Virtual Machine Failure Prediction using Log Analysis

MS Thesis Defense

Sukhyun Nam
Supervisor: Prof. James Won-Ki Hong
DPNM Lab, CSE, POSTECH, Korea
obiwan96@postech.ac.kr
Jun. 22, 2021
Table of Contents

◆ Introduction
◆ Background & Related work
◆ Methodology & Implementation
◆ Experiment & Evaluation
◆ Conclusion
Introduction
Network Function Virtualization (NFV)

- NFV technologies decouple functions (e.g. firewall, load balancer etc.) from hardware and move them to virtual servers.
- NFV reduced OPEX and CAPEX.
- Made difficult to monitor and take action on virtual machines (VMs) and server failures.
  - Faults in clouds system can take hours and days to fix [1]
Introduction

◆ VM Failure Prediction Tasks

- Predict the failures in VM in advance, to use in live migration of VNF before the failures occur to minimize service quality degradation
- Some of the failures have early errors or faults associated with
  - Errors or faults of computer equipment can be found in the log

◆ Challenges of Failure Prediction Tasks

- Complicated failure causes
- Complex failure-indicating signals
- Highly imbalanced data
Introduction

◆ Research Goal

❖ Predict at least one minute before a failure occurs using the logs that the VM outputs
  ● Failure is a state that the VM fails to network function
  ● Live migration takes an average of 45 seconds before based on VMs with size of 5GB on OpenStack
Background & Related work
Background

◆ Network Failure Prediction

❖ W. Ji et al (CCDC 2018) [2]
  ● Predict whether logs contain failure messages in wireless communication systems
  ● CNN showed best performance when experimenting with GRU, LSTM, CNN
  ● Accuracy 0.75 with gap 2000, accuracy 0.57 with gap 5000

❖ MING (ESEC/FSE 2018) [3]
  ● Predict node failure before 6 hours in cloud service
  ● Use temporal features (e.g. performance counters, resource usage) and spatial features (e.g. rack location, load balance group, update domain)
  ● Average recall of 0.63, precision of 0.92 and F1 score of 0.75

◆ Log based Anomaly Detection

❖ Deeplog (SIGSAC 2017) [4]
  ● Use deep neural network (DNN) to learn log patterns from normal execution
  ● Show F1 score of 0.98 in the OpenStack data set
Background

 Fault and Failure [5, 6]

- Mandelbug is a kind of bug whose activation and propagation are complex
  - Hard to reproduce
  - Takes longer time to fix than regular bugs
- In Linux, many failures related to networking are caused by Mandelbug [6]
- Two types of Mandelbug generate early symptoms
  - ARB (Aging Related Bug)
    - A kind of bugs that can cause an increasing failure rate and/or degraded performance, known as software aging
    - Symptoms: errors or faults due to overload (memory leaks or increase in total system runtime)
  - LAG
    - A kind of bugs that are non-aging related Mandelbug (NAM), but there exist a time lag between the activation of the bug and the occurrence of its failure
    - Symptoms: variety
Background

◆ CNN (Convolutional Neural Network)
  
  - Artificial neural networks specialized for learning that extract features without losing information from large amounts of data
  
  - Typically contains multiple convolution layers and pooling layers
  
  - The operation is simple and the number of parameters is small
  
  - CNN performs best in studies on sentence classification problems [7]
Methodology & Implementation
Overview of VM failure prediction model development

**Data Generation**
- Traffic Generation
  - Logs from VNF, systemd and kernel

**Preprocessing**
- Log Pre-processing
  - Sentence embedding within input window size

**Model Generation**
- Failure Tagging
  - Failure history
    - Failure info of time after gap
- CNN Model Training
- Model Evaluation

**Input**
- Sentence embedding within input window size

**Output**
Methodology

- Two sliding windows for input and output

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Logs</th>
</tr>
</thead>
<tbody>
<tr>
<td>05-21 02:09</td>
<td>autorefresh go cannot prepare auto</td>
</tr>
<tr>
<td>05-21 02:10</td>
<td>storehelpers go cannot refresh</td>
</tr>
<tr>
<td>05-21 02:11</td>
<td>our onion service received v and</td>
</tr>
<tr>
<td>05-21 02:12</td>
<td>snapd parts snapd deb build</td>
</tr>
<tr>
<td>05-21 02:13</td>
<td>stopped network service</td>
</tr>
<tr>
<td>05-21 02:14</td>
<td>tor has been idle for seconds</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>05-21 02:09</td>
<td>Normal</td>
</tr>
<tr>
<td>05-21 02:10</td>
<td>Normal</td>
</tr>
<tr>
<td>05-21 02:11</td>
<td>Normal</td>
</tr>
<tr>
<td>05-21 02:12</td>
<td>Normal</td>
</tr>
<tr>
<td>05-21 02:13</td>
<td>Normal</td>
</tr>
<tr>
<td>05-21 02:14</td>
<td>Failure</td>
</tr>
</tbody>
</table>

Input window size

Gap

Prediction object window
Input

◆ Pre-processing

- Remove numbers and replace symbols with space
- Translate time info as timestamps and remove VM name, application name
- Delete duplicate log

```
May 21 02:10:13 225-2c-4 snapd[2463]: storehelpers.go:551: cannot refresh: snap has no updates available: "core18", "lxd", "snapd"
May 21 02:10:14 225-2c-4 snapd[2463]: autorefresh.go:479: auto-refresh: all snaps are up-to-date
```

```
<table>
<thead>
<tr>
<th>Timestamps</th>
<th>Logs</th>
</tr>
</thead>
<tbody>
<tr>
<td>05-21 02:10</td>
<td>storehelpers go cannot refresh snap has no</td>
</tr>
<tr>
<td></td>
<td>updates available core lxd snapd</td>
</tr>
<tr>
<td>05-21 02:10</td>
<td>autorefresh go auto refresh all snaps are</td>
</tr>
<tr>
<td></td>
<td>up to date</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Word Embedding

- Express a word as a dense vector with preserving the characteristics of the word
- Public word embedding is not appropriate for log analysis

Embedding Vectors for Log Corpus

- Generated with Google’s open-source project word2vec [8]
- Contains 265,452 words
- ex) most similar words with ‘err’
  - errors, over, dropped, rx, crc, tx, collisons, miss

Input

<Corpus>

- Fat cat sleep on the mat
- My dream is to be an astronaut.
- Astronauts are bound for space on a spacecraft.

<Sparse Representation>

<table>
<thead>
<tr>
<th>fat</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>...</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>...</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<Word embeddings in 2D>

Embedding learning

<Corpus>

<Sparse Representation>

<Word embeddings in 2D>
Output – Failure history

- State checker send periodic ping to each VM and tag state based on DFA
  - Tag as failure if VM reject ping for a minute
- Save failure history for each VM

![State Checking DFA]

 VM1
 Normal
 Normal
 Failure
 Normal
 Normal
 Failure
 Normal
 Normal
 Failure

 VM2
 Normal
 Normal
 Failure
 Failure
 Normal
 Normal
 Failure
 Failure
 Normal
 Normal
 Failure

 VM6
 Normal
 Normal
 Normal
 Normal
 Normal
 Normal
 Normal
 Normal
 Normal
 Normal
 Normal

<Failure History>

Normal
Warning
Failure

Ping response
Ping rejection
Failure Occurs

Normal
Warning
Failure

<State Checking DFA>
Output - Pre-failure tagging

- CNNs are trained to extract features regardless of order
- Tags the states before the failure occurred to pre-failure rather than normal
- Tags pre-failure value during pre-failure size
- To enable pre-failure to be applied to loss, use KL Divergence-based custom loss
**Learning Algorithm**

- **CNN**

1. **Input Channel & Embedding**
   - root
   - post
   - failed
   - <UNK>
   - 
   - requests
   - <Sentence embedding>

2. **Convolutional Layer**
3. **Max Pooling and Concat layer**
4. **Fully connected layer with dropout and Sigmoid**

**Input Corpus**

- Log
  - Sliding
  - Window
  - Gap
  - Prediction Object

- Failure History
  - Normal
  - Failure

- Prediction Object
Experiment & Evaluation
Experimental setup

Controller Node

Monitorinig Node
- rsyslog server
- Pre-process
- Failure history
- syslog
- Application log
- VM states info

AI Node
- Log parsing module
- Word embedding
- Log embedding

Failure Prediction Module
- CNN model

Compute Node
- Log
- State Checker
- Ping
- VNF A
- Server
- Traffic
- Attack
- VNF F
- Client 1
- Client 6
- Output Label
- Label
Data Collection

- Generate multiple client-VNF-server chains
- Fault inject
  - In Microsoft cloud system, each day less than 0.1% of the nodes encounter failures [7]
  - Generate traffic, resource overload and external attack
Failure Data

- Collected 44 failures in 1 months
  - Server failures: 13
  - None symptom: 4
  - Failures with error before
    - ARB: 21
    - LAG: 6
    - 5 of them had a gap of more than 30 minutes
  - Total 22 number of failures were used for learning

<table>
<thead>
<tr>
<th>Type</th>
<th>gap</th>
<th>Application</th>
<th>log</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARB</td>
<td>5</td>
<td>systemd-timesyncd</td>
<td>Timed out waiting for reply from</td>
</tr>
<tr>
<td>ARB</td>
<td>2</td>
<td>apt-helper</td>
<td>Failed to retrieve unit state: Connection timed out</td>
</tr>
<tr>
<td>LAG</td>
<td>14</td>
<td>kernel</td>
<td>blk_update_request: I/O error, dev vda, sector op READ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>flags phys_seg prio class</td>
</tr>
<tr>
<td>LAG</td>
<td>1</td>
<td>kernel</td>
<td>fail to add MMCONFIG information, can't access extended</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PCI configuration space under this bridge.</td>
</tr>
</tbody>
</table>
Experiments

◆ Unbalanced data

- Total 43,364 number of windows are made, but only 22 of them are tagged as failure
- Apply oversampling with 2 to failure data and random undersampling with 60 to normal data
- Apply class weight for loss function as the reciprocal number of each class (normal/failure) in data

◆ Test

- Shuffle the data and divide by a ratio of 8:2 as train set and test set
- Divide by a ratio of 8:2 again the train set as train set and validation set
## Evaluation

- **Pre-failure size and Pre-failure value test**
  - Use 5 minutes for gap and input window size
  - Without pre-failure tagging
    - Acc: 0.95, Rec: 0.14, F1: 0.25

### Evaluation Table

<table>
<thead>
<tr>
<th>Value (min)</th>
<th>Size (min)</th>
<th>0.5</th>
<th>0.65</th>
<th>0.8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
<td>Acc: 0.98, Rec: 0.75, F1: 0.60</td>
<td><strong>Acc: 0.95, Rec: 1.00, F1: 0.67</strong></td>
<td>Acc: 0.97, Rec: 0.60, F1: 0.55</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>Acc: 0.97, Rec: 0.57, F1: 0.57</td>
<td>Acc: 0.91, Rec: 0.33, F1: 0.29</td>
<td>Acc: 0.96, Rec: 0.33, F1: 0.40</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>Acc: 0.86, Rec: 0.75, F1: 0.44</td>
<td>Acc: 0.90, Rec: 0.40, F1: 0.25</td>
<td>Acc: 0.93, Rec: 0.50, F1: 0.36</td>
</tr>
</tbody>
</table>
Evaluation

- **Gap and input window size test**
  - Use 3 minutes for pre-failure size and 0.65 for pre-failure value
  - Predict failures before 5 minutes with 0.67 of F1 score

<table>
<thead>
<tr>
<th>Win (min)</th>
<th>Gap (min)</th>
<th>5</th>
<th>10</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Acc: 0.93, Rec: 0.45, F1: 0.56</td>
<td>Acc: 0.94, Rec: 0.60, F1: 0.57</td>
<td>Acc: 0.88, Rec: 0.43, F1: 0.22</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Acc: 0.93, Rec: 0.37, F1: 0.46</td>
<td>Acc: 0.95, Rec: 0.33, F1: 0.43</td>
<td>Acc: 0.94, Rec: 0.57, F1: 0.44</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td><strong>Acc: 0.95, Rec: 1.00, F1: 0.67</strong></td>
<td>Acc: 0.95, Rec: 0.33, F1: 0.36</td>
<td>Acc: 0.82, Rec: 0.71, F1: 0.26</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Acc: 0.93, Rec: 1.00, F1: 0.17</td>
<td>Acc: 0.91, Rec: 0.43, F1: 0.35</td>
<td>Acc: 0.92, Rec: 0.40, F1: 0.24</td>
<td></td>
</tr>
</tbody>
</table>

LOG

Input Sequence

Input window size

gap

Prediction object ->
Label

Failure history

- **Evaluation Table**
- **Win (min)**: Indicates the window size.
- **Gap (min)**: Indicates the gap size.
- **Accuracy (Acc)**, **Recall (Rec)**, and **F1 score** are provided for each combination of window size and gap.
Performance Comparison with other models

- Use 5 minutes as input window size, gap and pre-failure size, 0.65 as pre-failure value
- Show angular line because test data is not large (num : 129)
- CNN show best performance
Conclusion
Summary

- We propose a model that analyze logs extracted from VMs which execute VNFs and determine whether failures will occur in the future
- Use pre-failure tagging method to get higher performance
- Could predict failures before 5 minutes with 0.67 of F1 score

Future work

- Gather more failures data
- Learn about failures that occur on the server
- Use CNN’s outputs as input of RNN
- Apply to container based environment
감사합니다
Publications (1/2)

◆ International Conference Papers (3, 1 under review)


◆ Domestic Conference Papers (5)


5. 남석현, 현종환, 유재형, 홍원기, "네트워크 텔레메트리를 활용한 머신 러닝 기반 네트워크 이상 탐지 기법 연구", KNOM Conference 2019, Daegu, Korea, May. 30, 2019, pp. 75-77.

◆ Domestic Patents (2)


Appendix. Loss with Pre-failure Tagging

◆ KL Divergence
  - Measure the different degrees of the two probability distributions
  - \( KL(p|q) := -\sum_{i=1}^{N} p_i \log q_i - (-\sum_{i=1}^{N} p_i \log p_i) = -\sum_{i=1}^{N} p_i \log \left( \frac{q_i}{p_i} \right) \) (N is # of classes)
  - Use custom loss based on KL divergence to learn even for pre-failure value

- Custom loss = \( y_{true} \times \text{classweight} \times \log \left( \frac{y_{true}}{y_{pred}} \right) + (1 - y_{true}) \times \log \left( \frac{1 - y_{true}}{1 - y_{pred}} \right) \)

◆ Keras KL Divergence loss function
  - \( loss = y_{true} \times \log \left( \frac{y_{true}}{y_{pred}} \right) \)
  - return 0 for all normal state (since \( y_{true} \) is 0)

◆ Cross-entropy loss function
  - \( Cross - entropy = -\sum_{i=1}^{N} p_i \log q_i \)
  - Only work when prediction object is 0 either 1


