CSCI 5105

Instructor: Abhishek Chandra

Today

- Process Resilience
  - Process Groups
  - Consensus Algorithms
  - CAP Theorem

Process Resilience

- How to protect against process failure?
- How to ensure correct results?

Process Groups

- Use replication:
  - Multiple replicas/copies of a single server
- Primary-based:
  - One primary, other redundant servers
- Flat group:
  - All are identical, need agreement among servers

Amount of Redundancy

- Depends on:
  - How many faults can a system handle?
  - What kind of faults can happen?
- k-fault tolerant system:
  - Can handle k faulty servers

5

Client-Server Environment

- Client needs only one response
  - Each server has ability to respond
- How many total servers do we need for a k-fault tolerant system if failures are:
  - Fail-stop/fail-silent?
  - Byzantine?

6

Consensus (Group Agreement)

- Servers need to agree on a common value/state
  - The state is distributed across servers
  - The value(s) may be proposed by servers
- Examples?
- Two desired properties:
  - Safety: Nothing bad will happen
  - Liveness: Progress will eventually happen
- Hard problem, depends on:
  - Reliability of communication channel
  - Behavior of faulty servers

7

Consensus: Feasibility

- Factors:
  - Process behavior: Synchronous or asynchronous
  - Communication delay: bounded or unbounded
  - Message ordering: Ordered or unordered
  - Message transmission: Unicast or multicast
- Can achieve agreement if:
  - Synchronous: Bounded delay or ordered messages
  - Asynchronous: Not possible in general

8

Two-Army Problem

- Simple scenario:
  - Two armies (perfect servers)
  - Communicate through a messenger that can be caught (unreliable communication channel)
  - Both need to agree on a common value
- Question: Can they agree?
- Example: TCP connection termination

9

Consensus Algorithms

- Crash failures: Paxos
- Byzantine failures: Byzantine Fault Tolerance

10

Consensus with Crash Failures

- Assumption: Only crash failures
- Goal: Process group appears as a single, highly robust process
  - Every nonfaulty process sees the same state sequence (values, commands) as every other nonfaulty process

11

Paxos

- Assumptions:
  - Partially synchronous system
  - Unreliable network
  - Nodes can crash, but have stable storage (so they can resume from the pre-crash state)
  - Fail-noisy failure model: Crash failures (eventually detected), no Byzantine faults or collusion
- Used in Google's Chubby, Apache Zookeeper coordination services

12

Paxos: Basics

- Group of nodes
- A general agreement algorithm
  - Nodes can propose different (and multiple) values
- Goals:
  - All nodes must agree on the same value
  - An agreed value must have been proposed

Paxos: Scenarios

- Multiple processes propose different values concurrently
- Some processes may fail, so that:
  - They may not receive the value
  - They may receive the value, but others may not be aware
- A failed process may return with an old value

Paxos: Entities

- 3 types of nodes:
  - Proposers: Propose values
  - Acceptors: Accept (or reject) values
  - Learners: Learn the eventually accepted value
- Different processes can have different roles, or the roles can be overlapping

Paxos Algorithm: Overview

- Each proposer can propose a value v with a timestamp t
  - All proposals have unique timestamps
- Goal: Pick a value v from all proposals
- Insights:
  - Only need a quorum of acceptors to agree on a value
  - Once a value is picked, then older proposals can be rejected
  - Once a value is picked by a quorum, then subsequent proposers must agree to this value

Paxos Algorithm: Phase 1

- Phase 1a (Prepare Phase): Proposer sends a proposal <t,v> to a quorum of acceptors
- Phase 1b (Promise Phase): An acceptor can reply with a:
  - Promise (not to accept a lower timestamped proposal, sends the value of previous highest timstamp accepted proposal)
  - Reject (Won't accept this proposal)

Paxos Algorithm: Phase 2

- Phase 2a (Accept Phase): Proposer getting Promises from quorum sends Accept
  - With a value (highest-timestamped proposal's value seen so far, or v otherwise)
- Phase 2b (Learn Phase): Accepters send notification of accepted value to learners

Paxos: Benefits

- Works if machines crash and resume:
  - If proposer sends low timestamp, it can be rejected
  - If an acceptor comes back with an old Promise, this can be superseded by a newer proposal
- The final value can be propagated by any learner
- Robust to network partitions
  - If one partition has a majority of acceptors

Byzantine Generals Problem

- N generals, M traitors (One commander)
- Problem: Traitors can lie, others don't know who the traitors are
- Question: Can trusted generals agree on whether to attack or retreat?
- Assumptions:
  - Reliable communication channel
  - Receiver of message can detect the sender

Byzantine Agreement: Requirements

- BA1: Every nonfaulty backup process stores the same value.
- BA2: If the primary is nonfaulty then every nonfaulty backup process stores exactly what the primary had sent.

21

Byzantine Agreement: Feasibility Condition

- Must have: N >= 3M+1 for agreement
- Why?

22

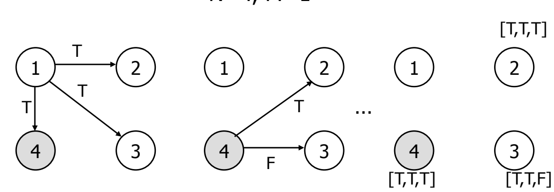
Byzantine Agreement Algorithm

- Recursive, Round-based
- Round 1: BAP (n, k): Commander sends a value to n-1 generals assuming k traitors (terminating condition: k=0)
- In each subsequent recursive round i:
  - Each general executes multiple instances of BAP (n-i,k-i+1)
  - Acts as a primary sending each value received in previous round to a subset of (n-i) generals (those not involved in the routing of the given value)
  - Each general determines the current round value by voting among received values
  - Pass the value up by recursion

23

Byzantine Agreement: Feasible Case

N=4, M=1

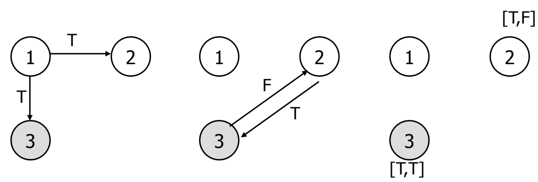


BAP(4,1)   BAP(3,0)

24

Byzantine Agreement: Infeasible Case

N=3, M=1



BAP(3,1) BAP(2,0)

N=3M+1 for agreement

25

Reliability with Network Partitioning

- What if network is partitioned?
- Can all processes still see the same state?
- Need to trade off safety with liveness

26

CAP Theorem

- C: Consistency
- A: Availability
- P: Tolerance to network partitions

- “2 of 3” rule: Can have only 2 of these properties
- Examples?

CAP Theorem - Revisited

- Is “2 of 3” rule misleading?

- Partitions are rare, can have all 3 most of the time
- Granularity of C and A can vary: whole system, subsystem, operation/data-specific
- C, A, P are continuous (non-binary) properties

- Partitions have to be managed

Partition-Latency Relation

- How would a node detect partitions in practice?
- Related to communication latency
  - Network latency
  - Can be defined based on app latency requirements
- No global definition of partitioning
  - Different nodes may (or may not) detect network partition

Partition Management

- How to handle partitions when they occur?
- Key questions:
  - How to operate during a partition?
  - How to recover after re-connection?

Partition Mode

- How should the two sides operate under a partition?
  - Allow some operations. E.g.: those that do not conflict or could be resolved easily
  - Delay some and prohibit some. E.g.: those that need to be globally consistent
  - Maintain operation history. E.g.: version vectors

Partition Recovery

- Make state consistent on both sides
- Roll forward from a state before the partition
  - Use the logs to apply/merge operations
- How to merge conflicts?
  - Use version vectors
  - Might need manual intervention
  - Automated if we allow only limited operations. E.g.: only commutative operations
- Compensate for mistakes
  - Cancel a duplicate operation