WELCOME!!!

CS 6650 *Scalable* Distributed Systems

(while maintaining some sanity?!) ☺

What are we doing?

- Weekly class time
  - Yvonne talking fast but recorded!
  - Industry mentors visiting
  - Working together to get the *crazy code* to RUN on AWS (or where you want to?)!!!

- Weekly mock interviews (sign up!) to walk through your code with the teaching team!

- Please read the syllabus and ask any questions!

How are we going to do it?!

- Weekly homeworks and Masteries are reviewed in *mock interviews*
    - 1:1 time with your TA!
- Homework will cover
    - Basics of concurrency and distributed system architectures
    - Server processing
    - Data, Data and more Data
    - Scalability issues throughout
- We will spend class time on homeworks together!

- Canvas, Teams, … talk to us!

The Homeworks build into larger Assignments!

- Client
  - Multithreaded exerting request load on dumb server to start ☺
  - Infrastructure to measure is on client-side
- Server evolves!
  - Database introduced along with “business logic”
  - Data storage management
  - Write-oriented workloads (making it harder to scale)
  - Threading on server
  - State management, horizontal scaling

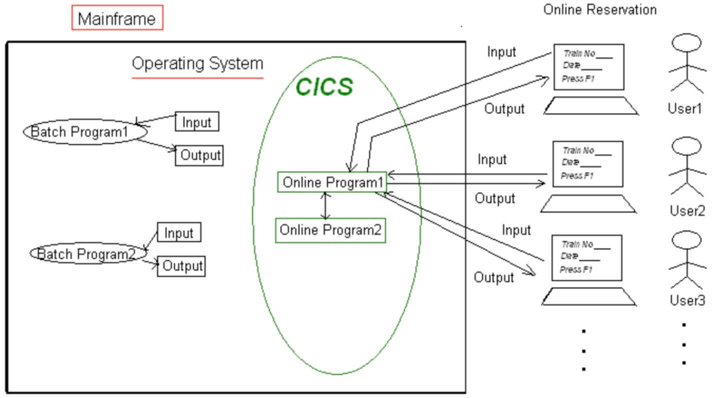
Server is Still evolving!

- Now scaling across more servers
  - Queuing for better responsiveness
  - Caching for performance
  - Monitoring
- And finally! ☺
  - Distributed database issues
  - More design patterns for data intensive computing
- Project ideas?!?

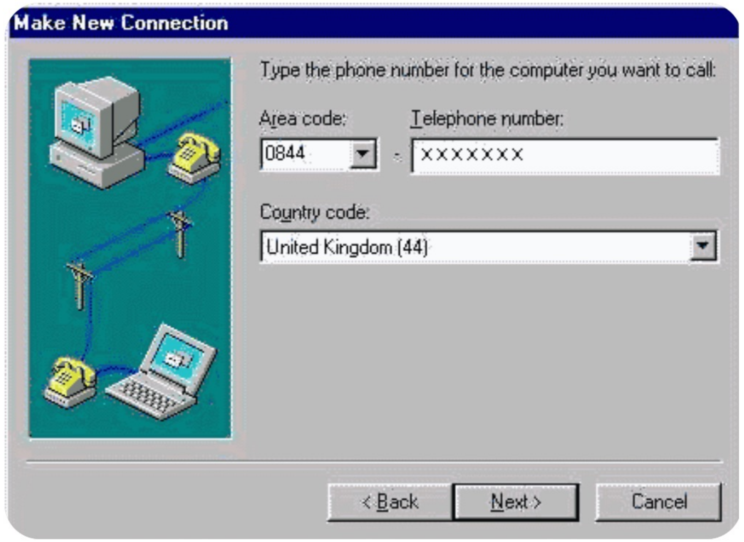
Quick intro, and history lesson!

- CICS IBM in the 60s! (still around)
  - Customer Information Control System
  - Mainframe and users
  - Timeshared server, dumb terminal
  - MTS (Michigan Terminal System)

- Along comes the Internet in the 90s
  - LANs, WANs, modems, and browsers (slow and simple!)
  - Eventually needed to start thinking about scale!
    - How many users/clients are hitting the banking server?
    - Amazon, Google, ebay?
  - Now we have MASSIVE data centers all over the world!
    - If we can process it!!!
    - Check out petabyte scale data sets
    - https://www.lifewire.com/terabytes-gigabytes-amp-petabytes-how-big-are-they-4125169



http://www.mainframegurukul.com/tutorials/programming/cics/introduction-to-cics.html



Software fabric that make distributed systems!

- Remote Procedure Calls
  - Java RMI
- Sockets (80s!)
  - Websockets
- CORBA (remote objects)
- HTTP
- More evolution to come…
  - Languages like Go and Rust!
  - Dask?!
  - Projects! ☺

Applications today

- These are just a few…
- Numbers a little out of date
- BUT YOU GET THE IDEA!

- More applications coming
  - Your project?!
  - The Earth Data Store

Dropbox – exascale storage system, millions of new files per hour, from 2012-16 seen over 12x growth.

Gmail – 1.2 billion users, blocks 10M spam messages every minute

Netflix – runs on AWS, took 7 years to migrate, uses 15% of global downstream internet traffic

Facebook - More than 300 million photos uploaded per day and 510,000 comments posted and 293,000 statuses updated per minute

Youtube - Users watch 4,146,600 YouTube videos every minute

What do these systems all have in common?

- Transactions, user interactions, analysis
- Many more have sensors, IoT
- Many more now have rich UIs
  - AR/VR!

- This means we have **large code** bases along with these massive data sets!
- And **large data sets** of performance monitoring results!
- They operate on multiple “nodes”, and are distributed systems

Nonfuctional Requirements (quality attributes)

- Performance, performance, performance
  - NOTE: collecting data on performance generates a lot of data!
  - Analysis of this is also challenging
- Scale
  - Performance is still good, even with more data, more users, more everything!
  - Note this can be very dynamic spikes
    - A Tweet from a well known person!
- Security
  - Data and access control
- Available
  - 24/7 (Leah knows about stress!)

What is SCALABILITY?

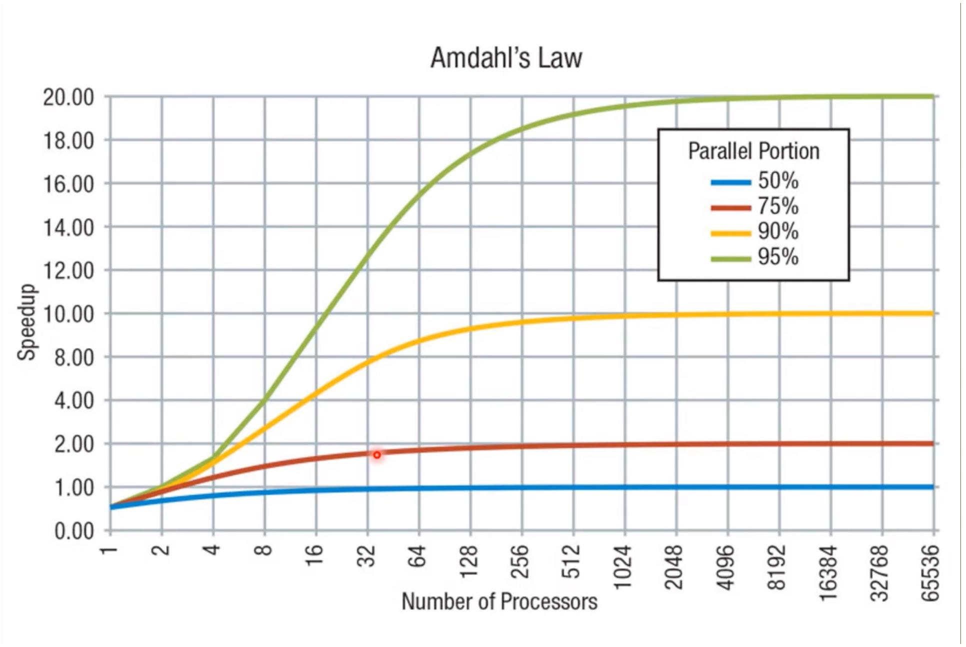
- The ability of a system to **increase** or **decrease** capacity in response to changing demands while continuing to satifsy service level agreements for latency and availability

- “Elastic Compute”

- Hyperscalability is the same as the above, but while supporting **exponential growth** with **linear costs**

Can we just throw hardware at it?

- Yes and no!
- We want linear “scale up”
  - With each additional machine, we want to be able to grow linearly
  - BUT there is Amdahl’s Law (wikipedia here!)
  - At some point we level out
    - Not getting faster because some things have to be done in sequence, one at a time
    - Example, a write to a database
    - “Serial bottlenecks”!



SOME problems are easily parallelizable

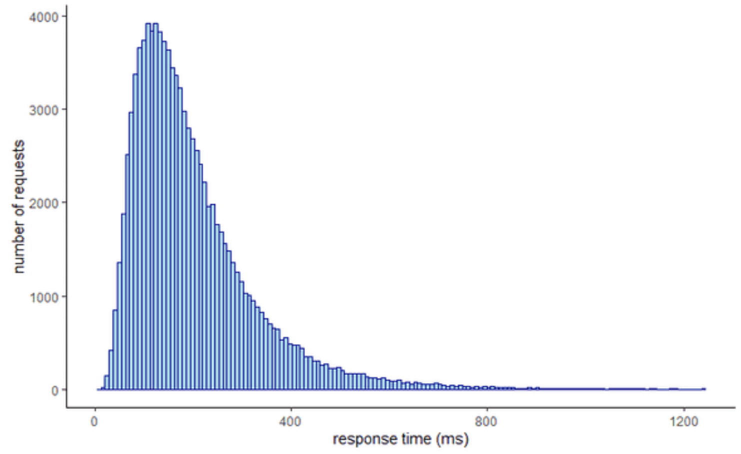
- Map/Reduce (Google paper, please share your thoughts on Piazza!)
- Some serialized behaviour but does well!
- Can be a little tricky to get right…
  - Data Skew
    - Breaking the data set down into “chunks”
    - Not every chunk takes the same amount of time to execute

- Please take a look at the paper (link provided on Canvas) and share your thoughts!

Response times are critical!

- Within *40 milliseconds*
  - A user will LEAVE!
  - Impacts google rankings!
  - There are delays in data transfer (latencies per request)
  - You will love measuring these! ☺

- Fun fact is that there is a VERY LONG TAIL graph
  - Slow responses DO happen
  - So how can we mitigate those?!



https://robertovitillo.com/why-you-should-measure-tail-latencies/

Throughput

- Typically reported as *requests per second*
  - Note: Latency is for a single request
  - Throughput is a bulk measure of **many** requests
  - You will love measuring this!

- When you have millions of requests per second *and* bursts
  - We need to split the load across multiple nodes!

- “Scale-out”
  - Add more nodes to increase throughput…
  - Does it always work?

Availability!

- 24/7
  - 99%, 99.999%, 99.9999%...

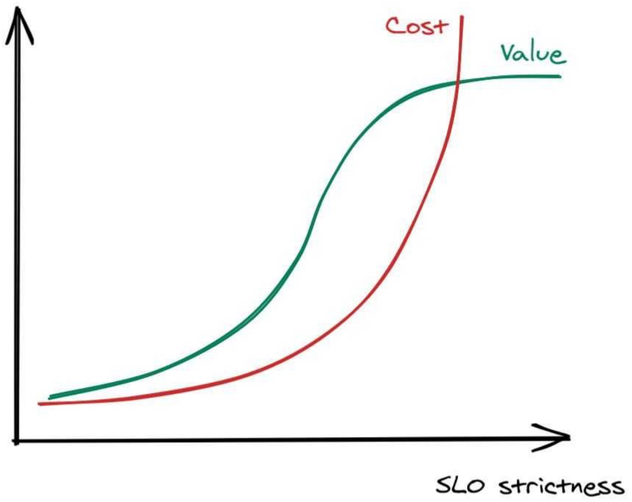
Availability %	Downtime per year[note 1]	Downtime per quarter	Downtime per month	Downtime per week	Downtime per day (24 hours)
90% ("one nine")	**36.53 days**	**9.13 days**	**73.05 hours**	**16.80 hours**	**2.40 hours**
95% ("one and a half nines")	18.26 days	4.56 days	36.53 hours	8.40 hours	1.20 hours
97%	10.96 days	2.74 days	21.92 hours	5.04 hours	43.20 minutes
98%	7.31 days	43.86 hours	14.61 hours	3.36 hours	28.80 minutes
99% ("two nines")	**3.65 days**	**21.9 hours**	**7.31 hours**	**1.68 hours**	**14.40 minutes**
99.5% ("two and a half nines")	1.83 days	10.98 hours	3.65 hours	50.40 minutes	7.20 minutes
99.8%	17.53 hours	4.38 hours	87.66 minutes	20.16 minutes	2.88 minutes
99.9% ("three nines")	**8.77 hours**	**2.19 hours**	**43.83 minutes**	**10.08 minutes**	**1.44 minutes**
99.95% ("three and a half nines")	4.38 hours	65.7 minutes	21.92 minutes	5.04 minutes	43.20 seconds
99.99% ("four nines")	**52.60 minutes**	**13.15 minutes**	**4.38 minutes**	**1.01 minutes**	**8.64 seconds**
99.995% ("four and a half nines")	26.30 minutes	6.57 minutes	2.19 minutes	30.24 seconds	4.32 seconds
99.999% ("five nines")	**5.26 minutes**	**1.31 minutes**	**26.30 seconds**	**6.05 seconds**	**864.00 milliseconds**
99.9999% ("six nines")	**31.56 seconds**	**7.89 seconds**	**2.63 seconds**	**604.80 milliseconds**	**86.40 milliseconds**
99.99999% ("seven nines")	**3.16 seconds**	**0.79 seconds**	**262.98 milliseconds**	**60.48 milliseconds**	**8.64 milliseconds**
99.999999% ("eight nines")	**315.58 milliseconds**	**78.89 milliseconds**	**26.30 milliseconds**	**6.05 milliseconds**	**864.00 microseconds**
99.9999999% ("nine nines")	**31.56 milliseconds**	**7.89 milliseconds**	**2.63 milliseconds**	**604.80 microseconds**	**86.40 microseconds**

https://en.wikipedia.org/wiki/High_availability

- Don’t want a single point of failure (SPoF)
  - So we need to make sure we have replicas/redundancy!

- When you have 1000s of components, some will fail!
  - Need to incorporate FAILURE into our designs

It is all about the *Tradeoffs*!



https://robertovitillo.com/why-you-should-measure-tail-latencies/

- Throughput versus cost
  - More hardware means more $$
  - When is it worth it? When isn't it?
- Performance versus availability
  - Fast but components may fail!
  - Remember the old mainframes, they were fast-ish ☺
- ***Need to be able to measure and discuss the tradeoffs in the work you do in this course!***
- This all depends on the context of the requirements of the system you are building, there is no ONE right answer!