452 Lecture 1: Introduction

Course title: Distributed Systems
(both 452 and M552)
My name: Tom Anderson
TA's: Umar and Will

Admin: guest lecture Wed, NO SECTION Thursday
Sorry about the early morning Thursday section: I didn't notice it until too late to change.

I'm excited about this class, as distributed systems is probably the most complex and difficult topic in all of CS.

Central part of distributed systems is failures. Google has more than a 1M machines in their data centers. What does that mean? System has to keep working even when some of the machines have crashed — with 1M machines, some machine is actively crashing all the time.

Failure is the normal case.

<Airline anecdote: 2 engines -> better, but more downtime. With web, can't afford any downtime!>

Failures means you can't test your code — does your code work in all failure cases, or just some? How can you test that? Try a failure here, try one there -- have to design from first principles, and use testing only as a backstop.

Here's an example of why your intuition is wrong.

Two generals problem. Coordinate attack on a valley, using messages. But the messages are unreliable — can't be sure they get through.

Can you come up with a protocol that works?

Try it: what should we send?

Key point: problem occurs even though no message is in fact lost.

In fact: impossible to coordinate with any finite protocol. If you could, then you didn't need the last message. So there's a shorter finite protocol. Take the shortest one. Then the last message wasn't needed! Contradiction.

Can generalize: impossible to coordinate two nodes to do some joint action, if nodes can fail and messages can take varying amounts of time to be delivered (e.g, because they can get dropped.) Can't tell if someone is failed, or just slow because the network is slow. So any finite coordination protocol can't achieve both timeliness, and correctness.

Mode of thinking is similar to that in 451 and 461 and 444, but combines the complexity of all three.

How many have taken 451? 461? 444? None of the above?

Central part of the class is a software project: so important you already know how to debug code.

However, you don't need to have had 451 — many distributed systems use threads (covered in 451), but we've designed the project so you don't need that. Instead, project is event-driven: each node waits for an event, processes it, and goes back to waiting for the next event.

Complexity is in understanding how to manage a distributed set of these event handlers, one per node, especially when some can fail.

A key part of the class: methodology for writing

working distributed applications.

Project: to build a peer to peer twitter app.

What do I mean by that? You and some friends can run the app, and it will do the things twitter does: post messages, see each other's postings, manage lists of people following a particular account, all in a fault tolerant fashion — that works even if some of the machines crash.

By peer to peer → without a central server.

Three steps, equally spaced.

1) client server (this week: how to do a simple procedure call from one node to another). "simple" -> well, what do we mean by RPC?

2) distributed transactions

3) high availability

M552 students: a fourth part of the project, open ended

Form groups of 2-3 today; get started tomorrow.

Class cancelled last week of quarter, to give you more time for that assignment. Stay up to date, as that assignment is very very hard.

Course mechanics:

No textbook. Instead research paper readings. How many of you have read a research paper?

Key is not to let yourself get bogged down if you don't understand something.

First half of the class: main focus of the papers will be the algorithm.

Second half of the class: main focus of the papers will be on the system design

Both 452 and M552 students:

Blog entry for 5 (10) papers, due at 1pm on the day of class. We'll pose a question for the paper, as a starting point for the discussion.

Two problem sets, and a final.

Collaboration policy: problem sets and exams done individually. Project done in groups. Dept policy applies.

What is a distributed system?
  multiple computers
  interconnected
  cooperate to provide some service
  [Example: Akamai, Google, catalyst, ...]
  [Counter example: shared UNIX time-sharing system]

Why care about distributed computing?

1. conquer geographic separation

2. build reliable systems out of unreliable components

3. aggregate many computers for high capacity
    aggregate cycles+memory: (ThreadMarks, Dryad)
    aggregate bw (Coral, Shark)
    aggregate disks (Frangipani)

4. customize computers for specific tasks
    email server
    backup server

Challenges

1. system design
    what does the client do, what does the server do? which servers?
    what are the right protocols?
    What are the right abstractions?

2. consistency
    shared data with multiple readers and writers

3. failures
    communication and hardware
    how do you tell the difference?
    which node is the last one to fail?

4. security
    adversary may compromise machines or manipulate messages

5. performance
    how do we make a system fast when it needs to coordinate across multiple machines
    network is often scarce resource
    avoid disk writes

6. variable network properties
    very slow networks, aka sensors net
    very fast networks, inside a data center
    very high latency networks, in the wide-area

7. implementation/testing
    concurrency with servers and clients
    ensuring code works despite failures

Easy to make distributed system less scalable, less reliable, etc. than a centralized system.

Lamport's definition: a distributed system is one where a computer you don't know about renders your own useless.

Example for the next two lectures, and beyond!, and the first project:

How do we build a network file system?

Any user can read/store data, process it, send back modifications

Server or clients can fail

Simple System Design
  One server w/ disk to store directories and files
  [picture: file server, "clients", read file, write file, create,
  etc.]

Question: who should do what?

Server stores data
Clients read/write data

Should applications know whether the data is stored locally or remotely?

On the web: everything is remote!
For file systems though, does your text editor know if it's a local file or a remote file?

Can we make it transparent? E.g., app does a file read, file system determines if its local or remote, and returns the data.

What are the consequences if it is transparent?

Performance: local system call vs. a remote message

Reliability: what happens if the other side fails?

Next time: we'll talk about RPC, designed to make this work well.

Topic: consistency & protocol design
  What if user operations require multiple file system operations?
    If I move a file from one directory to another, does another user see intermediate states?
    What if two users move a file to the same destination directory?

  To offload network and server probably cache files at client
    When does a write on a client become visible to other clients?
    Do we want it to behave like a single-machine file system?

Topic: system design
  What happens if your file server must serve a large community?
  What if more clients than one server can handle?
  What if more users than one server can store files and directories?
  How to use more servers to handle more clients?
  Idea: partition users across servers
    How to do load balance of users? Statically? Partition name space?
    What if some user need suddenly a lot more space?
    What if some files are much more popular than others?

Topic: fault tolerance
  Can I get to my files when some servers are down or network fails?
  Yes: replicate the data.

Problem: replica consistency. delete file, re-appears.

Problem: physical independence vs communication latency

Problem: partition vs availability. airline reservations.

Tempting problem: can 2 servers yield 2x availability AND 2x performance?

Topic: security

Internet provides global exposure to random attacks from millions

of bored studentsand serious hackers, e.g. intrusions for spam bot nets

How does the server know that a request is from me?

How is the file server protected?

How much do i have to trust the system administors?

Etc.

Topic: system design

What if we want to provide an Internet file system?

aggregate all computers in a gigantic file system

How do we need to do this?

more failures, longer delays, multiple administrative domains

We want to understand the individual techniques, and how to assemble them.