Control loops:

- How to make & check

Misc.

→ Some observations from the midterm

- RPC message size vs CPT
  \[ \text{Local} \quad \downarrow \quad \text{Local} \]

[Diagram of network connections]
- SPANS vs. LOGS

```
A
  ▼
  B
  ▼
  C
```

```
  A: RID, T_0, ...
  B: RID, T_1, ...
  C: RID, T_2, ...
```

Today: Control Loops

- What are they?

  Things that observe:
  - cluster state
  - request rate
  - workload b: where requests are coming from
  - ...

\[ 1 \text{ms} \quad 20 \text{ms} / \text{rate} \quad 300 \text{ms} \]

A take action to meet some objective

\[ \text{Min SLO violations} \]
\[ \text{Max throughput} \]
\[ \text{Min resource waste} \]
An idea that predates computers by several decades

How to steer torpedoes

Ships

You have already seen one control loop.

DeBange

But we did not talk about the design thereof but instead let the model do heavy lifting.

Today a deeper look at the actual loops.

What does this have to do with tracing?

Common theme of all three papers

Local decisions require global information

What global information is required?
A single service can have a lot of different requests to process

From different upstream services

Workflows (in the sense we saw earlier)

With different - resource requirements
- execution times
- SL0s
- Utility

Scheduling: Decide order in which
requests are to be processed
Admission control

From DABrange: Sometime best not to issue a request that is unlikely to complete in time.

More generally: Need to limit number of requests in the system for stability.

Why?

Admission control: How many requests to "allow into the system"?

How to disallow requests?

⇒ Return an error.
⇒ Silently drop them.
⇒ Make them not issue a request.

Request routing
Many services are replicated

If a request is going to be admitted, which replica should process it?

\[ \text{[Chosen replica then schedules the request]} \]

For all three: We should ask what is desirable

\[ \rightarrow \text{Across the cluster} \rightarrow \text{UTILITY FUNCTION} \]
\[ \rightarrow \text{Within a service} \]

Possibilities

\[ \rightarrow \text{Fairness b/w workflows, upstream services, user} \]

\[ \rightarrow \text{First come first serve} \]

\[ \rightarrow \text{Priority based} \]

\[ \rightarrow \text{Minimize violated SLO} \]

\[ \rightarrow \text{...} \]

Wise A C3 focus on minimizing violated SLOs.
Why?

- Common concern in multi-tenant environments for control loops managed by operators that manage tenant behavior

Sort of reflect economic incentive

- Less common in academic settings
  - networking
  - operating systems

Firmenich or some other metric is more common

Remember this figure.

Many options
- FCFS
- Round Robin

Order in which requests are processed.

Request scheduling
- Processing Time Fairness

- 000

Looking at one service might not be good for minimizing SLO violations

If A goes first (assuming nothing else is in the system)

If I go first,
Observe: Choice at \( x \) determines
- Overall completion time on \( I \)
- But has no (or smaller) impact on \( A \)'s completion time.

But delayed \( I \) could have violated SLO?

(I know this example sounds contrived, but...)

One option: Strict priorities: More stringent SLO first

Lower latency first.

Problem:

(Ignore the graph for now.)
Can delay low priority jobs arbitrarily.

What is desirable

→ Account for time left when scheduling.

(To S20 violation)

But, at a service, time until S20 violation is likely depends on

→ Time spent upstream
→ Wait time at service
→ Execution time at service
→ $E[Wait\ time\ downstream]
→ E[Execution time downstream]
Control loop

Request (time spent upstream)

Scheduler

Execute time of other requests

Downstream

Service

How to estimate? EWMA

\[ z_t = \lambda x_t + (1-\lambda)z_{t-1} \]

Contribution from old measurements drops exponentially!!

Estimates useful for many policies

→ EDF (Earliest deadline first)
→ SRTF/SRPT (Shortest remaining proc. time first)
→ SJF
Admission Control

How many requests/second (to avoid violating SLO)?

How many RPS to avoid SLO violations for X?

Capacity of?
Needed by $\square$ on upstream.

But how to estimate?

- Changes depending on workload

- Might also vary over time
  - S/W changes
  - Automated updates
  - ...

Control loop: Estimate (guess) capacity. How?

Assumption: Back pressure

Can use queue length at $\square$ to estimate
Can use processing capacity.

\[ \Rightarrow \text{Limit RPS to keep queuing within a small range?} \]

**Problem:** Both □ □ contribute to queue at □, as does downstream overload.

\[ \Rightarrow \text{For □ □ need to split capacity b/w them} \]
\[ \Rightarrow \text{For downstream overload: need to reduce rate} \]

\[ \Rightarrow \text{Track \# of requests from self} \]
\[ \Rightarrow \text{Track response time for each request} \]
But what if we can select among many different replicas?

Which should go to?

@ Shortest queue?

⇒ All of might make same decision. ⇒ Overload?

⇒ No coordination!
Queue length ≠ Response time.

Why?

(1) Fastest?
   → How to predict?

(2) c3??

"Based on historic information?"

Why would this possibly work in our case?

The case of symmetric decisions.
The case with different request lengths.

Rate at which a control loop can adapt?
Testing & debugging control loop

Challenges

- Defining correct vs incorrect is hard
- Hard to figure out sequence of feedback to use to drive the tests
- Hard to associate action with a particular
A.R.T targets a case where some of these problems can be assumed away.

What this enables:

- Mutation based input generation
- Associating (maybe) action & input
- Test oracles

Extending Beyond this setting