Distributed Systems

Arvind Krishnamurthy
University of Washington

Today's Lecture

- Introduction
- Role of knowledge in distributed systems
- Course details
- Start discussion on key building blocks

Course Notes

- Derived from research papers and course notes from other distributed systems classes, particularly:
  - Tom Anderson, Doug Woos, Dan Ports (UW)
  - Lorenzo Alvisi (Cornell)
  - Robert Morris (MIT)
  - James Aspnes (Yale)

Distributed Systems are everywhere!

- Some of the most powerful services are powered using distributed systems
  - systems that span the world,
  - serve billions of users,
  - and are always up!
- … but also pose some of the hardest CS problems

What is a distributed system?

- multiple interconnected computers that cooperate to provide some service

- what are some examples of distributed systems?

Why distributed systems?

- Higher capacity and performance
- Geographical distribution
- Build reliable, always-on systems

- What are the challenges in building distributed systems?

(Partial) List of Challenges

- Fault tolerance
  - different failure models, different types of failures
- Consistency/correctness of distributed state
- Performance
- Scaling
- Security
- System design, architecture, testing

- We want to build distributed systems to be more scalable, and more reliable

- But it’s easy to make a distributed system that’s less performant and less reliable than a centralized one!

Challenge: failure

- Want to keep the system doing useful work in the presence of partial failures

Consider a datacenter

- E.g., Facebook, Prineville OR
- 10x size of CSE building, $1B cost, 30 MW power
  - 200K+ servers
  - 500K+ disks
  - 10K network switches
  - 300K+ network cables
- What is the likelihood that all of them are functioning correctly at any given moment?

Typical first year for a cluster

- ~0.5 overheating (power down most machines in <5 mins, ~1-2 days to recover)
- ~1 PDU failure (~500-1000 machines suddenly disappear, ~6 hours to come back)
- ~1 rack-move (plenty of warning, ~500-1000 machines powered down, ~6 hours)
- ~1 network rewiring (rolling ~5% of machines down over 2-day span)
- ~20 rack failures (40-80 machines instantly disappear, 1-6 hours to get back)
- ~5 racks go wonky (40-80 machines see 50% packetloss)
- ~8 network maintenances (4 might cause ~30-minute random connectivity losses)
- ~12 router reloads (takes out DNS and external vips for a couple minutes)
- ~3 router failures (have to immediately pull traffic for an hour)
- ~dozens of minor 30-second blips for dns
- ~1000 individual machine failures
- ~thousands of hard drive failures
- slow disks, bad memory, misconfigured machines, flaky machines, etc

- At any given point in time, there are many failed components!

- Leslie Lamport (c. 1990): "A distributed system is one where the failure of a computer you didn't know existed renders your own computer useless"

Challenge: Managing State

- Question: what are the issues in managing state?

State Management

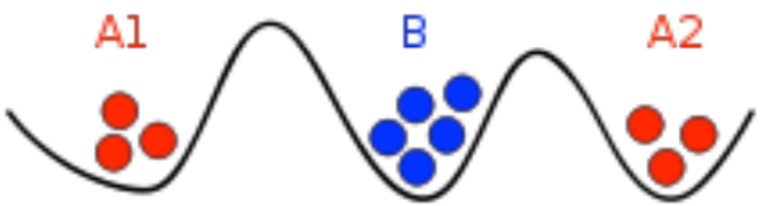
- Keep data available despite failures:
  - make multiple copies in different places
- Make popular data fast for everyone:
  - make multiple copies in different places
- Store a huge amount of data:
  - split it into multiple partitions on different machines
- How do we make sure that all these copies of data are consistent with each other?
- How do we "transactionally" update multiple data items?
- How do we do all of this efficiently?

Lot of subtleties

- Simple idea: make two copies of data so you can tolerate one failure
- Lots of subtleties in how to do this correctly!
  - What if one replica fails?
  - What if one replica just thinks the other has failed?
  - What if each replica thinks the other has failed?

The Two Generals Problem

- Two armies are encamped on two hills surrounding a city in a valley



- The generals must agree on the same time to attack the city.
- Their only way to communicate is by sending a messenger through the valley, but that messenger could be captured (and the message lost)

The Two-Generals Problem

- No solution is possible!
- If a solution were possible:
  - it must have involved sending some messages
  - but the last message could have been lost, so we must not have really needed it
  - so we can remove that message entirely
- We can apply this logic to any protocol, and remove all the messages — contradiction

- What does this have to do with distributed systems?

- What does this have to do with distributed systems?
  - “Common knowledge” cannot be achieved by communicating through unreliable channels

Another Example: Muddy Foreheads

- “n” children go playing
- Children are truthful, perceptive, intelligent
- Mom says: “don’t get muddy”
- Some number (say k) get mud on their foreheads
  - but a child doesn’t know if there is mud on his/ her forehead
  - each child can look at others’ foreheads

Muddy Foreheads (contd.)

- Dad comes, looks around, and says:
  - “Some of you got a muddy forehead”
- Dad then asks repeatedly:
  - “Do you know whether you have mud on your own forehead?”
- What happens?

Muddy Foreheads (contd.)

- Claim:
  - The first k-1 times the dad asks, all children will reply “No”
  - The k-th time all dirty children will reply “Yes”
- Reasoning by considering cases and using induction:
  - k=1: the child with a muddy forehead will say yes
  - k=2: let X and Y have muddy foreheads
    - Each sees exactly one other person with muddy forehead
    - In round 1, X noticed Y didn’t say “Yes”
      - Possibly only because Y must have seen a child with a muddy forehead ==> X must have mud

Distributed Systems are Hard

- Distributed systems are hard because many things we want to do are provably impossible
  - consensus: get a group of nodes to agree on a value (say, which request to execute next)
  - be certain about which machines are alive and which ones are just slow
  - build a storage system that is always consistent and always available (the "CAP theorem")

- We need to make the right assumptions and also resort to "best effort" guarantees

This Course

- Introduction to the major challenges in building distributed systems
- Will cover key ideas, algorithms, and abstractions in building distributed system
- Will also cover some well-known systems that embody such as ideas

Topics

- Implementing distributed systems: system and protocol design
- Understanding the global state of a distributed system
- Building reliable systems from unreliable components
- Building scalable systems
- Managing concurrent accesses to data with transactions
- Abstractions for big data analytics
- Building secure systems from untrusted components
- Latest research in distributed systems

Course Components

- Assignments (40%)
  - Deep dive into some of the papers; done individually

- Class participation (10%)

- Project (50%)
  - course-long project; done as a group