06-06798 Distributed Systems

Lecture 9:
Distributed Algorithms

Overview

- Distributed algorithms
  - achieve co-ordination, agreement, etc
- Examine sources of difficulties
  - timing
  - interaction model
  - failures
- and effect on distributed algorithms
  - impossibility results
  - increase in complexity and sophistication

Distributed algorithms

- Sequential algorithm
  - sequence of steps to be taken (by a single process) to arrive at a solution
- Distributed systems
  - multiple processes, each with own variables
  - communication by exchanging messages
  - form a graph, some topology (ring, arbitrary)
- Distributed algorithm
  - sequence of steps to be taken by each process, including transmission of messages, to arrive at a solution

Why difficult?

- In sequential algorithms
  - steps taken in strict sequence
  - rate of execution immaterial
- In distributed systems
  - no global time
  - processes execute at different, unpredictable rates
  - communication latency and delays
  - failures must be dealt with
  - processes have local state: true global state of the system difficult to observe

Examine combined effect of:

- Timing
  - clocks, local/global time
  - time used in: timestamp, event ordering
- Interaction model
  - synchronous/asynchronous
- Failures
  - benign (omission, timing)
  - Byzantine

Clocks and timing

- Internal clocks
  - record local time
  - count at different rate
    - clock drift = relative amount of time by which clock differs from a perfect clock
  - different time values if read at the same time
- Problems
  - local time unreliable when used as timestamp
  - correction must be applied (clock synchronisation)
  - event ordering difficult (logical time)

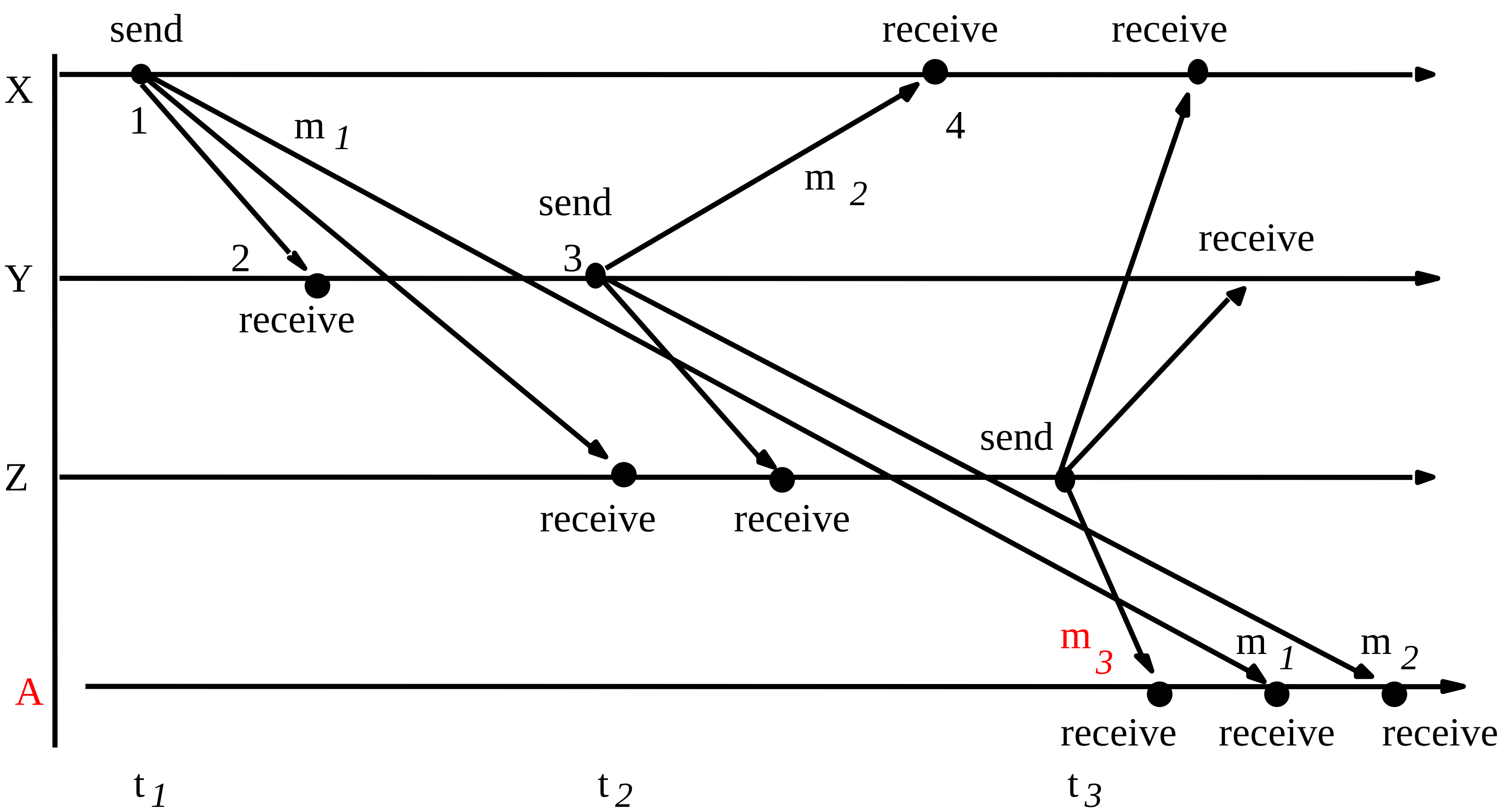
Event ordering

- Scenario: group of email users X, Y, Z, A
  - X sends message to group
  - Y, Z reply to group
- In real-time
  - X sends message first; Y reads it & replies
  - Z reads both & replies
- What can happen…
  - A sees messages in this order: from Z, X, Y
- Solution [Lamport'78]
  - record logical time

Logical time

- Now known as Message Sequence Charts
- Each process
  - has local time axis
  - records own events in linear order
- Communication
  - represented by arrows between processes
  - ordered locally according to send/arrival time
- Global event ordering
  - can be deduced without global time
  - partial order

Example of logical time



X, Y: send before receive, local order within process yields order 1,2,3,4

Interaction models

- Synchronous:
  - known upper/lower bounds on execution speeds, message transmission delays and clock drift rates
    - each takes at least MIN but no more than MAX time units
  - conceptually simpler model
- Asynchronous:
  - arbitrary process execution speeds, message transmission delays and clock drift rates
  - more general: if solution valid for asynchronous then also valid for synchronous

The synchronous model

- Simpler:
  - can make assumptions about delays, drift rates, etc
- But more difficult/expensive to build
  - need guarantees of delivery times, clock drift, ...
- Some algorithms easier: coordinated attack
  - two armies: initiator leads, both must attack together
  - suppose know bounds on message delays (MIN, MAX time units) and no failures
    - (One) sends Charge!, waits for MIN time units and charges
    - (Two) receives Charge!, waits for 1 time unit and charges
  - then One leads, Two is guaranteed to charge within MAX-MIN+1

What if asynchronous?

The asynchronous model

- More realistic:
  - no assumptions about delays, drift rates, etc
  - cf Internet, WANs:
    - routers introduce delays (messages may take a long time)
    - unpredictable load on server (affects response time)
    - processor sharing (affects execution time)
- But algorithms more difficult:
  - previous solution to co-ordinated attack does not work: suppose no bounds on message delays and no failures
    - choose sufficiently large T
    - (One) sends Charge!, waits for T time units and charges
    - (Two) receives Charge!, waits for 1 time unit and charges
    - cannot guarantee One leads (message may take longer than T)

Failures...

- Make the situation much worse:
  - message may fail to arrive (omission failure)
  - process may stop and others may detect this (stopping failure)
  - process may crash and others cannot detect this (crash failure)
- Types of failures
  - benign
    - omission, stopping, timing/performance
  - arbitrary (called Byzantine)
    - corrupt message, wrong method called, wrong result

Distributed consensus

- Often needed to
  - commit/abort transactions in distributed databases
  - agree on altitude on board of an aeroplane
- Here: coordinated attack (synchronous model, omission failures)
  - graph (processes are nodes, links are arcs)
  - initial opinion Charge! or Surrender!
  - all must attack together, otherwise destroyed
  - communicate via messengers (can be captured or lost)
  - must agree whether to attack or not, & attack if possible
- Solution possible if messengers reliable (see earlier)

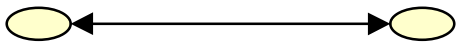
Consensus requirements

- Agreement
  - no two processes decide on different values
- Validity
  - if all start with Charge! then this is the only possible decision value
  - if all start with Surrender! then this is the only possible decision value

    (other variants possible…)
- Termination
  - all processes eventually decide

Impossibility result

- There is no deterministic solution that solves the co-ordinated attack problem even on on this graph:



- Solutions
  - make probabilistic assumptions about the loss of messages while keeping processes deterministic
    - errors may happen with some probability
  - use randomisation while allowing some violation of validity/agreement

Outline of argument

- Assume there exists an algorithm (for contradiction)
  - processes propose Charge!, Surrender!
  - exchange a set of messages
  - eventually agree
- Consider the last message in exchange
  - messenger could be captured!
  - result the same if message deleted, can dispense with it
- Repeat for the remaining messages
  - left with no message
- Conclusion: no algorithm exists (for this graph)

Process crash failures

- Crash failures
  - process stops executing, does not respond
- Crash detection
  - use timeouts
  - in synchronous model: can detect crash
    - how?
  - in asynchronous model: cannot distinguish if
    - it has crashed,
    - is slow, or
    - message failed to arrive!

Stopping failures

- Stopping failure (or fail-stop crash)
  - process stops executing
  - others can for certain detect this
- Detection
  - in synchronous model: use timeouts plus guaranteed message delivery
    - if message failed to arrive can deduce stopping failure has occurred
    - but if it arrives can we deduce no stopping failure has occurred?
  - in asynchronous model: more difficult! cannot distinguish if
    - message takes too long to arrive, or
    - stopping failure has occurred

Byzantine failures

- Also called arbitrary
  - worst possible error
  - system or component malfunction
  - wrong values, wrong method
- Examples
  - memory fault
  - where no checksums: corrupt messages
  - where no message sequence numbers: duplicate messages

Byzantine failures

- Many difficulties!
  - in asynchronous model impossibility result:

    three processes cannot solve Byzantine agreement even in the presence of one failure
  - need $n > 3f$ where n number of processes, f failures
- Solutions
  - can tolerate up to a certain number of failures
  - increased complexity
  - use of randomisation

Timing/performance failures

- Can occur in synchronous systems
  - server overloaded, slow response
  - often not critical (poor response time)

Class of Failure	*Affects*	*Description*
Clock	Process	Process's local clock exceeds the bounds on its rate of drift from real time.
Performance	Process	Process exceeds the bounds on the interval between two steps.
Performance	Channel	A message's transmission takes longer than the stated bound.

Summary

- Distributed algorithms are sensitive to:
  - types of interaction models
  - types of failures
  - timing
- Impossibility results
  - very common!
- Design issues
  - control timing if possible, allows timeouts
  - partial synchrony
  - guaranteed delivery of messages