Learning to localize faults using fault injection

Speakers:
Jesus Rios, Karthik Shanmugam, Qing Wang
IBM Research
Agenda

• Our approach to fault localization
• Fault Injection based causal learning
• Implementation and experiment results
Background

In practice
• many large distributed applications suffer from limited observability
• making very difficult to quickly find the exact location of a fault

We propose
• using interventional causal learning
  • interventions = fault injections
• to automate the task of localizing the root cause of a fault
• using only application logs data
Two-Phase Cyclic Approach to Fault Localization

- **Staging environment**: Use fault injection to learn an error propagation model
- **Production environment**: Monitor logs and apply model to detect and localize root faults

**Process Flow**
- Change
- Deploy

- Change
- Deploy
Learning Causal Model from Fault Injections

1. Define Fault
2. Inject Fault
3. Sim User-Flow
4. Delete Fault
5. Collect Logs
6. Filter Logs

Causal Model
- Learn causal relations
- Computing statistical correlations of error log locations
- Until enough confidence in the model is achieved

Fault Injection

RHEL OpenShift
WOLFFI
App
Logdna
Empirical Demonstration

Benchmark applications
• Train-Ticket
  • user-flow simulation covered 33 of its 41 microservices
• Day-Trader
  • 5 microservices all covered by user-flow
• Observability assumption
  • access only to microservice logs

Performance measures
• Learning accuracy
  • Learnt causal model vs. true underlying graph
• Localization of new injected faults
  • Accuracy
    • estimate contains true fault location
  • Informativeness -> estimate set size
    100% -> 1
    0% -> max
Fault Injection based Causal Learning

Karthikeyan Shanmugam
IBM Research, NY
Active Learning

What is the best set of node interventions to know Causal Graph G?
**Micro Services and UserFlow Model**

Example: $x_6$ - Variable that counts error logs of microservice 6

$x_6 = f_6(x_5, x_3, u_6)$

Error counts $x_6$ is a function of:

a) whether services 5 and 4 throw an error or not, and

b) whether user flow touched service 6 along with 3 or 5 or both
Micro Services and Userflow Model

P(u) - Distribution of user flows

In general, it's a Pearlian Structural Causal Model where noise variables confound multiple equations.
Micro Services and Userflow Model

Intervention on X2 - stopping the service

If user flow distribution is generic, it should roughly make services both 6 and 3 throw an error

E[X3], E[X6] are good simple tests that identifies ALL descendants
Algorithm for Finding Ancestral Graph

Condition: A Correlation test \((X, Y)\) has sufficient strength \(\text{iff} X\) is a descendant of \(Y\) (an intervened variable)

**Lemma 1.** [Pairwise Conditional Independence Test] Consider a causal graph with latents \(D_\epsilon\). Consider an intervention on the set \(S \subseteq \mathcal{V}\) of observable variables. Then, under the post-interventional faithfulness assumption, for any pair \(X_i \in S, X_j \in \mathcal{V}\setminus S\), \((X_i \not\perp \!\!\!\not\perp X_j)_{D_\epsilon[S]}\) if and only if \(X_i\) is an ancestor of \(X_j\) in the post-interventional observable graph \(D[S]\).
Algorithm for Finding Ancestral Graph

Condition: A Correlation test \((X,Y)\) has sufficient strength iff
\(X\) is a descendant of \(Y\) (an intervened variable)

**Lemma 1.** [Pairwise Conditional Independence Test] Consider a causal graph with latents \(D_{\ell}\). Consider an intervention on the set \(S \subset V\) of observable variables. Then, under the post-interventional faithfulness assumption, for any pair \(X_i \in S, X_j \in V \setminus S\), \((X_i \perp\!\!\!\perp X_j)_{D_{\ell}[S]}\) if and only if \(X_i\) is an ancestor of \(X_j\) in the post-interventional observable graph \(D[S]\).

**Algorithm 1** LearnAncestralRelations- Given access to a conditional independence testing oracle (CI oracle), query access to samples from any post-interventional causal model derived out of \(\mathcal{M}\) (with causal graph \(D_{\ell}\)), outputs all ancestral relationships between observable variables, i.e., \(D_{tc}\).

```
1: function LEARN_ANCESTRAL_RELATIONS(\(\mathcal{M}\))
2: \(\quad E = \emptyset\).
3: \(\quad\) Consider a strongly sep. system of size \(\leq 2 \log n\) on the ground set \(V\) - \(\{S_1, S_2...S_{2[\log n]}\}\).
4: \(\quad\) for \(i\) in \([1 : 2[\log n]]\) do
5: \(\quad\quad\) Intervene on the set \(S_i\) of nodes.
6: \(\quad\quad\) for \(X \in S_i, Y \notin S_i, Y \in V\) do
7: \(\quad\quad\quad\) Use samples from \(\mathcal{M}_{S_i}\) and use the CI-oracle to test the following.
8: \(\quad\quad\quad\) if \((X \perp\!\!\!\perp Y)_{D_{\ell}[S]}\) then
9: \(\quad\quad\quad\quad\) \(E \leftarrow E \cup (X, Y)\).
10: \(\quad\quad\) end if
11: \(\quad\quad\) end for
12: \(\quad\) end for
13: \(\quad\) return The transitive closure of the graph \((V, E)\)
14: end function
```
For single node Fault injections, Condition 1 is trivially true. Injecting Faults in every node will identify all ancestral relationships.
Algorithm for Finding Ancestral Graph

For single node Fault injections, Condition 1 is trivially true. Injecting Faults in every node will identify all ancestral relationships

The graph we return: **Transitive Reduction**

preserves ancestry AND they are causal if correlation tests are accurate. But we don’t return all edges.

References (the above algorithm of finding the ancestral graph is common to all these references first introduced in Ref 1)

Experiments

Set-up

• Two benchmark microservice applications are deployed on OpenShift Container Platform (OCP).
  • DayTrader: an online stock trading system.
  • TrainTicket: a train ticket booking system.

• logDNA service is installed in the OCP cluster to collect logs.

Fault Injection

• User-flow simulation
  • DayTrader
    • All five microservices can be covered by the simulated user-flow.
  • TrainTicket
    • 31 of all 41 microservices are covered.

• Fault injection framework
  • Developed in Python
  • Fault types: Http faults, latency and saturation faults, etc.
Experiments

Log Data Generation

- For both two applications, we inject one fault at a time and run the user-flow.

Log Data Preprocessing

- Label each log line by defined error patterns (e.g., “http 500 internal server error”)
  - Error logs
  - Normal logs

<table>
<thead>
<tr>
<th>Log File of microservice C</th>
</tr>
</thead>
<tbody>
<tr>
<td>_time</td>
</tr>
<tr>
<td>t</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>T</td>
</tr>
</tbody>
</table>

Table 1. The observed log data by injecting fault to microservice C. Microservice A and B are dependent on C.
Experiments

Causal Learning Performance

• Evaluation metrics:
  • Precision, recall, F1-score and SHD (structural hamming distance).
  • $\tau$ and bin_size

<table>
<thead>
<tr>
<th>Application</th>
<th>$\tau$</th>
<th>SHD</th>
<th>Precision</th>
<th>Recall</th>
<th>$F_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>DayTrader</td>
<td>0.01</td>
<td>0</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td>0.03</td>
<td>0</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>0</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>2</td>
<td>0.75</td>
<td>0.75</td>
<td>0.75</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>4</td>
<td>0.50</td>
<td>0.50</td>
<td>0.50</td>
</tr>
<tr>
<td>TrainTicket</td>
<td>0.01</td>
<td>36</td>
<td>0.68</td>
<td>0.54</td>
<td>0.60</td>
</tr>
<tr>
<td></td>
<td>0.03</td>
<td>33</td>
<td>0.73</td>
<td>0.54</td>
<td>0.62</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>37</td>
<td>0.71</td>
<td>0.44</td>
<td>0.54</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>52</td>
<td>0.40</td>
<td>0.08</td>
<td>0.13</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>51</td>
<td>0.33</td>
<td>0.22</td>
<td>0.04</td>
</tr>
</tbody>
</table>

Table 2. Fault Injection Causal Learning results by comparing with transitive reduction of ground truth with the setting bin_size=1s.

Fault Localization Performance

• We build the ancestral relationships using the causal learning results in the error propagation for localizing a new fault.

<table>
<thead>
<tr>
<th>$\tau$</th>
<th>Accuracy (%)</th>
<th>Informativeness (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.03</td>
<td>93.75</td>
<td>86.93</td>
</tr>
<tr>
<td>0.1</td>
<td>90.63</td>
<td>79.55</td>
</tr>
<tr>
<td>0.4</td>
<td>15.63</td>
<td>21.69</td>
</tr>
</tbody>
</table>

Table 3. Fault localization performance on TrainTicket using the output of Algorithm 1 with different $\tau$ values and the fixed bin size = 1,000 ms in terms of Accuracy and Informativeness.
Experiments

**Figure 5.** (a) Adjacency matrix of ground truth; (b) Adjacency matrix of transitive reduction (TR) of ground truth; (c) Adjacency matrix of Algorithm 1 ($r=0.03$, bin.size=1,000ms) before Step 12; (d) Adjacency matrix of Algorithm 1 ($r=0.03$, bin.size=1,000ms).

e.g. ui-dashboard -> ticketinfo
Experiments

Figure 4. (a) and (b) are ground truth graph and transitive reduction of it. (g) and (h) are the outputs of fault injection causal learning results with different $\tau$ values and the best bin_size=1s.
Thank you

Jesus Rios
Research Scientist
IBM Research AI
jriosal@us.ibm.com

Karthik Shanmugam
Research Scientist
IBM Research AI

Qing Wang
Postdoctoral Researcher
AI for IT Ops - Hybrid Cloud
IBM Research

Laura Shwartz
Distinguished Engineer
AI for IT Ops - Hybrid Cloud
IBM Research

Naoki Abe
Distinguished Research Scientist
IBM Research AI