Root Cause Analysis (RCA) for Future Networks:
Technology, Vision and Challenges

Armen Aghasaryan, Bell Labs, Paris-Saclay

Shonan Seminar on Causal Reasoning in Systems
24-27 June, 2019
Management of Future Networks with Root Cause Analysis

Introduction

The digital era calls for an unprecedented need of real-time monitoring and Root Cause Analysis (RCA) of complex systems deployed at a global network scale.

**RCA duality** character: RCA is applied to methodically identify and correct the *root causes of events*, as opposed to simply addressing their *symptomatic result*.

**RCA impact** perspective: “Root cause” may be described as the point in a causal chain where applying a corrective action or intervention would prevent the problem from occurring.

Model-based approach

- Model explains the occurrence, manifestation, and propagation fault situations
- Model learning techniques: interventional / observational / knowledge-based

**Correlation is not Causation**
RCA for Cloudified Networks

High volume of measurements and alarms to be handled at different layers
• Relevant alarm definitions and measurements not trivial to identify

Ambiguity in the interpretation of observations by the human operator
• Single fault may produce multiple alarms, a given alarm can be caused by different fault conditions
• Number of possible failures grows exponentially in the future (terminals – IoT, network components – 5G)

Delayed understanding of faults leads to negative impact on businesses
RCA for Cloudified Networks
Example of a Web app

Symptoms: (web) application performance degradation - generation of infrastructure and application level alarms

Multiple possible underlying causes
- User request peak or permanent increase (App extra demand)
- Underlying resource overload by other tenants (host network interface, host CPU saturated)
- Remote fault propagation via app dependencies (e.g. BD delay)
- Remote host failures, etc.
Fault model-based approach

Setting up the RCA design as a model-based fault diagnosis using a “model inversion” approach

• Fault model replicates the fault behavior of the network
  - representation of initial faults, fault propagation, and alarm generation
  - different types of models representing the interplay between observed variables and hidden states
    • Labelled automata, Petri Nets, Pattern matching, Hidden Markov models, Bayesian Networks, ...

• Diagnosis algorithms inverse the model
  - without reconstructing the global state (to avoid state space explosion)
Framework: from model discovery to fault diagnosis

- **Automated (fault) model discovery modes**
  - Interventional learning: stimulus-based App Profiling Sandbox
  - Observational learning: lagged correlation analysis & pattern learning
  - Self-modelled data/systems, e.g. perf. logs allowing auto-extraction of a Bayesian Network structure

- **Model Inversion: from symptoms to causes**
  - Identification of possible causes - provides the most likely explanations given the observed alarms.
  - Using Bayesian network inference, tile-based trajectory composition / Petri Net unfolding, Viterbi-like algorithms, etc.

**Model discovery modes**

- Interventional Learning
- Observational learning
- Parameter Inference

**Model inversion**

- Symptom Explanations
Interventional Learning
Stimulus-based App profiling

Intervention generation

Correlation analysis

Causal modeling

Symptom Explanations
Stimulus-based App Profiling Sandbox

**Goal:** Automated fault model discovery

**Approach:** Stimulus creation framework for virtualized distributed apps
- Interventional learning of causal Tiles via systematic resource perturbations and their impact analysis
- Algorithm for causal chaining of fault trajectories

---

**Intervention Generation**

**“Stimulus” creation:** system-level fine-tuned control of available computing resources on a given app node (CPU, RAM, Disk I/O, Ntwk I/O)

**Discovery of causalities:** analysis of the reactions of other app nodes to the current stimulus
Stimulus-based modeling

Correlation analysis

One datasheet per stimulus

DB     WS1    WS2    LB
hosts

Positive correlations
Negative correlation

DBserver-v2
WebServer1
WebServer2
WordPress-LB
blacksabbath
deeppurple
pinkfloyd

LB
hosts
Stimulus-based modeling

Causal Modeling: Interpretation of correlations given the stimulus

Current stimulus is on DB: network Bw down
Stimulus-based modeling
Tile extraction

Hidden faults (primary or propagated)
- we stimulate (and know) the ground truth
  • Network Bw, CPU, Memory, Disk I/O
  • Input Load

Hidden states – alert enablers
- enable alarm generation
- defined by OpenStack & proprietary meters

Visible events - alarms
- defined on the basis of the existing meters
Stimulus-based modeling

Tile extraction: example of stimulus on DB network bandwidth down

<table>
<thead>
<tr>
<th>resourceType</th>
<th>resourceID</th>
<th>meterType</th>
<th>CorrCoeff</th>
<th>resourceType</th>
<th>resourceID</th>
<th>meterType</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB</td>
<td>db_server_v2</td>
<td>network.outgoing.bytes.rate</td>
<td>0.87</td>
<td>WS</td>
<td>WebServer1</td>
<td>network.incoming.bytes.rate</td>
</tr>
<tr>
<td>DB</td>
<td>db_server_v2</td>
<td>network.outgoing.bytes.rate</td>
<td>0.85</td>
<td>WS</td>
<td>WebServer1</td>
<td>cpu_util</td>
</tr>
</tbody>
</table>

**Primary Cause Occurrence:**
Network BW down

**Local Propagation:**
Directly impacted meters

**Alarm(out.bytes.rate)**

**Primary Cause Occurrence:**
Insufficient CPU

**Local Propagation:**
Directly observed meters

**Alarm(cpu_util)**
Stimulus-based modeling

Diagnosis

- **Interventional learning of causal Tiles:**
  stimulus-reaction transitions (tiles) represent causal relationships between resource stimuli and the reactions at local & remote nodes

- **Causality chaining via net unfolding:**
  the alarm flow guides the on-line composition of reusable tiles into hypothetical fault propagation trajectories

- **Fault trajectory inference & diagnosis**
  Iterative selection of the best explanations (trajectories) using Max likelihood estimation algorithms (Viterbi algorithm variants)
Observational learning
Correlation and Causality Analysis with CloudBand (CB) alerts
Causality inference from CB alert logs

**Goal:** automatic inference of correlation & causality rules from observed alert logs

**Approach:**
- Optimized computation of type-based correlations
- Causality inference based on time lagged correlations

Implemented by OpenStack Vitrage
Presented at OpenStack Summit 2017, Boston
Causality inference from CB alert logs

Correlation Analysis

Datasets from CB:
- e.g. Cloud platform @Naperville: ~ 14K alarms, 1,5K distinct resources

<table>
<thead>
<tr>
<th>resourceType</th>
<th>resourceId</th>
<th>alertType</th>
<th>activeTimestamp</th>
<th>updateDate</th>
<th>inactiveTimestamp</th>
<th>close</th>
</tr>
</thead>
<tbody>
<tr>
<td>MACHINE</td>
<td>12345</td>
<td>VM_STORAGE_SUBOPTIMAL_PERF</td>
<td>05/08/2017 08:09</td>
<td>05/08/2017 08:20</td>
<td>05/08/2017 08:30</td>
<td></td>
</tr>
</tbody>
</table>

Computing correlations between alert time series:
- identified by <ResType x ResId x AlertType>
- using co-occurrence measures: Jaccard, Pearson,…
- filtering according to topological constraints
Causality inference from CB alert logs

Correlation Analysis (cont.)

Identify frequent co-occurrence patterns in terms of Types

- using Spark computing

Sparse representation of the Correlation matrix

Inferred set of co-occurrence rules

Group by Types

Aggregation functions (counter, mean)

Threshold constraints
Causality inference from CB alert logs

Causal analysis

Transform co-occurrence rules into causality rules

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>Alert Type</th>
<th>Agg. Value</th>
<th>Resource Type</th>
<th>Alert Type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Symmetric co-occurrence rules

Lagged correlation Sensitivity tests

expert knowledge on Resource & Alert Type semantics (if available)

Suggested causality rules

Maintain unresolved co-occurrences without causality
### Correlation threshold: 0.9

<table>
<thead>
<tr>
<th>resourceType</th>
<th>alertType</th>
<th>resourceType</th>
<th>alertType</th>
<th>Count</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOST</td>
<td>HOST_CEPH_PROBLEM → MACHINE</td>
<td>VM_STORAGE_SUBOPTIMAL_PERF.</td>
<td></td>
<td>5742</td>
<td>0.96</td>
</tr>
<tr>
<td>HOST</td>
<td>HOST_HIGH_CPU_LOAD → MACHINE</td>
<td>VM_CPU_SUBOPTIMAL_PERF.</td>
<td></td>
<td>47</td>
<td>0.98</td>
</tr>
</tbody>
</table>

### Correlation threshold: 0.7

<table>
<thead>
<tr>
<th>resourceType</th>
<th>alertType</th>
<th>resourceType</th>
<th>alertType</th>
<th>Count</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOST</td>
<td>HOST_CEPH_PROBLEM → MACHINE</td>
<td>VM_STORAGE_SUBOPTIMAL_PERF.</td>
<td></td>
<td>23490</td>
<td>0.96</td>
</tr>
<tr>
<td>MACHINE</td>
<td>FAILED_TO_GET_METRICS ~ MACHINE</td>
<td>VM_STORAGE_SUBOPTIMAL_PERF.</td>
<td></td>
<td>5953</td>
<td>0.72</td>
</tr>
<tr>
<td>HOST</td>
<td>RESOURCE_INVALID_STATE ~ MACHINE</td>
<td>VM_STORAGE_SUBOPTIMAL_PERF.</td>
<td></td>
<td>5525</td>
<td>0.77</td>
</tr>
<tr>
<td>HOST</td>
<td>HOST_CEPH_PROBLEM ~ HOST</td>
<td>RESOURCE_INVALID_STATE</td>
<td></td>
<td>232</td>
<td>0.77</td>
</tr>
<tr>
<td>HOST</td>
<td>HOST_HIGH_CPU_LOAD → MACHINE</td>
<td>VM_CPU_SUBOPTIMAL_PERF.</td>
<td></td>
<td>47</td>
<td>0.98</td>
</tr>
<tr>
<td>HOST</td>
<td>RESOURCE_INVALID_STATE ~ MACHINE</td>
<td>FAILED_TO_GET_METRICS</td>
<td></td>
<td>8</td>
<td>0.73</td>
</tr>
<tr>
<td>CLOUD_NODE</td>
<td>RESOURCE_INVALID_STATE ~ COMPONENT</td>
<td>RESOURCE_INVALID_STATE</td>
<td></td>
<td>6</td>
<td>0.82</td>
</tr>
<tr>
<td>MACHINE</td>
<td>FAILED_TO_GET_METRICS</td>
<td>CLUSTER_INST.</td>
<td>SYSTEM_ERROR</td>
<td>1</td>
<td>0.85</td>
</tr>
<tr>
<td>HOST</td>
<td>HOST_CEPH_PROBLEM ~ MACHINE</td>
<td>VM_STORAGE_SUBOPTIMAL_PERF.</td>
<td></td>
<td>6608</td>
<td>0.75</td>
</tr>
<tr>
<td>HOST</td>
<td>HOST_CEPH_PROBLEM ~ HOST</td>
<td>RESOURCE_INVALID_STATE</td>
<td></td>
<td>232</td>
<td>0.77</td>
</tr>
<tr>
<td>HOST</td>
<td>RESOURCE_INVALID_STATE ~ MACHINE</td>
<td>FAILED_TO_GET_METRICS</td>
<td></td>
<td>60</td>
<td>0.64</td>
</tr>
<tr>
<td>HOST</td>
<td>HOST_CEPH_PROBLEM ~ MACHINE</td>
<td>FAILED_TO_GET_METRICS</td>
<td></td>
<td>58</td>
<td>0.56</td>
</tr>
<tr>
<td>HOST</td>
<td>HOST_HIGH_CPU_LOAD → MACHINE</td>
<td>VM_CPU_SUBOPTIMAL_PERF.</td>
<td></td>
<td>47</td>
<td>0.98</td>
</tr>
<tr>
<td>CLOUD_NODE</td>
<td>RESOURCE_INVALID_STATE ~ COMPONENT</td>
<td>RESOURCE_INVALID_STATE</td>
<td></td>
<td>6</td>
<td>0.82</td>
</tr>
<tr>
<td>CLOUD_NODE</td>
<td>METRICS_READING_ERROR ~ HOST</td>
<td>RESOURCE_INVALID_STATE</td>
<td></td>
<td>2</td>
<td>0.53</td>
</tr>
<tr>
<td>CLUSTER_INST.</td>
<td>MODULE_ERROR → CLUSTER_INST.</td>
<td>SYSTEM_ERROR</td>
<td></td>
<td>2</td>
<td>0.69</td>
</tr>
</tbody>
</table>
Causality inference from CB alert logs

Diagnosis

Topology based RCA templates [Nokia CloudBand / OpenStack Vitrage project]

- Expert rules + **Inferred Rules** + dynamic instantiation of RCA templates => root causes
Inferring propagation scenarios from alarm logs (Radio Access Network)

Complexity reduction and automated causal analysis

**Goal:** Reducing the time to dispatch KPI (time to resolve an incident)

**Approach:** Structural and Causal analysis of the alarm “chaos”
- Using statistical causal inference techniques enforced with exogeneous knowledge, e.g. physical/logical topology, or expertise

**Impact:** Number of possible explanations reduced by 10K

**NSW DI demonstrator on a smartphone**

Presented at Global Analyst Forum 2018
Inferring propagation scenarios from alarm logs (2)

Complexity reduction and automated causal analysis

Dataset: large radio access network
- 200K network elements, 1 Million alarms every day (40 000 alarms/ hour), several hours per incident, up to 24 hours in some cases

Structural and Causal analysis of the alarm “chaos”

Using statistical causal inference techniques enforced with exogeneous knowledge injection (e.g. physical/logical topology, or expertise)
- Dynamic correlation graphs based on local/vertical/remote resource relations
- Graph summarization
- Number of possible explanations reduced by 10k
Root Cause Analysis of Future Networks

Summary of important challenges

Model-based approaches are challenges by the fact that in real systems models are often unknown!

- Fault injection and interventional approaches to causality discovery - *completeness, combinatorics*
- Observational analysis / statistical causal inference - *missing sufficient fault data volumes*
- Observational analysis assisted by some structural /topological knowledge - *partial topology knowledge*

Problem translation into the event space : adequate anomaly detection mechanisms

- Not specified in complex cloud-based environments
- Need for methods coupling anomaly detection and fault diagnosis

Challenges for Model Learning and Diagnosis

- High-dimensionality
- Dynamically reconfigurable systems, dynamically tracking the changes in the model
- Partially specified and partially observed models (controlled environments)
- Non-causal observation
- High-responsiveness : self-healing control loops
Fault model-based approach

Tile-based model

- elementary fault behaviors of various node types: <pre-conditions, alarm/event, post-conditions> 

Petri Net semantics

- distributed state / local conditions
- concurrency

Uncertainty representation

- unobserved tiles
- likelihood measure

Model execution: fault trajectory building & diagnosis

- Max likelihood estimation / Viterbi algorithm variants
- Stream-based puzzle assembling using tiles (Spark Streaming)
Trajectory building

**Trajectory 1**

**Primary Cause:** Network BW down

**Alarm:**

- `out.bytes.rate` → `DB` → `DB` → `...` → `WS1` → `WS1` → `...` → `CPU` → `WS2` → `WS2` → `...` → `CPU` → `Alarm(cpu_util)`

**Alarm:**

- `out.bytes.rate` → `DB` → `DB` → `...` → `WS1` → `WS1` → `...` → `CPU` → `Alarm(cpu_util)`

**Primary Cause:** Insufficient CPU

**Alarm:**

- `cpu_util` → `WS1` → `Alarm(cpu_util)`

**Trajectory 2**

**Primary Cause:** Insufficient CPU

**Alarm:**

- `cpu_util` → `WS1` → `Alarm(cpu_util)"
Model discovery in controlled environments
Stimulus-based modeling for Cloud: Open questions

Verification & testing of derived tiles
- Model properties, coverage of real system faults and behaviors

Automatic learning of “complex” behaviors
- Forward propagation “Conflict” behaviors,
- “Synchronization” behaviors induced by the co-occurrence of multiple faults

Diagnosis algorithms under non-causal ordering of observed events
- On-line vs batch processing with sliding window, distribution vs stream analytics

Dealing with dynamically reconfigurable systems, e.g. auto-scaling resources
- Relevance of the tile model learned in the Sanbox => asymptotic analysis of correlations, parametric tile model?
- (distributed) diagnosis algorithms for reconfigurable models (changing the number of resources)

Dealing with partially modeled systems, e.g. unknown types of resources (not covered by the model)
- On-line micro-stimulus approach, ...?