On Fault Resilience of Open Stack

Subho Banerjee
OpenStack

- Control and automate pools of resources
- Efficiently allocate resources
- Empower admins & users via self-service portals
- Provide APIs to make apps cloud-aware
Broad Commercial Support
Grizzly Logical Architecture
Overview

• Fault Injection Framework for OpenStack
  • Target intra/inter service communication during processing of an external request

• Study fault resiliency of OpenStack
  • Fault Resiliency — Maintain correct functionalities under faults
  • OpenStack essex and grizzly

• Categorize bug categories dealing with fault resilience issues
Fault Injection Framework
Design Principles

- Inject faults in communication flows
- Expose OpenStack’s high level semantics to the fault injection module
- Consider 11 most commonly used API endpoints
Parts of the stack being tested

Identity Service (keystone)

Image Service (glance)

Compute Service (nova)

AMQP Broker
1. Construct execution graphs for external requests

2. Performing fault injection and collecting results
   - Test Plan
   - Execution Graph
   - Fault Type: Receiver Crash
   - Fault Location: B1
   - Current Execution
   - Fault Injection Controller
   - Faulty Execution Result
   - State: S
   - Behavior: B

3. Specification checking
   - Specification
   - Expected State: S'
   - Faulty Execution Result
   - State: S
   - Behavior: B
   - Specification Violations
1. Construct Execution Graph (Logging and Coordination Framework)

- System-wide unique tag assigned to each external request
- Introduce tag fields request context and thread-local storage
- Trace processing of tag within scope of stack

Execution graph for VM_Create
- Layers in white are instrumented with the new tag based logging scheme.
- Consolidate communication between services
- Shared across the stack
2. Fault Injection

Constructing Test Plans

**Procedure 1 Test Plan Generation**

```python
test_plans ← an empty list
for all node in exe_graph do
    for all fault in fault_specs do
        if fault can be injected to node then
            new_plan ← TestPlan(exe_graph, node, fault)
            test_plans.append(new_plan)
    return test_plans
```
Inject Faults

- Only single faults injected
- Two types of faults considered
  - Service Faults
  - Network Partition Faults
- Suggest clustering nodes on execution graph
3. Specification Checking

**Specification 1** VM State Stabilization Specification

```plaintext
query = select VM from compute_database
   where VM.state in collection(VM unstable states)
if query.count() = 0 then
   return Pass
return Fail
```

**Specification 2** Ethernet Configuration Specification

```plaintext
if (VM.state = ACTIVE) and
   ((VM.host.Ethernet not setup) or
    (network_controller.Ethernet not setup)) then
   return Fail
return Pass
```
Fault Resilience of OpenStack
<table>
<thead>
<tr>
<th>Category</th>
<th>Count</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Common</td>
<td>essex only</td>
<td>grizzly only</td>
</tr>
<tr>
<td>Timeout</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Periodic checking</td>
<td>6</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>State transition</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Return code checking</td>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Cross-layer coordination</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Library interference</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>13</td>
<td>9</td>
<td>1</td>
</tr>
</tbody>
</table>

![Bar chart showing specification violations for bugs labeled as Essex and Grizzly](chart.png)
<table>
<thead>
<tr>
<th>API</th>
<th>Faults</th>
<th></th>
<th></th>
<th>Specification Violations</th>
<th></th>
<th></th>
<th></th>
<th>Bugs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Essex</td>
<td>Grizzly</td>
<td>Essex</td>
<td>Grizzly</td>
<td>Essex</td>
<td>Grizzly</td>
<td>Essex</td>
<td>Essex</td>
<td>Grizzly</td>
<td>Essex</td>
</tr>
<tr>
<td></td>
<td>Crash</td>
<td>Part.</td>
<td>Crash</td>
<td>Part.</td>
<td>Crash</td>
<td>Part.</td>
<td>Crash</td>
<td>Crash</td>
<td>Part.</td>
<td>Crash</td>
</tr>
<tr>
<td>VM create</td>
<td>217</td>
<td>133</td>
<td>311</td>
<td>229</td>
<td>93</td>
<td>43</td>
<td>150</td>
<td>49</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>VM delete</td>
<td>79</td>
<td>61</td>
<td>102</td>
<td>82</td>
<td>51</td>
<td>15</td>
<td>45</td>
<td>23</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>VM pause</td>
<td>24</td>
<td>17</td>
<td>35</td>
<td>29</td>
<td>16</td>
<td>13</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>VM reboot</td>
<td>64</td>
<td>36</td>
<td>139</td>
<td>104</td>
<td>9</td>
<td>11</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>VM rebuild</td>
<td>159</td>
<td>106</td>
<td>242</td>
<td>183</td>
<td>103</td>
<td>67</td>
<td>0</td>
<td>13</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>VM image create (local)</td>
<td>142</td>
<td>119</td>
<td>171</td>
<td>150</td>
<td>59</td>
<td>106</td>
<td>90</td>
<td>79</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>VM image create (HTTP)</td>
<td>107</td>
<td>92</td>
<td>171</td>
<td>150</td>
<td>24</td>
<td>84</td>
<td>79</td>
<td>71</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>VM image delete (local)</td>
<td>59</td>
<td>44</td>
<td>22</td>
<td>15</td>
<td>23</td>
<td>37</td>
<td>12</td>
<td>9</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>VM image delete (HTTP)</td>
<td>59</td>
<td>44</td>
<td>22</td>
<td>15</td>
<td>23</td>
<td>37</td>
<td>10</td>
<td>8</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Tenant create</td>
<td>7</td>
<td>6</td>
<td>7</td>
<td>6</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>User create</td>
<td>7</td>
<td>6</td>
<td>7</td>
<td>6</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Role create</td>
<td>7</td>
<td>6</td>
<td>7</td>
<td>6</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>User-role create</td>
<td>9</td>
<td>8</td>
<td>10</td>
<td>9</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>9</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Sum</td>
<td>940</td>
<td>678</td>
<td>1246</td>
<td>984</td>
<td>401</td>
<td>439</td>
<td>392</td>
<td>288</td>
<td>42</td>
<td>38</td>
</tr>
</tbody>
</table>
Types of Errors

• Timeout mechanisms missing in critical OpenStack paths, e.g., REST
  • CreateVM — Compute service querying image service, a network partition blocks system
  • Fixed in grizzly — 600s timeout by default.

• Cross layer coordination — How do you coordinate across layers?
  • OpenStack’s AMQP Wrapper waits indefinitely for answer. QPID client drops request after n tries.
• Library Interface —
  
  • Block compute service — QPID client uses a read/select which is different from the Python implementation. Hence the client performs read on pipe before it is ready.

• Return Code Checking

  • Incorrect use of return codes

  • Checking disabled
Discussion

• How does the OpenStack configuration affect the fault to failure propagation?

• Test plan creation — What do you do in cases where there are multiple ways faults can be injected?

• Finding the bug is still a manual process

• Does changing system’s state make a difference to the result?