Big Data

and

Machine Learning

Spring 2023

Hi, I am Panda.

This is a seminar on Big Data and ML

But we are not really going to look at this.

Why? Too broad, unclear what I can say that is not better said by an ML class.

Instead going to take a page from last year...
Focus on a single area/topic.

Last year: Scheduling

This year: Distributed tracing for debugging & profiling.

Three initial questions

- What?
- Why?
- How?
  \[\rightarrow\text{Class mechanics}\]

What?
Why is Alice unable to search for e-mails about Pizz A?

- Why is Alice unable to search for e-mails about Pizz A?
Often these problems are due to interaction b/w services.

For example:

- Recommendation engine assumes low DB latency
  \(\rightarrow\) makes many queries

- DB latency depends on # of concurrent queries
  \(\rightarrow\) latency increases as # of concurrent queries increase

- A recent search feature increases # of DB & recommendation queries

\(\Rightarrow\) search requires more resources
- Recommendations are slow
Need tools and methods to reason across all of these services—each of which might be comprised of several processes/services.

Distributed tracing is about collating and correlating information across processes.

Requirements?

- Record time—response time
  - Order
- Low overhead
- Always up/fault tolerant
- Log levels
Why?

- Useful across domains
  - Understanding bottlenecks/problems in ML
  - """" in data processing
  - ...

- We can actually generate traces, use & analyze them
  - Task for next class.
  - Doesn't require access to fancy H/W, etc.
  - We have pretty reasonable visibility into what people do in practice.

- There continue to be many open questions & problems that need to be solved.

How?

- Read & learn from
  - A recent book from people at Microsoft, Twitter, etc. on the topic
  - Access online, for free, from NYU library
- **PAPERS**

- **DOCUMENTATION & CODE**

- **GET YOUR HANDS DIRTY WITH COLLECTING & ANALYZING TRACES**

  - No stencil code -
    Really write something that exhibits phenomenon you think are interesting.

  - This week, get started with generating traces from a single process

  - Every few weeks snapshot exercises & submit as HW (3 x 10%)

  - Generating & Collecting Traces

  - Visualizing & Analyzing Traces

  - Using Traces to Debug Problems

- **FINAL PROJECT (25%)**

  - Interesting ways to collect or use traces

  - Interview trace frameworks
- Extensions to existing trace frameworks

- ... 

- Exams (10+20)
  - Understanding of class material

- Participation
  - In class (5%)
  - Helping people online, posting tutorials or instructions, presenting tools/techniques in class (10%)

What goes into a trace

Requirements

- Track causality

What?
Why?

- Support concurrent calls

Stack traces for programs

```
stack
f
b
main
--start
```

`bt` in gdb

```
A
\uparrow
B
\downarrow
\text{5ms}
\text{C}
\downarrow
\text{6ms}
```

return

```
B(C...) + C(C...)
```

RPC calls

Execution time
EXECUTION TIME

- Track time (or other attributes)

THE STRUCTURE OF TRACES

- Use IDs (Request ID or Trace ID) to distinguish between requests.

What is a request?
Within a request, model causality as a tree.

Representing this:

- Request Id
- Span Id
- Parent
- Start Time
- End

- Globally unique
- Unique to a request
Where are these stored?

How collected?

Overheads