Distributed Systems Fundamentals [Fall 2013]

Lec 1: Course Introduction
Know your staff

• Instructor: Prof. Roxana Geambasu (me)
  – Office hour: Thursday 1-2pm (CSB 461)
  – roxana@cs.columbia.edu

• Teaching assistants:
  – Peter Du (peter@cs.columbia.edu)
  – Yu Qiao (yq2145@columbia.edu)
  – Their office hours on the website
Important addresses

• Website: https://www.cs.columbia.edu/~du/ds/
  – 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 more privately, use the above mailing list or our email addresses
This class will teach you ...

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

2. Popular distributed **systems and tools used by big companies today**
   - E.g.: Google's protobuf/Bigtable/Spanner/MapReduce, Sun's NFS, Yahoo's Hadoop, Amazon's Dynamo, etc.

3. **How to build a real distributed system yourself**
   - Via a series of coding assignments, you will build your very own distributed file system
Relationship with other 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 data mining you need to build a DS
    - If you want to do mobile apps, you need to build a DS

- Class has distinctive focus on state-of-the-art systems being used today by big companies
  - Concepts classes followed by real-world practice classes
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)
  – Fairly outdated (compared to the modern focus of this class), but great for understanding core issues

• **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
  – Participate in class (ask and answer questions)

• Assignments
  – Six graded assignments (HW2-7) 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 a “sample” homework to help you figure out if you should take 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
  - Assigned today and due Sept 11, 2013

- **Homeworks 2-7** build a networked file system with detailed guidance

- 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
Grading

• Grading formula
  – 70%: six graded homeworks
  – 15%: final exam
  – 15%: class participation

• Grading policies
  – No deadline extensions: late submissions will 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 to build profile, 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><strong>Distributed file system</strong> (GFS, HDFS, NFS)</td>
<td><strong>Key/value store</strong> (S3, Dynamo, Cassandra)</td>
<td><strong>Distributed locking system</strong> (Chubby, Zookeeper)</td>
<td><strong>Distributed computing</strong> (MapReduce, Hadoop)</td>
<td><strong>Message queues</strong> (Amazon SQS)</td>
</tr>
</tbody>
</table>

TCP, UDP, HTTP, ... (low-level comm. interfaces)

Networking Stack
Why distributed systems? 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?
Why distributed systems? 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
Why distributed systems? 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? 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 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)
Topic: System 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”
Topic: Security

• 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

(Yu Qiao)
About Homework 1

• A warm up exercise.
• Determine whether you are comfortable with this class.
• Will be graded pass/fall. Will NOT be counted towards final grade.
• Read the submission instructions very carefully. It may be complicated.
• Always post your question at Piazza.
Next Time

• TODO for you: Homework 1

• We'll do a couple of case studies to understand what distributed systems really are:
  – Web architectures
  – A short history of cloud computing