Distributed Systems
CPSC 416
Winter 2021

Jan 12 Lecture (first class!)
Online

Oh yeah, pandemic

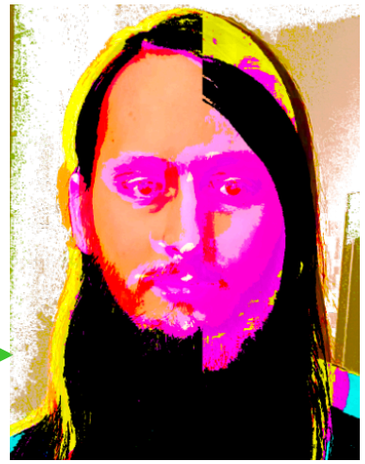
- Not a great time to be taking courses
- My first time teaching a large course over zoom
- Lots of resources, but this course may not be the right one for you (timezone/workload/content/etc)
    - Please consider carefully before committing
    - First two assignments before add/drop are a litmus test

Course staff

- **Ivan** Beschastnikh, instructor
- At UBC since 2013
  - Previous taught 416 four times (in person)
  - Research distributed systems, networks, security, program analysis

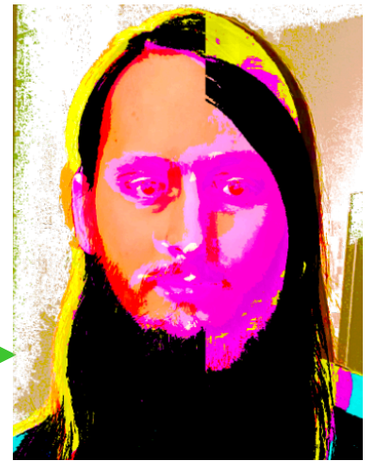
Course staff

- **Ivan** Beschastnikh, instructor
- TAs (all grad)
  - Finn
  - Mayank
  - Shayan
  - Shiqi



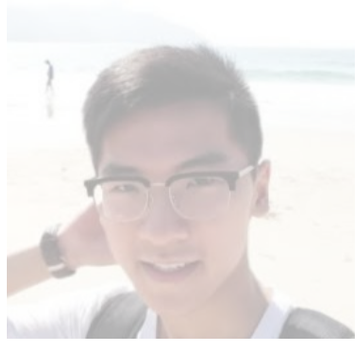
Course staff

- **Ivan** Beschastnikh, instructor



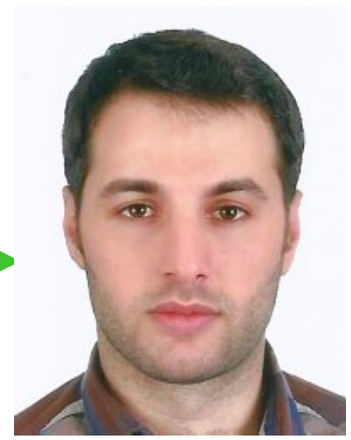
- TAs (all grad)
  - Finn
  - Mayank
  - Shayan
  - Shiqi

Arrives February



Arrives February

- PostDoc
  - **Jaafar**



Logistics

- 2016: 77 students (open-ended project)
- 2017: 117 students (assignment hell)
- 2018W: 160 students (assignments + projects)
- 2018F: 44 students (mix of above)
- 2021W: 160 students (assignment… hell?)
  - Zoom zoom zoom

Waitlist

- Waitlist has about 50 people!
    - Keep joining and working on assignments, some people will drop, *but not everyone will get in*
- To others: consider dropping if you have other courses that look more interesting

Basic resources

- Everything on the website, updated continuously:
  https://www.cs.ubc.ca/~bestchai/teaching/cs416_2020w2/

- Use Piazza for **all** course-related communication

- January office hours:

  - 6 hrs of office hours per week (see piazza/canvas for links)

  - Every day with Jaafar

  - + with Ivan on Thursdays

  - + with Shayan on Fridays

Course overview via the website

- Learning goals
- Go programming language (start learning!)
- Schedule (a work in progress)
  - Assignment 1 due Jan 15 (soon!)
- Exam (‘just’ a final)
- Advice for doing well
  - learn Go (a must to pass the course)
  - don’t hack, engineer
  - choose team, wisely
  - reach out on Pizza for help.
- Collaboration guidelines

Learning goals

- Understand key principles in designing and implementing distributed systems
- Reason about problems that involve distributed components
- Become familiar with important techniques for solving problems that arise in distributed contexts
- Build distributed system prototypes using the Go programming language

Learning goals

- Understand key principles in designing and implementing distributed systems
- Reason about problems that involve distributed components
- Become familiar with important techniques for solving problems that arise in distributed contexts
- Build distributed system prototypes using the Go programming language (the key to all the above)

Some workload comments from previous courses

- *The workload for this course is easily double that of any other course I had this term.*
- *Ivan has very high expectations of his students.*
- *I love and hate the fact that this class was a "sink or swim" approach to learning*

Assignment 1: BigUInt

- Okay, it's a little boring, but it will help you to:
  - Learn Go
  - Learn Go
  - Learn Go
  - Learn Go

Assignment 1: BigUInt

- https://www.cs.ubc.ca/~bestchai/teaching/cs416_2020w2/assign1/index.html

Assignment 1 note

- Last last year's 416 TA rant:

> TEST YOUR CODE ON THE UGRAD MACHINES!!!!!!!!!!!!!!!!!!!
>
> YOU WILL GET ZERO IF IT DOESN'T RUN OR COMPILE. WE HAVE NO SYMPATHY FOR THESE TYPES OF ERRORS.

… you've been warned

Zoom zoom out

- What are some examples of distributed systems?
  - Cloud: machines in a warehouse. Get AWS credits -> **Spin instances (VM)** -> SSH -> get things done.
    - Distributed accounting; distributed provisioning (request->exec, hypervisor); DC **F**ault **T**olerance (AWS buckets); storage services
  - HDFS: distributed file system for “big data compute” (provides data to compute instances; replication; FT; lookup)
  - Internet: global DNS (lookup: name -> ip); AS (autonomous systems) ~ ISP ~ network: BGP for coordination
  - Google drive: store a ton of data internally across many machines, FT (replicated), *“acts as one machine” ~ Consistency*
  - BitTorrent: “P2P” ~ free-for-all topology (“peer” or client is empowered); exchanging blocks of files; *ephemeral swarm*
  - Microservices ~ cool new trend for building cloud-based systems (service per task and interconnect them)
  - IPFS: “cool” “new” “file system”
  - Kubernetes: system for managing lots of resources
  - Zeronet: ?
  - BitCoin: scam ;-)
  - Twitch: video thing ~ Zoom
  - I2p: ?

Zoom zoom out

- What are some examples of distributed systems?
- Why not a distributed **application**? (DApps on blockchains)
  - System versus application: ?
  - Abstracted away from users
  - App is for clients, internals are systems
  - *System provides a “service” to other programs / **API***
  - *App usually interfaces with a person*

Zoom zoom out

- What makes a system ***distributed***?
  - Communication (networking)
  - Concurrency/async (threads/processes/machines/Pis)
  - Multiple machines/decentralization
  - Replication (coordination) for fault tolerance/fail over
  - Division of tasks (compute)
  - Scalability/high perf ~ nice to have for a dist. sys

Distributed system examples

- YouTube
  - Videos are **replicated** (multiple machines host the same video)
  - **Scalable** wrt. client requests for videos (internally **elastic** — can throw more machines at the service to have it scale out further)

Distributed system examples

- DropBox (or google drive)
  - **Replicated** content across personal devices
    - Supports **disconnected operation** (can work while disconnected, and synchronize when re-connected)
    - Maintaining data **consistent** across devices
- Supports sharing; **access control** policies (security!)

Distributed system examples

- NASDAQ
- **Transactions** (e.g., ACID semantics from databases). Many DBMS concepts apply to distributed systems!

- Strong **consistency** and **security** guarantees (otherwise people would not trust it with money)

Some D.S. challenges

- Synchronizing multiple machines (protocol complexity)
- Performance (how do you define/measure it?)
- Maintaining consistency: strong models (linearizable) to weak models (eventual) of consistency
- Failures: machine failures (range: failure stop to byzantine); network failures (just a few: disconnections/loss/corruption/ delay/partitioning)
- Security (how to prevent malicious control of a single host in a system escalating into control of the entire system?)

For Thursday

- Install Go on your personal machine
- Work through *Tour of Go!* and other tutorials.
- **Practice Go!**
- **Start on Assignment 1**