Hello! This is

Distributed Systems

Spring 2024

Today’s Plan

- What are distributed systems

- Why
  - Useful
  - Study

- How
  - General
  - Course specific

What
What changes

- Logic is concurrently executing on many machines

- Each of which can fail independently

Almost all of distributed systems is about reasoning about failures.
What does failure mean
- Crash (Software or hardware)
- Crash-Recover
  . . .
- Byzantine (Arbitrary Behavior)

Why

Failures appear rare.

Why do all of this?

Originally: Systems that needed to remain available
- ATC, Defence

Internet: ~1970 - to

Interactions between machines spread out geographically & owned by different entities
Large Clusters/Datacenters:

Failures for a single server are rare
MTBF 2-10 years

But at scale,

Google reports

- 1 net failure every 5 minutes
- 1000 machine failures per DC per year
- 1000+ disk failures
- ...

More recently

Cloudflare

-> 3x redundant power
Failedjon ~3 days

Azure

-> Several hours

Bottom line, consider enough machines,
something will fail

**OUR FOCUS**:

- What problems can distributed systems solve (given failure & other **ASSUMPTIONS**)
- How? What algorithms (protocols) do they use.
- What happens when **ASSUMPTIONS** are violated.

**Why study**

- **Deep Learning**
  - Training
  - Inference
- **Microservice Application**
  - **Cloud Computing**

- **React**
- **Javascript**
- ...
Distributed Systems

- Note, ABSTRACTIONS HIDE MOST OF THE COMPLEXITY

But

- Debug BAD PERFORMANCE \rightarrow\ Peek INSIDE ABSTRACTION
- Debug CORRECTNESS
- Reason about COST/PERF

Useful to understand how abstractions work & trade-offs they make!

Also, (I think) the problems & analysis are fun puzzles!

How To Study

* For most of us Failures are vanishingly
RARE

- Not something we are used to reasoning about

- HARD TO TEST
  - Need to control for when the failure occurs
  - Lots of ongoing work on testing behavior under failures

- Analyze behavior assuming some model
  - Models simplify and abstract reality
  - Usually impose stricter limits
  - Hope
    
    CORRECT UNDER MODEL
    
    $\Rightarrow$ CORRECT IN REALITY
- Analysis usually involves
  - Pen & paper
  - Proof assistants

Course Mechanics

- Papers
  - Models: Distributed Systems
    Failure Models
    Correctness Conditions

- Why?

- Expectations

- Labs
  - Elixir
    - Really a small subject
    - Code more closely resembles what is in the papers (mostly)
Exams (40%) → Readings → Labs (40%) → Extend → Final Project (15%)

Collaboration

Communication

Me → You: Website, Campuswire

You → Me: Campuswire + EMAIL

When possible: Please make posts public

Help others!

Asynchronous Model + Message Passing
Processes \( \Pi = \{ p_1, p_2, \ldots, p_n \} \)

Message Passing

Asynchrony Assumption

- Processes communicate by sending and receiving messages
- No bounds on time taken \( B/W \) \( \text{SEND}(m) \) & \( \text{RECV}(m) \)
- No bounds on time taken to process a message \( m \)
- No global clock

Consequences:
- Cannot decide whether process
  - has completed
p has failed.

Why?

Additional Assumptions

- Unreliable network: Messages can be dropped or reordered

- FAIRNESS: Any message sent infinitely often must be received infinitely often

```
\[ \text{A} \]

while(1) \&
\text{send}(B, m);

\[ \text{B} \]

\[ \text{FAIRNESS allows us to ignore unreliable network} \]

\[ \text{\& for } i = 0 \text{ to 10) \&} \]
\text{send}(B, m)
```
- Theory:
  \[
  \frac{P_a}{P_{\text{Net}}}
  \]

- Practice:

Going back to threads:

- Has equivalent results.

- This class largely focuses on message passing, commonly adopted model.