CSE 452/M552
Distributed Systems

Tom Anderson

Course Overload Information

tinyurl.com/hjl3tpj

How This Course Fits in the UW CSE Curriculum

- CSE 333: Systems Programming
  - Project experience in C/C++
  - How to use the operating system 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

Closely 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 (large) data, reliably and efficiently
  - Primary focus is single node databases
- CSE 550: Systems For All
  - One quarter firehose version of 451/452/461/444
  - Primarily for PhD students

A Thought Experiment

Suppose there is a group of people, two of whom have green dots on their foreheads.

- Without using a mirror or directly asking, can anyone tell if they themselves have a green dot?
- What if I say to everyone: someone has a green dot
  - Something everyone already knows!

There's a difference between what you know and what you know others know.

What is a Distributed System?

- Multiple interconnected computers that cooperate to provide some service
- Examples?

Why Distributed Systems?

- Conquer geographic separation
  - Facebook and Google customers span the planet
- Build more reliable systems
  - Out of unreliable components
- Aggregate systems for higher capacity
  - Aggregate cycles, memory, disks, network bandwidth
- Customize computers for specific tasks
  - Ex: email server, backup server

The Distributed System Challenge

Do useful work in the presence of partial failures with reasonable performance.

A Thought Experiment

- Consider a Facebook data center
  - e.g., Pineville Oregon
- About 10x the size of the Allen Center
- Approx $1B to construct (buildings and contents)
- Approx 30MW power draw
- Sidebar: How do you do cooling?

Data Center Layout

Pineville Data Center Contents (approx)

- 200K+ servers
- 500K+ disks
- 10K network switches
- 300K+ network cables
- User data is spread across multiple data centers

- What is the likelihood that all of the components are correctly functioning at any instant in time?

MTTF/MTTR

Mean Time to Failure/Mean Time to Repair

Disk failures (not reboots) per year ~ 2-4%

- At data center scale, that's about 2/hour.
- It takes about an hour to restore a 1TB disk.

Suppose each server reboots once/month

- 30 seconds to reboot => 5 mins/year offline
- 500K minutes in a year => 2 rebooting (on average)

We've Made Some Progress

Leslie Lamport, circa 1990:

"A distributed system is one where you can't get your work done because some machine you've never heard of is broken."

We've Made Some Progress

Today a distributed system is one where you can get your work done (almost always):

- wherever you are
- whenever you want
- even if parts of the system aren't working
- no matter how many other people are using it
- as if it was a single dedicated system just for you
- that (almost) never fails

Yet Another Thought Experiment: Local vs. Remote Operations

- How long does it take to do a simple procedure call on a modern server?
- How long does it take to do the same operation on a different server in the same data center?
- On a server in a remote data center?
  - Speed of light is ~ 1ns/foot

Why Is DS So Hard?

- System design
  - Partitioning of responsibilities: what should client do, what should server do? Which servers should do what?
- Failures are endemic, partial and ambiguous
  - If the server doesn't reply, how do you tell if it is (a) the network, (b) the server, or c) neither: they are both just being slow?
- Concurrency and consistency
  - Distributed state, replicated state, caching
  - How do we keep this state consistent?

Why Is DS So Hard?

- Performance
  - Generating a single FB page involves calls to hundreds of different machines
  - Performance can be variable and unpredictable
  - Tail latency: only as fast as the slowest machine
- Implementation and testing
  - Nearly impossible to test/reproduce all failure cases
- Security
  - Adversary can silently compromise machines and manipulate messages

Properties We Want (Google Paper)

- Fault-Tolerant: It can recover from component failures without performing incorrect actions. (Lab 2)
- Highly Available: It can restore operations, permitting it to resume providing services even when some components have failed. (Lab 3)
- Scalable: It can operate correctly even as some aspect of the system is scaled to a larger size. (Lab 4)
- Recoverable: Failed components can restart themselves and rejoin the system, after the cause of failure has been repaired. (Lab 5)

Other Properties We Want (Google Paper)

- Consistent: The system can coordinate actions by multiple components often in the presence of concurrency and failure. This underlies the ability of a distributed system to act like a non-distributed system. (Labs 2-5)
- Predictable Performance: The ability to provide desired responsiveness in a timely manner. (Week 9)
- Secure: The system authenticates access to data and services (Week 10)

Project

- Build an distributed key-value store
  - To clients, a distributed hash table
  - Stores arbitrary content per key (NoSQL)
- With:
  - Scalable to arbitrary size
  - Fault tolerant (continues to operate despite node and network failures)
  - Consistent (correct regardless of failures)
  - Timely progress (under certain conditions)
  - Failed nodes can recover

Project Management

- Lab 1 (mapreduce) due **next** Wednesday
  - Section Thursday: introduction to Go
- Labs 2-5 (key-value store)
- Think and plan very carefully before writing any code.
- OK to ask for help
  - Irene and Ray have done the project (I've done parts of it)
  - Also ok to ask other students for advice

Some Career Advice

- Create a portfolio
  - Course projects
    - Support building a tool to make it easy to share gitlab projects with employers
    - Edits to public source code projects
- To employers, code quality >> grades
  - Design, structure, tests, comments, …
  - Create a portfolio

Project Rules

- OK
  - Consult with us or other students in the class
- Not OK
  - Look at solutions posted by people not in the class
  - Cut and paste code

Readings and Blogs

- There exists no (even partially) adequate distributed systems textbook
- Instead, 14 research papers
  - How do you read a research paper?
- Blog
  - For seven of the papers, write a short (2-3 sentence) **unique** thought about the paper to the discussion board

Problem Set

- One problem set, available now
  - Equivalent to a take home, open book final
  - Done **individually**

The Science of Computers in the Classroom

- Don't

MapReduce

A programming model to help unsophisticated programmers use a data center without thinking about failures and distribution.

- Popular distributed programming framework
- Many descendants frameworks

Lab 1:

- Help you get up to speed on Go and distributed programming
- Exposure to some fault tolerance
- Motivation for better fault tolerance in later labs

MapReduce Computational Model (Document Processing)

For each key (k1, v1), compute

map (k1,v1) → list(k2,v2)

For each key (k2, list(v2), compute

reduce (k2,list(v2)) → list(v2)

User writes a map function and reduce function

Framework takes care of parallelism, distribution, and fault tolerance

MapReduce Steps

1. Split document into set of <k1, v1> pairs
2. Run Map(k1, v1) on each element of each split -> set of <k2, v2> pairs
3. Coalesce results from each split into a list for each key
4. Run Reduce(k2, list(v2)) -> list(v2)
5. Merge result