Mechanisms and Architectures for Tail-Tolerant System Operations in Cloud

Qinghua Lu, Liming Zhu, Xiwei Xu, Len Bass, Shanshan Li, Weishan Zhang, Ning Wang

China University of Petroleum, Qingdao, China
NICTA, Sydney, Australia

dr.qinghua.lu@gmail.com
Outline

- Motivation
- Tail-Tolerant Mechanisms and API Wrapper
- Deployment Architecture Tactics
- Evaluation
- Conclusion
Motivation

- System operations (such as upgrade, deployment and backup) in cloud are performed through cloud APIs provided by cloud providers
  - The completion time and reliability of operation tasks depends on the reliability and performance of API calls
- We observed cloud API issues during the development of our commercial product Yuruware Bolt
  - Yuruware Bolt relies EC2 to perform disaster recovery operations
  - e.g., when we detach/attach a volume, it is stuck at detaching/attaching
Motivation

- We performed searches on EC2 forum
  - 5 API calls: launch instance, start instance, stop instance, attach volume, detach volume
  - extracted API related issues: 922 cases out of 1109 API related cases are API failures
  - 81% of 922 failures are timing failures (stuck API calls and slow responded API calls)

<table>
<thead>
<tr>
<th>Posted on Aug 27, 2012 11:57 AM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Symptom:</strong> It took 16 minutes for an instance to stop.</td>
</tr>
<tr>
<td><strong>Root cause:</strong> n/a.</td>
</tr>
<tr>
<td><strong>Solution:</strong> The AWS engineer advised to try “force stop” twice if this happens next time.</td>
</tr>
</tbody>
</table>
Motivation

- We conducted experiments on the timing behaviour of 5 EC2 API calls and observed that around 4.5% have long tail characteristics.
Motivation

- Cloud API timing failures: major causes of the long-tail of operation tasks
  - Existing research focuses on reducing errors and repair time
- One step of an operation: parallel or deep hierarchical
  - One slow API response will cause the initial operation to be slow to respond
Tail-Tolerant Mechanisms

Diagram showing states and transitions:
- Requested
- Allocated
  - Normal request
  - Hedge request
  - Continue allocate or reallocate
- Started
  - Start
  - Reallocation
- Failed
  - Force fail a
  - Fail
  - Force fail s
- Completed
  - Complete
  - Force complete a
- Cancelled
  - Cancel allocate
  - Cancel start
Issue more request than we need and then cancel the remaining immediately after the required number is reached.
An alternative API is requested at the same time as the original API is requested.
Tail-Tolerant Mechanisms

If the API request sent to an instance is failed or there is no response from the cloud infrastructure
Tail-Tolerant Mechanisms

When an API call has been retried for several times and continue to fail
## API Wrapper

<table>
<thead>
<tr>
<th>API wrapper</th>
<th>Pattern</th>
<th>Implementation details</th>
</tr>
</thead>
<tbody>
<tr>
<td>launch-instance</td>
<td>hedge-request; continue-allocate</td>
<td>The API wrapper launches two instances when it receives a request. If one instance is launched within the time specified in the timing profile of launch-instance, the API wrapper will kill the other one. If neither of them launches, then the API wrapper re-launches another two instances.</td>
</tr>
<tr>
<td>start-instance</td>
<td>alternative-request</td>
<td>The API wrapper starts an instance and launches a new instance using the same image simultaneously, and cancels the one with longer return time.</td>
</tr>
<tr>
<td>stop-instance</td>
<td>force-complete-a</td>
<td>The API wrapper launches a call to the stop-instance API, waits for the time specified in the timing profile of stop-instance. If the call is not completed, the API wrapper forces the instance to stop using “force-stop” API.</td>
</tr>
<tr>
<td>attach-volume</td>
<td>alternative-request</td>
<td>The API wrapper attaches volume to an instance and launches a new instance at the same time. The wrapper waits for the time specified in the timing profile of attach-volume. If the call is not completed, it re-attaches the volume to the newly launched instance.</td>
</tr>
<tr>
<td>detach-volume</td>
<td>force-complete-a</td>
<td>The API wrapper waits for the time specified in the time profile of detach-volume. If not completed, then the API wrapper force-detaches the volume.</td>
</tr>
</tbody>
</table>
Evaluation of API Wrapper

- Evaluate API wrapper on EC2
- For each API we wrapped, we measure the return time 1000 times respectively

Measurement results of “start instance”.

Measurement results of “stop instance”.

EC2 API ▲ API wrapper

Return Time (s)

0 20 40 60 80 100 120 140 160 180 200

20.0% 18.0% 16.0% 14.0% 12.0% 10.0% 8.0% 6.0% 4.0% 2.0% 0.0%
Deployment Architecture Tactics

- Immutable server
  - Operators make an image which contains a new version of everything an application needs. After the image is launched, nothing more is added or allowed to be changed.

- Micro service
  - Operators break down an application into micro-services and make each service run on different VMs.
  - Lightweight instances and less performance interference

- Redundancy:
  - Operator can run more than the required number of VMs to avoid long-tail operations.
Evaluation of Deployment Architecture Tactics

- We evaluate the deployment architecture tactics through automatically upgrading 50 AMP stacks (Apache + MySQL + PHP) by shell scripts.
  - ran on EC2
  - upgrades the AMP stack from Apache 2.0.65, MySQL 5.1.73, and PHP 5.2.17 to Apache 2.2.22, MySQL 5.5.35, and PHP 5.3.10 respectively.
We implemented the three deployment tactics, and compared the number of the successfully upgraded VMs using different tactics with a baseline, which represents upgrade without any tactics.

Ran each of the 4 test cases 100 times

- **Baseline:** upgrade AMP running on 50 VMs to the recent versions directly on the original VMs.
- **Immutable server:** create an image of VM which runs the new version of AMP and launch 50 VMs using the image. Then we terminate the VMs running old versions of AMP.
- **Micro services:** run Apache and PHP on 50 VMs and run MySQL on another 50 VMs, then we upgrade them on the original VMs directly.
- **Redundancy:** launch 3 extra VMs with AMPs before we do upgrade. After the 3 extra VMs are successfully launched, we start upgrading the 53 VMs with AMPs.
Evaluation of Deployment Architecture Tactics

![Graph showing measurement results of deployment tactics.](image-url)
Conclusions

- We proposed API mechanisms and deployment architecture tactics to tolerate long-tail issues of operations in cloud
- We implemented our mechanisms as a tail-tolerant wrapper around EC2 APIs
- Our initial evaluation shows that the mechanisms and deployment architecture tactics can remove the long tails
- Future work:
  - implement the rest of mechanisms in API wrapper
  - model the reliability of cloud operations in SRN
Thanks you!