Learning From Traces?

Midterm

- Next week in class

- Open book
  - ↳ Notes, papers, ...

- Can use a tablet/laptop
  - ↳ But please, no chat, e-mail, Google.

- Types of questions
  - Some subtle hints: reminders of terminology

→ Some subsets / simple versions of terminology

↳ What is a span?

→ Ways to sample?

→ Some about applying ideas to simulated traces

↳ Figure out critical path

↳ Figure out dependency graph

→ Some about extending ideas from the papers we read

↳ Identifying the potential benefit of scaling

→ Aggregating traces?

Project Proposals

- Heard from 3 groups (~6 people) so far

↳ 1 Question about potential ideas

→ 2 proposals

- Poll: Is everyone on track?

- Where are people?

Last week

- Given semi-structured logs
  - ↳ <Timestamp, Request ID, service name>: ....

MYSTERY MACHINE

Infer causal graph of services
- ↳ Why was inference necessary? Would spans have helped?

- Given causal graph
  - ↳ Infer critical path for a request.

→ Compute a model for service latency & slack.

- Inputs?

→ Use model to improve quality.

Zooming Out

Overall Observation

Can use trace data to learn about system

↳ Bottlenecks for types of requests
→ Places with high likelihood of errors
→ How they use storage or the network or...
→ ...

↳ Can use this information to

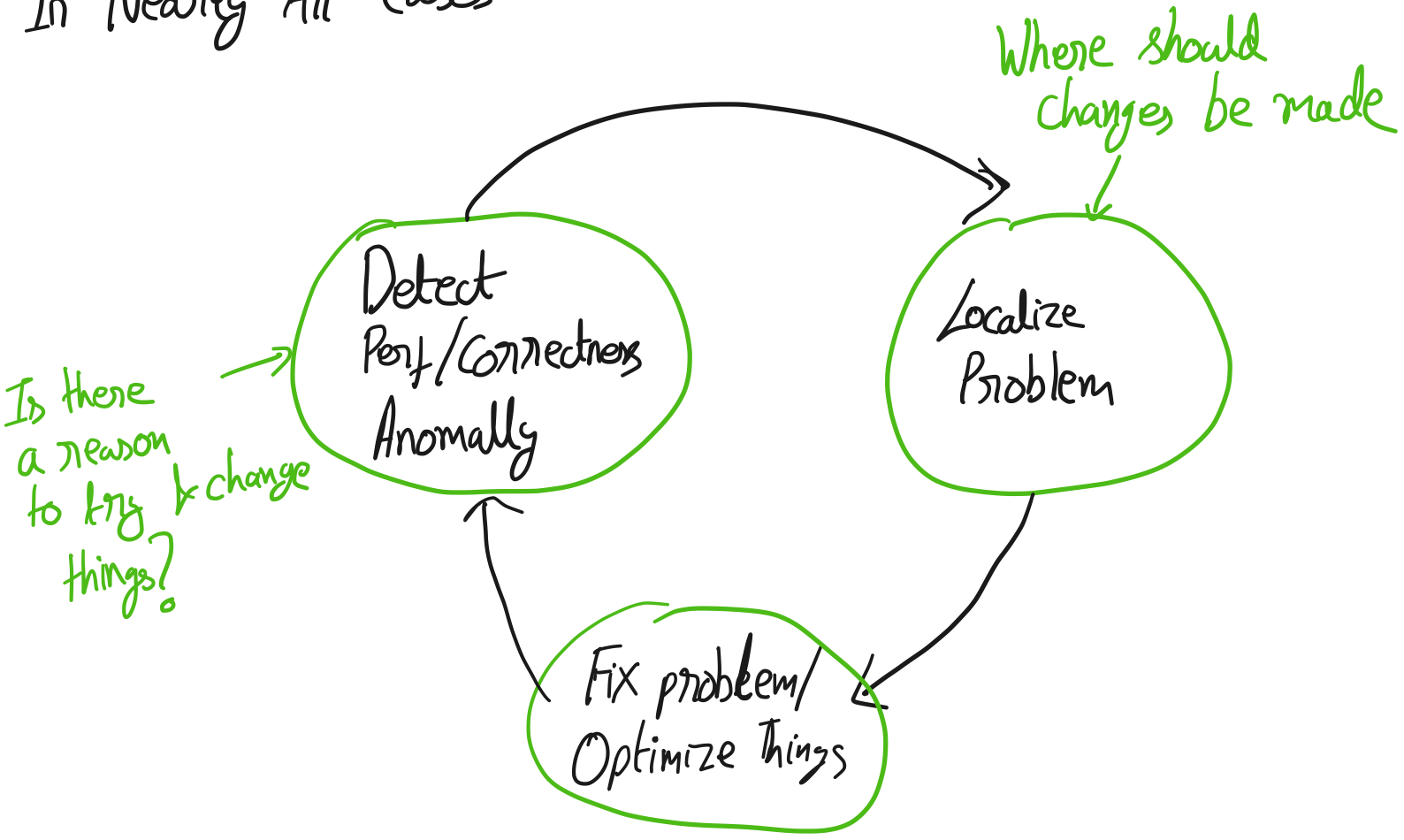
↳ Triage problems / allocate development time
- Mystery Machine, Critical Path Tracing

↳ Change application behavior or configuration
- DB Bourge

↳ Change deployment environment

– Learning on Distributed Traces for DC Latency

In Nearly All Cases



Traces help with all steps (maybe)

↳ Question really is on tools to automate portions of each

- Omega Gen: [Localize]
- ACT Now: Localize
- Learning on Distributed Traces: Opt

- DQBarge: Optimize
- Critical Path Tracing/Mystery Machine:

Critical Path Tracing

+ Google's approach to getting Critical Path

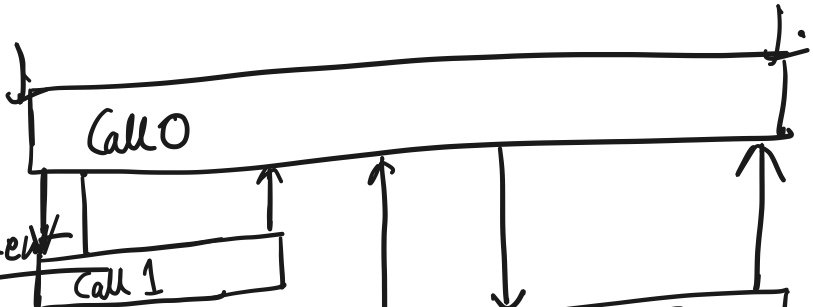
+ What is different
  - Shouldn't need to infer service level causality

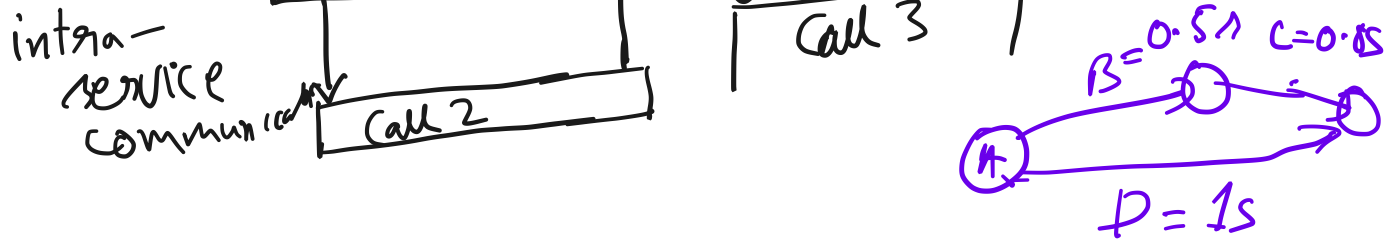
Dapper

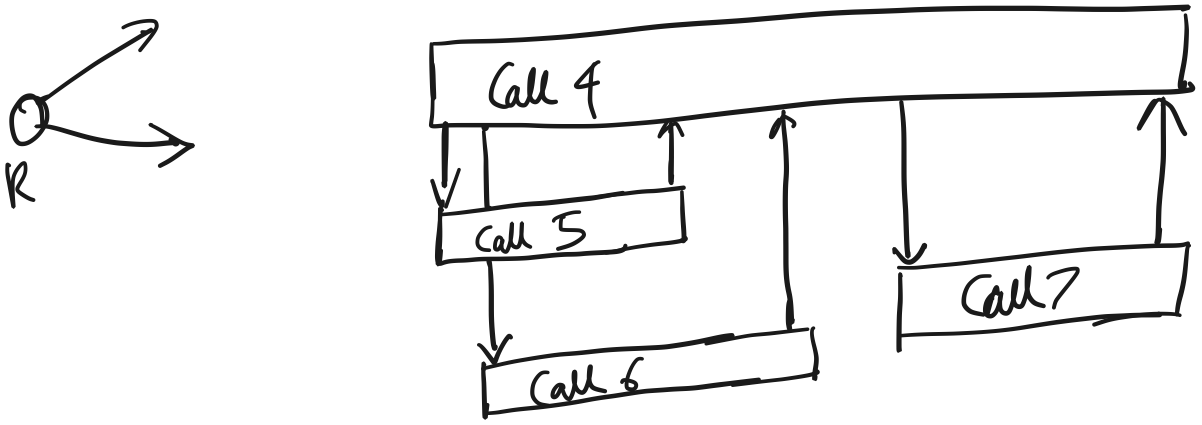
<rid, parent, st, et, id>

  - But Dapper also collects response latencies
    [Remember: each span has start and end time]
  - So why do we need this new system?

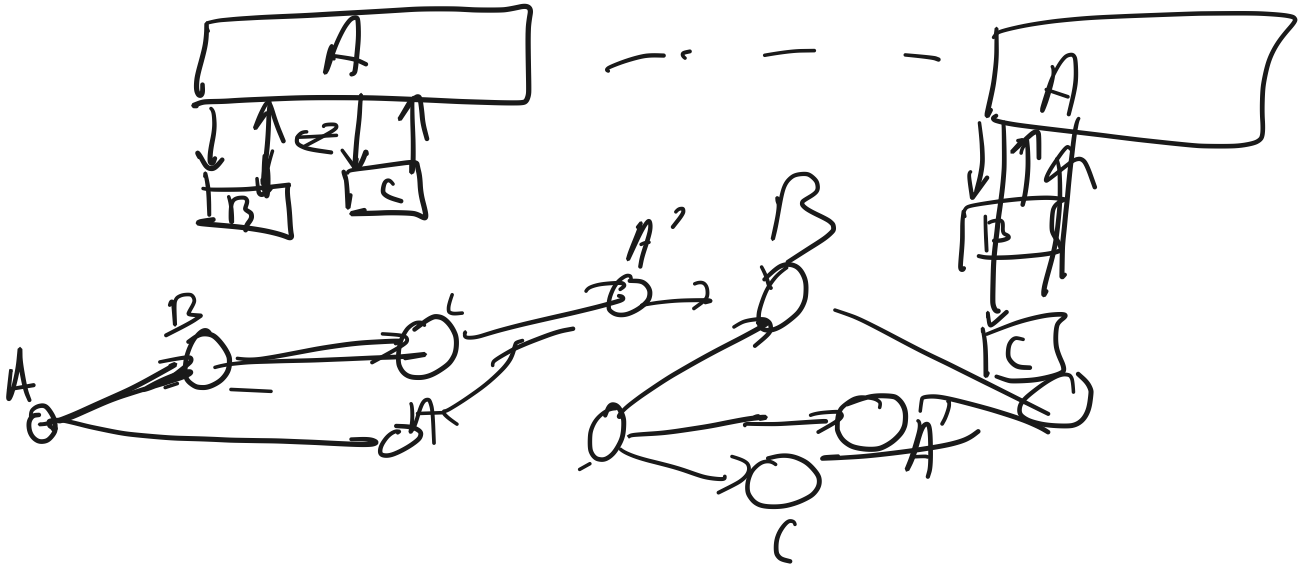
- Granularity
  - ↳ subcomponent







- How does Mystery Machine handle this problem?



- Other differences: On sampling
  - Assumptions on where sampling is initiated

- Consistent sampling

how (grid)?:

OmegaGen: Automating anomaly detection?

- Focus: Performance degradation in a single service
  - Hypothesis: Large enough performance degradation at one service likely to make service a bottleneck.

→ Automatically detect when this happens + notify administrators

- Ok; but how to detect anomalous performance?
  - Problem: Request might take a long time because
    - Complex Request
    - Problem in execution environment

→ Approach: Focus on problems likely due to I/O [etc]

→ Assumption: Execution problem [illegible] I/O

↳ Decide request processing is anomalous if I/O is slow or erroneous

↳ Why reasonable?

- Problem: How to separate out I/O from other processing?

↳ Requirements:

- Fidelity

↳ Identify correct I/O call

↳ Hard with OS based ptrace / syscall trace (maybe).

...

fn foo(a) {

let fname = a;

open(fname) - write

...}

→ Approach: Don't identify, instead mimic

Trade-Offs:

↳ Fidelity: Mimic must resemble actual execution

How?

How this approach fits into things so far

- Dapper, Mystery Machine, ... focus on
  - ↳ Limiting performance impact
  - ↳ Limiting what services have to change
- Does OmegaGen fit?

ACT Now: Localize problems by 'detecting anomalies'

Core hypothesis: PROBLEMS IN A REQUEST ARE LIKELY DUE TO UNEXPECTED/UNUSUAL INTERACTIONS BETWEEN SERVICES [Though we can generalize, see below]

↳ NOTE: DIFFERENT FROM OMEGAGEN
↳ NOT FOCUSED ON BUGS WITHIN A SERVICE

DOES THIS MAKE SENSE?

↳ WHY OR WHEN DOES THIS MAKE SENSE?

- Services have fail-over or other fall-back mechanisms: c.f., DaBarge

- Load balancing

- ...

- Approach: Diff successful & unsuccessful traces
  - ↳ Hypothesize that problem lies in the diff.

Problem: Diff is a super-set of problematic services. Why?

- How to solve this problem?

→ Try applying ideas from Mystery Machine.

$A = \{x, y, z\}$

$B = \{y\}$

$C = \{z\}$

$A - B = C$

$\{x, z\}$ $\pi \hookrightarrow T_k$ ✗

$\pi - \pi_0' = D_0 - \pi_1'$

$\pi_0' \hookrightarrow T_0$

$\pi_1' \hookrightarrow T_1$

$\pi_2 \hookrightarrow T$

A-C = $\pi - \pi_1'$
{X, b} ⋮ ,
π_n

- But how many traces?

  Thresholding:

- Generalizing this approach
  → Consider attributes in the diff.

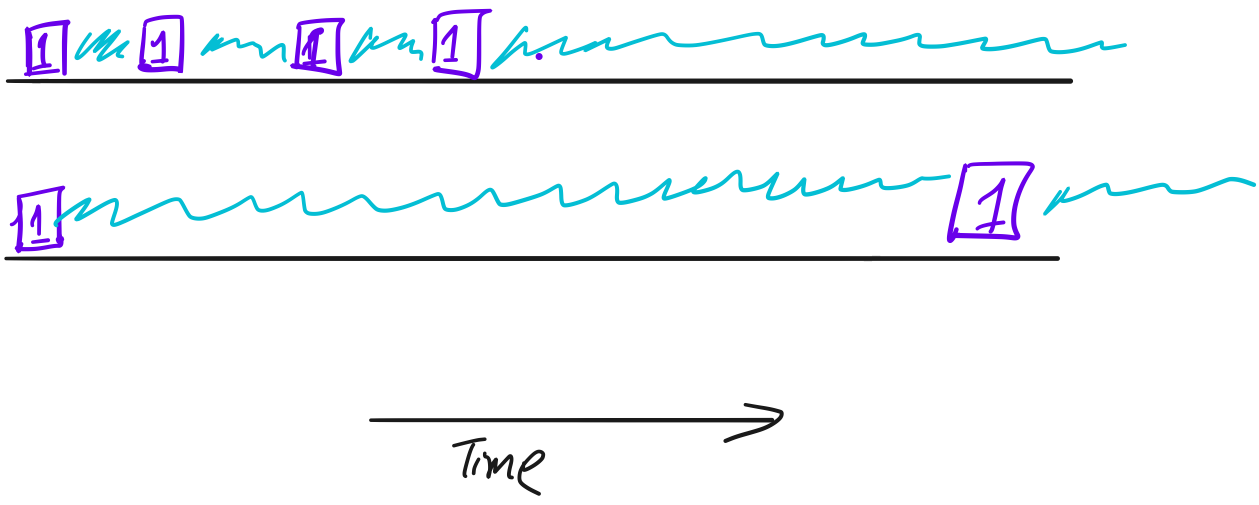
  → Challenge: What attributes to include?

  → How to compare attributes?

Which brings us to Learning From Traces.

Core observation: The IDEAL CONFIGURATION FOR STORAGE SYSTEMS DEPENDS ON WORKLOAD

- Will data be modified or not?
- Access pattern: do accesses exhibit temporal locality or not?



- Expected file size

→ Expected file lifetime

SOTA: Set parameters based on average workload

Problem: Average workload is really no workload.

Long standing tension in systems

Generality ⟷ Specialization

This paper: We have traces, can we automate specialization?

→ Actually specialization is hard: can we even answer questions about files based on writer.

Given

- Traces of previous requests that create file
  - → Attributes
  - → Spans
- Ilo log with FS information of interest (e.g. access locality)

Construct algorithm $A \rightarrow A_0$: Job attributes → FS information

→ Note: Similar to DQBarge.

But make no/few assumptions about attributes

But make no fewer assumptions about ...

on jobs or...

Why?

Does this actually work/challenges