CSE 452/M552 Distributed Systems

Doug Woos (and Tom Anderson)

About me

I’m Doug, one of Tom’s students

Mostly using Tom’s materials

Work on distributed systems verification

He/him or they/them

Logistics

Course website

- Important: Office Hours (none today)

Piazza

Code word is “leopard”: http://tinyurl.com/m9eg43b

Names

Place in Curriculum

CSE 333: Systems Programming

- Projects in C++
- How to use the OS interface

CSE 451: Operating Systems

- How to make a single computer work reliably
- How an operating system works internally

CSE 452: Distributed Systems

- How to make a set of computers work reliably and efficiently, despite failures of some nodes

Related courses

CSE 461: Computer Communication Networks

- How to connect computers together
- Networks are a type of distributed system

CSE 444: Database System Internals

- How to store and query data, reliably and efficiently
- Mostly single-node databases

CSE 550: Systems For All

- One quarter firehose version of 451/452/461/444
- Mostly PhD students

Thought experiment

Imagine a group of people, two of whom have green dots on foreheads

Without using a mirror or communicating, can anyone tell if they have a green dot?

What if I say: someone has a green dot

What you know

vs.

What you know others know

Distributed systems

Multiple connected nodes that cooperate in performing a task or providing a service

- Examples?

Why distributed systems?

Communicate across geographic separation

- Locality is super important

Ensure availability

- Whole system shouldn’t fail when one node fails

Aggregate systems for higher capacity

- Nodes fail all the time
- Whole system shouldn’t fail when one node does

Why are distributed systems cool*?

Extremely important in practice

- Crucial to bottom-line of huge companies
- Crucial to the daily lives of many users

Rich, well-studied theory

- Long tradition of formal reasoning
- Neat mathematical results

* For some values of “cool”

Why are distributed systems hard?

Asynchrony

- Different nodes run at different speeds
- Messages can be unpredictably, arbitrarily delayed

Failures (partial and ambiguous)

- Parts of the system can crash
- Can’t tell crash from slowness

Concurrency and consistency

- Replicated state, cached on multiple nodes
- How to keep many copies of data consistent?

Why are distributed systems hard?

Performance

- Have to efficiently coordinate many machines
- Performance is variable and unpredictable
- Tail latency: only as fast as slowest machine

Testing and verification

- Almost impossible to test all failure cases
- Proofs (emerging field) are really hard

Security

- Need to assume adversarial nodes

Sense of scale

Wide-area matters (across continents)

Local-area also matters (within a data center)

Correctness is the same

- Have to account for failures either way

Performance is different

Prineville Data Center

Huge FB data center in Oregon

Contents:

- 200K+ servers
- 500K+ disks
- 10K network switches
- 300K+ network cables

How likely is it that everything is functioning at once?

MTTF/MTTR

Mean Time to (Failure/Repair)

Disk failures per year: 20% or so

- So like 2/hour
- Takes about an hour to restore

If each server reboots once/month

- 30s reboot -> 5 mins/year offline
- 500K mins/year -> ~2 rebooting

… and not all of FB's servers are in Oregon

Local vs. Remote Operations

How long to do a procedure call locally?

- 10 instructions

How about to another node in the same DC?

How about to a node in some other DC?

- Speed of light = 1ft/ns

Properties we want

Fault-tolerant (Lab 2)

- Doesn’t go wrong when components fail

Highly available (Lab 3)

- Doesn’t go down when components fail

Scalable (Lab 4)

- Can grow to more (nodes, memory, etc.)

Other properties we want

Consistent (All labs)

- Appears as one node

Predictable performance

- Consistently stays within SLAs

Secure (Week 9)

- Can grow to more (nodes, memory, etc.)

Guaranteed Correct (Week 10)

- Formally proven to follow spec

Labs

Implement a sharded, replicated key-value store

- Lab 1: MapReduce
- Lab 2: Primary/backup
- Lab 3: Paxos
- Lab 4: Sharding

In Golang

- New-ish language, developed at Google
- “Easy” to learn, “easy” to write concurrent code

Labs

The labs are hard

- Based on MIT's grad-level course
- Nontrivial for me, TAs, Tom

General tips

- Start early
- Think before you code
- Ask for help! (classmates, us, Piazza)

Good candidates for code portfolio

Readings and blogs

No good textbook in this area

~14 papers (first one this Wednesday)

- "How to read a paper," Keshav 2007

Blog

- For 5 papers, write a short, *unique* thought (2-3 sentences) on the discussion board

Problem sets

5 problem sets

- First one due in 3 weeks, out next Friday
- To be done individually
- Short answer questions
- Should be quick (< 1 hour)

Another thought experiment

Two generals have to coordinate a time to attack

Messengers can be killed, arbitrarily detained

No other communication

If either attacks alone, army will be destroyed

Design a protocol to coordinate an attack