Models of Distributed Systems

Ashvin Goel

Electrical and Computer Engineering
University of Toronto

Distributed Systems
ECE419

With thanks to Tim Harris and Martin Kleppmann,
Lecture notes on Concurrent and Distributed Systems

Overview

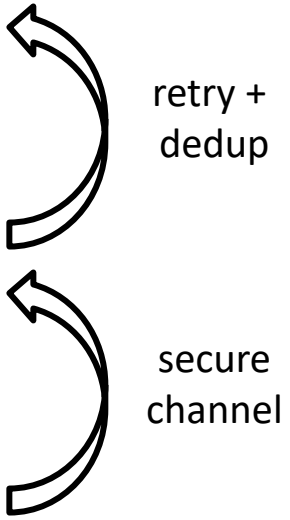
- System models
- Failures and failure detectors

System models

- A machine (computer, phone, car, etc.) is called a node
- A model of a distributed system specifies assumptions about faults that may occur
  - Network behavior (e.g. message loss)
  - Node behavior (e.g. crashes)
  - Timing behavior (e.g. delays)

System model: network behavior

- Assume bidirectional, point-to-point communication between two nodes, with one of:
  - Reliable links
    - A message is received if and only if it is sent
    - Messages may be reordered
  - Best-effort links
    - Messages may be lost, duplicated, or reordered
    - If you keep retrying, a message eventually gets through
  - Insecure links
    - A malicious adversary may interfere with messages, e.g., eavesdrop, modify, drop, spoof, replay



- Network partition: some links drop/delay all messages for extended periods of time

System model: node behavior

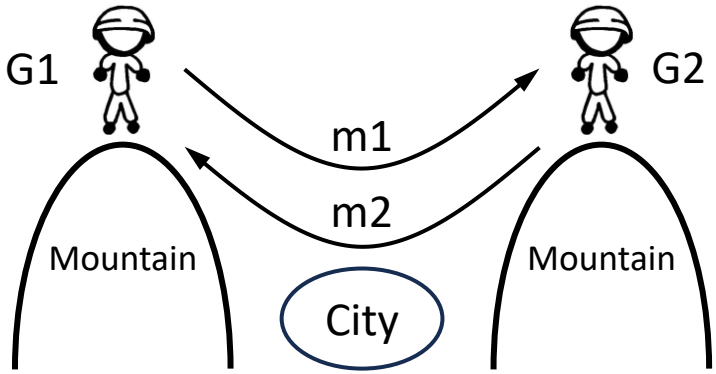
- When executing an algorithm, assume node may have:
  - Crash-stop (fail-stop) failure
    - A node may crash at any time, e.g., due to power failure
    - After crashing, it stops executing forever
  - Crash-recovery (fail-recovery) failure
    - A node may crash at any time, losing its in-memory state
    - It may resume executing sometime later
    - Data on non-volatile storage (e.g., disk, SSD) survives crash
  - Byzantine (fail-arbitrary) failure
    - A node is faulty if it deviates from the algorithm
    - Faulty nodes may execute incorrectly, including being malicious
- A node that is not faulty is called correct

System model: timing behavior

- Assume one of the following for network and nodes:
  - Synchronous
    - Message latency no greater than a known upper bound
    - Nodes execute algorithm at a known speed
    - Algorithms easier to design, but unrealistic assumptions
  - Asynchronous
    - Messages can be delayed arbitrarily
    - Nodes can pause execution arbitrarily
    - No timing guarantees at all
    - Algorithms more robust, but hard to design, sometimes impossible …

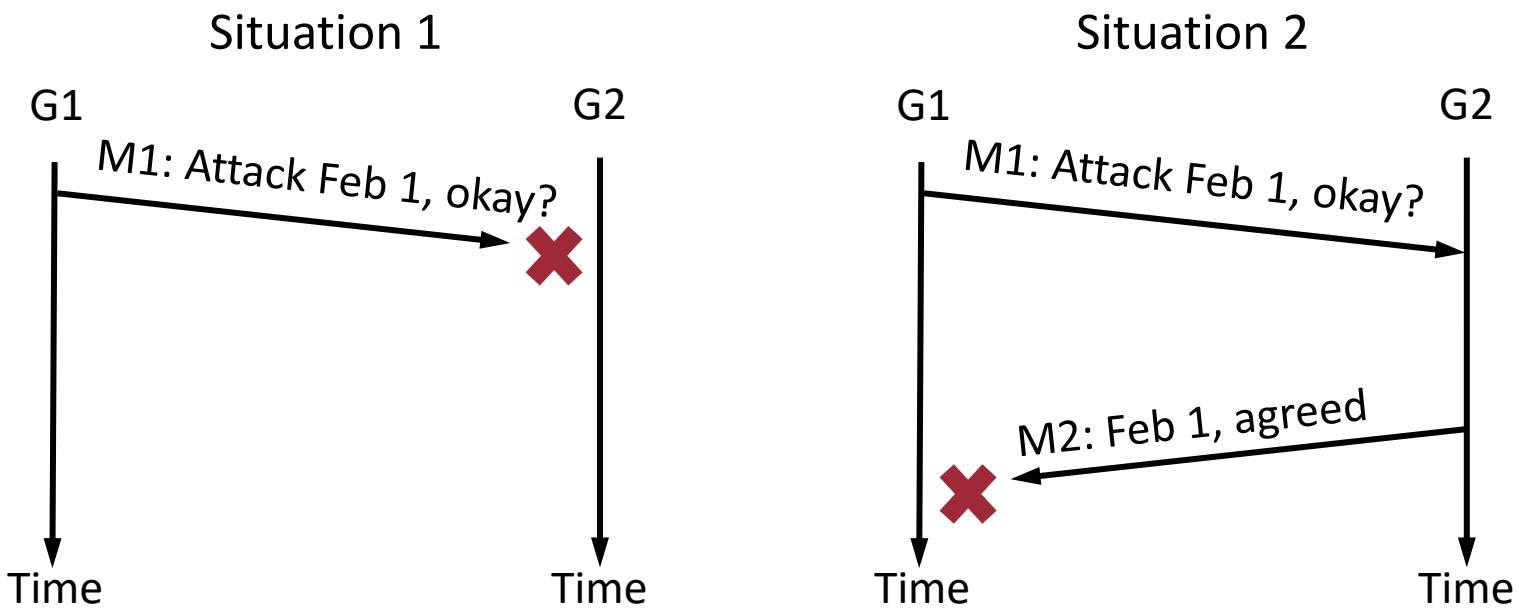
Two Generals problem

- A thought experiment that shows the challenge with coordinating actions over asynchronous links
  - Used to motivate the atomic commit problem (discussed later)
- Problem:
  - Two generals need to agree to attack to win (or else they lose)
  - Only communicate by sending messages (that may not arrive)



Two Generals dilemma

- For G1, Situations 1 and 2 are indistinguishable
- Should generals attack before or after receiving reply?
  - If before, then general may lose
  - If after, then generals wait forever … why?



System model: timing behavior

- Let's look at a third timing model:
  - Synchronous
    - Message latency no greater than a known upper bound
    - Nodes execute algorithm at a known speed
    - Algorithms easier to design, but unrealistic assumptions
  - Asynchronous
    - Messages can be delayed arbitrarily
    - Nodes can pause execution arbitrarily
    - No timing guarantees at all
    - Algorithms more robust, but hard to design/make guarantees
  - Partially synchronous
    - The system is asynchronous for some finite (but unknown) periods of time, synchronous otherwise
    - Practical algorithms designed for realistic environments

Why partially synchronous?

- Networks usually have predictable latency,
  but latency may increase due to:
  - Message loss requiring retry
  - Congestion/contention causing queueing
  - Network/route reconfiguration
- Nodes usually execute code at a predictable speed,
  but occasional slowdown may occur due to:
  - IO accesses
  - Operating system scheduling
  - Stop-the-world garbage collection pauses
  - Page faults, swapping, thrashing

Summary of system models

- For each of the three, pick one:
  - Network: reliable, best-effort, or insecure
  - Nodes: crash-stop, crash-recovery, or byzantine
  - Timing: synchronous, asynchronous, or partially synchronous
- These models are the basis for any distributed system!
  - If your fault assumptions are wrong, all bets are off!

Failures and Failure Detectors

Availability

- Online store wants to sell stuff 24/7!
  - Service unavailability = downtime = financial loss
- Availability: fraction of time service functions correctly
  - Two nines = 99% up = down 3.7 days/year
  - Three nines = 99.9% up = down 8.8 hours/year
  - Four nines = 99.99% up = down 53 minutes/year
  - Five nines = 99.999% up = down 5.3 minutes/year
- Service-Level Objective (SLO): availability expectation
  - e.g., 99.9% of requests in a day get a response in 200 ms
- Service-Level Agreement (SLA): contract specifying an SLO, with penalties for violation

Achieving high availability

- Failure: system stops working, causes unavailability
- Fault: part of system fails, may also cause unavailability
  - Node fault: crashes, malicious behavior
  - Network fault: packet drops, delays, network partitions
- Improve availability by
  - Reducing frequency of faults
    - E.g., use higher quality and redundant hardware
  - Software fault tolerance
    - Ensure that system continues working, despite some faults
      - E.g., avoid single point of failure, a single node/link fault leads to failure
    - As a first step, requires detecting failures

Failure (Fault) detectors

- Failure detector: algorithm that detects whether another node is faulty (typically, crashed)
- Perfect failure detector: labels a node as faulty if and only if it has crashed
- Typical implementation for crash-stop/crash-recovery
  - Send message, await response,
    label node as crashed if no reply within some timeout
- Problem: cannot reliably tell the difference between crashed/delayed node, lost/delayed message

Failure detection in partially synchronous systems

- Perfect timeout-based failure detector only possible in synchronous crash-stop system with reliable links
  - But real systems are partially synchronous or asynchronous!
- Eventually perfect failure detector possible in partially synchronous systems
  - May temporarily mislabel a correct node as crashed
  - May temporarily mislabel a crashed node as correct
  - But eventually, labels a node as crashed iff it has crashed
- Reflects the reality that detection is not instantaneous, so spurious timeouts may occur

Conclusions

- The system model specifies assumptions about 1) faults in networks, 2) faults in nodes, and 3) timing behavior
- Correct design of distributed algorithms and systems depends on these assumptions
- Distributed systems strive to provide high availability
  - With large systems, h/w failures become more common
  - Need to detect and tolerate faults in software
  - Cannot assume accurate failure detection in real systems