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 subject, kind reminders, terminology.
Some subset of simple streamline 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
  \[ \langle \text{timestamp}, \text{Request ID}, \text{service name} \rangle : \ldots \]

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

- Given causal graph
  \[ \text{Infer critical path} \text{ for request.} \]

- 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 on the network on...

Can use this information to

- Triage problems and allocate development time
  - Mystery Machine, Critical Path Tracing
- Change application behavior or configuration
  - DAG baggage
- Change deployment environment
  - Deploy to Tier 1 or DC storage
In Nearly All Cases

- Detect Peer/Connection Anomaly
- Fix Problem/ Optimize Things
- Localize Problem

Where should changes be made

Is there a reason to try & change things?

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
Critical Path Tracing

+ Google's approach to getting critical path

+ What is different

  - Shouldn't need to infer service level causality

  ▪ Dapper

    - But Dapper also collects response latencies
      [Remember: each span has start and end time]

    - So why do we need this new system?
- How does Mystery Machine handle this problem?

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

\[ \text{has (\text{null})?} \]

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?

  \[ \rightarrow \text{Problem: Request might take a long time because} \]

  \[ \rightarrow \text{Complex Request} \]

  \[ \rightarrow \text{Problem in execution environment} \]

  \[ \rightarrow \text{A problem likely due to I/O} \]
Assumption: Execution problem likely due to 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 100 5
  -> 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
  \( \rightarrow \) limiting performance impact
  \( \rightarrow \) 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]

\( \rightarrow \) **Note**: Different From OmegaGen
\( \rightarrow \) Not focused on bugs within a service

Does This Make Sense?
Why or when does this make sense?

- Services have fail-over on other fall-back mechanisms: e.g., DaBange

- 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.

\[ A = 3 \times 1 \times 7 \times 2 \]
\[ B = 2 \times 6 \]
\[ C = 3 \times 2 \]

\[ A - B = C \]

\[ 9 + 1 = 10 \]

\[ D_0 - D_1 = \]

\[ T_0 \]

\[ T_1 \]

\[ \pi \]

\[ 1 \]
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?

---

**Time**

- 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 $\leftarrow$ Specialization
This paper: We have traces, can we automate specialization?

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

Given

- Traces of previous requests that create file
  - Attributes
  - Spans

Construct algorithm

A \rightarrow A^0 \rightarrow \text{Job attributes} \rightarrow \text{FS information}

→ Note: Similar to DQBorge.

But make no/very assumptions about attributes
But make no fewer consolidations with the & 
on jobs or...

Why?

Does this actually work/challenges