One-phase commit protocol
    - Established by means of a coordinator
    - Coordinator tells all other participant processes that are involved whether or not to (locally) perform the operation in question
    - Drawback
      - If one of the participants cannot actually perform the operation, there is no way to tell the coordinator

Two-phase commit…

- Two-phase commit protocol (2PC)
  - Objective: atomicity
    - Given a computation distributed across a process group, how can we ensure that either all processes commit to the final result, or none of them do?
  - Model

    Consider a distributed transaction involving the participants of a number of processes each running on a different machine

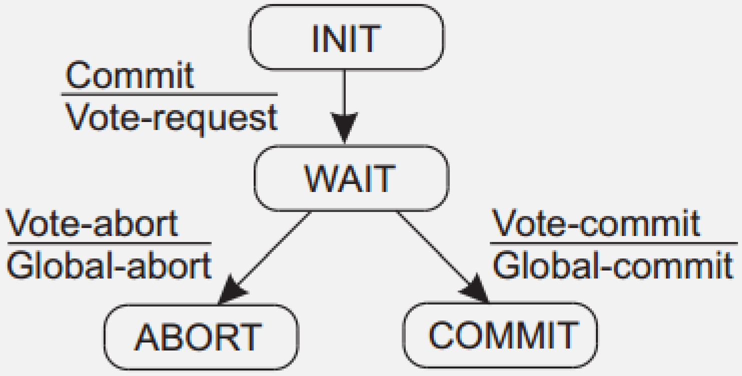
    - Coordinator: process which initiated the computation process
    - Participants: processes that are involved whether or not to (locally) perform the operation in question

Two-phase commit…

- Two-phase commit protocol (2PC)…
  - Phases 1 (Voting phase)
    1. The coordinator sends *vote-request* message to all participants
    2. When participant receives *vote-request* it returns either *vote-commit* or *vote-abort* to coordinator
       - If it sends *vote-abort*, it aborts its local computation
       - If it sends *vote-commit*, it prepares to commit by saving objects in permanent storage
  - Phase 2 (Completion phase)
    3. Coordinator collects all votes
       a) If all are vote-commit and no failure, coordinator sends *global-commit* to all participants
       b) Otherwise it multicasts *global-abort*
    4. Participants that *vote-commit* wait for *global-commit* or *global-abort* and handles accordingly and, in case of commit, confirm to coordinator

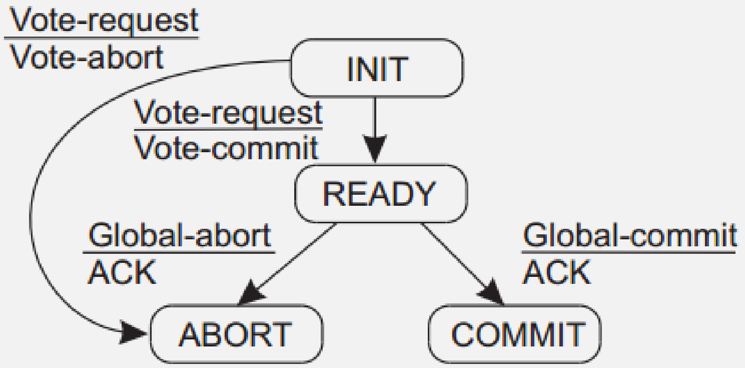
Two-phase commit…

- Two-phase commit protocol (2PC)…



(a)

Coordinator



(b)

Participant

Two-phase commit…

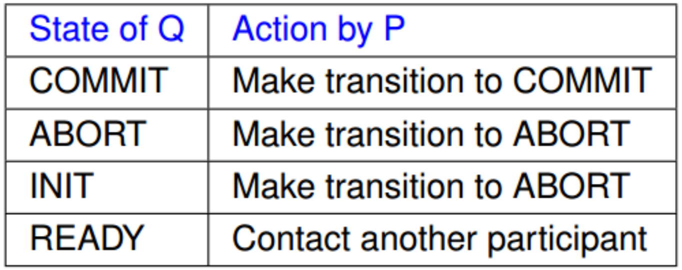
- Two-phase commit protocol (2PC)…
  - – Failure of 2PC
    - Processes could crash
    - Messages could get lost
  - – Solutions
    - Each process saves information relating to the 2PC in permanent storage
    - Timeout

Two-phase commit…

- Two-phase commit protocol (2PC)…
  - States where either coordinator or participant blocks
    - Initial state (INIT)
      - Participant waiting in init state for a *vote-request* message from the coordinator
      - If participant timeout, it will decide to locally abort the transaction and send *vote-abort* message to coordinator
    - Waiting for vote (WAIT)
      - Coordinator waits for vote of participants
      - If coordinator timeout before all votes are collected, it votes for an abort and subsequently send *global-abort* to all participants

Two-phase commit…

- Two-phase commit protocol (2PC)…
  - States where either coordinator or participant blocks…
    - READY
      - Participants blocked waiting for the global vote
      - If timeout, participant, P, could do either of the following
        - » Wait until the coordinator recovers again
        - » Contact another participant Q to see if it can decide from Q’s current state what it should do



State of Q	Action by P
COMMIT	Make transition to COMMIT
ABORT	Make transition to ABORT
INIT	Make transition to ABORT
READY	Contact another participant

- If all participants are in READY state, all has to block waiting for the coordinator to recover => blocking commit protocol

Two-phase commit…

- Two-phase commit protocol (2PC)…
  - Summary: failing participant
    - While at initial state
      - No problem, participant was not aware of the protocol
    - While at READY state
      - When participant recovers, it cannot decide on its own what it should do next, i.e., commit or abort
      - Solution
        - Contact other participants to find out what it should do
  - Performance while having N participants
    - Best case
      - 3N messages (vote-request + vote-commit + global-commit)
      - 2PC works fine without confirmation message
    - Worst case
      - There could be arbitrarily many messages

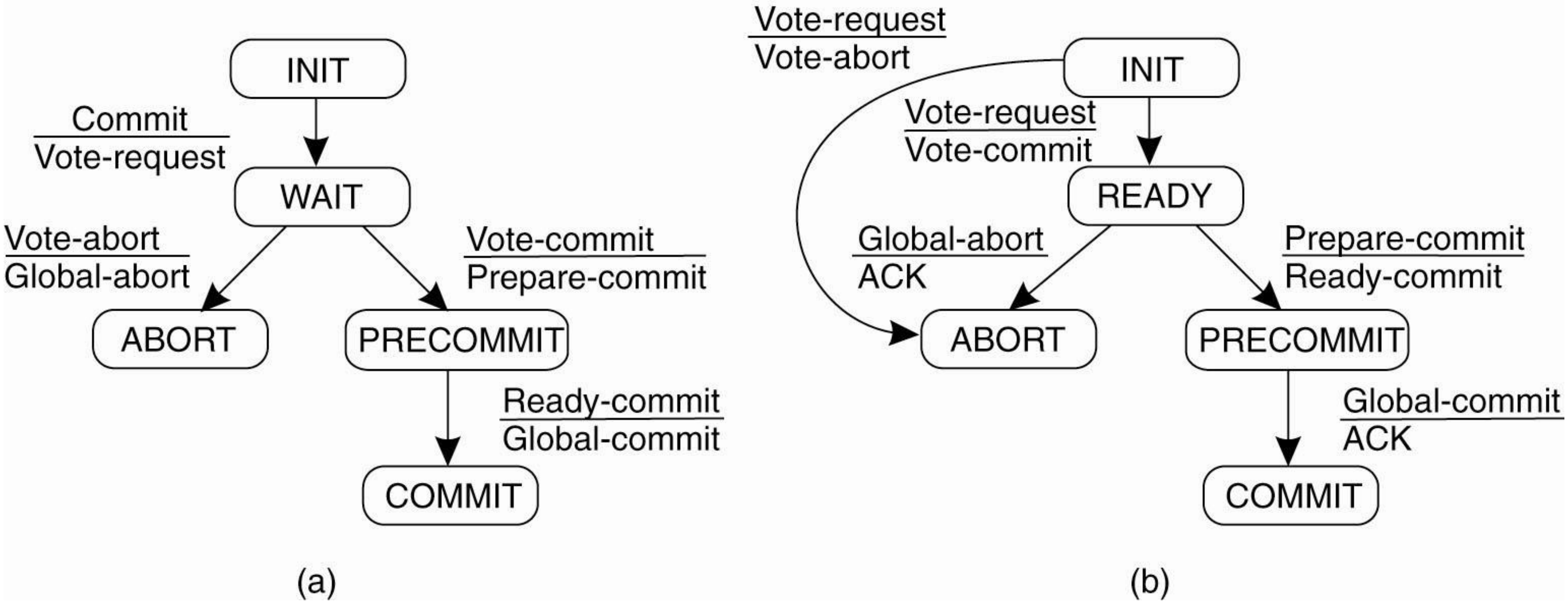
Three-phase commit

- Problem with 2PC
  - 2PC can cause a considerable delays to participants in the uncertain states
    - Occurs when the coordinator failed and cannot reply to *getDecision* requests from other participants

- Solution
  - Three-phase commit (3PC)
    - Avoids blocking processes in the presence of fail-stop crashes

Three-phase commit…

(a) coordinator, and (b) participant



Three-phase commit…

- Blocking states
  - INIT state
    - Participant waiting for a vote request from a coordinator
      - Assume that the coordinator has crashed
      - Make a transition to state ABORT
  - WAIT state
    - Coordinator waits for votes from participants
      - Assume that a participant crashed
      - Abort transaction by multicasting a GLOBAL-ABORT message

Three-phase commit…

- Blocking states…
  - READY or PRECOMMIT state
    - Participant blocked
      - Assume coordinator failed
      - Same as 2PC, contact any other participant
      - If contacted participant is in state COMMIT or ABORT, then move to the same state
      - If contacted participant is in state INIT
        - » ABORT
      - If majority of contacted participants are in state PRECOMMIT
        - » COMMIT
        - » All other processes will either be in state READY or at least, will recover to state READY, PRECOMMIT or COMMIT when they have crashed

Three-phase commit…

- Blocking states…
  - READY or PRECOMMIT state…
    - Participant blocked…
      - If majority of contacted participants are in state READY
        - » ABORT
        - » Participant may have crashed and nobody knows the state it will have when it recovers
      - Note
        - » No other participant can be in state INIT, while others are in PRECOMMIT
        - » If any operational process is in state READY, no crashed process will recover to a state other than INIT, ABORT, or PRECOMMIT
  - PRECOMMIT state
    - Coordinator blocked
      - Assume a participant that voted to COMMIT has crashed
      - Multicast GLOBAL_COMMIT
        - » Relies on a recovery protocol for the crashed participant