Distributed Monitoring and Analysis for Telecom Requirements

Yuki Kasuya <yu-kasuya@kddi-research.jp>  Ryota Mibu <r-mibu@cq.jp.nec.com>  Tomofumi Hayashi <tohayash@redhat.com>
KDDI Research  NEC  RedHat
Members
1. Goal
2. Requirements / Challenges
3. Solution / Architecture
4. Feasible Study
5. Conclusion
1. Goal
Telecom Operation Goal

- High Quality Network
  - High Speed
  - Low Latency
  - Not Congested
  - Wide Area
  - etc...

- NO Network Outage
How to Operate NFV

- Complex
  - Massive Component
  - Virtualisation Layer

Figure 4: NFV reference architectural framework
Changing Operation Style

- **Proactive Operation**
  - Agility
  - Before broken, do prevention

- **Periodic Operation**
  - 9am - 5pm maintenance

- **Reactive Operation**
  - 24/7 maintenance

- **Cost Reduction**
  - Reactive
3 Steps for Proactive Operation

Reactive

Fault Detect | Resolve | Recover

time

recover
1st Step

Reactive

Fault Detect | Resolve | Recover

Fault Detect | Recover | Resolve

recover
2nd Step

Reactive

Fault Detect  Resolve  Recover

Recover

Fault Detect  Recover  Resolve
3rd Step

Reactive

Fault Detect  |  Resolve  |  Recover

Proactive

Fault Detect  |  Recover  |  Resolve
Proactive Operation

Reactive

Fault Detect → Resolve → Recover

Proactive

Fault Detect → Recover → Resolve
2. Requirements / Challenges
NFV Applications

- Remote Access
  - L2TP/IPSec, SSL-VPN

- Security
  - Firewall, URL Filtering, Antivirus, WAF, DPI, IPS/IDS

- Storage
  - NAS

- Traffic Control
  - DNS, Wi-Fi Control, LB

- And More...
  - IP-PBX, Traffic Optimizer, Content Cache, Content Compression
Requirements for Telecom’s Proactive Operation

We need to adopt more advanced monitoring

- Increase Monitoring **Targets**
- Increase Monitoring **Items**
- Shorten Monitoring **Interval**
Increase Monitoring Targets

Network functions (Firewall, DPI, LB, …) are provided as VMs.

Service function VMs are divided for various reasons:
- Performance
- Security
- APL Configuration

Telecom needs to monitor a large number of targets:
- 1000 – 10000 VMs

Future vision: Service Function Chaining

About 100 VMs per service

10-20 service function

Thousands instances for monitor
Increase Monitoring Items

- **General cloud monitoring**
  - detect server stop

- **Telecom’s monitoring**
  - detect silent failure
  - Novel approaches like AI
  - We want to use a large quantity data
    - About 100 - 1000 data /1 server
    - Performance
      - CPU, Memory, Load, Swap, Network, Disk IO, ...
    - Fault
      - Alarm, Error message, ...

---

**Example:**

NEC’s MasterScope Invariant Analyzer

Performance data of about 1000 items per server

Learning

CPU is used for network processing

Analysis

CPU util is still high without network

The more data, the more unknown failure AI can discover
Shorten Monitoring Interval

- **Existing data for metering** (Ceilometer etc.)
  - Collect data every few minutes

- **Telecom high availability**
  - Service stop should be less than 1 second
  - We aim for fault detection in 0.1 second

- **Metering data does not suit the purpose**
  - Recovery start is delayed
  - Fault detection is missed
We need an architecture capable of collecting a large amount of data.

- 1000-10000 targets
- 100-1000 items
- 0.1 second interval

How do you collect these data?
3. Solution / Architecture
Limitation Monitoring Function

- **1 Monitoring Function**
  - 1000 Targets - 10000 Targets
  - * Items
  - * Interval

- **Reduce Targets**
  - 1 Monitoring Function manage
Each Computing has Its Monitoring Function
- Monitoring process is complete in each computing node

What is Good?
- Real-time (Fine-grained) monitoring
- Less Load
- Scalable
Total Architecture

Data Analysis (ceilometer)

- poller
- collector
- evaluator
- analyzer
- DB
- detail data
- traffic data
- VNF
- VM

Fault Management (DMA)

- DB
- analytics result
- VM
- concise data
- VNF
- poller
- collector
- evaluator
- analyzer
- VM
- analyzer
4. Feasible Study
DMA Architecture Detail

- **Collector**
  - Poller/Notification
  - SNMP Get
  - libvirt API
  - OpenStack API
  - SNMP Trap

- **Database**
  - Perf. Data
  - Fault Data

- **Analytics Engine**
  - Fault Detection (Prediction)

- **Evaluator**
  - Statistical Analysis
  - Alarm Correlation

- **Transmitter**
  - Fault Alarm
  - Statistical Data

- **Guest.**
  - Perf. data (CPU/Memory...)
  - Fault data (Alarm)

- **Host.**
  - Perf. data (CPU/Memory...)
  - Fault data (Alarm)

A Computing Node
DMA Architecture Detail

Guest.

Perf. data
(CPU/Memory...)

Fault data
(Alarm)

Host.

Perf. data
(CPU/Memory...)

Fault data
(Alarm)

Poller/Notification

SNMP Get

libvirt API

OpenStack API

SNMP Trap

Collector

Meter Translator

Database

Collector

Database

Analytics Engine

Fault Detection (Prediction)

Evaluator

Statistical Analysis

Alarm Correlation

Transmitter

Fault Alarm

Statistical Data

A Computing Node
What is the Goal?

- The Goal: Identify which framework can be suitable/feasible to our requirement on statistics collection
  - Scenario: Detect micro burst traffic?
  - Key Metric: Less performance impact on scaling situation (VM and node)
    - CPU Usage on node

- Target Framework
  - Ceilometer
  - Collectd
  - Monasca
Equipments

OpenStack

- Newton (RDO)  
  - CentOS 7.3  
  - MariaDB 10.1.20-1.el7  
  - RabbitMQ

- Controller x 1  
  - CPU: Xeon E3-1230v5 3.40GHz 8core  
  - Memory: 32G  
  - NIC x 2 (IPMI,Mgmt / API,Tenant)

- Compute x 4  
  - CPU: Xeon E5-2420/E31240/X3450  
  - Memory: 24G/32G  
  - NIC x 2 (IPMI,Mgmt / API,Tenant)

Target Framework

- Ceilometer 7.0.2  
  - Output: Gnocchi 3.0.5

- Collectd  
  - Output: Redis 3.2.8-1.el7

- Monasca 1.3.1  
  - Agents are in compute node  
  - Metrics DB: Influxdb 0.9.5-1
The Study Details

■ Environment
  ● Micro burst traffic case (one VM in node does huge traffic in 1sec every 1min)

■ Examination
  ● Can Framework detect microburst?
  ● How much CPU does the framework use?

■ Target Process
  ● Ceilometer
    • api, collector, notification, polling, polling-central, gnocchi-api, gnocchi-metricd, gnocchi-statsd
  ● Collectd
    • collectd, redis
  ● Monasca
    • api, collector, forwarder, kafka, persister, storm-supervisor, supervisord, zookeeper

■ Parameters
  ● Interval to get metrics: 300 sec, 60 sec, 10 sec, 1 sec, 0.1 sec (collectd supports 0.1sec interval)
Ceilometer (Rx Traffic)

The graph shows the received traffic in Mbps over time in seconds, with different intervals (1 sec, 10 sec, 60 sec, and 300 sec). The data fluctuates considerably, with peaks and valleys indicating varying traffic levels. The graph compares the traffic patterns at different intervals, highlighting how the receive traffic changes over time.
Collectd (Rx Traffic)
Monasca (Rx Traffic)
# CPU Usage (Sum of each processes’ CPU%)

<table>
<thead>
<tr>
<th></th>
<th>min</th>
<th>max</th>
<th>mean</th>
<th>median</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ceilometer</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1sec</td>
<td>16</td>
<td>493</td>
<td>234.4429371</td>
<td>226</td>
</tr>
<tr>
<td>10sec</td>
<td>11</td>
<td>286</td>
<td>65.4810989</td>
<td>24</td>
</tr>
<tr>
<td>60sec</td>
<td>10</td>
<td>409.32</td>
<td>48.07490509</td>
<td>18</td>
</tr>
<tr>
<td>300sec</td>
<td>9</td>
<td>479</td>
<td>44.21182817</td>
<td>17</td>
</tr>
<tr>
<td><strong>collectd</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.1sec</td>
<td>3.97</td>
<td>121</td>
<td>30.30854146</td>
<td>7</td>
</tr>
<tr>
<td>1sec</td>
<td>0</td>
<td>104</td>
<td>8.273686314</td>
<td>1</td>
</tr>
<tr>
<td>10sec</td>
<td>0</td>
<td>104</td>
<td>5.763236763</td>
<td>0</td>
</tr>
<tr>
<td>60sec</td>
<td>0</td>
<td>105</td>
<td>5.66033966</td>
<td>0</td>
</tr>
<tr>
<td>300sec</td>
<td>0</td>
<td>101</td>
<td>6.524475524</td>
<td>0</td>
</tr>
<tr>
<td><strong>monasca(controller)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1sec</td>
<td>0</td>
<td>30</td>
<td>3.424285714</td>
<td>3</td>
</tr>
<tr>
<td>10sec</td>
<td>0</td>
<td>29</td>
<td>3.29953047</td>
<td>3</td>
</tr>
<tr>
<td>60sec</td>
<td>0</td>
<td>64</td>
<td>3.554995005</td>
<td>3</td>
</tr>
<tr>
<td>300sec</td>
<td>0</td>
<td>19</td>
<td>3.743066933</td>
<td>3</td>
</tr>
<tr>
<td><strong>monasca(compute)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1sec</td>
<td>0</td>
<td>205</td>
<td>133.8796004</td>
<td>148</td>
</tr>
<tr>
<td>10sec</td>
<td>0</td>
<td>6</td>
<td>0.635334665</td>
<td>0</td>
</tr>
<tr>
<td>60sec</td>
<td>0</td>
<td>6</td>
<td>0.375624376</td>
<td>0</td>
</tr>
<tr>
<td>300sec</td>
<td>0</td>
<td>6</td>
<td>0.356593407</td>
<td>0</td>
</tr>
</tbody>
</table>
So what is learned?

- **Ceilometer (designed for telemetry)**
  - Gnocchi is good performance
  - Gnocchi CLI shows the latest data after 1min
  - Shorter intervals make rabbitmq busy state

- **Collectd**
  - Good performance in short interval
  - Redis is feasible to our requirements in data storing point of view
    - No need to store in long time (last 1 day is enough)
    - Need quick access (in-memory DB)

- **Monasca (designed for centralized monitoring)**
  - Current monasca-agent designed for 15sec/30sec interval, usually
    - Less than 3sec interval, monasca-agent does not work (not supported?)
    - But it is reasonable interval for usual monitoring
  - Asynchronous processing (to put DB) makes 1-3 sec delay to visible
    - Monasca-api receives the data but it is not appeared immediately
    - Kafka seems bulk/batch data
5. Conclusion
High Quality Network Service

Proactive Operation

Monitoring

DMA
Future Work

A Computing Node

Guest.
- Perf. data (CPU/Memory...)
- Fault data (Alarm)

Host.
- Perf. data (CPU/Memory...)
- Fault data (Alarm)

Flowchart:
- Poller/Notification
- SNMP Get
- libvirt API
- OpenStack API
- SNMP Trap
- Collector
- Meter Translator
- In-Memory
  - Perf. Data
  - Fault Data
- Analytics Engine
  - Fault Detection (Prediction)
- Evaluator
  - Statistical Analysis
  - Alarm Correlation
- Transmitter
  - Fault Alarm
  - Statistical Data

OpenStack API...
- Perf. Data (CPU/Memory...)
- Guest.
- Perf. Data (CPU/Memory...)
- Fault Data (Alarm)
Join us

- Appendix:  http://goo.gl/oAeGC5
- Etherpad:  https://etherpad.openstack.org/p/dma-discussion
- Contact:   dma-discuss@redhat.com