CS 686: Special Topics in Big Data

Networking and Messaging

Lecture 7

Today's Agenda

- Project 1 Updates
- Networking topics in Big Data
- Message formats and serialization techniques

Today's Agenda

- **Project 1 Updates**
- Networking topics in Big Data
- Message formats and serialization techniques

Project 1 Updates

- There have been a few minor tweaks to the P1 spec
  - Most important: **Due Oct 6**
- Deliverables are posted on Canvas
  - You can submit your design documents via Canvas
- Protocol buffers and Apache Maven are now installed on the bass cluster
- Store your chunk data in /home2/<username>

A few things to think about…

- How you'll configure your chunk sizes
- File placement
  - One approach: random.nextInt()
- Replication strategy
  - HDFS has its rack abstraction; we can come up with something simpler

Today's Agenda

- Project 1 Updates
- **Networking topics in Big Data**
- Message formats and serialization techniques

Network Concerns

- For the most part, we can rely on the network to do its job and live at a higher level of abstraction
- Many networking concerns still creep up in distributed systems
  - For instance, do we use TCP or UDP?
- We still need to think about:
  - Bandwidth
  - Latency

Bandwidth

- Also known as ***throughput***
- How many bits we can push through the network
  - Trending upward over time… slowly
    - 1000 Mbps networks are common in data centers
    - 10 Gbps is gaining some traction
    - Many home internet services are still in the range of 25-100 Mbps
- If we're going to use plumbing metaphors, it's the size of the pipe

Latency

- How long it takes your bits to get from one point to another
  - Latency has been trending downward overall, but not by leaps and bounds
  - We are limited by the laws of physics here
- How long the pipe is and how fast data can travel through it
  - Some communication mediums are more prone to latency: Ethernet vs WiFi

Round Trip Times

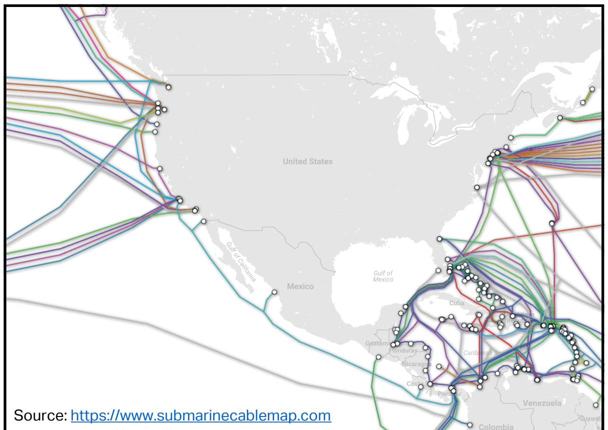
- Another factor to consider is the **round-trip time**
  - **RTTs**
- Another way of looking at latency
- Communication rarely goes one way
  - Even if you're uploading a file, you'd like to confirm that it actually made it over in one piece
- Testing RTTs: pinging a host

Ping Round Trip Times

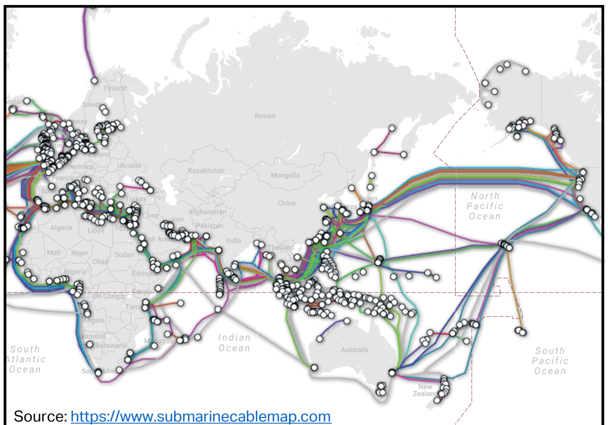
- PING **alpha.lan** (10.0.0.1): 56 data bytes
  64 bytes from 10.0.0.1: icmp_seq=0 ttl=64 **time=1.133 ms**
  (From my couch to the closet)
- PING **stargate.cs.usfca.edu** (138.202.168.21): 56 data bytes
  64 bytes from 138.202.168.21: icmp_seq=0 ttl=53 **time=15.640 ms**
  (Down the road to USF)
- PING **ruby.cs.colostate.edu** (129.82.45.204): 56 data bytes
  64 bytes from 129.82.45.204: icmp_seq=0 ttl=50 **time=70.991 ms**
  (Fort Collins, Colorado)
- PING **speedtest.shg1.linode.com** (139.162.65.37): 56 data bytes
  64 bytes from 139.162.65.37: icmp_seq=0 ttl=50 **time=124.017 ms**
  (Singapore)

Interesting: Physical Distances

- From my router to USF:
  2 miles / 3.2 km
- From USF to Colorado:
  ~950 miles / 1609 km
- From USF to Singapore:
  ~8,434 miles / 13,573 km
- https://www.submarinecablemap.com



Source: https://www.submarinecablemap.com



Source: https://www.submarinecablemap.com

Bandwidth-Delay Product (1/2)

- Link capacity (bps) * Round-trip delay time (s)
- This measures how many bits are "in flight"
  - How much data is sent before the receiver gets anything
- A network with a large bandwidth-delay product is called a **long fat network**, or **LFN** (*elephen*)
- Satellite networking has a large bandwidth-delay product

Bandwidth-Delay Product (2/2)

- Important because of its interplay with TCP
  - TCP dynamically tunes its window size
  - If the window is too small, link capacity is wasted
  - If the window is too large, the other end gets overwhelmed!
    - Queuing delays
- Thinking back to our ping example:
  - My local network (300 Mbit/sec, 1.1 ms) = **0.14 MB**
  - Singapore (50 Mbit/sec, 124.0 ms) = **0.78 MB**

To wrap up…

- Most big data applications operate on top of enough abstraction to almost forget the network exists
- This is fine until we hit a situation where one of our nodes experiences packet loss or congestion
  - Distributed computations often wait on **all** participating nodes' replies
  - You're only as fast as your slowest worker
- In the cloud, this can become a major issue
  - Where are your nodes? What is the network like?

Today's Agenda

- Project 1 Updates
- Networking topics in Big Data
- Message formats and serialization techniques

Messaging

- Previously, we talked a bit about network design
- The messages you send between components and the network design you choose are closely related
- For instance, recall our ring overlay: we can get by with just a single message type

Simple Messaging (1/3)

- If you want to send one well-defined message between components, all you need is a fixed-size buffer:
  - `byte[] buffer = new byte[25];`
- Remember: sockets are **byte** streams, *not* "message streams"
- A much more common format:

Message Size	Message Payload

Simple Messaging (2/3)

- Once you've unpacked the message payload, it can contain more fields:

Message Size	Message Payload		
	Message Type	Version	Message Data

- This allows for a layered approach:
  - Network code
  - Object creation code
  - Pass through a chain of handlers

Simple Messaging (3/3)

- If you don't need advanced features, size prefixed messages work well
- Exceptions:
  - You'd like to avoid reading the entire message before you start processing it
  - You don't even need to process the whole message (perhaps you are forwarding it somewhere else)
- Distributed systems' wire formats have a huge range of features and complexity

Serialization

- **Serialization** transforms an object, structure, or application state into a format for transmission
  - (and often storage to disk)
- Most common: **binary** formats
  - Better performance
- When you receive a serialized message, transforming it back into its original representation is called **deserialization**

Java Serialization (1/2)

- Another option is Java's built-in serialization
  - My advice: don't use it for anything but prototyping
  - Python has similar functionality in the **pickling** module
- These types of serialization are language-specific, brittle, and can lead to application errors
  - Memory leaks
  - Broken messages between versions

Java Serialization (2/2)

- Automated serialization is often not very performant
  - May produce large object graphs
  - Class members may be serialized that you don't need
- What's the big deal though? Aren't there more important things to worry about?
  - Distributed systems, whether they're storing data or processing it, have to communicate frequently
  - In some applications you'll speed ~50-70% of your CPU time serializing / deserializing messages

Alternative Approaches (1/2)

- One is Protocol Buffers, of course
- Another is manual serialization
  - In Java: **DataOutputStream**
  - Write an int for the message size, followed by an int for the message type, then a string, etc…
  - This can be extremely error prone
- Apache Hive *SerDes* require manual work but automate some of the processes

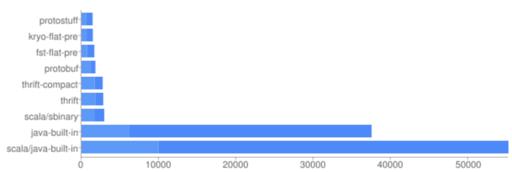
Alternative Approaches (2/2)

- Apache Thrift is quite similar to Protocol Buffers
  - Designed by Facebook
  - Lets you define your messages using a DSL, generate code for different languages, etc.
  - Also includes an remote procedure call framework
- XML ☹
- JSON
  - Not highly performant but user friendly, widely supported, and easy to debug

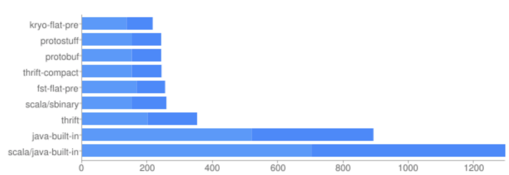
Benchmarks

- The *JVM Serializers* project by Eishay Smith makes it easy to compare serialization technologies
  - See: https://github.com/eishay/jvm-serializers
- Results shown here were collected on my laptop (more or less similar specs to a commodity server… at least back when it was new!)
  - macOS, Quad Core I7-4770HQ, 16 GB RAM

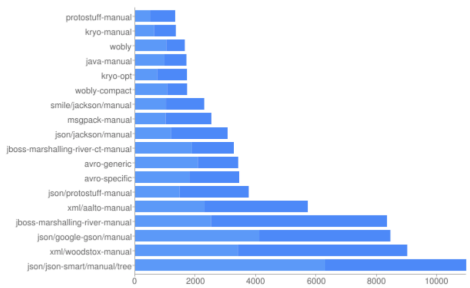
Automatic Serialization + Deserialization (ns)



Size (bytes)



Manual Optimization (ns)



9/8/17 CS 686: Big Data 31