CSE 5306: Distributed Systems

Fault Tolerance

Jiayi Meng

http://ranger.uta.edu/~jmeng/

[Recap] Goals of Distributed Systems

1. Performance and dependability
2. Scalability
3. Distribution transparency
4. …

[Recap] Dependability

- Availability: Ready to be used immediately
- Reliability: Run continuously without failures
- Safety: No catastrophic event
- Maintainability: Easy to repair

Different Types of Node/Communication Failures

Type of failure	Description of server's behavior
Crash failure	Halts, but is working correctly until it halts
Omission failure	Fails to respond to incoming requests
Receive omission	Fails to receive incoming messages
Send omission	Fails to send messages
Timing failure	Response lies outside a specified time interval
Response failure	Response is incorrect
Value failure	The value of the response is wrong
State-transition failure	Deviates from the correct flow of control
Arbitrary failure	May produce arbitrary responses at arbitrary times

How to Achieve Failure Transparency?

Failure Can be Masked by Redundancy

- Information redundancy
  - Extra bits are added to be able to recover from errors, e.g., Hamming code
- Time redundancy
  - The same action is performed multiple times
- Physical redundancy (in both hardware and software)
  - Extra equipment or processes are added to tolerate malfunctioning components

How to Tolerate a Faulty Process?

- Key approach: Organize several identical processes into a group
  - Replicate processes and organize them into a group
  - A group of processes jointly behave as a single, highly robust process

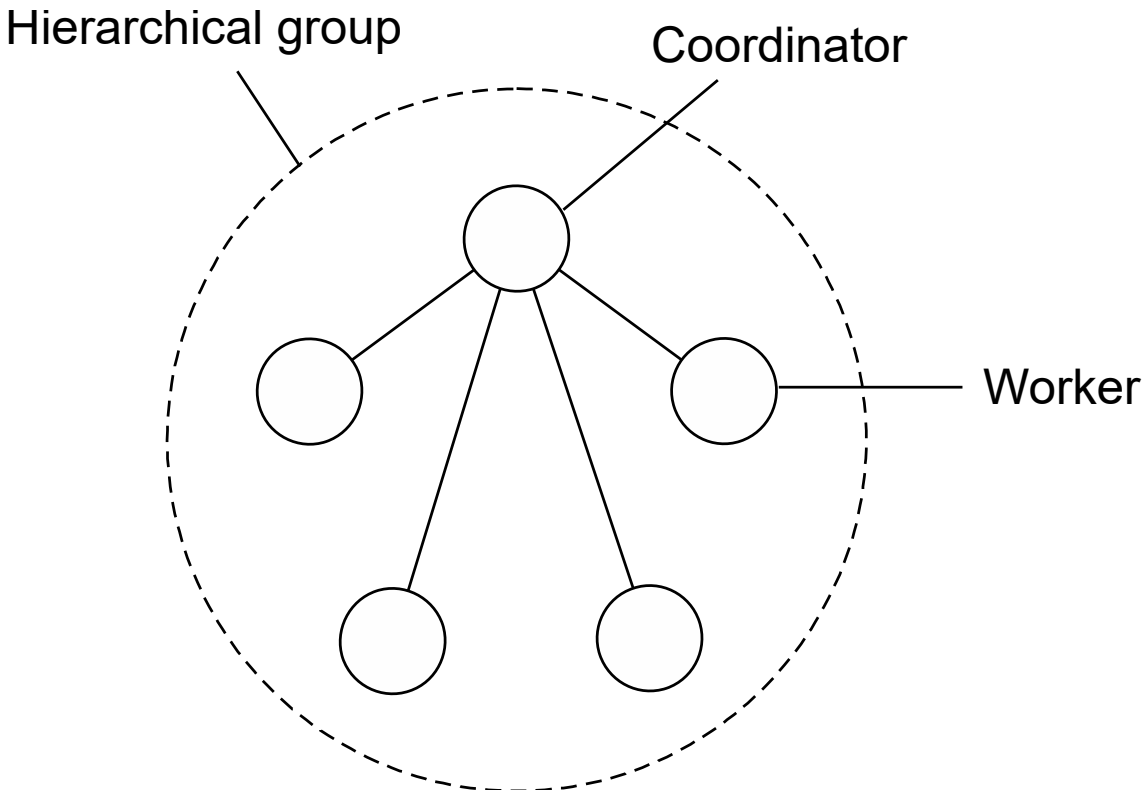
- Key property of the group
  - When a message is sent to the group itself, all members of the group receive it

  → If one process in a group fails, some other processes can take over for it

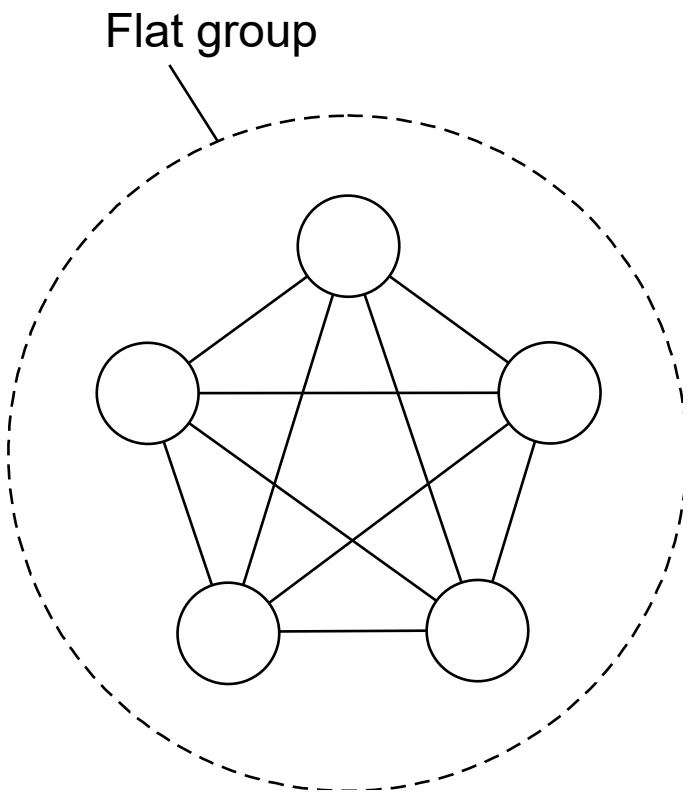
- A process can send a message to a group of servers
  - The group appears to be a single, logical process
  - W/o having to know who they are, how many there are, or where they are

How to Approach Such Replication?

- Primary-based protocols
  - A group of processes is organized in a hierarchical fashion



- Replicated-write protocols



How to Reach Consensus?

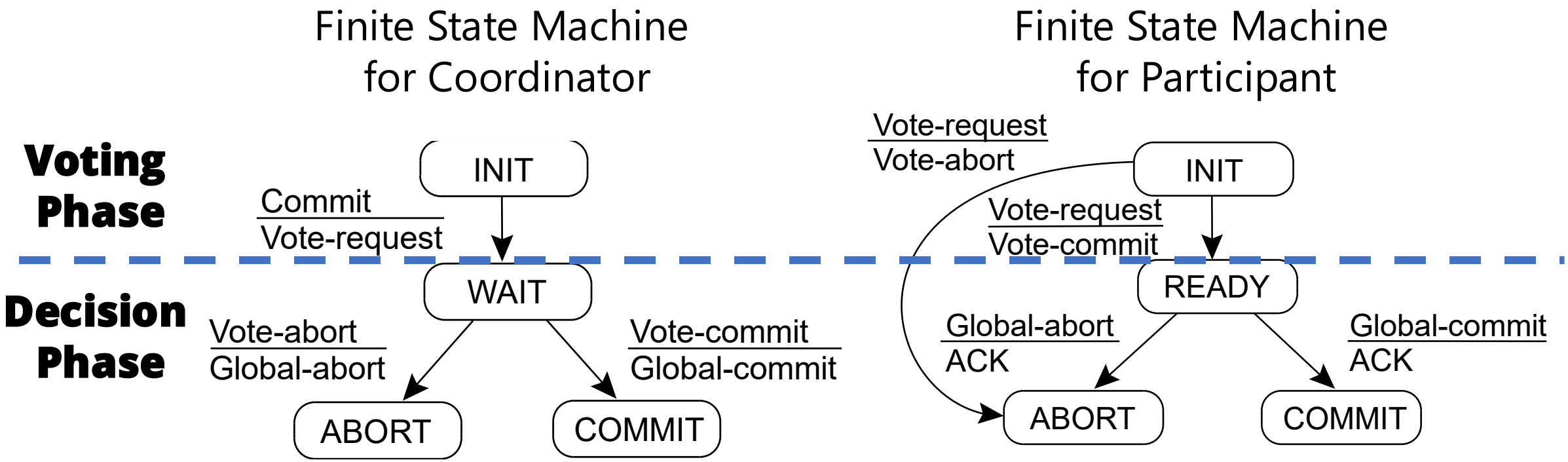
- A fundamental problem in fault-tolerant distributed systems
- A consensus algorithm is a process to achieve agreement on a single data value among distributed processes
- To replicate changes on different processes
  - A simple approach is to send them in order
  - What's the potential issue?

Distributed Commit

- Requires an operation being performed by all processes in the group or none at all
  - Distributed atomic transaction

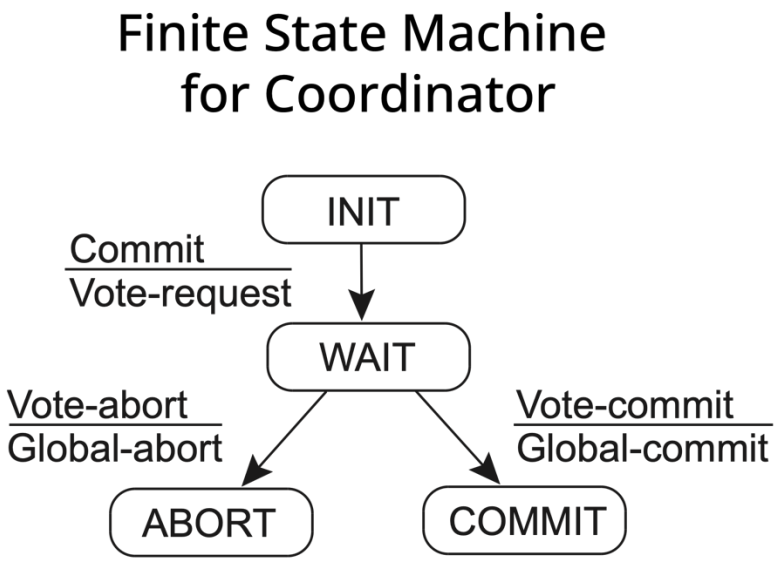
- It is often achieved by means of a coordinator
  - One-phase commit protocol
    - The coordinator tells everyone what to do
    - However, no feedback when a member may fail to perform
  - Two-phase commit protocol (2PC)
  - Three-phase commit protocol (3PC)

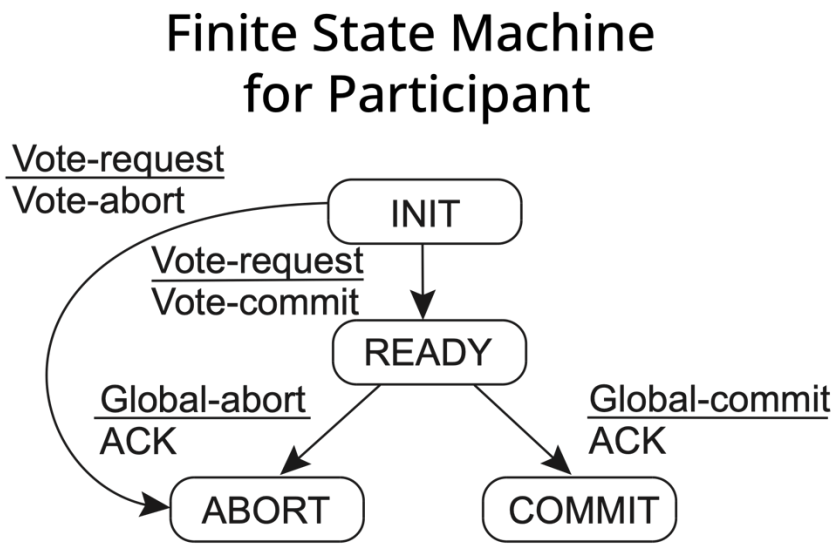
Two-Phase Commit (2PC)



Brainstorm: How to Deal with Failures?

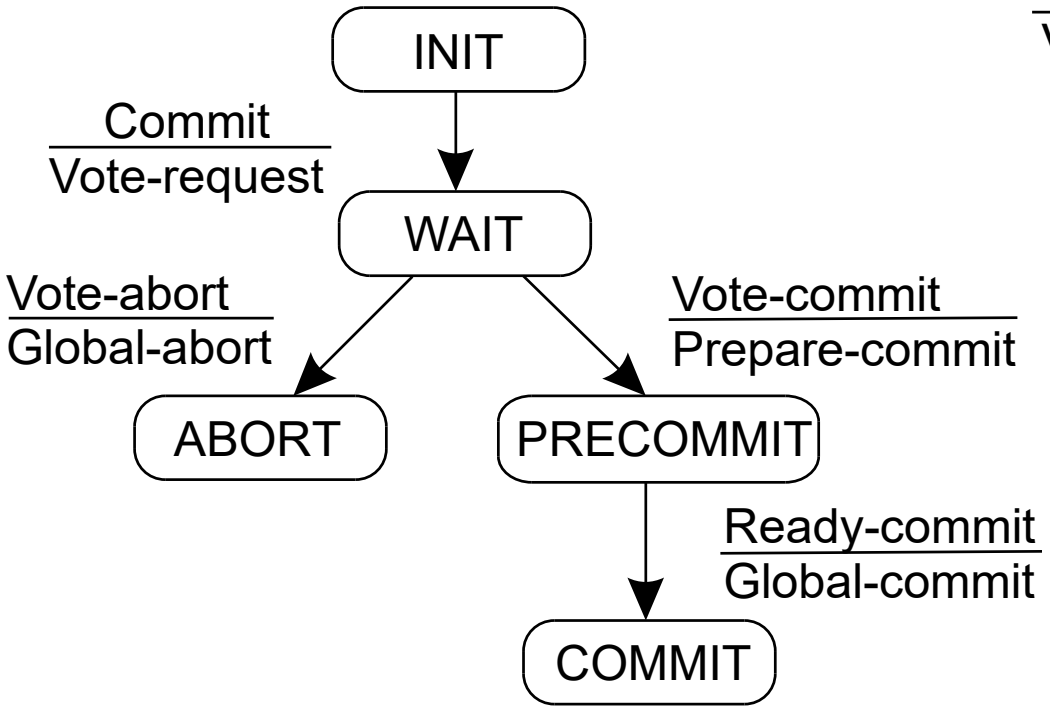
- What if coordinator is blocked in WAIT?
- What if participant is blocked in READY?



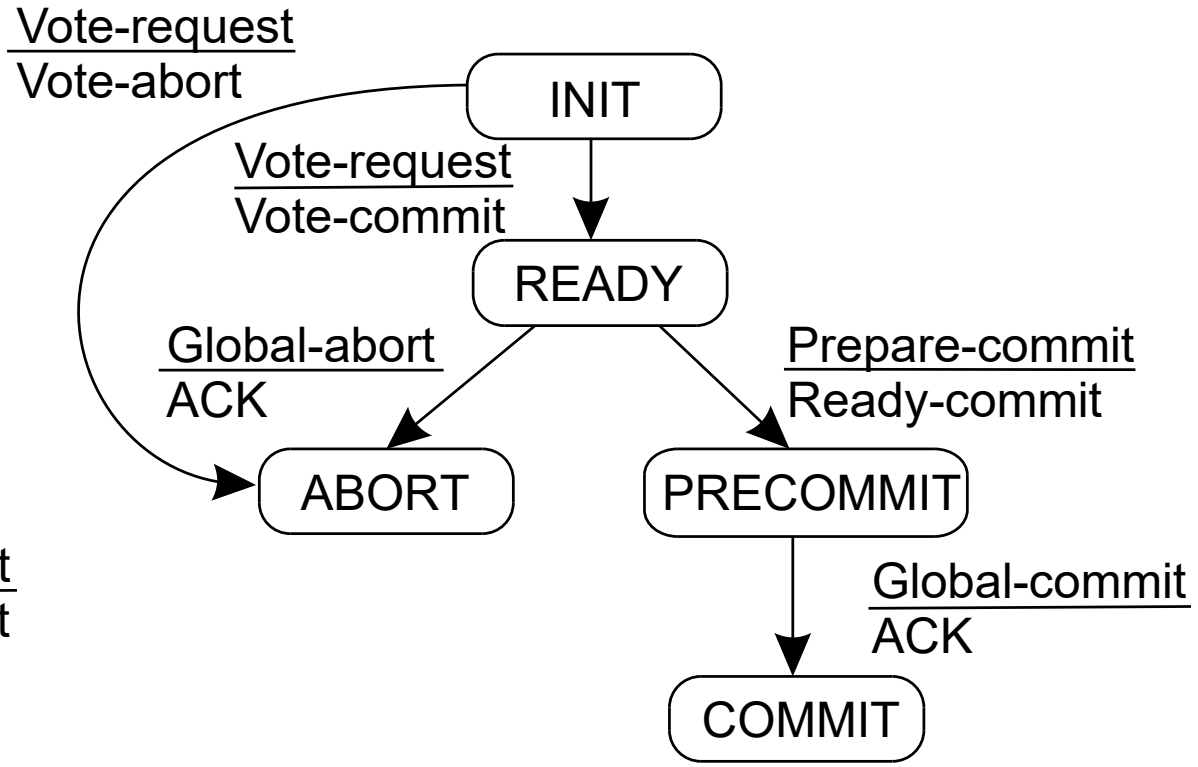


Homework: Three-Phase Commit (3PC)

Finite State Machine
for Coordinator



Finite State Machine
for Participant



How to Achieve K-Fault Tolerance?

- Consider replicated-write systems
- K-fault tolerance
  - The system can survive faults in k components and still meet its specifications
- How many processes at least are needed?
  - If k components fail silently
  - If k components exhibit Byzantine failures
    - Continue to run and send out erroneous or random replies

How to Reach Consensus in Faulty Systems with Crash Failures?

- Consensus: Commit operations in the same order
  - Values are the same

- Achieving consensus is easy when there are no failures
  - Either need a system-wide coordinator
  - Or perform totally-ordered multicast or casually-ordered multicast

Raft

- Raft: A distributed consensus algorithm
  - Primary-backup protocol
    - Primary: Leader
    - Backups: Followers

- Each node/server maintains
  - Three states: leader, follower, candidate
  - A log of operations
    - Log contains operations: (1) have already been committed; (2) are pending
    - Log: <o, t, k>
      - t: The term under which the current leader serves
      - k: The index of o in the leader's log
  - Current election term

Raft: Leader Election

- Two timeout settings are maintained: heartbeat and election

- Leader send periodic heartbeats to all followers
- If follower receives no communication over a period of time (election), it begins a new term and a leader election → Follower becomes candidate
  - Increment its current term
  - Request all servers to vote
    - Each server votes for at most one candidate on a first-come-first-served basis
    - *Server will not vote for candidate, if it has more up-to-date log*
      - Compare index and term of the last entries in logs
        - If logs end with the same term, whichever log is longer is more up-to-date
  - Wins the election if it receives votes from a majority of the servers
    - Then sends heartbeat to all followers

Raft: More about Leader Election

- No winner due to a split vote → Some follower starts a new term

- How to avoid split vote?
  - Election timeouts are chosen randomly from a fixed interval
    - E.g., 150-300 ms
  - This spreads out the servers → In most cases only a single server will time out