Lecture 5: Mystery Machine.

Administrivia

Project proposals & groups

Perhaps useful to send me a short description of what you were thinking.

Two options:

(a) Post a note, only visible to instructor, on Campuswire [PREFERRED]

(b) E-mail me to set up time to talk in-person
    [Will do it in 15-30 minute meetings next week]
- Poll on office hours.

- Class feedback?

localizing problems & using localization information

- Why?

Again a use of trace data & infrastructure

But less influenced by my tastes & preferences. Few big companies doing the same

Presents two different approaches
OmegaGen - White box / intrusive
[But does it gather the same information]

Mystery Machine - Black box

Assumes no knowledge about
- what services communicate.
- Service dependencies

Focus in both cases is on finding PERFORMANCE BUGS.

Critical Paths & Performance in Concurrent Systems
What do we need to determine critical path

1. Knowledge about dependency relations

- 1 → 2 → 3 → 5
- 1 → 2 → 4

3 Paths.

Core message: These dependency relations are often unknown.

Huh? Is this a Facebook only problem?
Why?

Approach: Mine dependency from traces

What do we need to determine critical path?

1. Knowledge about dependency relations
2. Time spent on each component in a path.

Mining Dependency Relations HoBr

1. Assume all possible relations
2. Use trace events as counter examples
Soundness: Will this method ever incorrectly decide two related services are unrelated? (In example above, no HOB.)

Completeness: Will this method ever incorrectly decide two unrelated services are related?

Impact on tool's critical path?
Assumptions: Why Is This A Reasonable Design In Practice?

- Natural Perturbation

② Identifying Critical Path, And Bottleneck
Is the critical path stable? Do all requests have the same critical path?

Why? Important to how we use critical path information.

Using critical path information

- Targeting optimization information resources
- Making resource provisioning decisions
  \[\rightarrow \text{Slack}\]
Using trace information in real-time

Q: When to give up on 4→6?
- 50-6 ms after 4 received request.

Pros

Cons
6 has a chance to respond

- Before invoking the call

Pros

6 does not meltdown

Cons

6 no chance

- Others?

Pros

Cons
Question: How do we decide (a-priori) whether to make 4-76 request?

Make Request

+ Likely have better quality result (How likely?)
- Increase load on 6, perhaps w/o benefit to this request

Do Not Make Request

- Guaranteed to have lower quality result
- Has no impact on 6.

Desirable: Make request only when success is likely. But how do we know?
Execution time depends on:

- Load (# of outstanding requests)
- Background tasks (GC, rebalancing, etc.)
- Request type
- Request arguments
  - Request history
  - Downstream tasks

Depends on other services + likely varies gradually

Maybe unpredictable, but likely to be periodic

Requires semantic knowledge

Maybe best captured by trace except on caveats above.

Core argument: Request execution time roughly depends on
- Other requests in the system
- System health
  - Execution time when processing request at other processes

Proxy for request type argument

When running, see a large enough variety of requests, workloads & system load conditions

Use data to learn a model for system behavior M

No inputs include

- Metrics at current service (e.g. 4)
- Metrics at upstream services (e.g. 2, 1)
- Accesses, etc. at upstream services

- Output: Yes or no on call + what to do
Challenge: How to collect metrics for the current request.

Sol:\ Do what PivotTracing Did.

OmegaGen

- How to improve accuracy for detecting perf bugs/anomalies
  + Localize them at a finer granularity

w/o many other requests.

Core Problem w/ localization

Programmer doesn\'t a-priori know critical path/Critical Op
Mystery Machine: Use data to find critical path/critical op post facto

OmegaGen: Guess potential bottleneck

Decide if they are actually a problem

1) Potential bottlenecks

I/O?

Why? What else could/should include?

2) Decide if they are a problem

Monitor execution

Minimize recursive operations
Soundness?

Completeness?
Do these systems really do same/similar things?

Returning back to DQ Badge

How have others solved this problem?