CS 4530 March 15, 2021 - Distributed systems & Team meetings

Agenda

1. Distribute systems discussion + review
2. Team meetings + work on projects
3. Project/distributed system discussion

Distributed systems

Q: What are the nice things we can from distributed systems?

- Remove a single point of failure (fault tolerance)
- Scalability: Maybe one server can't handle load, but put together it works
- Availability: Make sure the whole system stays up, even if part goes down
- Latency: Less time between request and response
- Performance: Increase in throughput - be able to do more things at a time
  - Q: How to improve performance without going distributed?
  - "Scaling up rather than scaling out" - If you have a single server, you can beef it up, giving it more processing power to handle more load. Scaling out - get more systems, might not be that strong each

Q: What are the constraints that make it hard to get these nice things?

- "More machines, more problems" - more

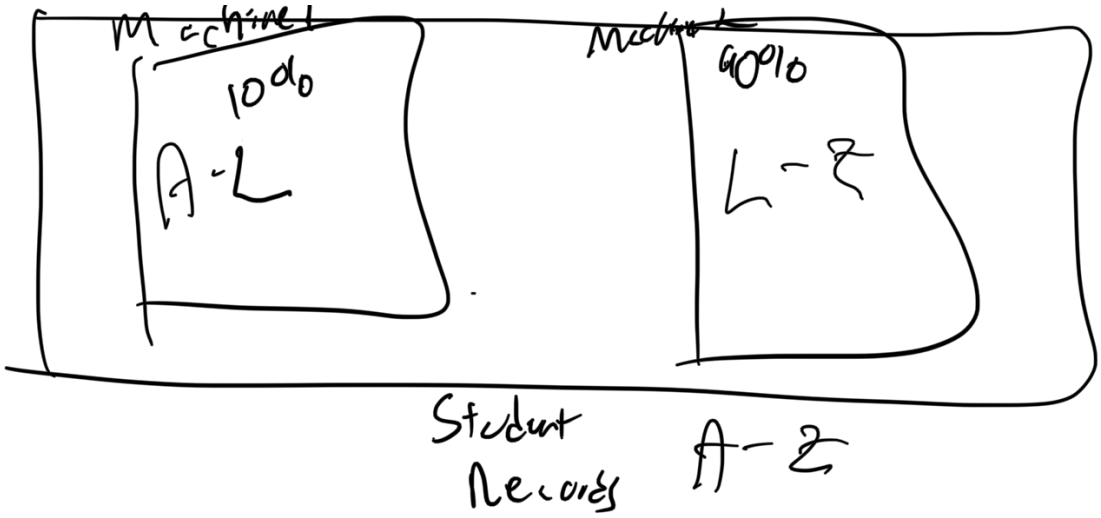
points of failure
- 8 fallacies of networks - network is not reliable. Multiple administrators - multiple opinions imposing constraints on how your system works. In general, more to maintain, more can go wrong

Q: Partitioning....? What does this mean and how does it help us achieve distributed systems goals?

- Split data up onto multiple machines
- “Chunking” or “bucketing”
- Scalability: store more data!
- Performance
  - Everything is broken up - “performance is isolated to each partition” (and won’t effect each other)
- Fault tolerance?
  - Yes - no single point of failure, if one machine goes down, can still do something, but can’t necessarily process an arbitrary request
- How to determine the split of data?
  - 1. Create a static rule, tell everyone about it [Sharding]

- Cons:
  - One server might go offline, then what do we do?
  - Have to update rules on clients as partitions change
- Pros:
  - No worry about fault tolerance/ performance of that central server keeping track of where files are

- 2. Create a central server, that central server knows where each student record/ piece of data is. [Partition]
  - Pros:
    - If we need to change the split, it's all in one place and maybe easy to do
    - Can evenly balance - just say: "put 50% in each place"
  - Cons:
    - Single point of failure
    - Additional network call required - need to go through some directory server to find your data



- Q: How does replication work + help us achieve our distributed systems goals?
  - Scalability:
    - More requests supported - can balance load across replicas
  - Performance:
    - Higher throughput, by processing requests on different machines
  - Latency:
    - Latency accumulates from the slow-processing components, maybe having more of the same component means they will respond faster
    - Replicate data closer to users
  - Availability:
    - System will keep working even if one or a few nodes fail

- Fault tolerance:
  - Tolerate more faults, assuming that they occur independently between our replicas
    - Example of independent fault (would effect just a single node):
      - Hard disk crashes, power supply burns out
    - Example of non-independent fault (would effect more):
      - Security breaches (are the services isolated - does attacking one mean both go down?)
      - ISP goes down/Data center goes offline
      -

- Q: What is the problem with replication?
  - CAP theorem - we can’t have nice things: Can’t have a system that stores data exactly like a single machine would, is distributed, and works despite network failures
    - Might do this wrong, and end up with inconsistentcies
  - Choose between using/building a system that:
    - Guarantees consistency, some number of node failure might make it unavailable (works as long as a simple majority is not crashed)
      - Zookeeper or RAFT, RabbitMQ
    - Guarantees availability, some crashes or failures might compromise consistency
      - Redis,