Diagnosing Distributed CPS with Timing Provenance

Yang Wu  Linh Thi Xuan Phan  Andreas Haeberlen
Problem: Timing faults

- Many CPS are time dependent
  - The “right thing” must happen at the “right time”!

- What if this goes wrong?
  - Reasons: attack, bug, misconfiguration, ...

- Goal: A powerful diagnostic capability
  - Can we find the root cause of both functional and timing issues, such as low throughput, oscillations, high latencies, ...?
Challenge

Root cause!

State of the art

- Distributed tracing: explain what was computed when, but not why
- Network provenance: only reason about functional causality

Cannot reason about timing

@2018 Linh T. X. Phan – Timing Provenance
A generalization of provenance that tracks both functional causality and **temporal causality**

- i.e., causes that affect the timing of the observed symptom
- may involve requests that are functionally independent

Result: Can explain both the ‘what’ and the ‘when’
How to capture temporal causality?

- Intuition: Represent ordering relationship between exec.
  - We need to know not just what the system did, but also in what order (queuing and scheduling semantics)

- Extend critical-path analysis in a novel way for the analysis

Request D can only be dequeued after C is dequeued and finished processing

Timing provenance of D must include C
Insight #1: Sequencing edges

Add a sequencing edge from execution X to execution Y if X immediately precedes Y in the queue.
Challenge: Usability

- Not all executions are equally important
- How to isolate executions that contribute substantially to the overall delay?
Insight #2: Delay annotations

- Annotate vertexes with the delays that they contribute

Slow Compute Response

@2018 Linh T. X. Phan – Timing Provenance
Insight #2: Delay annotations

- Annotate vertexes with the delays that they contribute
- Goal: Delay annotations should correspond to “potential speedup”
Delay annotations: How to compute?

- **Rule 1**: Subdivide delay among the preconditions in the order in which they are satisfied
- **Rule 2**: Attribute the remaining delay to predecessors along the sequencing edge

```
A@X :- B@X, E@Y
B@X :- C@X
E@Y :- C@X
C@X :- Z@X
G@X :- F@X
```

Diagram showing sequence and delays:

- **INS(Z)**
- **SND(+E)**
- **RCV(+E)**
- **INS(F)**
- **SND(+E)**
- **RCV(+C)**
- **INS(Z)**
- **DRV(A)**
- **DRV(G)**
- **DRV(B)**
- **DRV(E)**

Time intervals:

- [0s, 4s]
- [4s, 5s]
- [4s, 6s]
- [0s, 8s]
- [0s, 2s]
- [4s, 7s]
Insight #3: Provenance aggregation

- Aggregating subgraphs that are structurally similar
- Pruning zero-delay subgraphs
Putting everything together

- Detailed and weighted causal explanation of the delay
- Can find off-path root causes!
Implementation, experimental setup

- Zeno, a debugger for timing-related faults

- Support for declarative + imperative systems
  - Interfaces with NDlog and Zipkin
  - Gathers data from switches w/P4

- Evaluation
  - Evaluated with 9 realistic bugs from Google Cloud platform*
  - Used networks that contained 8-700 nodes
  - Results are promising
Evaluation results

- Correctly identifies 11-28 relevant events

11-35 vertexes contributing delay

Produces readable explanations

Size of the provenance for different example scenarios

Diagnosis in less than 10s

Low run-time overhead

Timing provenance is useful, compact and efficient!
Summary: Timing Provenance

- A generalization of provenance to explicitly represent temporal causality
  - The provenance tracks both functional and temporal causality through sequencing edges
  - Delay annotations + provenance aggregation improves usability
  - Applied to RapidNet and Zipkin: Can find off-path root causes

- Benefit: Precise reasoning of both functional and timing faults
  - This will be useful for CPS diagnostics where time matters!

- On-going work
  - Generalize to more complex scheduling policies