CS 188 (Scalable Internet Services) Notes

W 0 R Lec 9-22-16

- The Internet takes many forms and people have wrestled this for thousands of years.
- We need to make sure with billions of users, these things continue to function
- If you want to scale up a very simple web service to millions or hundreds of billions of users, you need to scale your applications.
- Scaling something like Twitter is very difficult even though it is a simple app.

- You have some idea and a lot of stories and building something that becomes important to use.
- As time goes on, your user base grows exponentially!
- This becomes a problem and your growth is growing really fast.

- What is an Internet Service?
- HTTP - a ton of traffic goes through using HTTP
- Two servers wanting to talk to each other is using HTTP
- When a native mobile application uses this, it probably sends structured JSON across

- Do we mean web browsers?
- Yes but so much more!
- HTTP is the foundation for most APIs on the internet
- Building an application and process payments
- The company that transacts the money will create an HTTP interface
- The core of what we are talking about extends to the experience from a browser to a native application

What do we mean by scalable?

- When your users grow and you want to make sure your code doesn't break or have issues.
- As your users grow over time, your code doesn't break.
- A long time to return.
- More and more users == traffic
- requests per second
- Even if Yelp had a billion users, they have to handle hundreds of millions of restaurants.

"Effectively meets demands"

- Service is just available

Counter example

- We put things on AWS

- When traffic gets to be too big, we deploy a larger instance on EC2.
- Why is this NOT scalable?
- Eventually you run out of instances.
- Eventually, they have a biggest instance.
- This is NOT a scalable solution.
- We would call these vertical scaling.
- We would prefer horizontal scaling!
- More resources rather than more powerful resource(s).

Scaling is the core of this course, but we will see other things important to internet services

- Performance != scalability
- Up to a certain level of load before it falls over.
- Web service that responds more slowly
- Low performance but scalable solution
- There will be techniques that provide both scalability and performance
- Security
- How HTTP keeps data secure
- It can be difficult to understand performance limitations
- Agile software development
- Test driven development
- Important part of building industrial software
- The increasingly rich client
- Focus on the backend
- Web browsers are becoming more sophisticated
- This can influence the topic as well.

In summary…

- This course will teach you how to build an application **that can respond to** worldwide attention and usage.

Course Structure

- Labs on Fridays at 12 PM & 2 PM in Boelter 2760.
- Focused on the course project with a demo each week.
- Office hours Friday 10 AM to 12 PM in 4531M Boelter

This course is **project intensive**.

- Work in teams of four
- Develop an interesting internet service
- Deploy it on Amazon EC2
- Measure its performance and scalability
- Build out the functionality and use techniques to achieve higher and higher degrees of scale
- Use this technique to show quantitatively that this supports a higher number of users.
- Experience with load testing.

- Document these improvements and present them.

This course is demanding, but it is very rewarding.

- A new programming language (Ruby)
- An application development framework (Rails)
- Amazon Web Services: EC2, S3, CloudFormation
- How to load test an internet service.

This is **not a deep-dive** in Relational Databases, Networking, Distributed Systems, Network Security, or Cloud Computing.

This course has an **industrial focus.** We will use industrial software development techniques.

- It is possible for everyone to be successful and all the code will be open sourced on GitHub.
- Software development would be renamed for "Googling Stack Overflow"
- We want to solve problems because they are useful for the world.
- Encouraged to look at each other's code and use Piazza.
- Git(hub), Ruby/Rails, Travis CI, NewRelic
- Travis CI runs all your automated tests each time you run your code.
- NewRelic monitors your code

Why Ruby on Rails?

- Rails weaknesses: CPU usage, memory usage (performance is bad)
- Rails strengths: Building web applications quickly is important

Being able to scale your web application would be the same if you want a fast framework or a slow framework.

What does this course have to offer you?

- Ruby on Rails gives you a high paying job.
- If you are headed for academia, there is still a lot of useful information presented.

Course textbooks:

0. "High Performance Browser Networking" by Ilya Grigorik
0. Ruby on Rails Tutorial Book by Michael Hartl

Your grade will be based on three main factors

- **Your final project presentation**
- Your final project paper
- Your teammates' evaluation of your contributions

Also considered, to a lesser degree

- Weekly demo progress
- Assistance to other students on Piazza
- Each will grade the other four students.

Obviously, don't wait to the last minute because this takes a lot of time.

Goal: Gain hands-on experience building and deploying a scalable web service

- Choose interesting and large data set
- It doesn't need to be a genuinely novel thing or super useful to the world.

No Friday session tomorrow

Course Project

- One week sprints and write down in a backlog about what we want to accomplish
- Modern software engineering techniques
- Scrum, TDD, Pair programming
- Choose what you use, so Unix is **highly** recommended (Mac OS or Linux)

Past Projects
http://scalableinternetservices.com/projects

Important Links
https://scalableinternetservices.com
https://github.com/scalableinternetservices
http://piazza.com/ucla/fall2016/cs188

Lecture Structure

- The course lectures will cover material you need to build a successful project, and additional material that won't directly applied.
- Necessary content for successful projects is front-loaded so you can apply these concepts early.
- Second half will be additional material and it is interesting.
- CTO of Splice is coming and GitHub for music
- Guest speaker from Snapchat
- Guest speaker from Riot Games

First five weeks will cover:

- Intro to the basics: HTTP & HTML
- Industrial software engineering: Agile, TDD, CI, Pairing
- HTTP Applications Server architectures
- High availability via load balancing: a share-nothing web stack
- Client-side and server-side caching
- Using relational databases in web applications: concurrency control and query analysis
- Scaling via sharding
- Scaling via SOA
- Scaling via read-salves

Subsequent lectures will be guest speakers and

- Scaling via non-relational databases (NoSQL)
- Basics of web security: Firewalls HTTPS, XSS, CSRF
- HTTP 2.0
- Client-side renaissance: Client-side MVC
- Thicker clients: Asm.js, Emscripten, Webruby
- Content-delivery networks
- Elixir, Phoenix, and Erlang (maybe)

http://www.codeacademy.com/en/tracks/ruby/

W 1 T Lec 9-27-16

- Donald Trump's website crashed.
- Situation where in a moment, 8 million people go to the same website
- Google Chrome and go to the Web inspector and all this info should be here.

The Lifecycle of a Request

- We have a web browser, which is a process
- There is no magic, it is a process like any other.
- These apply just as well to a web browser
- A web server is a process that runs on an operating system
- If you have a fork bomb, it tends to be bad.
- Just use all that same intuition to reason about web browsers
- Web browser is running on the client's computer and they talk to each other over the Internet

A string isn't routable on the Internet! You need to route it.

- DNS resolution before you do anything and get back an IP address
- HTTP sits on top of TCP, so at the beginning of this interaction, we are going to set up a TCP connection.
- Classic three way handshake from TCP networking.
- You are really talking to just one server.
- There are very few single pieces of hardware that can handle that many simultaneous requests.
- Use multiple servers and get them to appear to the outside world.
- If you are going to have all these servers appear to the outside world, you make these stateless and hold these on backing information.
- In your project, focus on using relational databases as your backing store.
- We will be talking primarily about relational and SQL databases

HTTP

- Visually depict what Mutz is going to talk about at a given day.
- HTTP is this protocol and your web browser is talking to this server.
- Internet was used for things like telnet and email
- Two existing technologies could be connected for good purposes.

- HyperText is a text document containing links and these go to other text pages.
- Wouldn't it be great to have documents over links at the Internet.
- There were other projects like Xanadu but it never really took off.

NextStep is the descendant of Apple
- Andreessen (founder of Andreessen Horowitz) did some Internet stuff

HTTP Explained
- Very readable and if you were to attach a traffic analyzer, there are parts you can read and understand well.
- In its original, simplest version, it says to open a TCP socket and send over a text request and close the socket.
- Single TCP and request and get something back in.
- All we were doing is getting HTML
- Serve up the images on the page and serve up structured data and all sorts of things

HTTP Explained
- Request:
- Verb: GET, POST, etc.
- Resource: What are you getting? If I was getting /index, that is a resource
- HTTP Version: Easier to develop the protocol. The person developing can behave differently
- Headers: A lot of flexibility
- Response
- Status code: Tells what the return error is

HTTP Explained
- Some number of headers and we know we have completed the headers when we get two newlines
- Status code and the server also sends back headers

HTTP Verbs
- GET
- Get a copy of the resource
- Should have no side-effects
- If you do a GET any number of times, it shouldn't change the state of the server
- If there is a sign out link, it does a GET /users/sign_out, but this would betray the spirit of the verbs
- Parts of the web assume that this behavior is the case
- Web browsers cache a lot of information and these aggressively go out and cache data.
- This would go out and data before you go request them

- POST
- Sending data to the server
- When you have interacted with forms in the most vanilla way, this is submitting a POST request
- Assumed to have side effects
- Filling out information and push some state and modify state on a server
- Not idempotent
- Idea that for some types of actions (one time or many), it should have the same effect.
- If a user signs out once or 100 times, it pretty much has the same effect.
- A non-idempotent action but be something like creating a new blog post.
- This is the idea that if you issue multiple POST requests, it would have multiple instances of side effects
- Back button (do you want to submit this form again?)
- POST requests can do things multiple times, so do you want me to do this again?

- PUT
- Used to update an existing resource
- We are updating a single thing, so if we do multiple PUT requests, the output will be the same as doing a single PUT request with all the changes

- DELETE
- Destroys a resource
- Tends to be idempotent but has side effects
- Deleting a session and the outcome of the server will be the same

- HEAD
- Used for caching and this is how verbs should be used

Various assumptions that browsers and plugin designers use

- Less commonly used verbs:
- TRACE
- OPTIONS
- Tells you the verbs that are supported for all the resources
- CONNECT
- Connect to a third web server, which is a little weird for security purposes

- I would send a request, and it would send me back all the headers for debugging.

HTTP Resource

- Think of this as a logical hierarchy
- This is NOT the file path location.

- The Apache web server would just serve up files as though they were resources and they were really separate.
- You can have resources that don't map in any way to the file system and you can have any form of mapping.
- These are actual links here and there were categories of products.
- URLs are readable to people and we can understand this hierarchy a little bit.
- Anything past the question mark is called a query string
- Search terms and help locate the resource

HTTP Version

- There is HTTP version and this is included in the protocol to make it easier to transition from one version of the protocol to future versions.
- 1991, HTTP 0.9 was a single line protocol with no headers
- 1996, HTTP 1.0: Headers added so you can serve more types of content
- 1999, HTTP 1.1: Connection keep-alive and freshness of data
- 2015: HTTP 2.0: Modern day HTTP

HTTP Headers

- Accept: Indicates the format of data that it would like tor receive back
- Accept: text/html
- Get the version in text/html
- Accept: application/json:
- JSON is structured javascript object notation
- Accept: application/json, applciation/xml:
- You could have a situation where your client says to have applciation/json OR application/xml
- Server responds to see what it actually produced

- Accept-Encoding: I probably want to just compress it and that indicates that I as a client can handle zip data or gzip data
- Host: www.google.com (host header is that string)
- You could host multiple servers on the same machine, and that machine can accept different requests from different hosts)
- Accept-Language: indicates the preferred languages (in order)
- Accept-Language: es, en-US
- I prefer spanish, but will accept US english
- User-Agent: Indicates what type of browser or device is connecting.
- People occasionally will serve up different content based on the user-agent string and we have different browser manufacturers
- Everyone building a new browser can be seen as compatible.
- Browser will announce itself as other browsers
- Web developers made mistakes like checking if user-agents are Mozilla, do the following.
- They want to be appear the more dominant browsers and the user strings are unreadably ridiculous

- Set-Cookie & Cookie
- In the response, we will get Set-Cookie and the browser will present this cookie when it interacts with this web server.
- Persistent logins (here is a unique string, present this to me in the future)
- Advertisements (servers want to be able to track you as you travel across the Internet)
- Use cookies so you can uniquely identify yourself to the ad services.
- Connection: keep-alive
- Reuse TCP connections for multiple connections

- Security related headers
- Click jacking attack and surrounding with malicious software and X-Forwarded-Proto and we will talk more about security stuff
- If you don't know exactly the size of the data, you can use these headers to navigate the beginning and end of the data.
- Headers that begin with X- are not part of the HTTP specification but can be "standardized"
- All of your web servers will interact with these servers in the same way and not caught in RFC specs

HTTP Status

- 1XX - Informational
- 2XX - Success
- 3XX - Redirecting
- 4XX - Client Error
- 5XX - Server Error

- 200 OK: Everything works well
- 301 Moved Permanently: The server is saying what you are looking for can be found at a new location
- 302 Found: Temporarily redirected and go to some other place. Don't rely on this general
- 403 Forbidden: Client has not authorized himself correctly
- 404 Not Found: Specified resource could not be found
- 418 I'm a Teapot: April Fools!
- 500 Internal Server Error: Something crashed
- 503 Service Unavailable: Temporary failure.

HTTP Body

- The body is used for things like a POST request and the content would be encoded in some way in the body.
- Most commonly, it will come back in the response.
- GET request may not have a body but the response will have a body

netcat - nc: Used for networking things

Beyond one Request per Connection

- If you build an HTTP client, you can open and closer your TCP connection.
- This is inefficient. It takes a long time to establish the connection
- You have to do the handshake and you have to do all he congestion control stuff.
- Use multiple round trips to setup connection and TCP connections have low bandwidth

Before you can effectively communicate at all with the server, you have to go to the server and traffic has to come back to your client and then the data has to come back here.

- Kind of inefficient and your latency could be much higher.
- If you get your Internet through like a satellite connection, you can have a high latency.

TCP has slow start, and that is the slowest and it isn't until we have done multiple round trips that we get a fair amount of bandwidth.

- This is the effective bandwidth we can have here and we always have a low amount of bandwidth.
- As time goes on, we can only send small amounts of data.

In HTTP 1.1, they introduced "Connection: keep-alive"

- Send my request and get my response.
- Once I receive my response, I can send out more requests to the server

- The client here establishes connection and the orange bar corresponds to the server
- The client here sends another request and gets a response

People tend to NOT perform multiple GET requests without hearing a response back

- Head of line blocking
- You have multiple resources that are being served up and this cannot logically be served up to the client until the first one has finished.
- Depending on the order that the client has requested things, this resource could be more urgent.
- Due to the serialized order that it has been requested, this can cause problems
- If I have a bunch of requests, what happens if the system crashes?
- You won't know if the request got handled or not.

A browser will open up multiple TCP connections and send its requests to whatever is most appropriate and wait to see what is completed.

- A browser that needs N resources and in theory, we could open up all the TCP connections at once and issue them in sequence.
- We should use **six** as the number of concurrent connections.
- In HTTP 2.0, you will only need **one** connection
- If you wanted more of them, you can use “domain sharding” and all of these names would go back to the same actual machine.

HTML header contains information about the page that isn’t directly rendered

- We will talk more about CSS later in the slides.

- h1 - Header
- table, tr, td - display data in a tabular way
- span - inline, so text style layout
- div - two dimensional layout
- It is now more common to build using spans and divs and style appropriately with CSS

Intro to CSS

- The precedence order says that more specific has higher precedence than less specific
- inline > id > class
- !important can be used to increase the precedence of a CSS rule
- inline styles are frowned upon by web developers!

By separating out things into different stylesheets, your web application becomes more maintainable.

- React.js uses JS to construct the DOM and they advocate using the inline styles
- Outside of that world, we hear people frown upon inline styling.

Bootstrap is a library of UI controls and widgets and you get these off the shelf.

W 1 R Lec 9-29-16
Headline for TechCrunch

- There is tech talent shortage and there always will be
- IT jobs market booms, but talent is in short supply
- Managing scarce resources most of the time like managing CPUs and memory.

Introduction

- Modern software techniques are designed to optimize for time.

- Industrial Software Development Techniques
- Next steps between now and tomorrow in the lab section

For Today

- Sprint 2: Starts October 14, 2016
- What your team should be doing and if you don't have experience in industry, it won't make too much sense.

Agile & Scrum

- Scrum is a type of Agile process
- Agile software is not necessarily Scrum, but Scrum is a subset of Agile

Waterfall

- The term Waterfall comes from this paper "Managing the Development of Large Software Systems"
- This is one of the famous figures from that paper and it was very convoluted.
- Each box represents a task and each arrow gets passed down
- Analysis and program design and code the software and operate the software.
- Software was engineered like this in the 70s, 80s, and 90s
- The first type of software that Mutz worked at was a startup and there was no process
- A lot of them came from academia and they did what made sense
- As the company grew, they brought in an "adult" and he brought in the waterfall process and everyone was in some phase of development and each phase was well-defined.
- It was all done in Word documents and it was a nightmare.
- A lot of the software engineering world moved away from this type of planning

- Good for specialization, but the problem is that there are bottlenecks
- Sometimes when you test software, many assumptions in earlier stages are wrong.

Bad assumptions

- Assumes that software testers and divs know how everything work
- Software developers won't have many surprises
- Designers understand the difficulty of each design option
- We frequently find out things that invalidate assumptions earlier in the process
- Instead of embracing change, this fights change or assumes it isn't that common

Agile

- In 2001, the "Agile Software Manifesto" was written
- A few different individuals come together and write this book
- XP (Extreme Programming)
- Schwaber and Sutherland come up with Agile

- They described their values and Agile software development wants to **respond to change**
- Embraces change because the assumptions made at the beginning will not be true later on
- **Working software** takes precedence over comprehensive docs
- **Customer collaboration** over contract negotiation. Team would talk to people using their software and get them to agree upfront.
- Showing your software early and often and getting feedback
- **Individuals and interactions** are trying to achieve goals

Responding to change over following a plan
- When you start your journey, can you plan out the whole thing?
- In this world, you acknowledge that your map is hazy and you have some idea, but in this sort of world, you start walking with a compass and start backtracking as needed.

Working software over comprehensive docs
- Rather than describing to your customer what the software is going to do, you will demo and use the software with them.
- Automated acceptance tests are better than extensive documented requirements
- Don't use photoshop mockups. Write automated tests!
- Not all automated tests are acceptance tests, but all acceptance tests are automated tests.
- Descriptive unit tests: Larger codebases -> See a method in a long codebase with comments of that method
- After the code was written, someone changed it so the comments no longer reflect reality.
- In Agile software development, it de-emphasizes code comments
- In situations where you would have code comments, you would have descriptive unit tests.
- Instead of having a style guide, you would have a library
- Think of this as we are going to use Bootstrap and restyle Bootstrap factored out into a library

Customer collaboration over contract negotiation
- Now that I have seen it in action, I have changed my mind, and the non-Agile way is to assume that building what they asked for is fine.

Individuals and interactions over processes and tools
- Acknowledge that it is a full teamwork and focus on the goals to be achieved
- QA Tester but also knows how to write software as well
- Sees a bug and offers of fix it
- This is a good thing to do!

- Focus on pushing the same goal

Scrum

- Preceded the Agile software manifesto
- Build software in an industrial software
- To have that many roles in scrum, it doesn't make sense in the context of this class.
- Kansan: Focuses more on controlling the Work In Progress
- Whatever your team is working on at this moment
- XP: Automated testing and pair programming

Scrum roles:

- Product Owner: Someone who understands what you are trying to build and can answer the question of if it will work for the end user.
- If the engineers come to the Product Owner, they can answer questions based on the needs of the user.
- Be able to define how valuable different features are.
- Comes up with a backlog which classifies how vl
- The Team
- Intentionally vague roles
- Scrum doesn't care about if you are a QA Engineer vs a software development engineer
- Scrum Master
- Helps makes sure the scrum process is mostly followed and if the team is blocked, the scrum master makes sure this happens
- Not a management role but rather making sure external issues get resolved

Story

- A feature (or a small slice of a feature)
- These stories are user visible.
- Won't have stories that are like "refactor code to be cleaner"
- Reflects some unit of customer value
- Order them so we get the most value first.
- A story tends to be written as a sentence and looks like "As a ____, I can ____, in order to ____"
- Have a brief description of what the user is able to accomplish
- Sprint
- A fixed length of time in which the team will work.
- Try to do these 4 stories and for the length of that Sprint, they will have those story.
- You can release your software every 2 months, but you would decide a new collection of stories to work on.

Scrum

- 1 week sprints

- Team will meet and based on product backlog, the team will decide what they want to make this week
- Teams give their best estimate of what can be accomplished in that week
- Pulls from the product backlog which is a list of stories curated by the Product Owner
- Try to figure out what they need and increase sales.
- Product Owner is constantly readjusting the list
- The top of the product backlog are the most important thing to accomplish

Spring Backlog

- We are going to accomplish these 4 stories and velocity is how many stories we plan to accomplish this week
- Not always accurate, but we tend to overestimate what we can accomplish.
- Keep track of velocity and this commitment gives visibility to the product owner
- Commitment is frozen so don't readjust content of the sprint.
- Standup meetings intend to make meetings faster!
- Typically done at the beginning of the day and discuss how things are going and what should be planned.

Each week, we conduct a retrospective

Scrum

- For our projects, we will use a lighter version of Scrum

Test-Driven Development

- As you are working, everyone is taking the application and taking it in 100 different directions
- Information isn't global in this world
- Take a certain object class and extend it in that way and those two ways can be contradictory
- How do we find out if we make errors?
- We could have a QA department and they can do a ton of regression testing and make sure that things that used to work still work.
- As the app gets bigger, it is harder to do this.
- Have the compiler warn us of problems!
- Languages with rich, strongly typed systems i.e. C++, C, Java
- Ruby is NOT strongly typed

Humans would be expensive resources to test

- Add 20-30% to our headcount and type systems and compilers work well for some defects but if person A and person B were taking the class, it could notify you of some problems

- What was previously a string is now a different type and they look fundamentally different
- You are writing things that should be checked as you go.
- Invariants are still the case.

Automated testing

- We are going to be writing code that test other code.
- They are generally in the same language and it is just code that executes the code we are written
- A code coverage tool runs automated testing suite and will see how many lines are executed by the automated testing suite
- If methods are never added, we need to add tests
- Ruby is a very permissive language and other classes of that object will never get tested

How do we get automated tests to cover it?

- Write the test first before you write the code, and then you write the code to make it pass.
- The code is supposed to do what it is supposed to do.
- Don't write any production code until there is test code that tests it
- Write the minimal amount of production code to make it pass.

- Once you have tests in place, refactor your design
- In terms of human time, it is generally worth the time
- Most of the code you write in school is generally bad to write tests for
- In industry, it is opposite because what you are using will be useful a year from now.
- At AppFolio, we have tens of thousands of automated tests so we want to run all of these tests, so we will spend money to automate all this.

- Tests aren't important in this class but it is a good practice!

Test-Driven Development Example

- Fizz buzz

- Controller Tests
- Integration tests
- State of your database
- Selenium tests
- Browser - automated tests like issuing click commands
- Pain to write and it causes the biggest headaches
- As you are adding code

Continuous Integration

- If all of us were building an application together, we could go off, make some changes, and want to reintegrate back into the codebase.

- Make things reconcile and run together
- This can be very painful!
- If we have a bunch of people making changes, getting them to all be coherent can be painful!
- This painful process of getting a working build should be done as soon as possible
- “The effort of integration is exponentially proportional to the amount of time between integrations”
- Every time you make changes, you will merge them back to master as often as feasible.
- Once you have merged things together, how do we know if it works?
- Run all of your automated tests.
- “Continuous Integration” server will monitor your source control system for newly checked in code.
- Every time there is a change, you will run all of your automated tests and notify people ASAP if there are problems within their code.
- Run all the automated tests before you checkin code but that could take too long.
- If we had 100 developers in this room and we check in 3 or 4 times a day, sometimes, we need a lot of machine to fill this problem
- Code quality tells you if things are simple or complex
- The reason it is not perfect is because it is a written algorithm

Continuous Integration

- As these things pass, we get more an more down here i.e. RC (release candidate)
- As you go further and further down the list, we get a higher degree of certainty
- Number 9 is sort of advisory, but we obviously prefer clean code
- In GitHub, you can integrate Travis CI with pull requests
- User wants to make a change to the software and if they want to merge that code back in, it will be a request for others to review and merge it.
- Want it to be run ready to be merged in.

GitHub Workflow

- Distributed version control system and we want to keep track of changes to files through history.
- Git-flow and Github-flow
- Git-flow: More complex and it is older. More powerful
- **We will be using Github-flow**

Github Flow:

- If you are working in Github work flow, you would get your copy of master up to date
- Regularly committing on this branch and pushing to origin

- When you are done and your code is sitting on a feature branch, you will open up a pull request

git rebase

- If your Git history pointed to a certain revision, you could rebase it to another branch
- If you commit regularly to a feature branch (don't want to lose work), you could squash your commits down to a smaller number so your code will be more sensible and readable to future generations

Pair Programming

- Most useful when developing a new feature
- When you are reading code, it is difficult for two people to do so at the same computer and probably not worth the time
- Not required, but recommended

W 1 Dis 9-30-16
Work on the database stuff

W 2 T Lec 10-4-16
Motivation

- We have this technology stack and we will be talking about what runs on this individual server

- GitHub README
- Choose from different cloud formation templates and these are different types of application servers.
- We will have opportunities when we go to load test things later in the class
- You can choose a type of virtual machine and thread/parallelism

- We've seen the HTTP protocol
- Out in the world, there are different browsers that speak HTTP and send over a request
- Receive some data in a response and to reuse that socket for multiple requests
- The software that powers these servers falls into two categories.
- It runs an HTTP server or an application server

- The HTTP server is the first point of contact
- Why do we break it up like this?
- Load balancing: So you can have one manage multiple requests and have them manage the same thing.
- In the process of forwarding this request, we can have multiple requests.
- More modular and improve them separately

- HTTP Server:
  - High performance piece of software that is pretty secure and pretty reliable
  - Apache has been around for 20 years and that piece of software has been relatively stable and very configurable over a long period of time
  - You will have concurrency concerns taken care of here.
  - The version of your web server will be pretty static

- App Server:
  - Specific to your application
  - Different for each person
  - 4 different people building 4 different Rails apps will use 4 app servers
  - Specific to your business or problem and it is optimized for changing rapidly
  - It wouldn't be uncommon to be releasing your new application server all the time.
  - Releasing it weekly, daily, or many times a day.
  - MVC architecture

HTTP Servers

- Apache owns most of the market share
- The green line is nginx and it came out about 10 years ago and is a high-performance alternative to Apache.
- Throughout the same time frame, Microsoft has been losing market share.
- In this other category, we have Open RESTY (?) - invariant of nginx

HTTP Server's responsibilities:

- Parsing the HTTP request and crafting HTTP responses very quickly
- We want it to handle a high degree of concurrency
- Responsible for figuring out the right handler
- If your HTTP server is facing a Rails-based app server, it can be satisfied from the REST server itself
- Needs to be stable and secure
- Computer security is obviously very important and add features and fix bugs and security holes
- Provide a clean abstraction so have a configuration file that easy to reason about, easy to understand so the user can understand what gets forwarded to where.
- Lots of ways to design and architect an HTTP server

Many possible ways to architect an HTTP server:

- Single Threaded
- Process per request: Know the difference between process and thread

- Thread per request:
- Event-driven: Different style of programming

HTTP Servers - Single Threaded

- Write your own HTTP server and in the homework, you would build your own HTTP server
- Bind to port 80 and listen()
- Enter a loop you would sit in forever
- Try to read from that socket and read a request
- Handle that request and write out the response.
- Close the connection eventually
- If we built it in this way, and we had a request come in
- What happens if we get more HTTP requests coming in before the software is processing the response.
- Wait for the other request to finish processing and get the backlog.
- Your OS would be kind and hold on to the first couple of TCP requests but then it would stop accepting TCP requests to handle these sockets.
- This wouldn't be ideal obviously because a lot of waiting around.
- You cannot just rip through all of this really fast.
- Especially processing this request, getting from disk is probably the fastest thing it can do.
- Every time it tries to process the request, it does outbound network activity

Problem!

- The act of responding to a request takes a while.
- Trying to do everything in one thread of control, so it is hard, but if we have multiple threads of control, we can fix this.

HTTP Servers - Process Per Request

- Loop forever and after accepting a socket connection, it will fork
- It would loop over this TCP socket and figure out how to respond to this.
- Process was getting stuck wouldn't happen anymore and all it would do is accept sockets and fork.
- **Forking is an expensive operation**
- Forking a new process for every request can involve a lot of processes

Strengths:

- Simplicity
- Great isolation between requests and if there is a problem in the code that handles one of these web requests, then the problem can die or seg fault.
- Parent process wouldn't be affected by any sort of errors or segmentation faults.

Weaknesses:

- Pretty heavyweight.

- Each time you fork, the more memory you will use.
- Depending on how complex the code is, this could be rather large.
- In addition to the intuition, when you fork the process, there is setup and teardown.
- If it was reading out of a database, each time you fork, it would take time to do that sort of thing.
- What if load just keeps rising and rising?
- The number of processes will exponentially increase and cease to be responsive.

HTTP Servers - Proces Pool

- Uses a process pool and sets up these processes at the beginning so that we don't wait all the time.
- The startup costs get done at the beginning and get amortized over time.
- Sets up previously all the environment variables before spawning these processes.
- The children are responsible for accepting incoming connections and using shared memory to coordinate.
- Parent process watches the level of business of its children and controls the spawning.

Strengths:

- Many of the same strengths as the previous system
- Great isolation between requests. Children die after M requests to avoid memory leaks.
- If you have a memory leak, it won't be a problem anymore.
- Performance under high load will just stay at a fixed number.
- This technique doesn't require the developer to deal with multiple threads.

Weaknesses:

- More complex system since processes are starting and restarting.
- A decently good way to build an HTTP server but it has some downsides.

HTTP Servers - Thread per request

- Create new thread and while we can still read it
- Read the request
- Process the request
- Write the response
- Each thread can modify state that will be seen by other threads.
- That is the big weakness
- Thread safety is tough.

Strengths:

- Fairly simple

Weaknesses:

- Most code, you frequently didn't write so writing thread safe code may be more difficult than it sounds.
- Pushing thread-safety on the application developer isn't an ideal scenario.

HTTP Servers - Process/Thread Pool

- Do they want to have many processes or less?

Strengths:

- Tune between memory consumption and isolation

Weaknesses:

- Push thread-safety on the developer
- Developer can make mistakes

HTTP Servers

- C10K - proposed in early 2000s.
- This thought caused the creation of the nginx web server

- Assume you have a 1 GHz machine with 2 GB of RAM and a gigabit Ethernet card, can we support 10,000 simultaneous connections?
- This seems like it should work but why doesn't it work?
- It turns out to be difficult in practices but it is possible.
- Why are we less than the sum of our parts?

Let's say we have 10K connections. Each is doing this:

Read from network socket
Parse the request
Open the correct like
…

You have a series of system calls and it keeps Waiting
Most of the time, this process is around and we want to schedule something.

- All this concurrency slows down all the processes.
- If I go to read a file and there is no data available, the blocking approach is to wait until data can be given me to me and the process will be blocked.
- Non-blocking will never block or wait for something to progress

asynchronous:

- select(): Let me know when any of these file descriptors can be read to.
- epoll_*(): If this list of file descriptors is very large, we may not want it passed back in forth so keep the FDs in kernel space.

HTTP Servers - Event Driven
loop forever:
    select operations and block until one of this list is ready
    - After this method returns, go through each file descriptor in the list and go through a handler on this.
    - Each handler needs to execute very quickly.
    - It is never going to wait for IO and it will only do CPU operations
    - some_handler figures out the user wants a certain image off disk
    - It cannot do any IO because the handler can only do CPU operations.
    - Add those file descriptors

Q. What if *some_handler* is doing a lot of computation?
A. If any of the processes takes a long time, it gets starved and no one else does any sort of activity.

- Break it off into a separate process and interact with it using traditional IO mechanisms

List of well known example
- nginx
- OpenResty
- node.js (JavaScript)
- Node has a reputation of being able to handle a lot of clients.
- Pretty much in any language, there is a framework you can use to do Event Driven systems.
- These all have a high degree of concurrency and they all use the Event Driven systems pattern.

Strengths:
- High performance under high load
- Don't need to worry about threads.
- Predictable performance under high load.

Weaknesses:
- Very poor isolation
- They can affect other threads
- You have to worry about if one request takes too long, then it can have a huge impact on all your other clients.
- Fewer extensions and harder since many libraries are not written with the goal in mind of avoiding blocking IO

Code is dominated by callbacks:
- EM is event machine
- These braces enclose a block of code
- Each function is getting a request and then you define different blocks
- Inside this block, we have a second request that has its own blocks

- It can be complex and the developer will set some state and get it to another state elsewhere.
- You end up with systems that can have bugs and defects and it can be catastrophic for systems and availability.

Application Servers

- We have these advantages of breaking things out into separate processes
- Application logic will be very dynamic, and pool of application processes will get destroyed and deployed at all times.
- HTTP server will emphasize performance a great deal so a language like C is a fantastic language to write HTTP servers.
- You could write fast and efficient code, but you cannot write code fast.
- Security concerns are easier: HTTP server can shield the app server from some things.
- It is vulnerable to things that the HTTP server will handle.
- Startup/setup costs can be amortized if the app server is running continuously.
- Response goes back to the HTTP server and there are few different ways to manage this application.

CGI - create a new process and set ENV variables

- Only works while spawning a process.

FastCGI, SCGI - allow you to have a persistent process and communication over the network

HTTP - HTTP server accepting requests and forwards requests to the backend server using HTTP

- Your application server can speak HTTP and the reason is that it is logically correct and can communicate by HTTP
- Could be a very slow HTTP server

Many of the same concurrency issues haven't gone away:

- How should we approach this?
- Threads? Processes? Vented?
- Today, we will look at a quantitative look at a few of these approaches.
- One of the things is that you should experiment what will be faster and slower yourself.
- Trying out different servers and different configurations.
- Inform your intuition about what is faster or slower.
- You will be doing this same sort of evaluation.
- Done by load testing tool
- Ask Mutz questions about the graphs!
- It won't necessarily make a lot sense, but it takes practice.

The Demo App

- In the class GitHub, there is a demo app, and this is intended to show off a lot of concepts introduced in this class.

- If you have questions getting things working, look at the demo app and implement a lot of the concepts in this class.
- Written by Mutz and his friend.
- Different git branches on the demo app
- Think of it like a lightweight reddit application
- Multiple communities
- General areas and each community has many links and each submitted link has a tree of comments
- Use the demo app to do some performance testing and demonstrate some concepts in this class

Load-Testing tool

- These are the steps that each of the simulated agents goes through

0. Going to the homepage
0. Waiting for up to 2 seconds
0. Requests a form to create a new community
0. Waits up to 2 seconds
0. Submits the new community…

- Try out a few different operations and that is all we will do
- Simulated users are waiting up to 10 seconds and these take some time as well

The users stay around for many requests and the application as it is built is running against an unoptimized version of the application.

- The first version will be a very slow application and they are unoptimized.
- The tests are conducted on an M3-Medium instance.
- Tests use Puma CF and here is a link to this cloud formation template
- Has some minimal optimizations but not very many.

The Demo App

- Graphs
- Two axes
- Response time and time progression
- x-axis: as time is progressing, we are moving to the right here on this axis.
- y-axis: the amount of time it takes to respond to requests.
- Because this is our load progression, we will have more and more users.
- As a result, we will tend to see that the response time of the web server will be good for a while and it reaches a point where it cannot satisfy requests for a while.
- Instructive moment where things start to go back for the server.

There are also graphs showing the creation of new agents in the system against the number of agents being destroyed

- The beginning has lines that stay together and users are created and destroyed at the same rate.

- Users start to be created more and more because the simulator creates more users but the number of users being destroyed goes down.

Green line quickly falls to 0 and this is the point at which they diverge.
This is like 2 seconds here and at about 60 seconds in, it got terrible.

- One process can handle a new user arriving every second.
- Afterwards, it couldn't handle more load.

The Demo App - 4/1

- System is functioning for a much longer period of time and it starts to fall apart up until 2 users per second.
- If you look back here, that corresponds to steps 3. and 4.
- When you deploy this, we can see that quantitatively, we can handle far more load.
- You are just choosing a different degree of concurrency and you can support far more users.

The Demo App - 16/1

- With 16 worker processes, we have a small improvement of 4 but we have diminishing returns.
- 4 threads deliver much better performance than 4 processes (for the demo app)
- The final presentation will include a lot of these graphs
- Here, we can go up until 240 seconds.

The Demo App - 1/32

- 32 threads - perhaps there was insufficient memory for 32 processes

Application Servers

- All the previous slides were using JRuby
- The C implementation would be slow because of global interpreter lock
- In a given process, only one line can be executed at a given time
- If you have N processes, you can have a few locks
- When you ran these tests, you will see how slowly MRI works with threads.

The Demo App - MRI 1/64

- This was very high performance and we had to do a bunch of digging to see why it is high performance.
- The lock would show that database operations would fight with one lock.
- If you take an empirical and data-driven approach, the database driver was specifically designed against this lock in the interpreter
- You should take a very quantitative approach to your application, so deploy and test it.

Application Servers

- Webrick
- Small and can be deployed quickly
- Very little build time and generally used for development
- If you are having a bug, it is possible it would work on Webrick and not other things.
- When using Puma, choose your virtual machine.
- Choose either Java based VM or C based VM
- Threads worked better in Java but memory consumption is worse.
- Advanced students can install other application servers like Unicorn, Thin, Mongrel (?)

Q. Is there some middle ground for the processes?
A. Completely up to you for choosing what the users do.

- About half of the operations were write operations and the other half will be read operations
- It is that type of user profile that will affect your numbers.
- Interesting results will be good.
- Different usage patterns will affect these patterns.

Application Servers
Fusion Passenger

- Can work with nginx or Apache
- Takes information and uses that to fine-tune and control the pool for workers.
- It has a simple mechanism that you don't have to worry about tuning yourself.
- This feature that is very desirable is that the fusion passenger knows when ruby/rails is loaded.
- Starts the Ruby interpreter and then it will fork the process after.
- It takes the process and copies it over.
- There is a specific thing that this does on Linux that is goes. Copy-on-write semantics.
- If I have a process that is 200 MB and I make a copy of it, it won't copy the whole address space.
- When it writes to various parts of the process, it will retain pointers to parts that don't diverge.
- Because Ruby and Rails doesn't change a great deal, it will be much more memory efficient.

Puma

- Application server originally designed for Rubinius
- Ruby interpreter that is written in Ruby
- Works well with JRuby
- Can have a high number of threads.

Making your code thread safe isn't always obvious.

- Calls method transact with a different method of transact that calls the instances
- ||= means assignment if nil
- Assign get credentials if it is nil
- It has threading problems because this originally wasn't a class method
- These were originally stored on a per-instance method
- Classes persist between threads so if you have multiple threads and save state, then the credentials for one request will get shared to another request.

app/models/order.rb

```
class Order
    belongs_to : user, klass: User
end
```

- The reason is because you aren't ever requiring the User class and you rely on Ruby's autoload functionality.
- This works if you have a single thread of control, but there are subtle thread safety problems
- If you run this line of code, you will never tell the code where to require the user
- Asking your user to figure out thread safety on his own can be difficult for the software engineer.

W 2 R Lec 10-6-16
Vertical Scaling

- T and M series: Good mix of memory and CPU
- The T series is an instance that we should probably NOT be using for load-testing.
- Uses notions of budget, so it is unpredictable for resources.
- C series: emphasizes a fast CPU
- R series: emphasizes memory
- Does my application really respond to memory scarcity
- Look at the demo app and look at the effect of adding concurrent processes on system performance.
- What if you increase the size of your server?

M Series is balanced so it has two virtual CPUs and SSD storage

- This would cost someone about $100 a month if they run these servers
- The response time is very fast at first and then it sort of falls over.
- It failed to keep up with the load progression we were throwing at it.
- It responds well with 6 users per second and it kind of failed between 6 new users but second, but fails with 10

M3 X-Large has 4 CPUs and it goes about 360 seconds in the text.

- The relative benefit goes from 6 to 10.

- We are paying about $200 more per month.

C3 4X-Large Instance

- You are burning through money and marginally, you handle more and more users.
- There is a limit beyond what you can progress.

Load Balancing

- If I want many servers to appear to the outside world, then it needs to support this sort of behavior.
- If the client goes and writes to that server and later goes to read, they should see the changes.
- In order for land balancing to work, servers have to be stateless.
- These servers are stateless, so when someone comes in, it will get persisted in the database.

- You want clients to be requesting and interacting with your servers
- There should be no difference between these servers
- The users shouldn't have to know that there are many servers.
- Hide them behind one IP address
- For a named server, resolve it to an IP address
- This IP address might be the same, but it will then dispatch to different servers
- What IP does it point to?
- It produces multiple answers and this is a valid technique that is used.

Load Balancing on the Web
Idea #1: HTTP Redirects

- 300 response codes that indicate redirections
- This would always go to the same server and either issues to a 301 or 302 to redirect users
- Send back 301 or 302 and tell the client where to go.

% nc www.domain.com 80

Strengths:

- Simple
- You have a ton of control over where people end up.
- This server can talk to other servers and can tell who is more lightly loaded or heavily loaded
- Location independent
- These servers are behind a switch and these really could be anywhere.

Weaknesses:

- Visible to users in the browser bar
- www has a ton of load

Idea #2: Round Robin DNS

- If DNS server is always returning the same value, then everyone hammers that one server.
- Browser could cache it or other DNS servers can cache it.
- Everything we have just mentioned is a bad thing because the DNS server should be able to load balance in an intelligent way.
- Caching a big reason why this thing can be a problem!
- It is possible for having low traffic time and if we are going to return at peak time, then maybe we want to take our servers down in the evening
- Less control over load-balancing
- You don't want to just be taking these servers down

Strengths:

- Easy
- Cheap
- Simple

Weaknesses

- Less control over balancing

Most common ways in industry

Idea #3: Load Balancing Switch

- Make packet rewriting happen fast
- The expensive hardware used has to be in the same place
- If we want to deal with HTTPS traffic, this result will struggle unless it can decrypt HTTPS traffic

Strengths:

- Works fine with HTTPS
- If you give out 4 or 5 answers, then it is harder to remove things from the list.
- The next one struggles in particular.

Idea #4: Load Balancing

- Has an HTTP connection with one of your back servers
- Make your request and look at the request and decide which server to send it to.
- It should do an HTTP connection to somewhere else
- The previous solution was just using TCP connections and it didn't know what was going on with those TCP connections

Load Balancing on the Web

- This is the last method and we know because when we switch, we establish a connection with the server and the contact with the backend server handles earlier in the prompt.

- Watch which servers have the most connections to them and then direct these requests to those that have fewer connections
- Measure the response time going back and forth
- Whatever server has been responding fastest, send traffic over there
- Minimize bandwidth per server
- Based on URI (resources)
- For some reason, you want to send your traffic to two different types of servers in general.

- When you use the ELB in class, it uses round robin sequence with cookie-based stickiness

Connection Pooling

- Reuse TCP connections on the backend
- The total amount of time setting up new connections tends to be really fast because it is on the backend.

- You can use just a traditional PC that can do all this.
- If we are going to be using an HTTP server as a load balancer, then you use these as a layer

Amazon Elastic Load Balancer

- $0.025 per hour plus $0.008 per GB processed
- Works with EC2 autoscaling
- Supports SSL termination
- Can put private keys with AWS services
- Availability Zones and regions
- In AWS, there are all these geographic regions and there is an idea that any sort of failures that occur will be uncorrelated between availability zones.

Horizontal Scaling

- In the 12th minute, there will be a lot of new users arriving.
- These are new users that begin their journey every second.
- We have a database deployed that runs MySQL and we have a single application server initially.
- Load balancer and an application server with a database.
- Since we only had a single application server, it would be silly to have a load balancer.

1 M3 Large Instance App Server

- Pay for the database in this case.
- Handles about two new users per second but this is still slower than our fastest vertically scaled instance.
- A lot cheaper though!

- Each of these should interact with the database to store state and what should happen here in terms of performance.

For Next Time:
- Email Mutz the team name,
- Each person's name, email, GitHub account name
- Read/do chapters through 14 in Hartl's Rails book by next Tuesday

SoundByte -> Bitten marshmallow

W 4 T Lec 10-18-16
- We are going to be introducing material and we are going to be looking into the NoSQL data layer.
- Concurrency control and some query analysis

Motivation
- We are looking for the data layer.
- Understand the importance of relational databases and architecting scalable internet services.

All the changes will see the changes on the server because it gets persisted down to a data layer

Q. Why do we separate out the concerns of the application layer from the data layer?
A. Optimize them separately. One is a stateful system and the other is a stateless system.

Separate the needs and the concerns of the application layer, so application developers do NOT have to worry about those problems.
- Data durability -> if you are using cloud service or SaaS service, then the data won't be gone.
- Data isn't going to disappear and will be saved.
- It isn't always easy to make sure this is okay.

Instead of keeping your state in this database, then you have to figure out how to scale out the state.

The needs of these two systems has application servers changing very frequently.
- Data layer should NOT be deployed regularly. We want it to be stable.
- Data layer will push off these hard problems.
- If we have these as part of the same system, this database has to do far less.

A Stable Data Layer - Transactions
- Transaction is a way of being formal about when data changes are seen to other users in the system.

- Data does not need to be changed necessarily immediately.
- Want the system to have high availability.

Relational Databases

- They give you much more
- More versatile
- If you aren't sure what your system needs to do before you do it, pick a relational database.
- Like a Swiss army knife
- It does have a limited ability to scale horizontally
- You cannot just continue to throw hardware at the problem

Non-relational Databases

- Have a workload that is too large for a relational database to satisfy.
- Does a certain number of operations successfully so it can specialize

Initial starting point should be a relational database. If you know you need a massive number of user requests, you might want a non-relational database.

- You can have a relational database that powers most of your application, or you can apply a NoSQL database if you want.

Database Transactions

- T: R(X), R(Y), W(X), Commit
- First X was read, then Y was read, then X was written to, then we committed to the database

ACID properties in a database:

- Atomicity
- Everything happens collectively or not at all.
- Consistency
- Once you put data into the database, it should not change when you check back later
- Isolation
- A series of transactions running at the same time means that one transaction should not see the intermediate results of other transactions
- Durability
- Once data has been written to the system, it won't be lost.
- Journaling systems provide atomicity and durability

Consistency and isolation are provided by locking mechanisms

- Acquire certain locks before you do that.
- Operating Systems
- Learn how to manage concurrency for multiple threads inside the same process.
- Manage the data that it manages.

- No help with side effects
- If you open a transaction and you want it to be atomic, your relational database should be able to handle that.

Schedule (or “history”):

- Time passing on the left and time ends on the right.
- In the context of a web application, we can have an application server on the left and right.
- Read value of X, writing value of X, committing for the transaction.

Aborting is canceling the transaction so it goes away.

- The first transaction could have read incorrect data
- T2 ultimately does NOT want to be affecting anything from T1
- T1 could read the version of X and that read version could affect what was written to Y
- **Dirty read problem**

Serial schedule

- Two different schedules are **conflict equivalent** if they involve the same actions on the same transactions
- T1’s actions all happen before T2’s actions or T1’s actions all happen after T2’s actions

A schedule S is **conflict serializable** if

- S is **conflict equivalent** to some serial schedule
- The way all the conflicts are resolved is in a way that leads to a serial schedule
- You want it such that the database leads to a **conflict serializable** schedule
- Transaction will only commit after all the data that has been read has been committed.

Locks

- We can control the access to various at a elements by having locks and making sure that each piece of software can acquire certain sets of locks

Two types of locks:

- Write-lock
- Exclusive lock
- Read-lock
- Shared lock
- Any number of other people can acquire shared locks on that.

Two-Phase Locking

- This can end up in cascading aborts

- Transaction 1 reads a value at A and time passes and it decides to abort the transaction.
- Transaction 2 reads the value of A at this point in time and it releases the read lock.
- The database knows that there has been dirty data and it needs to abort this transaction as a result.
- We can end up with these cascading aborts using two phase locking.

Strong Strict Two-Phase Locking

- Release your locks all at once
- Each transaction must obtain a S (shared) lock on object before reading.
- Avoids these cascading aborts.

In practice, when you are building your Rails applications, you want to use two types of concurrency control:

- Optimistic
- Waits for concurrency problems and blows them up when necessary
- Pessimistic
- Prevents problems before they start

Optimistic locking in Rails

- Add a column to your table called *lock_version* and this version gets stored in memory.
- When you persist it back into the database, ActiveRecord wants to check that it is the same as when it writes it to the database.
- Active record object is using a stale version of the row.
- This locking technique is an application-level construct

Optimistic looking

- Strengths
- Predictable
- Lightweight
- Weaknesses
- Sometimes the errors will throw errors rather than slowing down
- Sometimes you can use re-tries, but that is a silver bullet because you don't want people hammering on the database
- If you have some operations in the system that aren't very important, you can potentially have retries.

Pessimistic looking

- The existence of these will slow down and block the instances of these things
- Unpredictable performance and these users will see their requests timing out and potentially taking a long time.

Deadlock

- Lock is acquired on order 7 and the order of lock acquisition occurs in another order.
- If your locks are acquired in different order, we can end up with deadlock in the system.
- It will probably detect that there is a deadlock and abort these transitions.

Concurrency Control in Rails

- Consequences of being wrong are somewhat high.

Why would you want to use optimistic locking in this example?

- Not mission critical data for likes on Facebook.

Query Analysis

- Demo app from “database_optimizations”
- If we are having performance problems, we will change higher performance on a system

Rails

- When it goes through the loop, we only go one item at a time.
- When you go and get your submission, you need to give Rails these hints and this tells Rails that I will be using these things.
- Pre-fetch this data in advance.

Rails goes and gets all submissions and gets all comments and where ‘submission_id’ is in a first statement.

- Not all the data that is returned here is important, but we are going to dig into these that are important for performance.

Query Analysis

- If you see some of these values with slow queries, this is an area of optimization
- Look at the estimate of rows to be examined

select_type

- The type of select that MySQL will perform
- A subquery is where you are doing something like joining a table into another table
- MySQL can handle this well if it is handling the inner SELECT statement once and it needs to re-execute that SELECT statement for every value of the JOIN.

possible_keys & key

- Does a massive scan as a result and you can give hints to MySQL if you don’t like the way it is constructing its query plan.

rows

- Estimate of how many rows need to be read.

- The larger this number is, the worse it can be.
- If MySQL has to read a bunch of rows, it can take a very long time.

Database index

- Fast, compact structure for identifying row locations
- B-trees that allow database to quickly find where values are
- If you can tell the database you want rows returned between 5 and 50, it can easily eliminate large collections of that index tree.

Query Analysis - Indices

- There are times when reducing the amount of memory it takes to store a value will lead to a big speedup.
- Less common performance optimizations, but it is possible to keep this relevant to our project.
- Instead of storing an integer value in a big int, you can store it as a tiny int.
- Store it in fewer numbers of bits.
- Instead of having a text-base value, you can use integer
- We aren't talking about the actual space on the disk it would take up, but rather the amount of spam in memory that this would take
- Change the way you are writing code from writing raw SQL versus using ActiveRecord
- Identify this join is a problem and this would identify that this join is a problem.
- By modifying, you move the 1600 to the other side and do some alright

W 5 W Lec 10-27-16

- The way to test theories empirically is if you have a ton of users, but in the context of this class, we don't have tens of millions of users, so we will be using load testing.
- We are referring to a pretty simple idea.
- If this is our technology stack and we normally have a web browser, we will be creating many simulated web browsers the exercise the full stack of technologies here
- Deploy some software on the server and the name of this is called **Tsung**
- Acts just like a web browser would but this piece of software is designed to be very high performance.

What should we observe?

- Response times
- The length of time between when a request is sent and received.
- Generally, you have a system responding fairly quickly and the response time generally stays about the same
- Server is on fire - no longer effectively serving requests
- Error rates

- If I write a load testing script, in general, when I issue requests to my server, I will get a response code 200 OK
- Sometimes, I will get 300 codes, which is a good thing, so it is fine for your website to redirect you.
- 500 is bad because it is an internal server error
- 400 is probably bad because you expect to be able to go to a page and the page doesn't exist.
- When the system is functioning well we will see 200 or 300 status codes.
- Are our synthetic users able to finish their tasks?
- Look at the rate at which new users are simulated in the system.

Some observations:

- When we build a load testing setup, we get a mixture of read and write operations.
- Demo app has these characteristics and he creates a new community.
- Why should our load testing script have a mixture of reads and writes?
- Helps us test all the different features i.e. cache invalidation via writes.
- If we only had READ operations, your system would respond perfectly and my system is super fast because if we think of something like the demo app, it won't be a useful website if no one ever wrote data to it.
- We also don't want users to have the same habits
- Users were all doing the same thing in the demo app, but in reality, users have very different habits.
- You could have admins who use the system in a very different way than someone who is merely a visitor to the site
- When we write a load testing script, we don't want to issue this HTTP request and repeat it over and over.
- We want to capture the output of previous steps and apply it to future steps.
- If you are submitting forms or creating new data elements, your web application will respond by thanking you for creating this thing, and you need to add 55 to the URL.
- A user will need to know what happened in the previous step to be successful.

High Performance

- apache bench
- httperf
- Both are written in C and they can generate a lot of load but it is very simple
- httperf has no notion of incorporating what happened in previous steps to subsequent steps.

Rich Feature Sets

- Funkload

- Written in Python and allows you to write load tests in Python and it is very powerful. You can include arbitrary Python libraries.
- It is weak in the sense that it struggles to generate a lot of load

Tsung combines high performance and rich feature sets
- Tax even our largest deployments
- It can be confusing to use at times and the graphs it generates is pretty ugly.

- It is a challenge to generate enough traffic to tax this system to find out where it's breaking point is.
- Trying to run Python scripts in parallel will only get you up to hundreds of concurrent requests.
- Funkload allows you to specify your load testing in a very expressive ay but it is slow.

Tsung
- This tool does not make you choose between performance and flexibility
- Funkload would require a fleet of funk load testing servers
- Deploying dozens of servers to load test our system!
- It uses a toolchain and virtual model that is very scalable.
- Some people use this toolchain to build this web application.
- T-series does things in bursts so you want to do it quickly (more bang for your buck)

- The user that is being simulated will do many different things, so the idea behind many different sessions is that we can simulate many different behaviors

Define all of these increasing amounts of load progression and we can uncomment them if we need them.
- It can handle a massive number of users arriving every second.

thinktime - wait some random amount of time so that real users do have pauses in their behavior

setdynvars - create a random number and create new user in order to generate unique email addresses or submission names

This load testing tool lets us see the output of previous requests.
- This is the contents of the POST request, and this can sometimes be unwieldy to write.
- When you see %%_varname, this is how the load tester tool does interpolation

You can see the body at the bottom and this is the actual text of what we were sending to the server.

- Many of these things we don't really need
- You can simplify this a bit and this is a great starting point of how to submit to these forms.

It is very good to be creating the data you are creating in your user scripts.

- This is because it can start to build up and slow down your system.
- Let's say I have 1000 communities in my system and I don't clean up, it pollutes the system and later tests will run much more slowly.
- Whatever test runs last will run the slowest.

W 5 Dis 10-28-16
Gem paper clip -> Example of this in the demo app

- scalableinternetservices/demo
- file_attachments branch
- The most recent couple relate to file attachments and the other show you how to load test file attachments
- To get file attachments working, in development mode, we need our AWS key
- If not, we need access keys to do file attachment and this allows us to do our file attachment and we need to figure out how to stream the music on the website

W 6 T Lec 11-1-16

- By now, we should have significant functionality (RIP)
- By now, we should have demos within our group and it isn't something to get too stressed about, and each team will demo in succession and demo it to the whole class.
- Basically, you stand up and connect your laptop to the screen and show what it does.

Using tsung for load testing:

- tsung -n - f simple.xml start

Turn off CSRF protection

- Focus on other things in the class and comment out the "protect_from_forgery with: :exception"

If you want to go into industry, this is valuable stuff to know, so don't worry about directly building these new products.

- How many basic functions in society are mediated by Internet systems
- Pull out a mobile app to deliver something to my house.
- If I want to save important documents, these can be stored electronically.

As more functions move to Internet systems, it is more important to keep those secure.

- What types of bad actors can we see in the world

- We can have a man in the middle attack
- This person can modify your traffic and the way to think about this is if you are Starbucks, there is someone who can do whatever they want with your traffic.
- You can have people in the middle who snoop on your traffic but not modify it.
- Traffic that was going out can see each other's traffic.

Privacy

- Idea that these people cannot read your traffic
- If you have communications that are sensitive, they stay private.
- If you open up the newspaper, there are a lot of counterexamples.

Web Security Basics

- HTTPS
- Designed to protect against bad middlemen and bad servers
- Bad clients are still around and aren't in much effect of HTTPS and they are affected by later attacks
- If you divide your intermediaries for those who you are trying to interact with, all of HTTP is built on TCP.
- TCP gives us some amounts of protection
- Are any of these people limited and you cannot casually insert packets saying I insert this request
- TCP has sequence numbers that get acknowledged on both sides.
- People cannot just send packets to the server and pretend to be you.

Man in the middle attack and this entity will be connecting and he can connect to the server on my behalf and make alterations here.

SSL

- Make sure you always speak with respect to security

Symmetric Cryptographic Algorithm

- Think of a cipher system and there is just one key.
- Sometimes this key looks like a password or a big blob of data
- If I take the message I want to communciate and I take the key and encrypt the message with that key, I will get the cipher text.

Asymmetric Cryptography

- Take a piece of plaintext data and "signing" it with your private key
- RSA, DSA, Diffie-Hellman*
- Diffie-Hellman is more of a key exchange mechanism

You have to have both sides aware of a key, but you need to pass it over unencrypted and people can still see and do everything.

HTTPS

- We want this to work where there are no *a priori* shared secrets
- How do we have this person and this server create a shared key in a way that these people cannot note

- Use asymmetric cryptography in order to get a shared key
- Both sides generate a key pair with a client key1 and client key2
- When they send randomness, they encrypt it to the other sides private key

Man in the middle attack
- Present public key to the server as the client's key
- Present his public key to the client as the server's key
- Establish a shared session key

HTTPS - Certificates
- Reduce the number of people the clients need to trust.
- Certificating authorities sign certificates indicating that the web application is in fact the right key.
- In this example, we have Alice, Bob, and Charlie
- Alice is the client, Charlie is the server, and Bob is the CA

"Root" CAs vs "intermediate" CAs
- This person is neither Alice, Bob, nor Charlie
- CAs are ways that we can build up trust that a server has the public key that it claims to have.
- They only know a small list of certificating authorities.

Symmetric cryptography is far faster and cheaper, but obviously less safe.
- TLS is HTTPS and allows for combinations of different cipher suites to be used.
- Pretty healthy over time, and the types of ways of encrypting things has been able to improve over time.

HTTPS - SSL Handshake
- Each of these represents a roundtrip between client-server
- This is going to be a slowdown and this could be slow performance wise.
- Client and server first do a three-way handshake
- Then, the client says Hello embedded with some randomness
- Server responds with a hello and some more randomness
- It adds more randomness and creates the session key
- Encrypts it with the server's public key and they send it back to start communication.

Only person who should have the private key is Wells Fargo
- A lot of roundtrips so in the real world, we care about performance and we want to walk across campus if the network is going in or out.

If we can cut down on the round trips, this would be a big win for us.

- The client and the server exchange enough information to reestablish information.

Certificates can be revoked.

- We have an offline mechanism (certificate revocation list): Print out a list of k elements
- Browsers use a combination of the two and you have this other connection you need to establish
- CA needs a high reliability system to tell if something has been revoked or not.

We can design specific hardware for load-balancing and converting it to FPGA

- If you are only encrypting it 99% of the way, it isn't.

SQL Injection

- Malicious user gets a database to execute a bad statement
- How do we mitigate this?
- Writing code that looks like this should not be done.
- You are using data sent as user input, so you need to be extremely careful and not do this.
- There will be something we didn't think of and we might end up with a security problem

We would have one bad client trying to attack a victim client

- Cross-SIte Scripting
- Javascript execute will have the same permissions that it served up from
- One of the earliest revisions of the demo app
- There is a title here on the submission and instead of actual text, we have some scripting
- When input is received, it goes into the database.
- Rails keeps track if data has been sanitized to the user.
- We have not made sure the string is safe to the user.
- Automatically convert it to text where it converts HTML tags to things that will display as this text.

SQL Injection & XSS prevention

- Continuous integration
- Tarantula is an automated tester
- Fuzzing is sending inputs that are likely to cause problems
- Send different configurations of JavaScript back to you
- Crawling and something that goes from page to page.
- Deploys your application and tries to get SQL injection attacks and XSS attacks to work

Cross-site Request Forgery

- Victim user and a victim website wit a malicious website
- Ingredients for this attack is to get the victim user to visit this malicious website and this triggers an interaction with another website.

Be sure that a page is generated by me

- Cryptographically sign something to get this to work
- Make sure the GET requests don't have side effects

The best way to load test your application is to comment this code out so you don't have to deal with these tokens

- If you were deploying something to production, you would want to turn that on to protect your users

When to use a Firewall

- Scalability Rules
- Failed firewalls are the #2 driver of site downtime after failed databases

W 7 T Lec 11-8-16

- No lab this week
- No lecture on Thursday (Snapchat) :(

Riot Games guest speaker (Sean Maloney)

- Big data engineer
- Student in this class 4 years ago
- Intern at Appfolio

Lead developer on Riot's ETL tools

- UCSB grad student

How do they handle LoL users throughout the world.

- Coordinate and strategize and work to the other corner of the game.
- Destroy the other side and progress by making money, killing opponents, and destroy the enemy team's Nexus.

This data is a little outdated, and we ingest about 50 TB daily from across the world.

- Since beta in 2009, they have collected about 26 petabytes stored in Amazon
- The reason we collect this data is to focus on being the most player focused company in the world.
- A lot of this is used for things like gameplay balance decisions i.e. one champion is too OP
- Used for toxicity detection and encouraging a healthy interaction in the game.

Actual game servers hosting League of Legends

- Not the exact client application across every different region.
- Some regions in Korea have different features enabled because they play in PC cafes.
- A lot of people in America play at home.
- If you pay $20 to pay at the PC cafe, you want to have a good experience so PC cafe owners provide extra incentives.
- Cultural differences - back 2 or 3 seasons ago, they had a skeleton mage named Carcus who was an undead person and in certain ancient customs, it is taboo to show skeletons.

Special arrangements moving great distances and a lot of variety within that data.

- We have game servers located throughout the world.
- Why do we do this?
- Latency issues!
- Lag in the game can impact your win rate a lot.
- It is important that the people who play our games are physically close to the servers we hope to play
- Because we have so many players, we have learned about sharding in order to break down the amount of data we have and make it more digestible.
- The infrastructure for our games is what we see for a standard website.
- Overall, we have a load balancer that directs traffic to our game servers.
- Game servers are like application servers and everyone logs in for the weekend and we can scale it back when you're done.
- Located within each data center.
- We scale out the number of application servers that are hitting those databases.
- Bottleneck becomes the database
- Challenging to scale out a traditional database system
- If we are playing League of Legends and we have to wait 10 seconds for your purchase to go through, you start to get frustrated.
- We put non a caching layer in between the primary databases.
- Any specific technology using memcache
- Keep paying Oracle for a long time for this caching mechanism
- When a player walks in, if they exist in that cache, it will be pulled from memory.
- Query the primary database.
- Sean's team is responsible for production data about chats and store purchases out of this system
- Running a query and not pulling things out of the database.
- Every query that you have some performance costs associated it, and if I am doing an intensive query, he could be potentially affecting our players.

Read slaves

- Back up the primary database and the primary reason for these read slaves is hot backup.

- Anytime the primary database goes down, we flip a switch and the hot backup database replaces it.
- Additionally, we don't want to ETL all the data because if the primary database fails, then the hot backup is primary and it affects this.
- We have a secondary read slave to pul a the data out.

We have other data sources we pull from like Twitch and Youtube

- Client card companies that help with store processes and connecting to those data stores and pulling them in as well.
- All of these databases are MySQL and various different types like FTP servers or REST API websites.
- All this data gets pulled on a repeating cycle, either hourly or daily depending on the system and stored on one location so that all the data is in the same place.

Copy the data from the MySQL database and storing it in a data warehouse in China.

- You have different clients so some situations in which the data model might be different.
- Clashing datasets in this case.

Allow applications to push data to us

- Once we get the data, we store everything in a flat file and you can think of it as a comma delimited file
- Inside of Amazon S3, we have tools that can query it and allow gamers to see gameplay balance reports as well as batch queries.
- We have visualization tools that allow them to create pretty reports for the MBAs who like pretty colors

ETL - Extract, Transform, Load

- Important term because a lot of companies do it!
- Business-way of moving data that exists from here to over here and making it a repeatable process.
- Loading them into S3 so there is a lot of open source tools and Riot built one of its own.
- They will do the ETL stuff but they have to make 20 ETLs that are exactly the same for all these different regions
- All of the analysts are saying why doesn't this work?!?!?!
- We created our own ETL solution and they wrote it in Ruby.

Friendly to an analyst who is not super technical

- Don't expect business analyst Bob to write any Python to connect to a database and change things.
- Take three identical tables that are hosted and combine them into one table that the analyst can query with a row that signifies which environment is is from

Create an ETL

- Fantasy League of Legends with fantasy LCS
- Choose professional League of Legends players and whoever has the most points wins!
- MySQL database server and pull the list of users in the 2nd column with a set of users.
- Load the data and store it into a tool called Vertical

FUETL can connect to a bunch of different endpoints and there are a couple of open source tools that work with this.

- Good model for what other companies are doing in the same space
- Craft a SQL statement and that format gets stored in the destination database.
- None of this has anything to do with regions and this is what FUETL can do what many other regions cannot do.
- It has to run against all these different servers
- From an administration perspective, we only have to maintain that everything on this page is correct.
- Schedule it and run it from here.
- Let's dig into how the code is architected a bit.
- FUETL uses a MVC-architectural design pattern with a scheduler.
- Right now, let's move data into this and all of these different tools have overlapping business rules that they need to adhere
- All these different tools are our views and one view is console and another is worker process
- Under that is a controller layer and all of this is built in Ruby but NOT Ruby on Rails
- Hybrid monster of some things on Ruby on Rails
- Break out the common elements of business requirements you want to execute.
- Helper service is responsible for checking if things connect to a database and making sure the database URL is correct.
- The task service is what the mapping is between and it does the transformation step in ETL and connects the source and the target database.
- Environment service is all he business logic that encapsulate different regions
- Below that is our model and this is how we use ActiveRecord
- All of the metadata about the tasks being run

FuETL Statistics

Q. Releases to the game servers and the data models change to a certain degree. How do we do deploys in different regions at different times and are there situations when the data model varies?

Q. Rephrased question: We are the data warehouse team and we don't get to control the application databases. We just consume from them. How do we respond when North America changes their data model and South Korea doesn't?

A. In our data warehouse, part of the reason we store everything in these flat files is because there are reduced guarantees and complaints regarding the structure of each individual row.

- Pull the data and the majority of them is stored somewhere in a map.
- Date column, environment column.
- That field doesn't exist and tell those guys "Hey you messed up and forgot to add this field"
- Byproduct is that everything in your database is stored and it is not very performant because you cannot build indices on it.
- String comparison search performance (shitty) instead of some of the optimizations you get from DynamoDB.

If you wanted, I can create 14 TB data daily from your mom's personal eBay website.

- These numbers don't really mean anything, but they are there.
- You will know whether a presenter is a better estimation of the presenter's problem space and it has to do with the infrastructure challenges they have to deal with.

FuETL Scaling

- Schedule and worker nodes
- All of this is done in Amazon EC2 and we can spin up clusters of workers over and over again ad nauseam until the source databases cannot handle it anymore.
- We haven't broken S3 but they have broken DynamoDB

Ensure that the same job doesn't execute twice and if the same job executes twice, it won't corrupt the data.

- Imagine that I am moving store data into the data warehouse and two different jobs pick that up and we might have duplication of the data in that destination.
- Two workers don't know about each other but we need to make sure to move the data.
- A very important concept involving data is building your architecture and keeping idempotency in mind.

Idempotency

- Operations will produce the same result if executed once or multiple times

Non-Idempotent: x = x * 5

- Value keeps changing

Idempotent: abs(abs(x)) = abs (X)

- Value is the same no matter how many times you run it.

In our workers, we need the ability to make these tasks idempotent and make it run in succession and we won't get duplication of data

- Here is once every data after October 25th
- Every execution after that will fail

- In the big data world, we don't have as many constraints as we do in the traditional relational database world.
- Big databases are designed to be performant and handle tons of data
- In a big data world, at least with AWS, it doesn't check for primary keys

Use idempotency when you have a bunch of processing units

- Come to terms with message queue architecture

FuETL - scheduler says worker A do this job, worker B do this job, …

- Puts it into a queue and whenever the workers are ready, they can pull from the top of the queue and get the next job to work on
- Third worker picks up ETL 3 and it is really short.
- Helps ensure that the services are utilized as much as they can be and if there is any communication issues with one particular worker, it doesn't block the rest of the workers.
- Message queues allow you to easily add additional worker nodes.
- These worker nodes come online and is there any work available for me.
- As opposed to an architecture that doesn't have message queues, it has to say "Hey scheduler, I'm online and available"
- More logic between scheduler and consumer when they are directly connected.
- Help abstract and decouple the purpose of these jobs.

No jobs in the ether if networking dies.

- If any network disconnection happens or a crash occurs in the middle of that process, the lock in that element will be released and the next person can start working on that.

Message Queues

- Amazon SQS (Simple Queue Service)
- AWS works well with this

Create delivery once guarantees and we are hosting our message queues on many different machines.

- Get a queue of data and we have this bottleneck between the scheduler and all the different workers.
- The distributed systems and SQS and you have to create an exactly once guarantee
- Majority of these tools on this slide actually have an at least once guarantee
- Your message won't get lost but you could get a situation where two different nodes are running it.

Health and monitoring your production system

- College is terrible about health and monitoring
- You don't lose $2 million/second that it is down

- If you think about those things, that is the defining bar that differentiates you from entry-level and mid-level engineer
- Think about how the people and engineer's impact things
- The service that he uses is called Kafka which is the new hotness is big data
- It has been out for a couple of years.
- Solid production ready state and it is a message queue.
- It is very fast, high throughput, and it doesn't get lost.
- It has at least once guarantee
- The revolutionary thing about Kafka is that traditional message queues were "Give me the last element of the queue"
- Adds a database commit log
- We can now give you the last 10 elements of the queue or another service can jump on and that service already read the last 3 elements of the queue
- This is why we have a log and we have a data source and every element that is stored has an offset ID (unique number that counts and grows)
- As consumers read from this queue, they don't remove the element from memory but just increment their own personal counter and look at the next one.
- Kafka - we have an always available friend app and we ingest metrics from that and proved its value.
- There is a situation in which a certificate expired in Turkey, and that certificate helped encrypt the communication channel that downloads the newest patch with all the upgrades from last season.
- Certificate failed and connections between players and the software you can download the latest updates from could not occur.
- Update the certificates and impact with different topics.
- Certificate error topics and network latency.
- All of the applications can push when these different events occur and you just log this stuff
- Previous elements are still there and add another consumer and this guy starts at one and you can replay data.
- Commit log that allows you to replay data for other things

Consumers say we don't have to process the same data more than once

- If the game that gets sent in exists in some memory collection, then do nothing. Otherwise, process the game.

Write expensive database aggregate queries of a champion from this date to this date.

- Because I am querying 3 years of data, it takes 50 minutes to get back.
- Set of consumer types that exist off of Kafka that provide aggregation.

How does Kafka let you interact with what we were talking about.

- String processing and Flink is a Google open source product.
- Increment the # of entries and add values, do division, and send it somewhere

- A lot of customizability in these processors and traditionally, you would look over the whole period of time throughout the database.

Taking care of aggregation because different consumers may have different logic.

- This is very new and in industry they are still trying to find the optimal architecture for this because they have different logic since each one does something different.

You have to wait a day to crunch numbers and figure out if the guy was an asshole.

- Create a string processor that categorizes a toxic message

SQL will query the tables and this will query Amazon S3

WASP - we don't know what all the data is and if the analyst says the data is wrong, it makes us look bad like we don't know what we are doing

- 10 engineer team trying to service a company of over 2,000 people
- Auditing service that will compare source and destination and when we are auditing, we have seen that people will make updates in previous days.

NoSQL databases are less efficient because they don't have as good indexing for aggregation.

- Made a mistake where they tried to use this tool called Vertica to support our Player Support database.
- Vertica is a column database but it is very poor at finding one record
- Tons of data but storage is cheap.
- NoSQL database is optimized for finding one column and returning things in sub seconds.
- Choose wisely!
- Scale and respond to that time of querying.

Choose the right tool for the right access pattern

- Tweak everything and when Sean was doing the project, he worked on a board game website that lets you stream video.
- Google Hangouts but you could play boardgames
- Tic-Tac-Toe but the majority of the time was spent on scaling and eke out that
- Task where Sean wanted to migrate a lot of files form one directory to another.
- Something as simple has additional complexities
- They aggregate every game for one of these champions to set the bar for performance
- Don't get angry at Sean because he didn't write the query, but he provided the services to access that data
- Player support and chat detection and that is it.

Q. Running a bunch of Kafka on EC2 instances? Opinion about a similar solution and Kinesis Firehose provides the same functionality that Kafka does and it works well with many companies but Netflix and LinkedIn didn't think it was good.

- Kafka clusters located in the game data centers and we want it written to a 100% and that extra jump to AWS may be more than we want to handle.

Q. Setting up the replays?
A. The last Sean heard about replays, they are a compressed time series of events at frame 23 and the replays are played sequentially from frame to frame. Tagged with an ID or database with a service that we request those endpoints from and the game client would see this game and send it to a REST API that will read and return a file from Amazon S3. A lot of stuff is prioritization.

W 8 T Lec 11-15-16
Matt - CTO of Splice

Splice is like the combined version of Git and Spotify

- Musicians use different tools!
- Small client that uploads in the background with all the information they have and you can see the different versions
- See the different plug-ins and get inline information and go back in time.
- As you were making music, you could never go back and everything was lost.
- Put it somewhere else and open source it somewhere else and released the entire song.
- Every single track and presets with information and modify it back to the community and get picked up by a label and become the next big star.

Plugins in music are hardware synthesizers that get visualized into software.

- This sound can be generated by your computer directly.
- Still costs a lot of money
- Serum is made here in Los Angeles, and they have been working with the creator of this plugin and they sell it each month.

Skrillex is using this and we get a lot of auditions

- We can do different searches and we could listen to all these sounds to create music
- Getting millions and millions of additions each day
- Pushing a lot of data out
- 9 - 10 TB of data

Startup advice

- Within 3 years, 92% of startups fail
- This number is pretty optimistic and we want to see the graduation from a certain stage to another stage.

- One of the easiest ways is to see how a startup makes and/or raises money
- Seed money can be from family, cousins, a rich friend, or angel investors
- In 2015, < 5% of companies were able to raise money a 2nd time

Design

- When you want to scale something, you need to know good design
- Frederick Brooks - good writer who is an amazing software designer
- Building the first IBM computer

0. The **formulation** of the conceptual constructs
0. **Implementation** in real media
0. **Interact** with the implementation

Go all the way into the user interaction with the system

- Reimplement things and design is this loop.
- Things take time, and the first project you do might not be the right one, and you have to rewrite the code many times.

- The hardest part of design is deciding what you want to design
- How do you organize your code and decide what you want to design.
- Part of what to do at the beginning is exploring what to design at the beginning.

Goodness function

- What is it improving, and why is this design better than another.
- What is it really helping?

What needs to be validated?

- Do you care about throughputs? Latency? Metrics you are monitoring?
- Do you care about how fast engineers write the code, or the response time?
- Whatever you do, you probably need to change eventually!
- Design with the idea of change!
- Some product decisions will change too.
- Encapsulate code.
- Keep the technical debt small
- Make compromises when we write code and all the time, you create technical debt.
- Things that will have a consequence in the future of how you do things.
- Example: Write an API endpoint that has a cache.
- Next time there is a bug, I created this block that is harder to read, and try to keep these things small.
- Don’t under or over engineer things.

Splice

#1: Avoid making obvious mistakes

- Understand where you stand when it comes to your project
- Are you an outlier?
- Are you something that most people do?
- SQS - queueing mechanism that can process information on the other side. RabbitMQ is an open source project that you run on your own.
- Put some work in there with workers picking up the data and processing it.
- Hosted service by Amazon where we don't have to do it.
- Rails is designed for specific things
- Rails was a backend and not a good idea to use it at Sony since it is for gaming.
- Don't roll out your solution if your solution already works.

Most important part of your stack?

- Whatever allows you to move fast without breaking too many things.
- Too much technical debt is NOT a good feeling.

Split the backend and frontend at Splice

- Backend is fully isolated while the frontend talks to the backend.

Backend

- Go a bit deeper later on
- Web API receives the request and we process parts of the information and anything too slow gets put into a queue.
- We don't want to block the request, and the async processor does its thing and goes onto a list of different jobs.
- Anything not visual is written in Go
- Mac App uses Objective-C
- Windows App uses C#
- Now, Windows/Mac App uses Go + Electron (embedded Chrome)
- Web uses Angular

- Have a server-client architecture!

benefits of a JS frontend app

- Javascript is a pretty terrible language; they use TypeScript, but JS runs well in a browser.
- You can start polluting constraints by adding information into the backend that leaks into the frontend
- Everything is API driven and we have a consistent interface
- Frontend/QA can check staging in a very safe manner
- Try to render things in a different way and load this new design and target predictions without going through the entire deployment process.
- Specialization

- People will know the frontend and the tooling that changes a lot.
- Very important that people really understand and own that domain.

Perceived performance

- Render the page server-side and then push it to a client.
- The first load can be slow and we have to evaluate all the code and make it smooth.
- Performance is good but we can have a problem with latency.

- Exciting way to build UX

GO

- APIs 50+x faster than in Ruby
- Compiles very fast and it feels like you just started a Python/Ruby script
- Better computational speed because garbage collection, but you don't have to manage your own memory.
- Memory: ~15 MB per server, which is pretty insane and it will start between 60 or 100 MEGS per process.
- You spend less time garbage collecting and the response time is much, much faster.
- You could run more servers for the same price.
- The new version of GO updated their garbage collector, and now the fastest process time is 100 ms
- Extremely easy to deploy since you don't have to worry about Ruby gems
- Unified "backend" language

ElasticSearch is a cache layer

- You can have multiple nodes that contain different parts of information, and you can start new nodes and distribute them across different nodes
- Not designed for caching, but it is a hack we made for scaling
- Lucene is the most powerful open-source search engine but it lacks certain things
- Schema less(ash) document store - **recommended to actually set the schema though!**

- Speaks JSON - we can move data before sending it back to the frontend.
- For every write you are making, you might be pulling out hundreds of thousands of tweets.
- Database is a primary place where we write and we don't have to do JOIN queries or anything
- JVM based, which requires a lot of RAM (pain in the ass)

Dashboard is like a FB feed of information and it is easier on the frontend.

- Duplicate information and keep it hot, and rebuild the search engine anytime without losing data.

CDN is a distributed cache and depending on the domain we are using, we are using ELB

- Elastic load balancer - Imagine you are going 100 requests per second and you are using the load balancer.
- We have a spike and have hundreds of thousands of people coming in.
- We cached this information and another option is to load balance and have multiple servers with people making the request.
- Load balancer is distributing the load and start a bunch of the servers.
- Two different load balances in the same area just for backup.
- Split the traffic in two to handle all the requests
- Write to the different data stores we are using.

- On the frontend, a user will get an HTML response and this page will talk to Google X. If this request is coming from a bot, crawler, or FB, instead of serving fresh content, we need to get the pre-renderer page.
- Load app.js, which is served on S3 so you don't go through the server's anymore.
- Make API calls on the other side and we can scale that.
- This simplifies a lot of things because we don't have to worry about the APIs

CI - Continuous Integration

- Run everything and to make sure it is good.
- Run up to 20-30 times a day and this doesn't go to production directly.
- Staging an environment where QA is testing, and if things are looking good, set up a new branch and make a pull request

- Like every company, they have isolated scalability issues, and this has been mostly due to data stores.

Lessons learned

- It doesn't matter what technologies you have used!
- Facebook is written in PHP, and PHP is one of the worst programming languages to use.
- Don't be arrogant and use the best technology (which is Go!)
- As long as you are a good engineer and know how to design certain things
- Know your objectives and be very explicit about it
- If I gave you a task and I need you to load-test it, we want to check certain conditions and see how things are present.

Optimize something and write some benchmarks, and most of the time, Matt is wrong, but this shows perseverance!

- If you create big blob of things, it is much harder to swap.
- Small and flexible

Docker - if you deploy to AWS, we share a machine with multiple people and you run inside a VM

- Allocated amount of resources and this is your virtual machine.
- Based on the concept of containers and is now available on different platforms
- Share resources without sharing access.
- When I deploy to CI, it is the same units thatI was using before.
- Everything gets isolated!

Q. Do you use a Docker orchestration tool?
A. We do barebones Docker and it doesn't have enough servers but they used Docker compost and their CI is the one controlling all their servers.

Q. DDOS, can't you do that in nginx?

- By the time DDOS comes to the load balancers, your engine would be dead!
- You need a really big architecture that can handle TB of data pushed to your server.
- Filter at the IP level and filter it before it comes to you.
- A lot of attacks that are crafted a certain way i.e. a firewall and router that fits your app
- Filter most of the stupid attacks!
- Be safer and Cloudflare offers CD and services all in one
- What's a big challenge in storing audio?
- What is the difference between hosting audio vs images?
- Privacy issues.
- Well-known artists using our platform and we make sure that if someone logs in, they can only see things they are allowed to see
- Uncached data is about 6-10 TB of data per day.
- People are making the requests and it comes out to the entire side.
- Monitoring and assets coming in and out.
- Nobody is really doing cloud audio and music. Inventing the tools from scratch is hard.
- People can send up to 4 GB so we don't want to blow up memory on the server.
- Most of the other problems are what you get on every single app.
- How do we handle payment processing?
- We use a bunch of different providers with hooks and asynchronous.
- There is a small niche problem of audio and once you work hard enough, you get it solved!

Q. Changing languages and changing your stack. How do you know the cost and benefit of changing?
A. When Matt started, he knew they would NOT stay with Rails, and it would be the fastest way of getting the frontend up and running and proving that they have a good project.

- Is there a risk in using that technology?

- On the frontend side, things would probably change!
- The frontend is shifting very quickly and they will change.
- Going back to effective design, why do you choose a technology and what do you get in exchange (since inevitably, there is a cost)?
- Merge the native applications on Mac and Windows, and we have frontend separated out.

- Open up new projects that we did NOT have before!
- Go through entire list of pros and cons and the engineering effectiveness and goodness you are getting.
- Recommend on pushing what you have and NOT changing due to every change you see.

Let's say you have an array of numbers and we want to sort it, but we sort on a key and we flush things and we have a concept that does it at insert time.

- Allows us to do ranking and statistics that we cannot do in Elastic Search
- Redis is used for sorted sets. It is also faster than Elastic Search
- Elastic Search lets you do cross-document search and allows you to get more rested query information.
- A lot of people using MongoDB instead of Elastic Search.

Postgres

- NOT good at replication
- Once you have multiple nodes and scaling the leaves, this becomes a problem!
- MySQL is hosted on RDS and they handle requests.
- Youtube, GitHub, etc. use MySQL and for them, it is enough.

W 10 T Lec 11-29-16

- Google wanted to understand how important performance was for Google users.
- Intentionally used delays in Google search results.
- Made their system just a little bit slower and watched what happened to users who experienced slowness.
- Slowed less than .5 seconds!
- There is a small but significant decrease in engagement.
- People were less likely to search on Google in the future.
- The black line is the shorter delay, and the y-axis is the decrease in future searches by users.

Small improvements will change user engagement.

- When you are Google and your ad revenue is tied into that, .8% of Google's ad revenue is a very large number.

- Walmart
- Found that fast and high performance page loads were correlated with conversion rate.
- conversion rate: someone becoming a customer
- Improving page load times by 1 seconds resulted in up to a 2% increase in conversion rate.

Network latency is bound by the speed of light and that isn't changing any time soon.

The user's client issues a GET request and we get it from the origin server.
- Owned by akamai, or owned by you
- When it is returned, the cache remembers it and the client gets the image.

If most of the time it doesn't have what it is looking for, it mostly slowed things down.
- High hit rate that this works.

Implementation
- Which endpoint should we send a person to?
- We have clients and users all over the nation and the circles represent the locations where our CDM providers set up servers.
- How do we actually send them there?

Akamai doesn't know your IP address at the time, but it knows the IP address of your DNS server.

- You tend to set DNS server's to 8.8.8.8 (easily remembered IP address that is a Google provided DNS server)
- If I am using 8.8.8.8, people on the East and West coasts could be using this DNS resolver.
- The load could be higher and the geographic distance between two places is NOT always the same as the network distance.

Virtualization
- Applications that are very memory are hit the hardest, but all applications are affected by some degree.
- Instruction-Level Parallelism
- Not all instructions can be executed in parallel
- There are dependencies between instructions, so you ultimately hit a limit in the degree of parallelism you can achieve.
- Some percentage of your application that can be parallelized and some that cannot.
- Heat and power are very important parameters in virtualization on the server side.
- Electricity usage is also a big deal.

Paravirtualization

- Modifying the operating system to allow virtualization to occur.
- Docker: Uses Linux LXC

Virtualization: Docker

- Uses Linux LXC, which is namespaces and Cgroups to provide virtualization
- Docker innovations were to create Docker files
- How you specify inheritance between images and changes + modifications

LXC: Namespaces + Control Groups

- You can pin a container to a specific core/CPU and make it so that some containers cannot activate or use devices.
- Control memory usage and the amount of I/O you have.
- Protect the databases from getting excessively hammering the disk.

Dockerfiles

- All these steps and instructions happen at build time.
- On my continuous integration, I can build the image and interact with that container and run my tests on it.
- When I push my image to production, when it is pulled back down, I can verify the image as correct.
- Add dependencies between these services and make it so that when you bring up web, you bring up DB first.
- Dependencies modify DNS requests inside web and if I try to resolve DB, it will return an IP of the database running container.
- I don't need to figure out at runtime what happens.

Docker orchestration

- Refers to the larger problem of figuring if you have many Docker containers, how do I connect them together so I know where they correspond to other containers.

Course Conclusion

- Needed a home to live in or go on a date?
- What would you do!
- These are timeless problems!
- We use internet services to do this everyday, and we are in an increasingly globally connected world.
- Building an internet service to scale is extremely difficult.
- Even for a simple web application like Twitter, scaling it can be very complex.
- You have had the good fortune of generating a web application that is very popular and the world agrees with you!
- You wake up one week and your performance is doubled.

- Handling parallelism and concurrency on your server
- Can you tune these numbers to get a greater degree of concurrency?

How to use load-balancing to set up a nice, horizontally scalable architecture

- Relational database can start to crumble and how to scale your database.
- Break up your application into multiple applications and deploying read-slaves.
- How to do sharding so you can apply all of these techniques.
- When those techniques are NOT enough, we talked about NoSQL datastore and what to do there.
- Content Distribution Networks (CDN) - billions of users and offload all of your static assets.

The basics of web security and client side MVC frameworks.

Gaining experience in modern web application technology

- Rails
- AWS
- Web programming

Helpful for getting a job, research, and starting a company

- Have fun!