LINFO2345

Teaser

Introduction to Distributed Systems

Instructor: Peter Van Roy peter.vanroy@uclouvain.be

Assistant: Mathieu Pigaglio mathieu.pigaglio@uclouvain.be

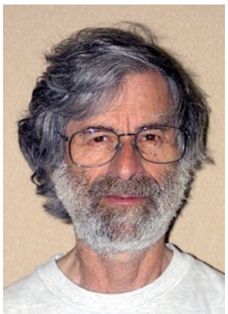
Web: Moodle

Slides by Seif Haridi, Ali Ghodsi, Peter Van Roy

1

What's a distributed system?

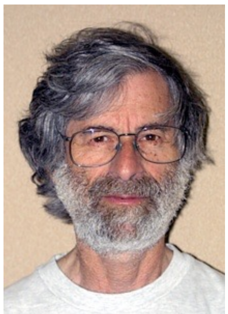
"A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable."



2

What's a distributed system?

"A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable."



Leslie Lamport

What's a distributed system?

- "A set of **nodes**, connected by a **network**, which appear to its users as a **single** coherent system"

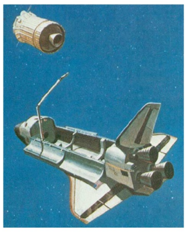
We focus on concepts, models, and foundations

Why study distributed systems?

- It is important and useful
  - Societal importance
    - Internet
    - WWW
    - Small devices (mobiles, sensors)
  - Technical importance
    - Improve scalability
    - Improve reliability
    - Inherent distribution

Increasing Reliability

- PASS developed by IBM in 1981, used in a space shuttle
  - Could have been done on one node
  - But 4 separate nodes used for fault-tolerance
    - Voting on outcome



Why study distributed systems?

- It is very challenging
  - Partial Failures
    - Network (dropped messages, partitions)
    - Node failures
  - Concurrency
    - Nodes execute in parallel
    - Messages travel asynchronously

  } Parallel computing

- Recurring core problems

Core Problems

What types of problems are there?

Teaser: Two Generals' Problem

- Two generals need to coordinate an attack
  - Must agree on time to attack
  - They'll win only if they attack simultaneously
  - Communicate through messengers
  - Messengers may be killed on their way

Teaser: Two Generals' Problem

- Lets try to solve it for generals g1 and g2
- g1 sends time of attack to g2
  - Problem: how to ensure g2 received msg?
  - Solution: let g2 ack receipt of msg
  - Problem: how to ensure g1 received ack
  - Solution: let g1 ack the receipt of the ack…
  - …
- This problem is impossible to solve!

Teaser: Two Generals' Problem

- Applicability to distributed systems
  - Two nodes need to agree on a value
  - Communicate by messages using an unreliable channel
- Agreement is a core problem…

Consensus: agreeing on a number

- Consensus problem
  - All nodes propose a value
  - Some nodes might crash & stop responding
- The algorithm must ensure:
  - All correct nodes eventually decide
  - Every node decides the same
  - Nodes only decide on proposed values

Consensus is Important

- Databases
  - Concurrent changes to same data
  - Nodes should agree on changes
- Use a kind of consensus: atomic commit
  - Only two proposal values {commit, abort}

Broadcast Problem

- Atomic Broadcast
  - A node broadcasts a message
  - If sender correct, all correct nodes deliver msg
  - All correct nodes deliver same messages
  - Messages delivered in the same order

Atomic Broadcast ≡ Consensus

- Given Atomic broadcast
  - Can use it to solve Consensus
- Every node broadcasts its proposal
  - Decide on the first received proposal
  - Messages received in same order
    - All nodes will decide the same
- Given Consensus
  - Can use it to solve Atomic broadcast [d]
- Atomic Broadcast equivalent to Consensus

Concurrency Aspects

How to reason about them?

Modeling a Distributed System

- Asynchronous system
  - No bound on time to deliver a message
  - No bound on time to compute
- Internet is essentially asynchronous

Impossibility of Consensus

- Consensus cannot be solved in asynchronous system
  - If a single node may crash
- Implications on
  - Atomic broadcast
  - Atomic commit
  - Leader election
  - …

Modeling a Distributed System

- Synchronous system
  - Known bound on time to deliver a message
  - Known bound on time to compute
- LAN/cluster is essentially synchronous

Possibility of Consensus

- Consensus solvable in synchronous system
  - With up to N-1 crashes
- Intuition behind solution
  - Accurate crash detection
    - Every node sends a message to every other node
    - If no msg from a node within bound, node has crashed
- Not useful for Internet, how to proceed?

Modeling a Distributed System

- Note that Internet is mostly synchronous
  - Bounds respected most of the time
  - Occasionally violate bounds (congestion/failures)
  - How do we model this?
- Partially synchronous system
  - Initially system is asynchronous
  - “Eventually” the system becomes synchronous
    - We don’t know when, but we know it will happen

Possibility of Consensus

- Consensus solvable in partially synchronous system
  - If less than N/2 crashes (i.e., majority must be correct)

- Useful for Internet

Failure detectors

- Let each node use a failure detector
  - Detects crashes
  - Implemented by heartbeats and waiting
  - Might be initially wrong, but eventually correct

- Consensus and Atomic Broadcast solvable with failure detectors
  - How? Attend rest of course!

Failure Aspects

What types of failures are possible?

Nodes always crash?

- Study other types of failures
  - Not just crash stops

- Byzantine faults
- Self-stabilizing algorithms

Byzantine Faults

- Some nodes might behave arbitrarily
  - Sending wrong information
  - Omit messages…
- Byzantine algorithms tolerate such faults
  - Byzantine algorithms only tolerate up to 1/3 faulty nodes
  - Non-Byzantine algorithms can often tolerate 1/2

Self-stabilizing Algorithms

- Robust algorithms that run forever
  - System might temporarily be incorrect
  - But eventually always becomes correct
- System can either be in a legitimate state or an illegitimate state
- Self-stabilizing algorithm iff
  - Convergence
    - Given any illegitimate state, system eventually goes to a legitimate state
  - Closure
    - If system in a legitimate state, it remains in a legitimate state

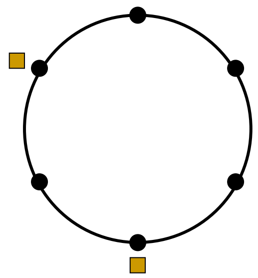
Self-stabilizing Algorithms

- Advantages
  - Robust to transient failures
  - Don’t need initialization
  - Can be easily composed

27

Self-stabilizing Example

- Token ring algorithm
  - Wish to have one token at all times circulating among nodes
- Self-stabilization
  - Error leads to 0,2,3,... tokens
  - Ensure always 1 token eventually



28

Future of Distributed Systems

- Large-scale systems
  - Previous theory assumes few nodes knowing each other
- Dynamic systems
  - Nodes joining, leaving, and failing
  - Dynamicity: rapidly changing number of active nodes
- Examples
  - Skype, BitTorrent, ppLive…
  - Peer-to-peer algorithms, gossip algorithms
  - Cloud computing
  - Edge computing (Internet of Things)

Summary

- Distributed systems are everywhere
  - Set of nodes cooperating over a network
- Many problems reduce to a set of core problems
  - Consensus, Broadcast, Leader election
- The Internet is the principal distributed system
  - It is a partially synchronous system
- Different failure scenarios are important
  - Crash stop, Byzantine, self-stabilizing algorithms
- Interesting new research directions
  - Large scale dynamic distributed systems

Let’s start