Fundamentals Large-Scale Distributed System Design
(a.k.a. Distributed Systems)

Lec 1: Course Introduction
Interested in...

1. Cloud computing?
2. Big data?
3. Scalable web services?
4. And the large-scale infrastructure systems that are making these possible?

If so, you're in the right room.
This class

1. **Core concepts** of large-scale distributed systems
   - Abstractions, algorithms, techniques

2. The **inner-workings** of several distributed systems serving as infrastructure for some **very big companies**
   - E.g.: Google's protobuf/Bigtable/Spanner/MapReduce, Sun's NFS, Yahoo's Hadoop, Amazon's Dynamo, etc.
3. **Gothas** of using some popular distributed systems, which stem from their inner workings and reflect the **challenges** of building large-scale distributed systems - MongoDB, Redis, Hadoop, etc.

4. **How to build** a real distributed system yourself! - Via a series of coding assignments, you will build your very own distributed file system
Related CU classes

• Multiple cloud computing classes are offered @CU
  – Those classes teach you how to use various popular distributed systems (particularly Hadoop)
  – This class will teach you the how those and other systems are built, so you can build and use them better in the future

• Similar to the OS class, but for the distributed environment
  – And in the “cloud” era, everything is distributed!
    • If you want to do “big data,” you need DS
    • If you want to do mobile apps, you need DS

• This class has a distinctive focus on state-of-the-art systems being used today by big companies
  – “Concept” classes followed by “real-world practice” classes
Know your staff

• Instructor: Prof. Roxana Geambasu (me)
  – Office hour: Mon/Wed after class (CSB 461)
  – roxana@cs.columbia.edu

• Teaching assistant and office hour info:
  – Mathias Lecuyer (mathias@cs.columbia.edu)
  – Ph.D. student of mine
  – Office hours on the website
Important addresses

• Website: https://roxanageambasu.github.io/ds-class/
  – Still in progress and may change at any time
  – Check regularly for schedules and deadlines!

• Discussions: Piazza
  – Please sign up ASAP

• Staff contact:
  – distributed-systems-class@lists.cs.columbia.edu
  – If you have a homework-related question, post it on Piazza, as it might help others
  – If you need to communicate something private, use the above mailing list or our email addresses
Prerequisites: C/C++!

• Pre-requisites:
  – You must have a solid working experience in C
  – Some knowledge of C++
  – Columbia courses (or equivalents):
    • COMS W3137 Data Structures and Algorithms
    • COMS W3157 Advanced Programming
    • COMS W3827 Fundamentals of Computer Systems
    • Optional, but very useful: COMS 4118 Operating Systems

• If you lack these prerequisites, do not take the class
  – Heavy coding accounts for a large portion of the grade!
  – Use first assignment to figure out if you have sufficient experience
Course readings

• Official textbook:
  – Distributed Systems (Tanenbaum and Steen)
  – Outdated (compared to the modern focus of this class), but great for understanding core issues of distributed systems

• Very useful references:
  – The C++ Programming Language (Bjarne Stroustrup)
  – Principles of Computer System Design (Saltzer and Kaashoek)
  – Advanced Programming in the UNIX environment (Stevens)
  – UNIX Network Programming (Stevens)
  – Google's C++ Coding Style Guide
Course structure

• Lectures
  – Read assigned chapters from Tanenbaum before class
  – IMPORTANT: Participate in class (15% of grade, plus my invaluable appreciation :) )
  – Ask/answer questions, contradict me and each other, propose new discussion topics, comment on the weather (well maybe not the last one, but BE ACTIVE!)

• Assignments
  – Multiple graded assignments (HW2 and after) plus one pass/fail assignment (HW1)
  – Each assignment has two parts: writing and coding
  – Coding component is a.k.a. a “lab”
Assignments

• **Homework 1** is sample to determine whether to take this course
  – It will be graded **pass/fail** and will not count to final grade
    • Pass means we welcome you to the course
    • Fail means drop this course and take a prerequisite
  – *Already available on website and due Sept 12, 2014*

• **The rest of the Homeworks** build a **networked file system** with detailed guidance
  – Number and contents of the homeworks is still TBD, but some rough (older) versions are already available online
Assignments (cont)

• All homeworks have written and coding parts
  – Written parts: loosely follow the coding portions and course concepts, and help you understand those better
  – Coding parts: heavy coding, need lots of time!
  – Start with written part then do coding part

• You will submit all homeworks via git
  – The repo will be set up by Monday Sept 8
Grading

• Grading formula
  – 65%: six graded homeworks
  – 20%: final and midterm exams
  – 15%: class participation

• Grading policies
  – No deadline extensions: late submissions get a 0!
  – Can discuss, but *not* look at others' code or answers
  – For coding: be as clean as you can possibly be
    • Test thoroughly, comment your code, and adhere to a strict coding style (we recommend one on website)
  – More policies on website, read them in detail!
Acknowledgements

• Course builds on several other distributed systems courses:
  – MIT‘s 6.824 (Robert Morris and Frans Kaashoek)
  – NYU's G22.3033 (Jinyang Li)
  – CMU's 15-440 (David Andersen)

• Lab assignments are taken from MIT, NYU courses
• Lectures are adapted from all three courses
• Website structure is adapted from NYU course
• This lecture is adapted from MIT and NYU lectures 1
Questions?
What are distributed systems?

- Examples?
- Counter-examples?

Multiple hosts

A network cloud

Hosts cooperate to provide a unified service
Example: Gmail

• What do you think happens when you click on “Inbox”?

(screenshot found through google images)
Example: Gmail

• What do you think happens when you click on “Inbox”?

• Lots of components accessed: load balancer, auth service, mem cache, gmail front end, storage service, ads service, batch computations, etc.
Distributed systems vs. networks

- Distributed systems **raise the level of abstraction**
- Hide many complexities and make it **easier to build applications**

### Applications
(Gmail, Facebook, mobile apps...)

<table>
<thead>
<tr>
<th>files, dirs</th>
<th>put, get</th>
<th>acquire, release</th>
<th>tasks</th>
<th>enq, deq</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributed file system (GFS, HDFS, NFS)</td>
<td>Key/value store (S3, Dynamo, Cassandra)</td>
<td>Distributed locking system (Chubby, Zookeeper)</td>
<td>Distributed computing (MapReduce, Hadoop)</td>
<td>Message queues (Amazon SQS)</td>
</tr>
</tbody>
</table>

TCP, UDP, HTTP, … (low-level comm. interfaces)
Q: Why distributed systems?  
A1: For location transparency

• Examples:
  – Your browser doesn’t need to know which Google servers are serving Gmail right now
  – Your Amazon EC2-based mobile app doesn’t need to know which servers in S3 are storing its data

• Why is location transparency important?
Q: Why distributed systems?
A2: For scalable capacity

• Aggregate resources of many computers
  – CPU: MapReduce, Dryad, Hadoop
  – Disk: NFS, the Google file system, Hadoop HDFS
  – Memory: memcached
  – Bandwidth: Akamai CDN

• What scales are we talking about?
  – Typical datacenters have 100-200K machines!
  – Each service runs on more like 20K machines, though
Q: Why distributed systems?
A3: For availability

• Build a **reliable system** out of **unreliable parts**
  – Hardware can fail: power outage, disk failures, memory corruption, network switch failures …
  – Software can fail: bugs, mis-configuration, upgrade …
  – To achieve 0.9999 availability, **replicate data/computation** on many hosts with automatic failover
Why distributed systems?
A4: For modular functionality

• Your application is split into many simpler parts, which may already exist or are easier to implement
  – Authentication service
  – Indexing service
  – Locking service

• This is called the service-oriented architecture (SOA) and much of the Web is built this way
  – E.g.: one request on Amazon’s website touches tens of services, each with thousands of machines (e.g., pricing service, product rating service, inventory service, shopping cart service, user preferences service, etc…)
Challenges

• Achieving location transparency, scalability, availability, and modularity in distributed systems is really hard!

• System design challenges
  – What is the right interface or abstraction?

• Achieving scalability is challenging
  – How to partition functions for scalability?

• Consistency challenges
  – How do machines coordinate to achieve the task?
Challenges (Continued)

• **Security challenges**
  – How to authenticate clients or servers?
  – How to defend against misbehaving servers?

• **Fault tolerance challenges**
  – How to keep system available despite machine or network failures?

• **Implementation challenges**
  – How to maximize concurrency?
  – What’s the bottleneck?
  – How to reduce load on the bottleneck resource?
A word of warning

“A distributed system is a system in which I can’t do my work because some computer that I’ve never even heard of has failed.”

-- Leslie Lamport
Topics in this course
Case study: 
Distributed file system

- Single shared file system, so users can cooperate
  - Lots of client computers
  - One or more servers
- Examples: NFS (single server), GFS (multi-server)

Client 1

Client 2

Client 3

Server(s)

$ ls /dfs
f1 f2
$ cat f2
test

$ echo "test" > f2
$ ls /dfs
f1 f2

$ ls /dfs
f1 f2
Topic: Interface design

• What is the right interface?
  – File interface: relay FS requests to server (NFS, GFS)
  – Block interface: expose disk blocks from server(s) and have FS logic in clients (Storage Area Networks)
  – Key/value: expose put/get interface (Amazon S3)
  – Database: expose a DB interface from the server (Google’s Bigtable, distributed RDBMS)

• There is no right answer
  – There are always tradeoffs: performance, ease of programming, scalability
  – Choice depends on the application
  – This will be a theme in this course
Topic: Scalability

• How to **scale** the distributed file system?
  – Lots of users with lots of data (e.g., all CU students/faculty)

• **Ideally**: having N servers supports Nx as many users as having one server

• Idea: **Partition data** across servers
  – By user
  – By file name

• But you **rarely get the ideal**… Why?
  – Load imbalance: one very active user, one very popular file
  – One server gets 99.9% of requests; N-1 servers mostly idle
Topic: Consistency

• When C1 moves file f1 from /d1 to /d2, do other clients see intermediate results?
  – f1 in both directories, f1 in neither

• What if both C1 and C2 want to move f1 to different places?
Topic: Fault Tolerance

• Can I use my files if server / network fails?
• Idea: replicate data at multiple servers

• But how to maintain consistency despite faults?
  – S1 misses updates while it reboots, so S2 must update it
  – If network’s down, S1 can’t get updates – should it resume execution?

• In general, consistency is tough and expensive
  – Hence, many applications opt for “weak consistency”
• Adversary can manipulate or sniff messages to corrupt or access files
  – How to authenticate?

• Adversary may compromise machines
  – Can the FS remain correct despite a few compromised nodes?
Topic: Implementation

• How do clients/servers communicate?
  – Direct network communication is painful
  – Want to hide network stuff from application logic (e.g., RPC, RMI)

• The file server should serve multiple clients concurrently
  – Keep (multiple) CPU(s) and network busy while waiting for disk
  – But concurrency is hard to get right (e.g., race conditions, live locks, deadlocks)
Overview of Homework 1:
C++ Warm-up

(TA will describe in more detail)
Homework 1

• A warm up exercise
• Determine whether you are comfortable with this class
• Will be graded pass/fail. Will NOT be counted towards final grade
• Read the submission instructions very carefully
• Also read our policies
  – https://roxanageambasu.github.io/ds-class/policies.html
• Always post your question on Piazza, as it may benefit others
Next Time

• TODO for you: Homework 1

• We'll do a couple of case studies to understand the role of distributed systems in enabling new and exciting trends today:
  1. Cloud computing
  2. Big data
  3. Scalable web services