A few ideas from distributed systems for PL folk

Lindsey Kuper (UC Santa Cruz)

lecture 1 agenda (time permitting):

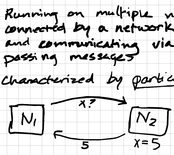
→ distributed systems: what & why?
→ time & clocks
→ causality & happens-before
→ Lamport diagrams
- Lamport clocks

"the failure of a computer that I've never heard of can render my own computer unusable"

— Leslie Lamport

A distributed system is:

- Running on multiple nodes connected by a network and communicating via passing messages
- characterized by partial failure.



What could go wrong?

- Request could get lost, or be slow
- Response could get lost, or be slow
- Byzantine faults!
- N_2 could crash.
- N_2 could be slow.

Partial Failure and unbounded latency.

FLP (1985)

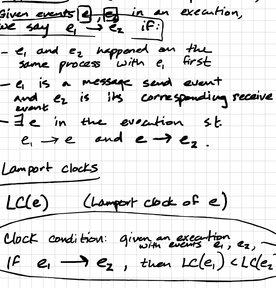
Time and clocks

What do we use clocks for?

→ marking points in time
"This lecture starts at 16:00 UTC+1:00"
→ track durations
"This lecture lasts 90 minutes"

Physical clocks

- time-of-day clocks
  - Synchronized with NTP
  - okay-ish for marking particular points in time.
  - bad for durations too.



- monotonic clocks
  - good for durations
  - bad for timestamping particular points in time

import time
time.monotonic()

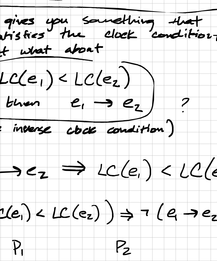
Logical clocks only track order of events.

if $A \rightarrow B$

we know

- B could not have caused A.
- A could have caused B.

Lamport diagrams



Happens-before ($\rightarrow$) (Lamport, 1978)

Given events e_1, e_2 in an execution, we say $e_1 \rightarrow e_2$ if:

- e_1 and e_2 happened on the same process with e_1 first
- e_1 is a message send event and e_2 is its corresponding receive event
- $\exists e$ in the execution s.t. $e_1 \rightarrow e$ and $e \rightarrow e_2$.

Lamport clocks

$LC(e)$ (Lamport clock of e)

Clock condition: given an execution with events e_1, e_2,
if $e_1 \rightarrow e_2$, then $LC(e_1) < LC(e_2)$

LCs consistent with happens-before.

The LC algorithm:

- Every process maintains a counter, initially 0.
- Events increment the counter by 1.
- When sending a message, a process attaches its current counter value.
- When receiving a message, a process sets its counter to max(local counter, message counter) plus one.

This gives you something that satisfies the clock condition, but what about

if $LC(e_1) < LC(e_2)$
then $e_1 \rightarrow e_2$?

(the inverse clock condition)

$$e_1 \rightarrow e_2 \implies LC(e_1) < LC(e_2)$$

$$\neg(LC(e_1) < LC(e_2)) \Rightarrow \neg(e_1 \rightarrow e_2)$$