Troubleshooting Distributed Systems

Sam Whitlock
samw@icsi.berkeley.edu

Colin Scott
cs@cs.berkeley.edu

Scott Shenker
shenker@icsi.berkeley.edu

Motivation

- Distributed systems have many bugs
    - asynchrony **X** partial failure **X** complexity = bugs

- Practitioners collect event logs for diagnosis

- When system encounters bug **B**, developers given event log **E** to troubleshoot

Common Case

- Log **E** is very large, hard to analyze
  - Best developers are often given task of troubleshooting

- Underlying bug **B** is "causally sparse"
  - Can be tickled with small external event sequence
  - But don't have any idea what that sequence is!

Our goal

- Minimal Causal Sequence (MCS)
    - Given a function replay() that can replay any set of events
    - **E** is an MCS iff, for all **e** in **E**
        - replay(**E**) triggers **B**
        - replay(**E** - **e**) doesn't trigger **B**
    - An MCS is only locally minimal

- MCS is fundamental to troubleshooting

Finding an MCS

- Conceptually easy
  - Given **E**, **B**, pick **e** in **E**, ask
    - if replay(**E**-**e**) triggers **B**, then **E** = **E-e**
    - Iterate!
  - Will always converge on an MCS

- Technical challenges all in replay()
  - Nondeterminism
  - Timings

Replay Timings

- Must interleave external with internal events
  - Need to maintain happens-before relation
  - If we get this wrong, won't trigger bug
  - Can't use clock time reliably
- Use causality as guide
  - Analyze causality in original run
  - Leverage it to understand causality in replays
- We are the early stages of this research
  - Trying to fundamentally change the way people troubleshoot distributed systems

Thank you

eecs.berkeley.edu/~rcs/research/podc13.pdf