Design and Scaling

CIS 322
Mar 2, 2017

A couple notes…

- Required pairing for assignment 9 is being dropped
  - Late export implementations are too much of a risk
  - Export format is more strictly specified
  - Should be able to test with anyone
  - I will provide the test files we will be using (hope to get those files written later today)
- Assignment 10 will still need pairing.
- I'm traveling for an interview on 3/16… Andy will probably have the lecture on 3/16

I've been concerned about assignment 9… data migration is frequently challenging and adding the dependency on other students that might be overloaded has been on my mind. I'm making changes to the assignment to address this.

I'm still concerned about assignment 10 for similar reasons… but I think I need this one to be paired.

Also, I'm interviewing which has added additional time pressure to me. Of the problems to have, this is a good one but may impact lecture on 3/7 if my flight gets delayed and will impact lecture on 3/16.

How do we find computers on the Internet?

- Internet Protocol (IP) only specifies numeric addressing
  - Each host has a unique numeric address
    - 32 bits in length
    - Some addresses are reserved (non-routable)
  - Ranges of addresses occur on the same network segment
    - Originally 8 bits per network “class”… replaced by CIDR to help alleviate address scarcity
    - Routing by sending packets up/down an overlay of address spaces
- Humans are terrible at numbers

DNS case study…

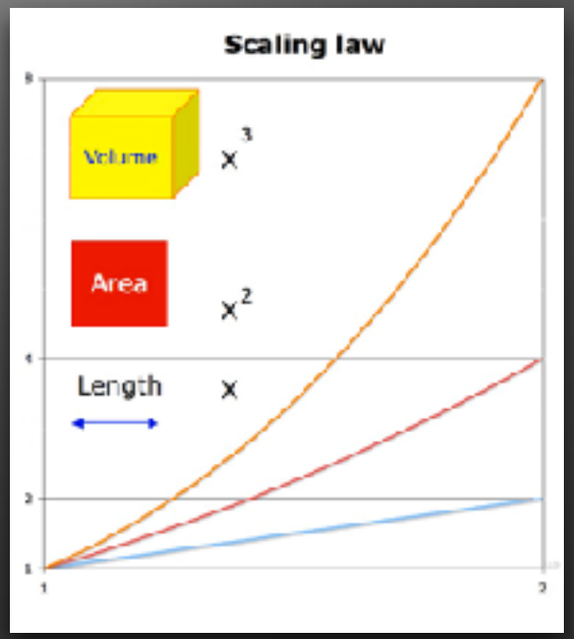
In the really early times there was a hosts file that contained all the public hosts on the internet with their IP address. It was distributed via ftp from a single well known computer.

This creates a huge bottleneck… a single IP address is a single host which needed to service all of the load. Many people needed the hosts file and would contend for the bandwidth; recall that early networks were comparatively slow.

DNS is the solution we converged on. There are some root DNS servers that are responsible for the TLDs. These redirect to DNS servers for specific domains, which may further redirect to subdomains. Caching is used to limit the amount/depth of DNS requests but introduces other problems.

Scaling

- Computation takes time and space
- At a high level:
  - Can't read data fast enough
  - Can't compute fast enough
  - Can't store enough data
  - Can't write data fast enough
- "Fixing" often shifts where the bottleneck is.
- Often solved by throwing hardware at the problem



Scaling a system is generally done in response to having detected some bottleneck somewhere, often caused by increasing the amount of input, output, or computation done by the system. Addressing the constrained resource moves the problem elsewhere in the system.

Over engineering for scale complicates system designs. Every design choice has an impact on scaling behavior; being able to scale a system over time is partly good engineering and partly luck.

Fundamentally most scaling solutions come down to throwing more hardware at the problem.

Add More Hardware

- Easier than changing the process or inventing new algorithms/hardware
- Perceived to be low risk since it is more of the same
- What to do:
  - IO - Add more IO adapters
  - Bandwidth - Add more network cabling
  - Compute - Add more processors
  - Storage - Add more memory/disk
- Dependancies/costs can cause adding more hardware to fail

A competent developer is going to cost the organization a lot of money per year. If you get $80k/yr in salary, you cost the organization roughly $160k/yr due to taxes and other overheads. That developer might take months to develop a better algorithmic solution or may never be able to deliver a better algorithmic solution.

For the same money, the business could by a lot of hardware or rent a lot of time on a cloud service. Throwing hardware at a problem is a no brainer for most organizations with money.

Of course throwing more hardware and problems doesn’t always work.

Two Is Better Than One

- Need twice as much bandwidth, use twice as many network interfaces
- Need twice as much memory, install more or bigger DIMMs
- Need twice as much disk, install another disk or two
- Need twice as much compute, install more cores
- Too much for one computer, install another one

http://luxenography.blogspot.ie/2013/12/many-hands-make-light-work.html

The basic idea is that rather than having one component do all of the work two components will balance the work across themselves. For trivially parallel problems, this will work.

More slow components can be better than fewer fast components… that is the story behind GPUs. Hundreds of slow cores are more powerful than a few really fast cores… but why not have both.
The number 3 supercomputer, Titan, has 18,688 16-core CPUs and 18,688 Nvidia K20 GPUs

More HW, More Concurrency

- Concurrency is trouble for some design components
  - Anytime something should happen exactly once
  - Anytime independently started computations modify the same data
  - Anytime an independently started computation writes data that impacts another independently started computation
- Sharing means ~~caring~~ headaches
- The shared thing is a bottleneck since access/computation generally needs to be serialized

In the general case, there are dependencies between computations. These dependencies will force ordering and waiting between computations. Coordination overheads cause diminishing returns as hardware is added.

These are super interesting problems, I would suggest taking a course on parallel algorithms to try to understand these problems and some solutions better.

Another potential problem with just throwing more hardware at the problem is failing to look at the real problem. A fibre optic cable comes with a cost and has a particular bandwidth and latency… so does a truck full of hard drives.

Let’s Design Something…

- Mockups and Process Diagrams
- Tech Stack
- Diagram some hardware
- Ultimate cat video archive?
- Twitter clone?
- Wire tapping system?
- Cloud storage system?

Let’s do a design exercise. Let’s choose something simple and try to build a good understanding of what a solution might look like. Then let’s imagine the technology we might need to implement it. How to we place/distribute that hardware initially?

After getting an initial solution together, let’s assume a bottleneck occurs somewhere and try to design around it.