RELIABILITY IN MODERN CLOUD SYSTEMS

Summer 2025

LOGISTICS

ASSIGNMENT 1

- Assignment 1 has been released
- Due: Saturday 10th May, 2025 5pm CEST.
- Each student registered on CMS has their own private fork of the assignments repo.
  - If you do not have access to your repo then contact the course staff immediately after class.
  - We will only grade the officially created forks of the main assignments repo.
  - Remember to pull test fixes from the main repository.

ASSIGNMENT 2

- Assignment 2 will be released on Saturday 10th May, 2025
- Due: Saturday 28th May, 2025 5pm CEST.
- Description: Assignment will be adding observability to the luggage sharing application
- We will push an assn2 branch to the main repo
  - Pull from the main repo to get assn2 in your private forks
  - Instructions will be posted

RELIABILITY BASICS DISCUSSION

PAPER SUMMARY

PAPER SUMMARY

- Retry bugs are common!
  - Retry behavior is difficult to test!
- Retry bugs are of the following type:
  - Incorrect decision of retrying
  - Incorrect timing and frequency of retrying
  - Incorrect clean-up and implementation of retrying
- Use LLMs to detect retry locations
  - Use static analysis techniques + unit testing to test retry behavior

DISCUSSION THEMES

- When to retry and how to retry?
- How should a load balancer balance requests across the replica group?
- Cancellations and Deadlines try to reduce wasted work. Which technique should a system employ?
- Are retries good?

DISCUSSION THEMES

- If to retry, When to retry, and How to retry?

DISCUSSION THEMES

- If to retry, When to retry, and How to retry?

If the error is transient, try infrequently and adaptively, while cleaning up any wasted resources and state.

DISCUSSION THEMES

- Are retries good?

DISCUSSION THEMES

- Are retries good?

Retries are only good for transient (recoverable) errors.
Retries are bad for systematic errors.

DISCUSSION THEMES

- Are retries good?

Retries are only good for transient (recoverable) errors.
Retries are bad for systematic errors.

DISCUSSION THEMES

- Cancellations and Deadlines try to reduce wasted work. Which technique should a system employ?

DISCUSSION THEMES

- Cancellations and Deadlines try to reduce wasted work. Which technique should a system employ?

We need both. Deadlines prevent timeout-based wasted work. Cancellations can terminate wasted work more generally.

DISCUSSION THEMES

- How should a load balancer balance requests across the replica group?

DISCUSSION THEMES

- How should a load balancer balance requests across the replica group?

Ideally, load balancer should distribute load equally. This is difficult for many reasons: request variability, diff hardware, etc

OBSERVABILITY

HOW TO KNOW THAT OUR SYSTEM IS BEHAVING AS EXPECTED?

HOW TO KNOW THAT OUR SYSTEM IS BEHAVING AS EXPECTED?

Observability is the ability for users to understand the internal state of the system so that operators can answer the following questions:

- ❖ Did something bad happen in the system?
- ❖ What bad thing happened in the system?
- ❖ Where did the bad thing happen in the system?
- ❖ Why did something happen in the system?

OBSERVABILITY NEEDS DATA

Observability is the ability for users to understand the internal state of the system so that operators can answer the following questions:

- Did something bad happen in the system?
- What bad thing happened in the system?
- Where did the bad thing happen in the system?
- Why did something happen in the system?

Impossible to answer these questions without having access to the relevant data

SERVICE LEVEL INDICATORS (METRICS)

SERVICE LEVEL INDICATORS (METRICS)

Metrics are numerical values measured during execution

- Example: 99^{th} percentile latency, goodput, etc

Metrics are good for answering the question:

- “Did something bad happen in the system?”
- They are cheap, require minimal intrusion, and requires almost minimal effort from developers

MONITORING + ALERTING

Operators configure metrics to automatically monitor the system

- Metrics can be super fine-grained
- Metrics can be customized to detecting 1 specific issue

Deviations from expected metric values produce alerts

- Alerts are passed on to the On-Call Engineers
- OCEs must triage the issue/alert

MONITORING + ALERTING

Operators configure metrics to automatically monitor the system

- ❖ Metrics can be super fine-grained
- ❖ Metrics can be customized to detecting 1 specific issue

Deviations from expected metric values produce alerts

Not all metrics are useful!

LOGS

LOGS

Logs provide more context into what happened in the system

Log data is super rich

- Captures various dimensions of data
- Event-based granularity
- Contains a human provided annotation (print statements or error messages)

LOG PREPROCESSING

….but logs are usually unstructured

- ❖ Mixed between structural and non-structural events
- ❖ Text is free-form
- ❖ Format is inconsistent across systems
  - ❖ Format is inconsistent across services
  - ❖ Format is inconsistent within a service

LOG PARSING

Logs must be parsed to produce clean data that can be used by operators to diagnose and debug their issues

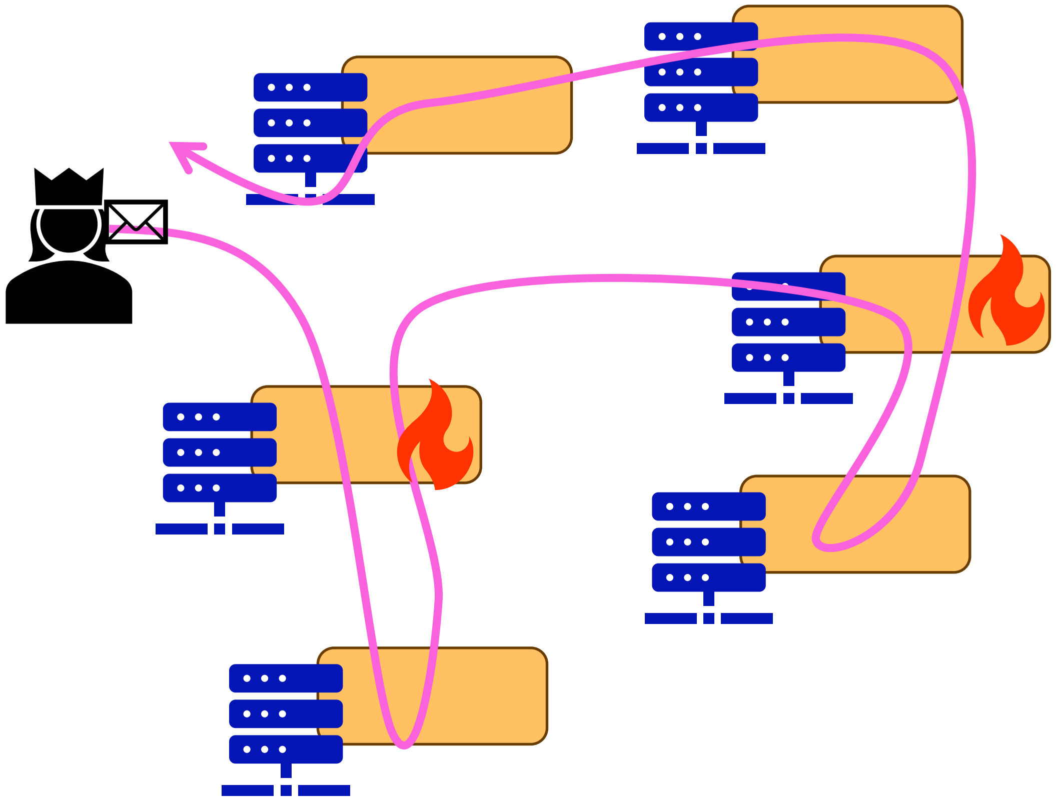
- ❖ Parsing is done in a bespoke form
- ❖ Parsing is not always fully accurate

Unsure what attributes to extract from logs ahead of time

- ❖ Logs are usually stored in raw form
- ❖ Multiple tools provide ability to “search” logs

DISTRIBUTED TRACES

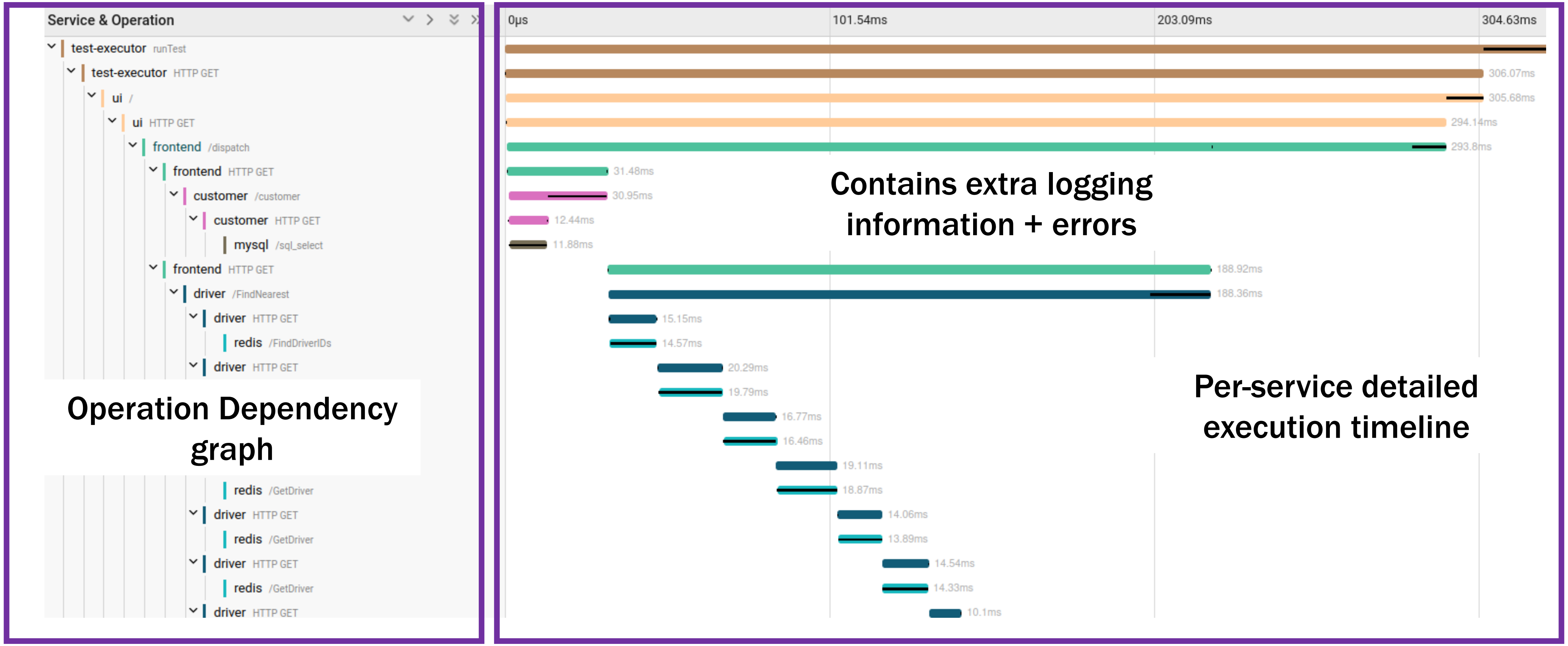
DISTRIBUTED TRACES



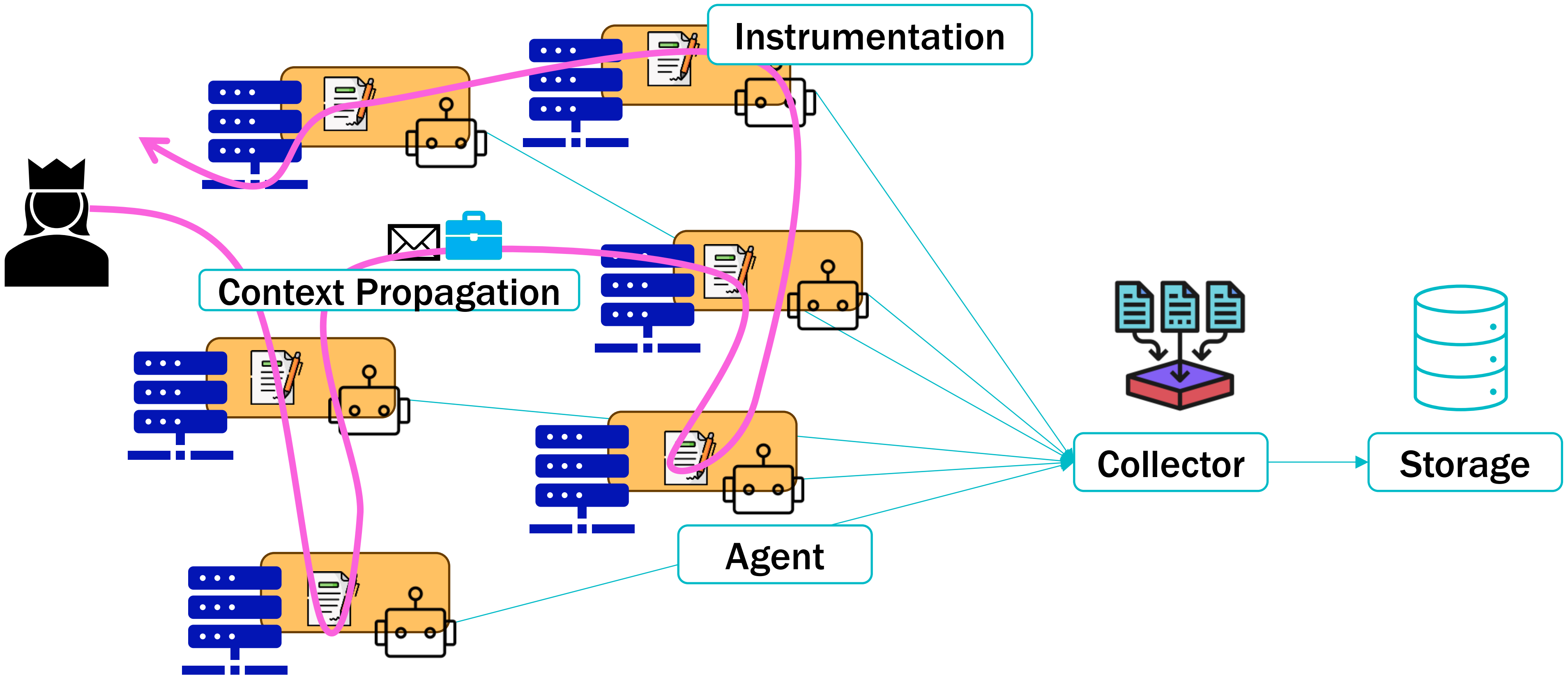
Distributed Trace captures the execution history of a request through the system

- Timing execution info at every node/service
- Partial order of execution across services
- User-defined logs + context

WHAT DOES A TRACE LOOK LIKE?



DISTRIBUTED TRACING COMPONENTS



INSTRUMENTATION & CONTEXT PROPAGATION

INSTRUMENTATION & CONTEXT PROPAGATION

Instrumentation allows request execution across boundaries

- ❖ Generates a unique ID for each request
- ❖ User's "instrument" i.e. modify the code to denote start and end of operations
- ❖ Logging events are linked with the specific operation
- ❖ Tracing libraries capture the duration and errors for each operation

Instrumentation can be done partially automatically but often requires manual intervention for the appropriate granularity

INSTRUMENTATION & CONTEXT PROPAGATION

Request context tracks the state for each request

- ❖ Context encapsulates the unique ID
- ❖ Might include any other request-specific metadata

Context must be propagated across network and process boundaries

- ❖ Must correctly handle various concurrency patterns
- ❖ Standard libraries can do this partially

TRACING BUT AT WHAT COST?

TRACING BUT AT WHAT COST?

Large systems generate large swathes of data

- Facebook generates millions of traces per day

Tracing comes at an overhead

- Network bandwidth
- Instrumentation on the critical path
- Trace Storage

HOW TO DECIDE WHICH REQUESTS TO TRACE AND WHICH TO STORE?

HEAD SAMPLING

HEAD SAMPLING

Decide to trace a request when it enters the system

- ❖ Random decision based on sampling rate (eg: 1%)

Advantage:

- ❖ Efficient as we are not tracing all requests

Disadvantage

- ❖ We won't catch the error requests (decision before execution)

TAIL SAMPLING

TAIL SAMPLING

Decide to trace a request when it leaves the system

- ❖ Decision could be random or based on attributes of the data

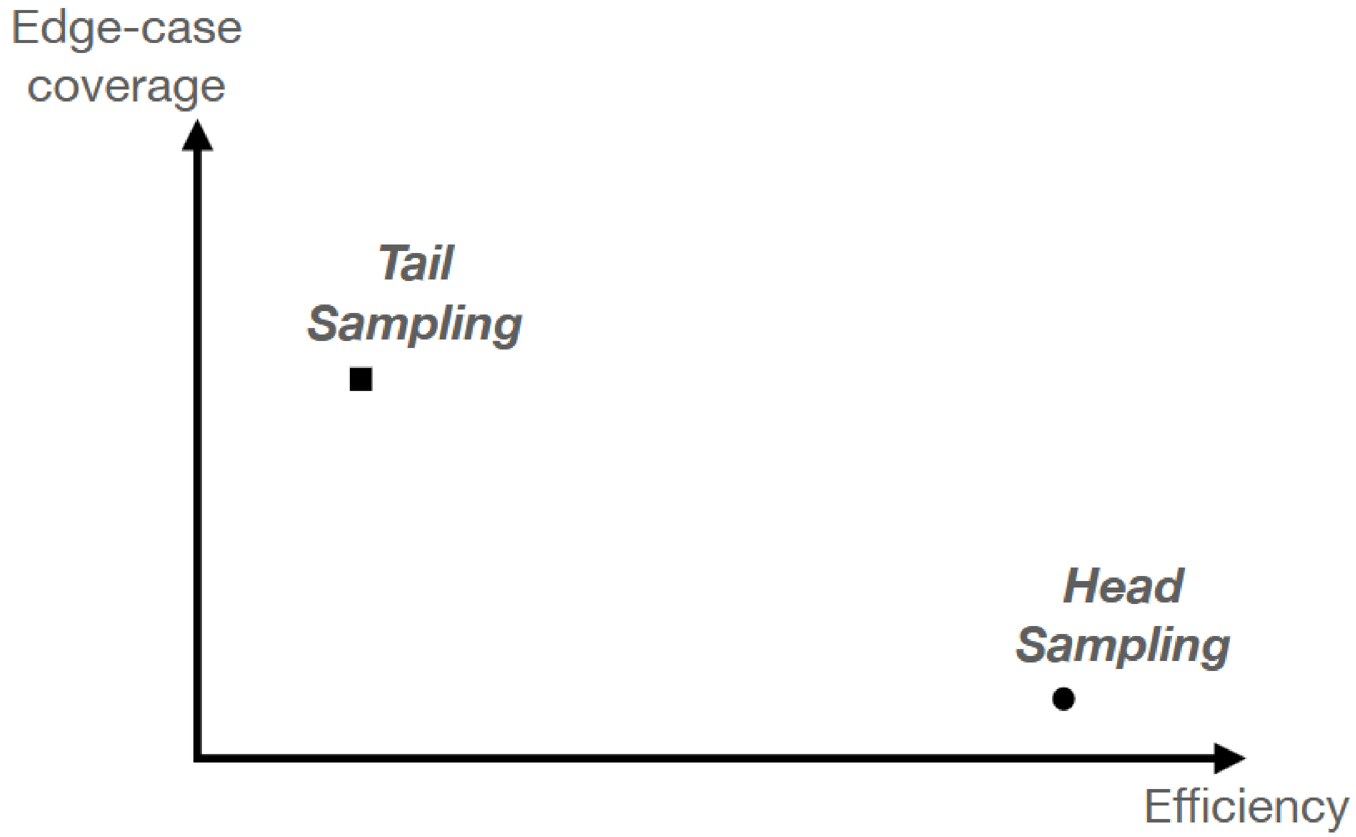
Advantage:

- ❖ We catch the low-frequency error requests

Disadvantage

- ❖ We have to trace everything and collect everything for each request

HEADS OR TAILS?



RETROACTIVE SAMPLING

RETROACTIVE SAMPLING

Data generation is cheap, bottleneck is storing + propagating

Symptoms for error requests can be programmatically detected

The Benefit of Hindsight: Tracing Edge-Cases in Distributed Systems, NSDI 2023, Zhang et. al

RETROACTIVE SAMPLING

Data generation is cheap, bottleneck is storing + propagating

Symptoms for error requests can be programmatically detected

Idea: Trace every request but leave the data ingestion for later

- ❖ Only ingest the trace data iff there was an issue in the request
- ❖ To store trace data for request, any component can trigger the saving mechanism if it detects an issue
- ❖ If request is not triggered, then the old data will be overwritten with new data

The Benefit of Hindsight: Tracing Edge-Cases in Distributed Systems, NSDI 2023, Zhang et. al

TRACE ANALYTICS

A collection of UI tools and scripts that allow users to extract answers to their questions from the underlying data

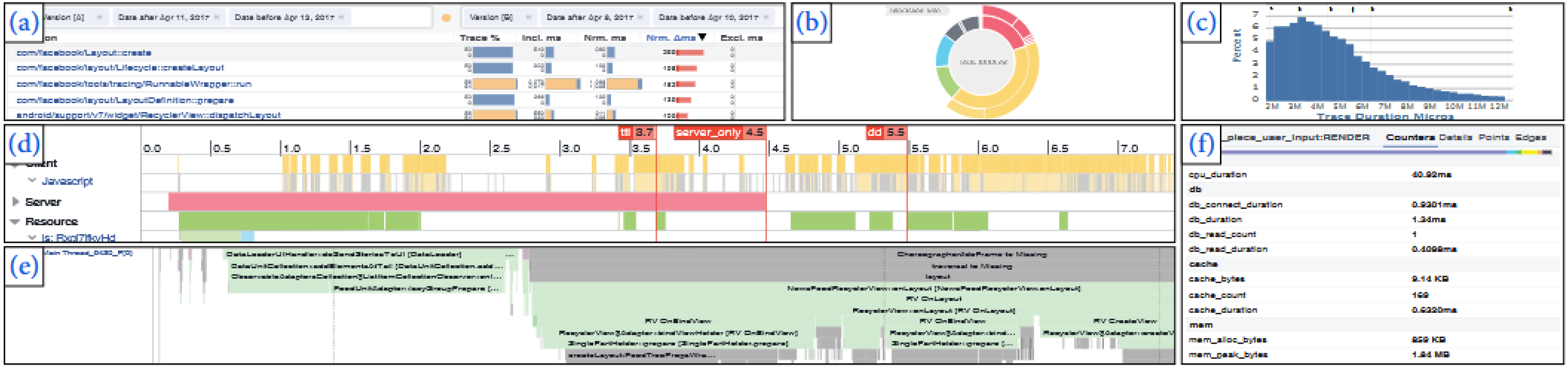


Figure 9: Engineers can use aggregate visualizations (a-c) to explore features. They can continue to drill down to individual traces (d-e) or elements within the trace (f). All visualizations support customizations to focus on relevant data for that view (cf. §4.5)

DISCUSSION THEMES

- If you had a finite operational budget for observability, how would you distribute it among metrics, logs, or traces?
- What is the correct sampling rate for an application? Should this rate be fixed?
- What is the biggest problem with using raw logs?
- Where should you add tracing/instrumentation points in your system?