Visualizing Distributed Traces in Aggregate

Adrita Samanta and Henry Han

Mentors: Darby Huye, Zhaoqi (Roy) Zhang, Lan (Max) Liu, Prof. Raja Sambasivan
What are Distributed Systems?

Distributed systems are environments where multiple computers work on numerous tasks within a network.

Email example:

- Front End
- Authentication
- Draft Email
- Sending
What is Distributed Tracing?

Tracing is a method of looking into requests in distributed environments.

Each request – any task performed by the system – offers visibility into interactions between services. (Latency, errors, etc.)
What is Distributed Tracing?

Tracing is a method of looking into requests in distributed environments.

Each request – any task performed by the system – offers visibility into interactions between services. (Latency, errors, etc.)

Let’s say you sent an email to a coworker:
Current Issues with Tracing

Developers have a hard time understanding an entire trace dataset.

- Applications like Jaeger allow visualization of one or two traces at a time
- There can be millions of traces/services to look into
Dealing with Millions of Traces

- Debugging and optimization becomes harder for developers without an understanding of the whole trace dataset.

- Software companies can generate millions of traces daily so, combing through all traces can be inefficient. But, often these traces can be similar [1].

- Group together similar traces reduce traces needed to understand overall system
Design

Encoding Traces

Build Similarity Graph

Identify Similar Groups

Inspect Representative Traces

Build Aggregate Group Visualizer
1. Service names - for high level understand of system
2. Full trace topology - for details of services and their requests
3. Latency - for optimizing applications
Defining Trace Similarity

Keep track of service names in the trace.

Similar set of service names \( \rightarrow \) traces are similar

---

Trace 1

- A
- B
- C
- D

Trace 2

- A
- B
- C
- D

Trace 3

- A
- D
- C

Set of service names:

- Trace 1: \{A, B, C, D\}
- Trace 2: \{A, B, C, D\}
- Trace 3: \{A, C, D\}
Measuring Similarity Between Traces

Jaccardian Similarity:

\[ J(A, B) = \frac{|A \cap B|}{|A \cup B|} \]

For two traces (trace_a and trace_b):

- A = set of service names in trace_a, B = set of service names in trace_b
- Threshold = 0.8 (min value of J(A, B) to consider A and B similar)

- J(A, B) > threshold \(\Rightarrow\) similar set of service names \(\Rightarrow\) traces are similar
Build Trace Similarity Graph

Trace 1
{A, B, C, D}  

Trace 2  
{A, B, C, D}  

J(trace1, trace2) = 1 > 0.8
J(trace1, trace3) = 0.75 < 0.8
J(trace2, trace3) = 0.75 < 0.8
Identify Similar Groups

- Use **Disjoint Set Union (DSU)** algorithm to find connected groups in the trace similarity graph.
Methodology

- Created a sample trace set of 24 traces that has variation.
Results

Trace 1

Trace 2

Trace 3

Trace 4

Trace 5

Trace 6

Trace 8

Trace 9

Trace 7

Trace 10

Trace 12

Trace 21

Trace 13

Trace 19

Trace 22

Trace 11

Trace 23

Trace 14

Trace 20

Trace 15

Trace 16

Trace 17

Trace 18

Trace 24
Design

- Encoding Traces
- Build Similarity Graph
- Identify Similar Groups
- Inspect Representative Traces
- Build Aggregate Group Visualizer
Choose a trace from each group that represents the group:
  ○ Trace connected to the most other traces in the trace similarity graph (i.e. trace with the highest degree).
Results
In addition to representative traces, we want to visualize traces such that developers can further their understanding of each group.

We want to highlight major services/interactions to show the most important information about a group.
Collecting Group Data

We want to measure the frequency of each node within the group.

<table>
<thead>
<tr>
<th></th>
<th>Node A</th>
<th>Node B</th>
<th>Node C</th>
<th>Node D</th>
</tr>
</thead>
<tbody>
<tr>
<td># Traces</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Trace 1

Trace 2

Trace 3
Visualizing a Group

Rules:
1. Yellow nodes: present in all traces
2. Gray nodes: present in some traces
3. Node size: corresponds to # traces it is in vs. total # traces in the group.
   a. Node B is $\frac{2}{3}$ the size of Node A
Interacting with Group Visualizations

Say we want to look into Node A:

Rules:

1. Chosen service (A) is highlighted in green
2. Arrows are shown to indicate services which A calls
3. Arrow size corresponds to how often A calls another service
Comparing Groups

Trace 1

Group 2

Trace 2

Trace 3

Group 2 visualization:
Future Work

● Implement other methods of encoding traces.
  ○ Full trace topology
  ○ Latency

● Build trace similarity graph more efficiently

● Implement our aggregate visualization ideas using graph-tool.

Visualizing Distributed Traces in Aggregate

1. Encoding Traces
2. Build Similarity Graph
3. Identify Similar Groups
4. Inspect Representative Traces
5. Build Aggregate Group Visualizer
References


Acknowledgements

We’d like to thank

● Our mentors: Darby Huye, Zhaoqi (Roy) Zhang, Lan (Max) Liu, and Prof. Raja Sambasivan for their guidance and time.

● MIT PRIMES: Dr. Slava Gerovitch and Prof. Srini Devadas for this great opportunity.