**Querying**

**Traces**

**Admin**

- Midterm scores, etc. will be sent out over weekend, by Monday

- Final Project
  - Haven't really heard back for the cases where I asked for clarification

  - New Task: This week: Post what you plan to work on + rough timeline

  *Start working on Final Project!!*

- State of the course
Topics of interest?

Any interest in student presenters?

Where We Are

- What are traces & how collected
  - Dapper
  - Pivot Tracing
  - Snicker

Potential (but uncommon) uses
- Replay

Real Uses
- Critical Path Tracing
- Find Bottlenecks

Today: "Querying" traces.

- What request traces do you want to look at?
  - Depends on what one is trying to do?
Why is tail-latency high?

- Cannot just look at critical path
- Must decide what requests to check
- Localize what part of a service

Why are requests slower now?

- Pick two or more requests?
- Traces & compare.

But which ones?

What is desirable
Another way to look at the problem is how to build an **Index**?

**Q:** What type of index is useful?

- Depends on common queries.

  - Find requests whose **latency** utilization is $>$ \( \text{abs value 10th percentile average} \)

- Find requests where component has

- Find requests corresponding to some business logic

- Find requests that are affected by a recent code change.

---

Of course we also want to allow people to combine these queries.

Wants indices with two types of information

<table>
<thead>
<tr>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>(business logic, code change, ...)</td>
</tr>
<tr>
<td>( R_1 )</td>
</tr>
<tr>
<td>( R_2 )</td>
</tr>
<tr>
<td>( R_3 )</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

Aggregate

**Statistic**
New Problems

(2) How to update index?

Request \[\rightarrow\] Type

\[\rightarrow\] Compute aggregate

(1) Index selection: What types to use:

\[\rightarrow\] Automatic [iptol] (don't do)

\[\rightarrow\] Semi-automatic [AMTA]

Observations for computing aggregates

\[\rightarrow\] Need to compute aggregates over groups

Q1: How to group?

-Determines what types of aggregates are meaningful

\[\rightarrow\] Or what they show.

Example: 99 percentile of B's latency when called from A must be computed only on A-2B

\[\rightarrow\] Motivates to use B when processing
- Median (put post is where many requests of type X)

- ...

Observe: Can always compute an aggregate on demand, but not efficient.

Desirable: Maintain stats ahead of time by choosing groupings that are likely to be DESIRABLE.

What Groupings are desirable

- Types indexed
  - Request type
  - User
  - Business flow

- Finer grained structures
  - Service
  - Request-level call graph
  - Subspace?

...
What aggregates to maintain?

Ex. Request type

Stats on response latency

Note: Related to but not the same as groupings.

Why?

So really, we are back to the question of what granularity of stats/aggregation are people likely to find useful.

The obvious ones:

- Request type

- By service.

Looking Beyond (sort of)
Similar control/data flow.

Why?

Within a service

```plaintext
... fl (...) lock(x) ? Time spent here? \rightarrow On elsewhere. unlock(x)
```
Why interesting?

Problem,

⇒ Generally do not have this information
⇒ Could add it but...

⇒ Can we infer it?

Answer: Not really, but can try to approximate in some cases.
More generally

- fØ cannot be waiting for an RPC to complete (?)

Problem: Not very accurate.

\[
\begin{align*}
\text{async } f_1(\ldots) & \quad \text{Asyncynchronous RPC call} \\
\text{async } f_2(\ldots) & \quad \text{Asyncynchronous RPC call}
\end{align*}
\]
Can we use previous tricks to do better?

- Aggregate by dependency/service

- Aggregate by “Business Process”
Where We Are

- Have a way to assign type to request

  → Reusing some ideas from before + information in traces
  
  * Request type

  * [Dependency information] → request type

  by mining previous traces

  * Call-graph → Spans

  * ...

- Have a way to aggregate metrics for each type

  → Using into contained directly in traces + some inferred information

Q: How to use any of this?
Great that one can query and get requests, but how do people use it in general?

CPA: Points out bottleneck—allows one to decide where to focus optimization efforts.

But in general often want to determine what is different

How do slow & fast request differ?

How do old & new system differ for requests?

..