The Innovation Lab

Summer Internship 2025

Mentor Intern Meeting 8

<MiCozServices>

Mentors:
Smaran Jawalkar, Animesh ND, Achyuth Yogesh

Interns:
Suchitra Shankar, Navneet Nayak, Nathan Paul, Lekhyasree M

Want to speed up your code? Profile it!

Just because a portion / line of the code runs the longest doesn't mean optimizing it is worth it!

Code runs in **parallel**, **concurrently** and **asynchronously**.

Code can be **dependent** on other code or in **contention** with other code!

What if:

We could experiment and measure real changes?

If I speed up (optimize) this line of code by **X%** how much will performance improve?

This would actually take in account all **causal relationships...**

What can we do?

Speeding up a line of code by X% == **Slowing** down **everything else** (other code in parallel) by X%

This is called **'virtual'** speedup

Problem Statement

Coz: A causal profiler for **monolithic applications (single binary programs)**.

Goal: Test whether the **principles** of causal profiling used by Coz can be applied to microservice architectures

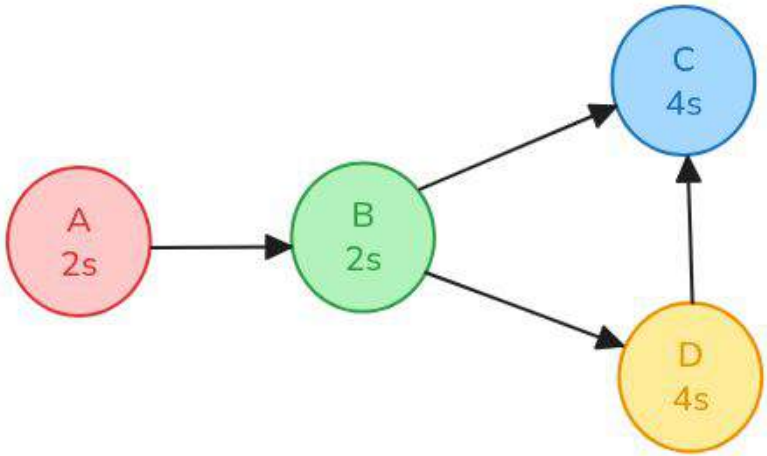
Approach

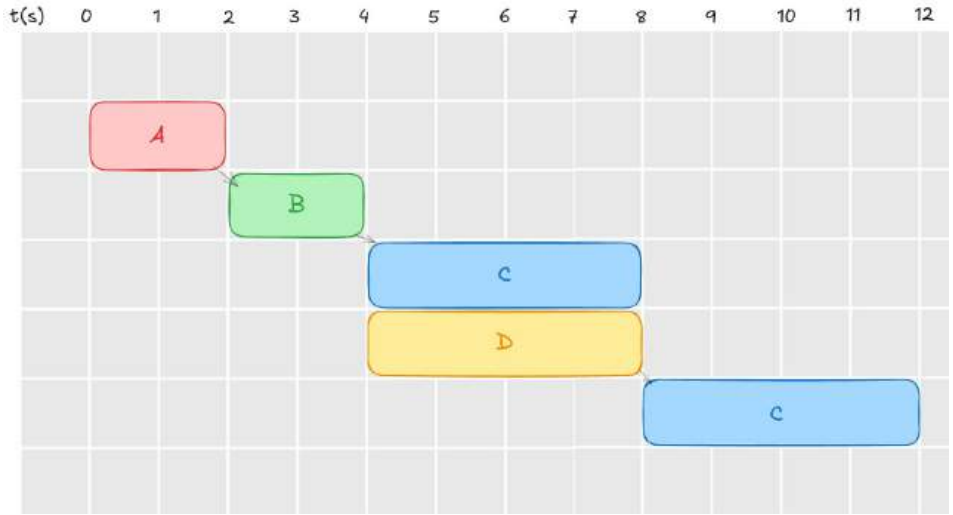
Lines of code/threads running, ≈ services in microservice architectures.

We shift our focus to which service is worth optimizing...

Proposed Solution

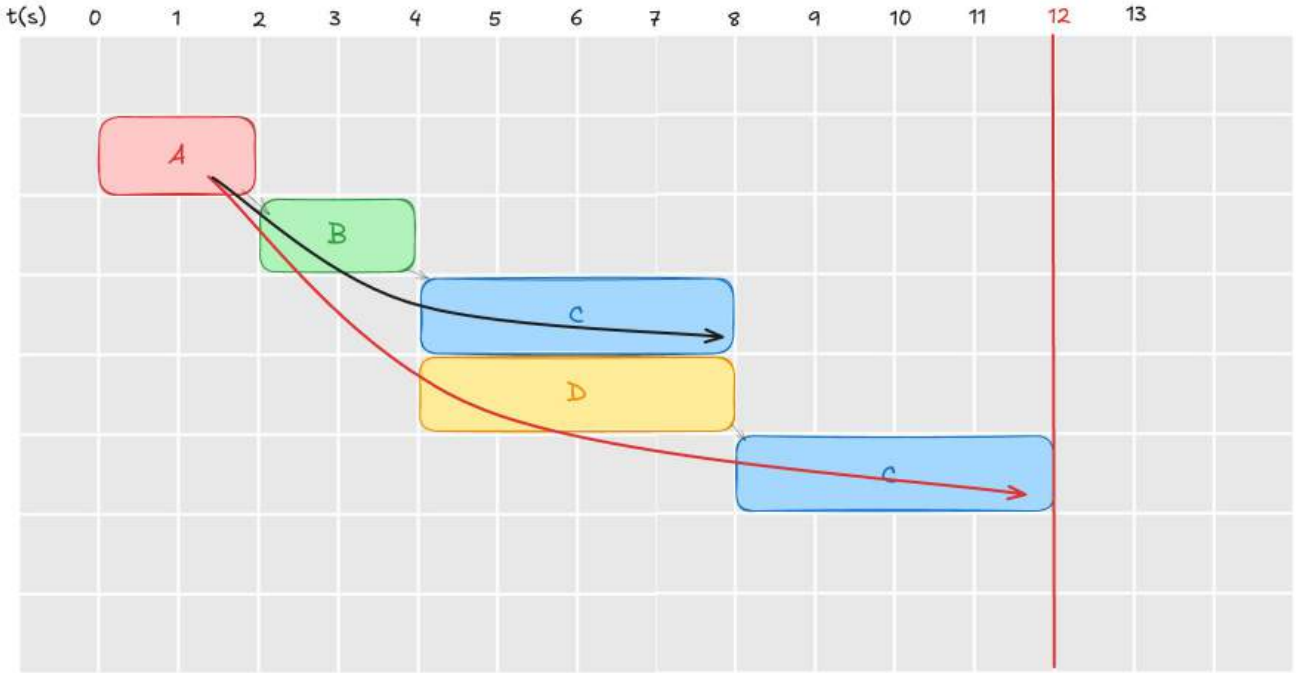
We can visualize this microservice architecture in the form of a Timing Diagram



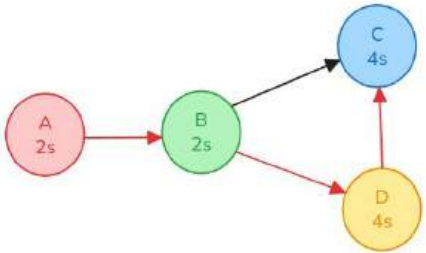


Proposed Solution

Consider the **critical path** (follow the red arrow)

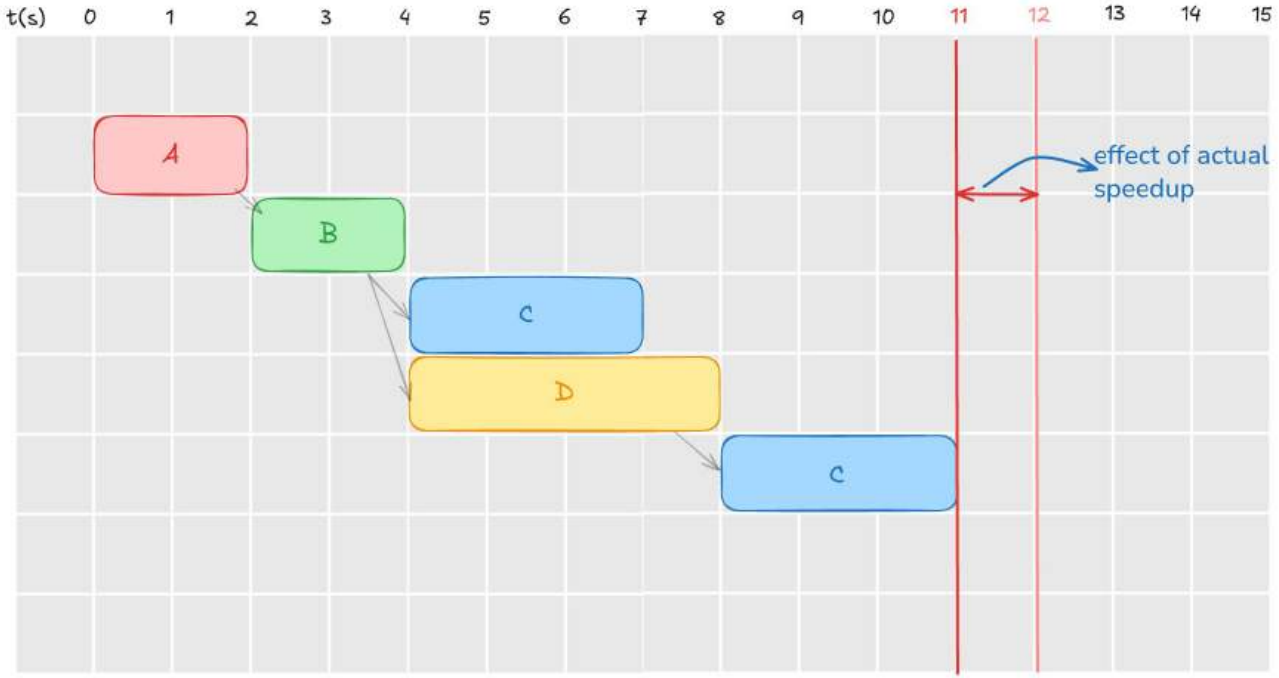


Runtime:
2+2+4+4 = **12s**

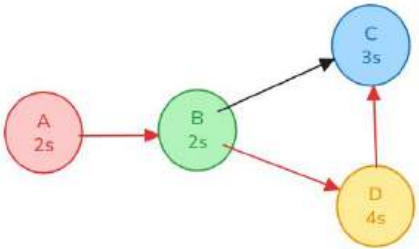


Proposed Solution

Consider the case of speeding up C



Runtime:
2+2+4+3 = **11s**

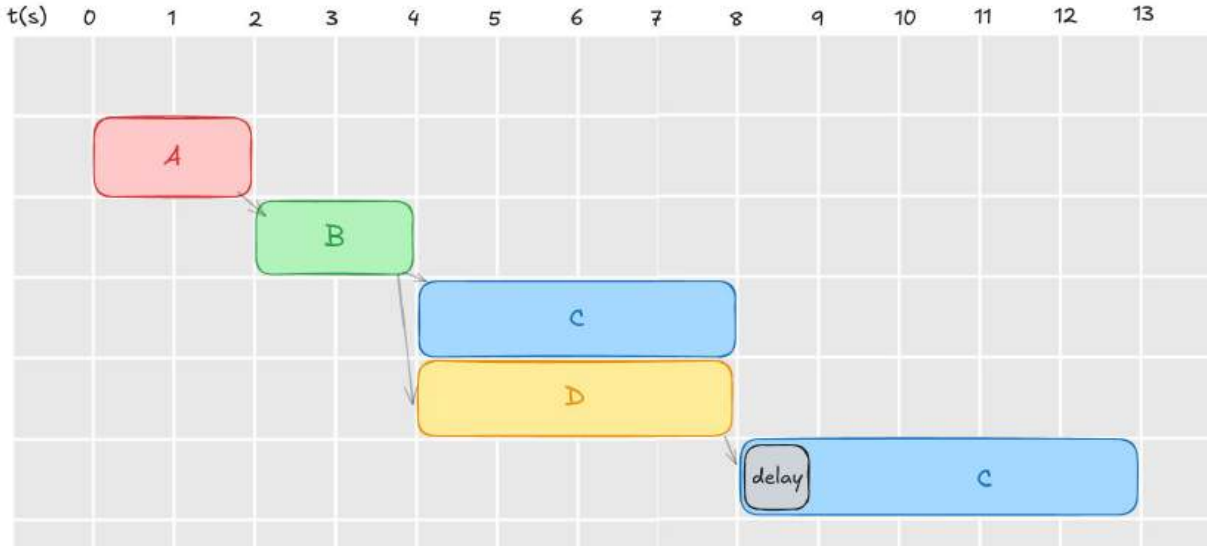


Proposed Solution

speeding up a microservice == **stalling** all other **branches of execution**

Proposed Solution

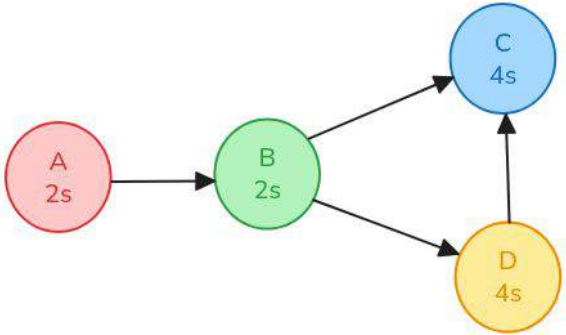
Let's **virtually** speed up C by 1s.



Slowing down **any one service** on a branch

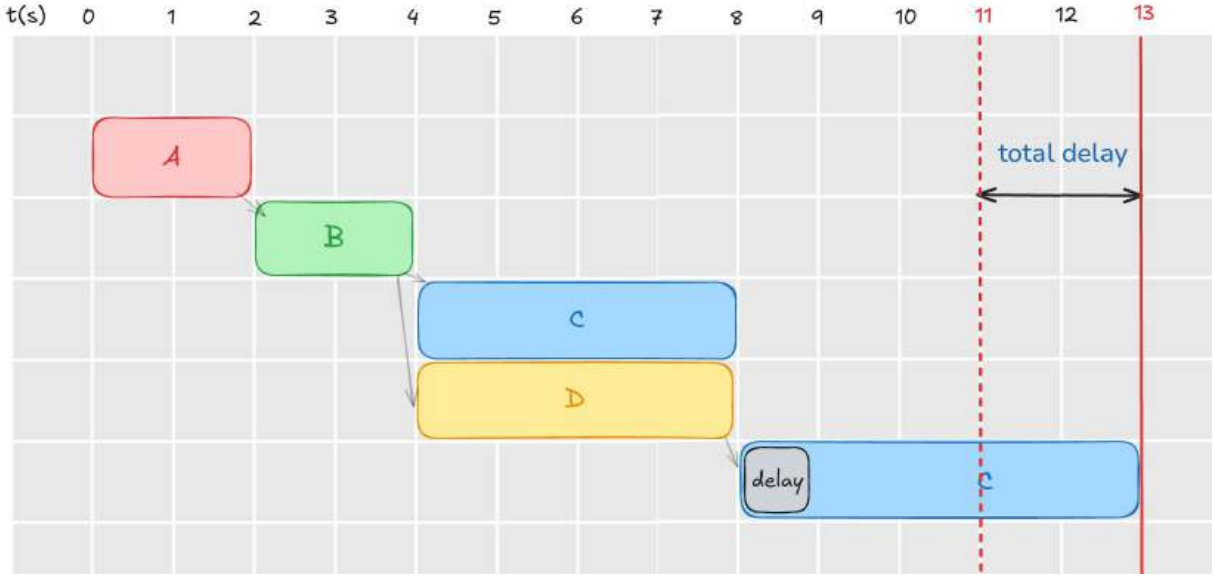
==

Slowing down the **entire branch**



Proposed Solution

Let's virtually speed up C by 1s.



Effective Runtime =

Runtime – n*d

n = no. of times C is instantiated

d = delay injected

= 13 - 2*1 = **11s**

Technical Model

When **virtually** speeding up a service:

1. **What** services should stall (sleep)?
2. For **how** long?
3. **When** should they stall?

Determined using **Counters**.

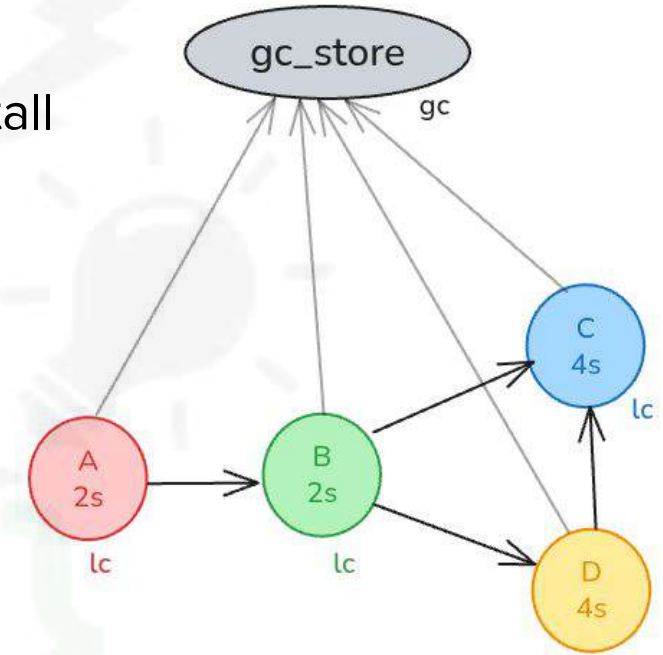
Counters

1. **Global Delay Counter**

   -> Time that **whole** architecture should stall

2. **Local Delay Counter**

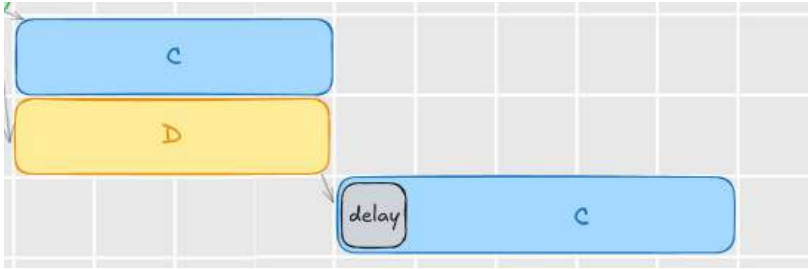
   -> Time that a node has **already** stalled.

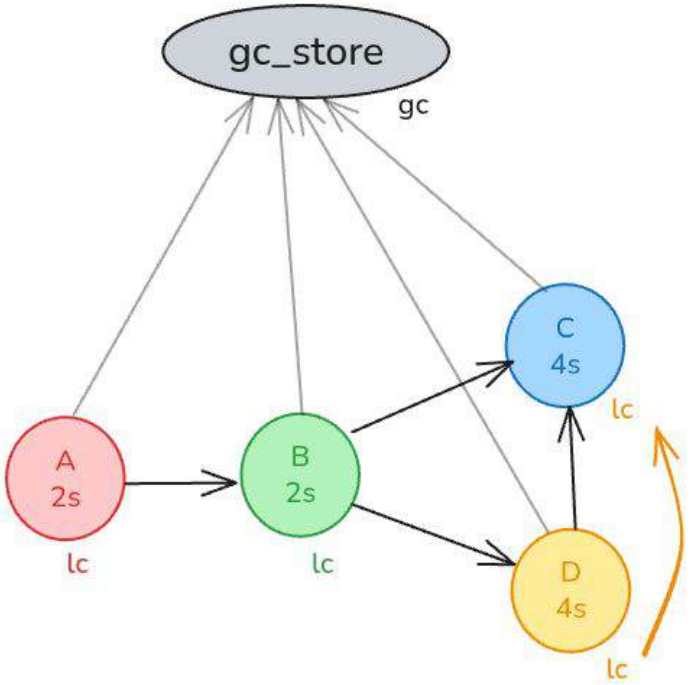


All microservices query a **global counter** store for the global delay.

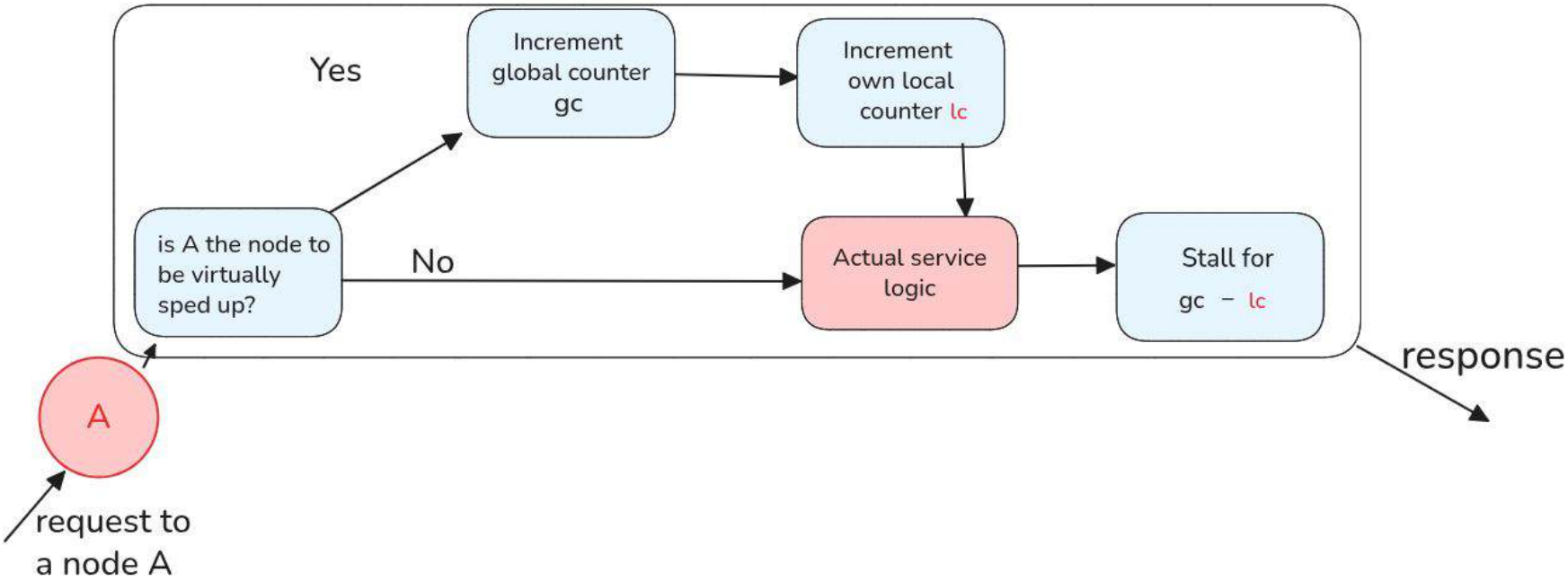
Propagation

To prevent stalling **multiple** services on a branch of execution. Local delay is **propagated** forward in the branch.





Middleware


Yes
Increment global counter gc
Increment own local counter lc
is A the node to be virtually sped up?
No
Actual service logic
Stall for gc − lc
response
A
request to a node A

How did we test our causal profiling model?

1. **Simulating** microservice call patterns

2. Testing on a **Real** microservice

Simulating microservices?

Each execution path for a request can be modelled as a DAG (**Directed Acyclic Graph**)

A DAG represents the **flow** of microservices, where each **node is a service** and **edges show dependencies**

The nodes in the DAG are deployed as services, which do a set amount of computation (by controlled **busy waiting**).

Approach

For every service in the DAG, we:

1. **Virtually** speed up the service

2. **Actually** speed up the service (by **reducing** busy waiting time)

Check if latencies **match**.

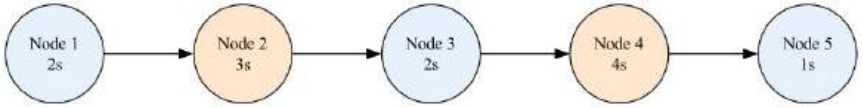
Simulating microservices

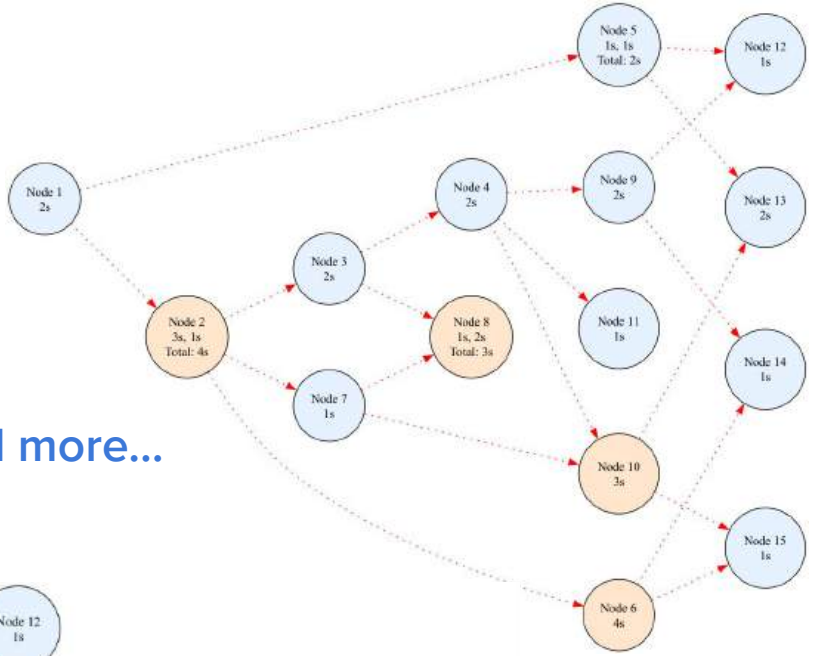
We tested our virtual speedup model:

1. **60+ randomly generated** microservice DAGs

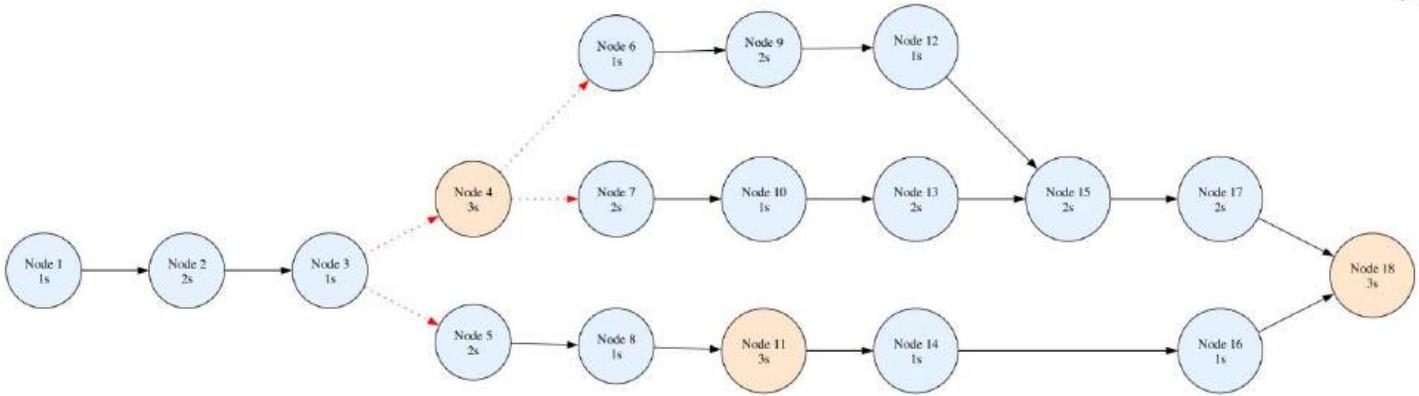
2. **Modelled real microservice** architectures

Testing random DAGs



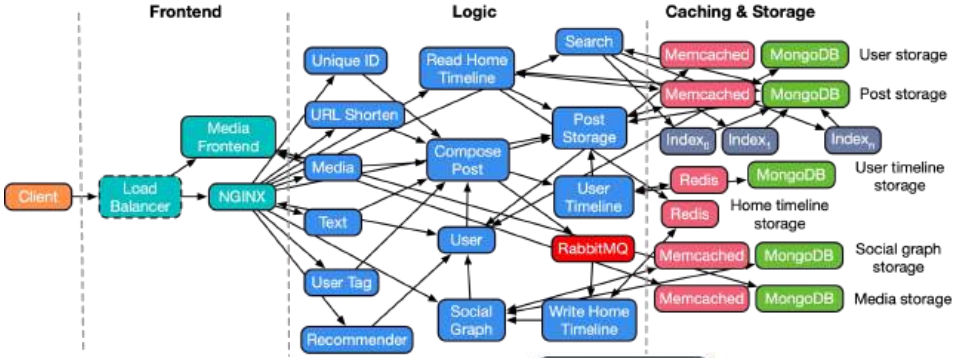


And more...

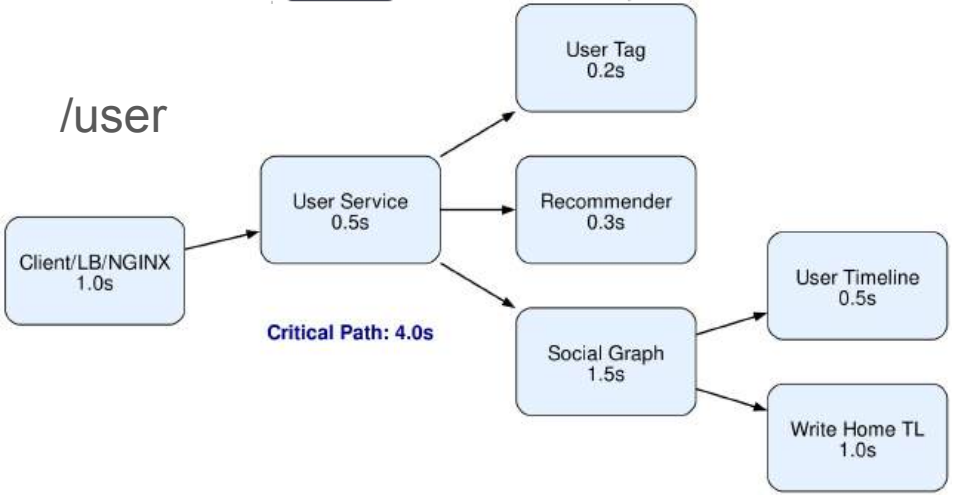


MiCozServices

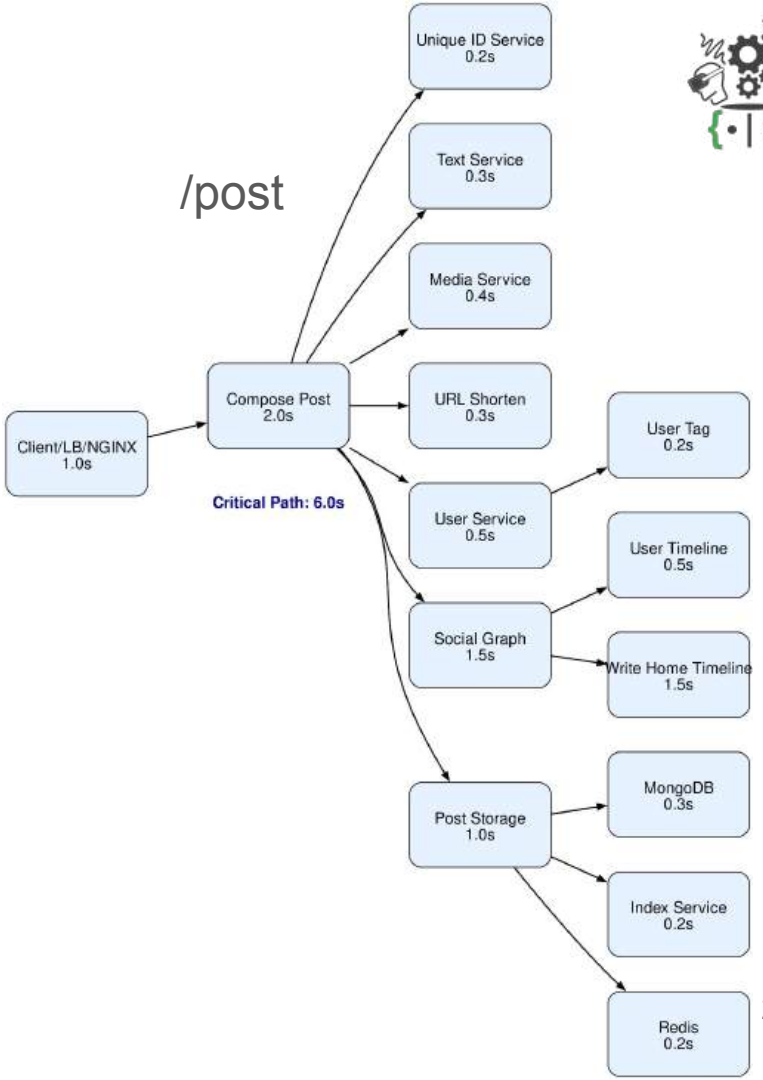
Deathstar benchmark - Social Network



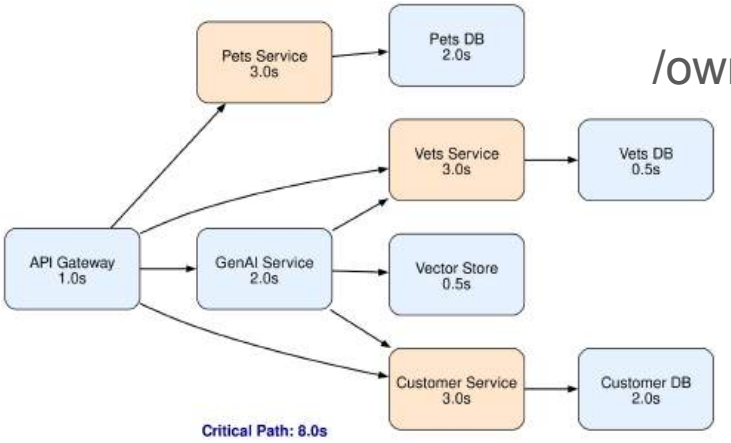
/user

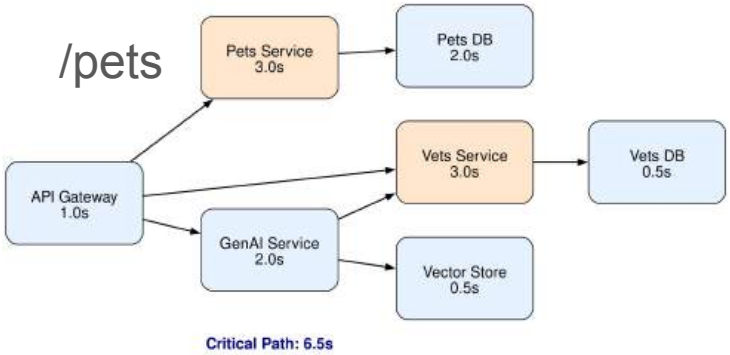


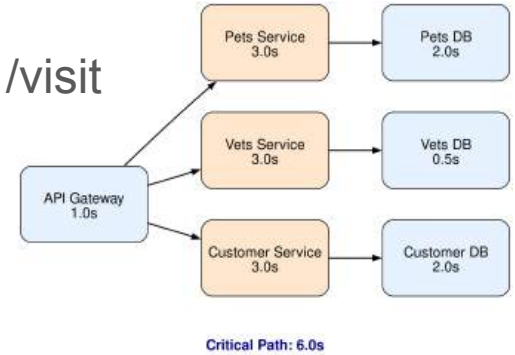
/post



PetClinic: A popular microservice benchmark







MiCozServices

Results

In **all** our tests on simulated microservices:

Latency after virtual speedup == Latency after real speedup

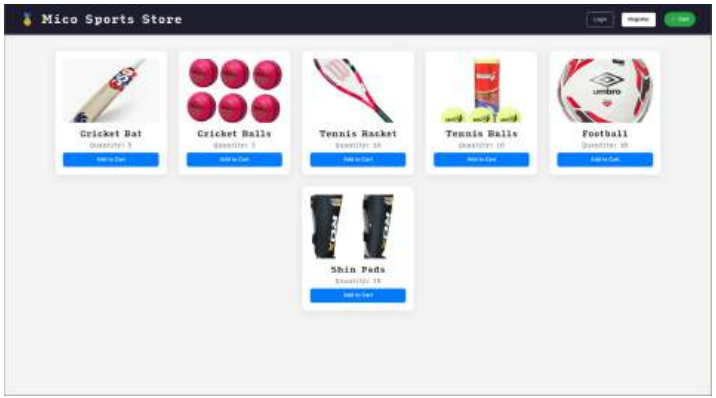
(within a **1% margin of error.**)

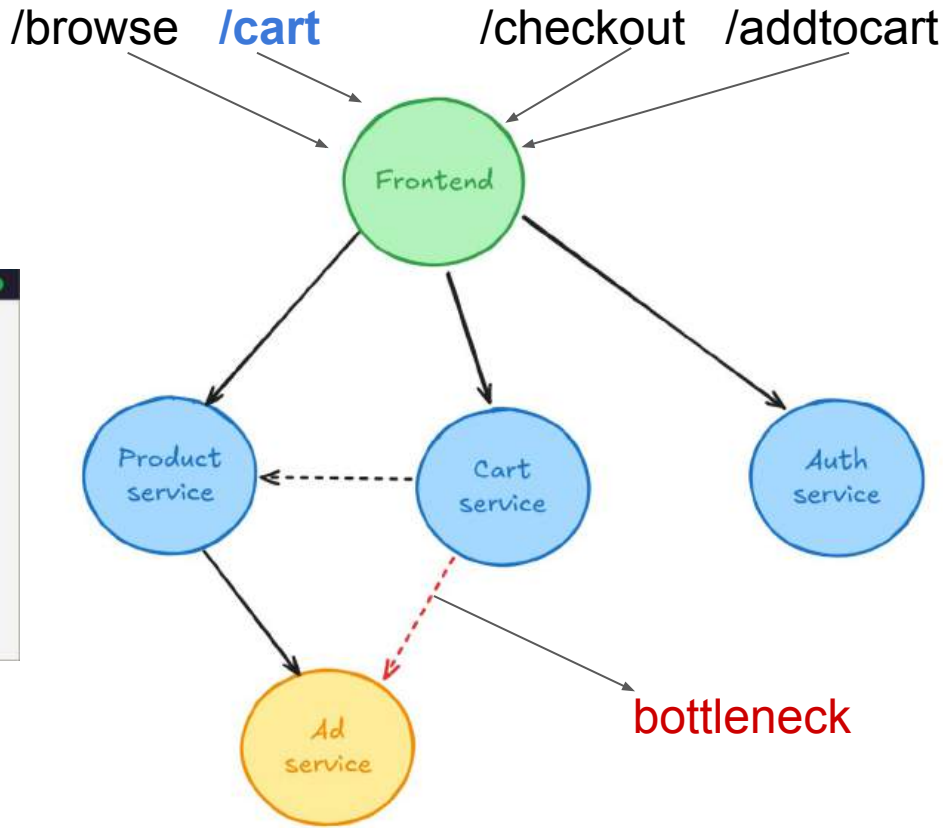
Testing on a **real** microservices

Real microservices have **unpredictable** runtimes

To get an idea of the response time of the system, we send a lot of requests and analyze a **distribution of latencies**

Testing on a **real** microservice





MiCozServices

What do we test?

Test effects of speeding services on the `/cart` endpoint, we get the following latency distributions:

1. No speedup

2. After **Virtually** speeding up a service

3. After **Actually** speeding up a service (in our case: **reducing** the amount of work done by it)

Load Testing

Modelling **real usage** of the system by simulating many concurrent **users** sending requests

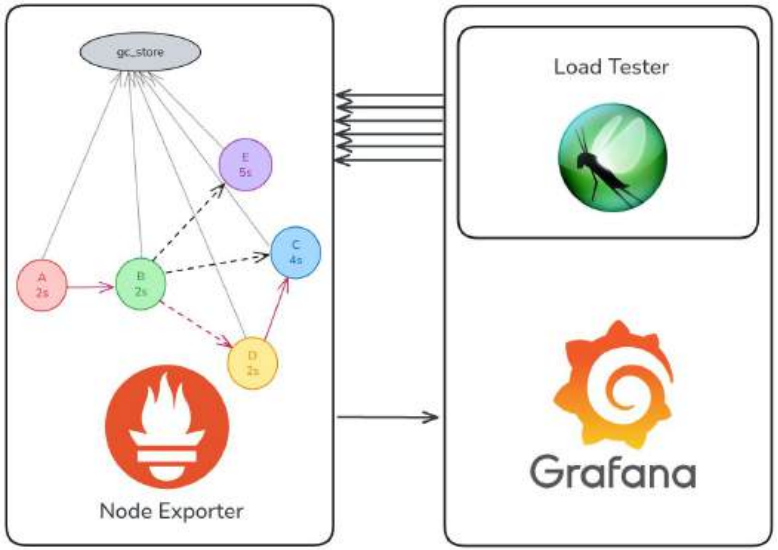
Requests are sent at **constant** Requests Per Second (RPS)

Test Environment

1. Services deployed in containers on one machine.

2. Load tester on another machine, connected via ethernet cable

3. Monitoring CPU, memory usage of services using `prometheus` & `grafana`



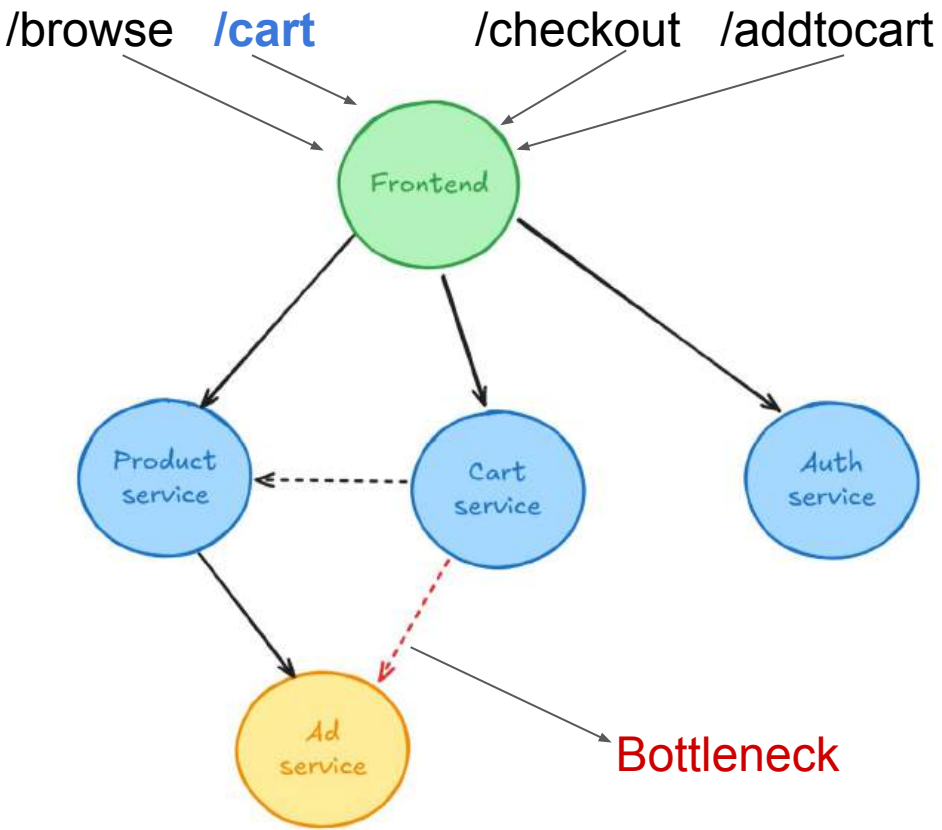
How do we compare distributions of latencies?

1. **Statistical Tests** (Mann Whitney U)
   Compares two distributions to say if values from one are likely to be lesser than values from the other

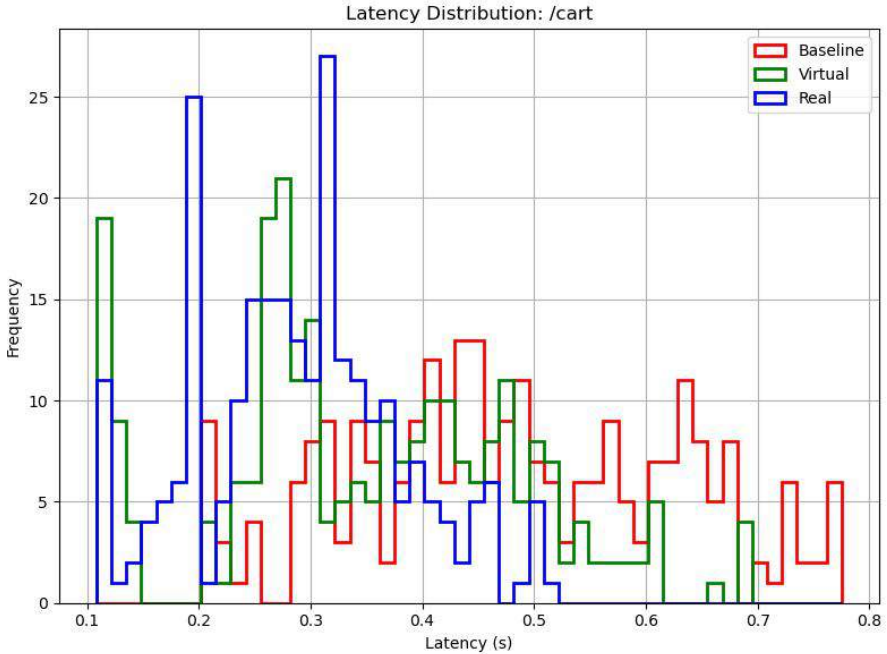
2. **Cumulative Distribution Function**
   Used to estimate percentile latency values of underlying distribution (25th, 50th, 99th percentile etc...)

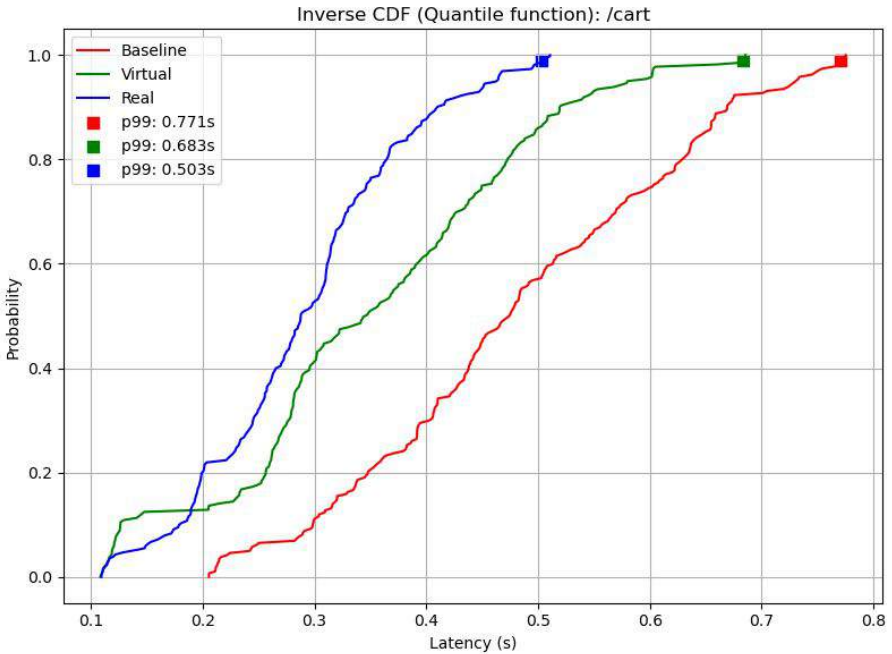
Reminder

Check if: our causal profiling model tells us which service is worth optimizing (`ad` service) and which is not (`product` service)



Speeding up the **Ad Service** by ~60% of it's runtime

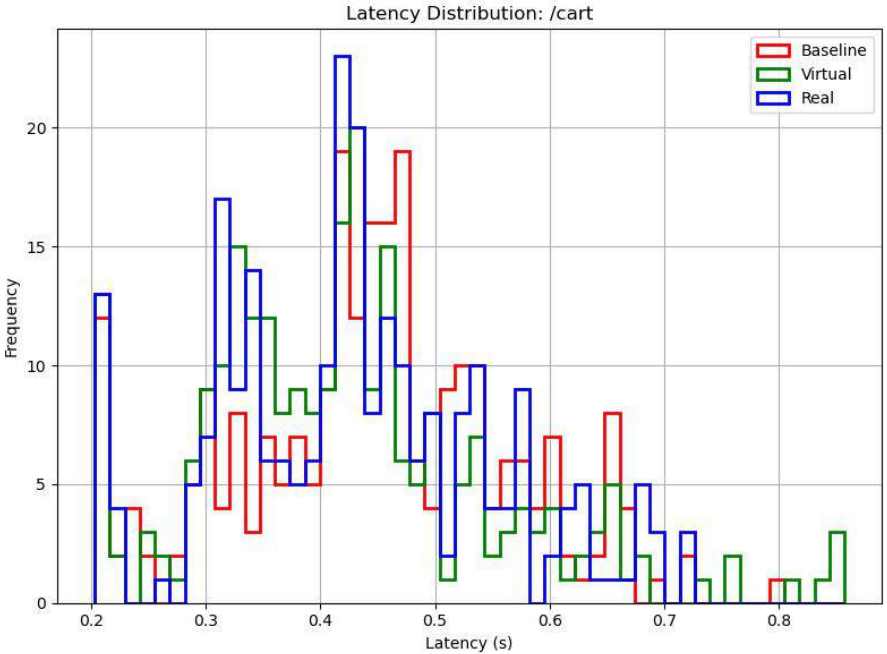


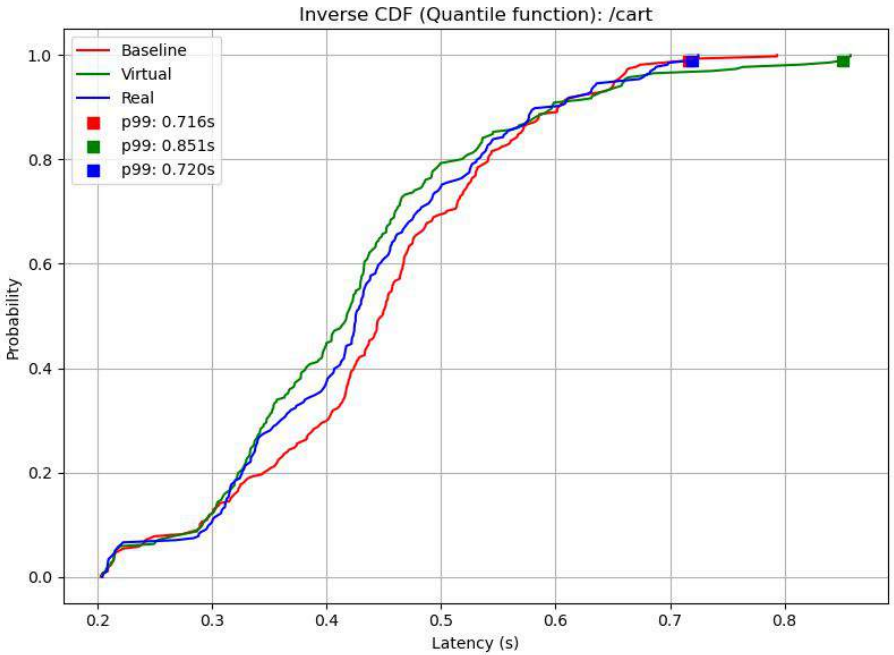


* more left skewed distribution indicates lesser latency

* cdf to the left denotes lesser latency at a percentile

Speeding up the **Product Service** by ~60% of it's runtime





* more left skewed distribution indicates lesser latency

* cdf to the left denotes lesser latency at a percentile

Conclusions

1. It is possible to causally profile some microservice DAGs
2. This principle is also applicable to real (non-deterministic) microservices, using statistical analysis methods.

Future Scope

1. Make virtual speedup more accurate to real speedup
2. Consider more microservice DAGs:
    a. With services make asynchronous (non-blocking) network calls
    b. DAGs with message queues
3. Tool that can be run on microservices without intrusive instrumentation like application layer middleware

References

- Coz paper: https://arxiv.org/pdf/1608.03676v1.pdf
- Coz GitHub: plasma-umass/coz

Questions?

Thank You

MiCozServices