Cloud API Issues: an Empirical Study and Impact
Qinghua Lu1,2, Liming Zhu1,2, Len Bass1,2, Xiwei Xu1, Zhanwen Li1, Hiroshi Wada1,2
1Software Systems Research Group, NICTA, Sydney, Australia
2School of Computer Science and Engineering, University of New South Wales, Sydney, Australia
Qinghua.Lu@nicta.com.au

ABSTRACT
Outages to the cloud infrastructures have been widely publicized and it would be easy to conclude that application developers only need to be concerned with large scale cloud provider infrastructure outages. Unfortunately, this is not the case. In-cloud applications heavily rely on cloud infrastructure APIs (directly or indirectly through scripts and consoles) for many sporadic activities such as deployment change, scaling out/in, backup, recovery and migration. Failures and/or issues around API calls are a large source of faults that could lead to application failures, especially during sporadic activities. Infrastructure outages can also be greatly exacerbated by API-related issues.

In this paper we present an empirical study of issues in Amazon EC2 APIs. Some of the major findings around API issues include:
1) A majority (60%) of the cases of API failures are related to “stuck” API calls or unresponsive API calls. 2) A significant portion (12%) of the cases of API failures are about slow responsive API calls. 3) 19% of the cases of API failures are related to the output issues of API calls, including failed calls with unclear error messages, as well as missing output, wrong output, and unexpected output of API calls. 4) There are 9% cases of API failures reporting that their calls (performing some actions and expecting a state change) were pending for a certain time and then returned to the original state without informing the caller properly or the calls were reported to be successful first but failed later. We also classify the causes of API issues and discuss the impact of API issues on application architectures.

Categories and Subject Descriptors
D.2.4 [Software]: Software Engineering – Software/Program Verification.

General Terms
Design, Reliability.

Keywords

1. INTRODUCTION
Public cloud platforms often provide only weak guarantees on the availability of their resources compared to traditional hosting services. Therefore, many applications in cloud are architected to achieve high availability by fully leveraging the nature of cloud platforms such as on-demand resource allocation, micro-rebooting, or replication and failover across-geographically distributed sites [1, 2]. In-cloud applications also need to perform some sporadic activities such as upgrade, backup, recovery and deployment changes. To achieve high availability designs and perform other sporadic activities, it is necessary to call the Application Programming Interfaces (APIs) provided by cloud providers since resources in cloud are manipulated through APIs (directly or through scripts or consoles). Therefore, in cloud, application availability depends not only on the availability of cloud resources and infrastructures but also on the reliability and performance of API calls. Designers and developers need to understand the nature of issues occurring on API calls and implement proper failure handling and fault tolerance mechanisms.

In this paper, we report an empirical study of issues related to APIs in Amazon Elastic Compute Cloud (EC2) [3]. The motivation of the study also comes from the facts that we observe a large percentage of the cases reported in the EC2 forum are related to API issues (53 of a recent sequence of 100 items in the forum are API-related issues). We also observed cloud API issues during the development of our commercial disaster recovery product Yuruware Bolt [2] that relies heavily on APIs to perform disaster recovery operations such as snapshotting, starting and stopping VMs and redeploying a system at the recovery site. We manually searched relevant key words in the EC2 forum [4] and classified the EC2 API issues and causes of the issues.

The major contributions of this paper:
• We extracted API related issues from the EC2 forum. 922 cases out of 1109 reported API-related cases (searched using selected keywords) in the EC2 forum from 2010 to 2012 are API failures (rather than feature requests or general inquiries).
• We classified the extracted API failures into four types of failures: content failures, late timing failures, halt failures, and erratic failures. We illustrated distribution of failure types among the studied cases and discussed some representative examples for each failure type.
• We classified the causes of API failures into three types of faults: development faults, physical faults, and interaction faults.
• We did initial analysis of impact of API failures and faults on in-cloud application architecture.
The major findings are summarized as follows.

- A majority (60%) of the cases of API failure are related to stuck API calls or unresponsive API calls.
- A large portion (12%) of the cases is about slow responsive API calls.
- 19% of the cases are related to the output issues of API calls, including failed calls with unclear error messages, as well as missing output, wrong output, and unexpected output of API calls.
- There are 9% cases reporting that their calls (performing some actions and expecting a state change) were pending for a certain time and then returned to the original state without informing the caller properly or the calls were reported to be successful first but failed later.

The current study only focuses on searching a limited set of API calls for manipulating EC2 instances. However, they are the API calls that have the most issues according to our initial EC2 forum search and experience from Yuruware Bolt development.

The rest of this paper is organized as follows. Section 2 describes the methodology of our study. Section 3 summaries the data collected from the EC2 forum. Section 4 classifies the API failures and discusses each API failure type in details. Section 5 analyses the causes of the API failures. Section 6 discusses the impact of API failures on in-cloud application architecture. Section 7 covers related work. Section 8 concludes the paper.

2. METHODOLOGY

Fig.1 describes our methodology for the empirical study on cloud API issues. The rectangle boxes describe the study steps and oval circles represent the study sources. The arrows between a rectangle box and an oval circle represent data flow while the arrows between rectangle boxes represent the transitions between study steps.

2.1 Study Sources

We have four major sources for our study: Amazon Web Service (AWS) support engineers in the EC2 forum [4], AWS outage reports [5-10], Netflix Tech Blog [1], and our commercial product Yuruware Bolt development and operation experience [2].

Amazon EC2 forum is one of the discussion forums Amazon provides to its users to post their questions and feedback. Amazon support engineers reply most of the cases and help users solve the reported issues.

When AWS experiences an outage, a report is published after the outage is resolved. In the report, AWS explains the causes of the outage and discusses what they did to resolve the issues and what they would do to improve the infrastructure reliability.

Netflix [11] is a provider of on-demand Internet streaming media. Netflix almost builds its entire system on top of Amazon public cloud and is one of the biggest Amazon customers. In order to deal with infrastructure issues, Netflix has built a set of open source platforms [12] to improve the availability of their streaming service. Some of the platform features are motivated by API related issues. Netflix Tech Blog is a blog where Netflix shares their perspectives, decisions, and challenges regarding the software they build and use to create the streaming service.

Yuruware Bolt is a commercial disaster recovery product that provides geographic resilience for systems operating in AWS. During the development and operations of Yuruware Bolt, we learned many lessons with respect to EC2 API reliability issues.

2.2 Study Procedures

2.2.1 Search and collect data

We did initial searches on the AWS EC2 forum to identify which API calls have the most issues as there are a large number of EC2 API calls. We identified five frequently used API calls based on discussions in Amazon EC2 Forum and our commercial disaster recovery product Yuruware Bolt development experience. These API calls are: describe an instance, start an instance, stop an instance, detach a volume, and associate an elastic IP. We used the following search strings and their variations for the five identified API calls: “describe instance”, “start instance”, “stop instance”, “detach volume”, and “associate elastic IP”. The identified search strings are not themselves API call names (e.g. ec2-describe-instances), since many users report issues using plain English in the Amazon EC2 Forum. In this study, we only analyzed the cases posted from 2010 to 2012 (See Section 3 for details). And we found that the most frequently reported case type is API failure, which we also experienced during from our Yuruware Bolt development and operation.

![Figure 1. Methodology of the empirical study](image-url)
2.2.2 Failure and Fault Classification
We classified these API issues using the taxonomy of dependable and secure computing [13] regarding types of failures and faults. A failure is an event that occurs when the delivered service deviates from correct service. We classified the API failures reported in the AWS EC2 Forum into four major classes based on the failure taxonomy proposed in [13]: content failures, late timing failures, halt failures, and erratic failures.

A fault is the adjudged or hypothesized cause of an error (that may or may not lead to a failure manifested at different levels) [13]. For all reported failure cases, we analyzed the causes and classified the causes based on the fault taxonomy proposed in [13]. The causes are analyzed based on the discussions between users and AWS support engineers in the EC2 forum [4], AWS outage reports [5-10], our Bolt development and operation experience [2], and Netflix Tech Blog [1].

2.3 Study Limitations
There are several limitations in the study:

- First, the current study is only based on searching using five textual strings. However, they are the API calls that have the most issues according to our initial EC2 forum search and experience from Yuruware Bolt development.
- Second, the current search is limited to the EC2 forum only. We have not searched in the forums for other AWS services (e.g., Simple Storage Service or Virtual Private Cloud) and other cloud providers’ forums or major programmer forums.
- Third, the study only looked at single API call failures and did not investigate API call sequence failures which may also be a major source of issues often discussed.
- Fourth, the cause and impact of the API failures rely heavily on the EC2 forum and Amazon outage reports. However, Amazon may not fully disclose the causes of issues due to commercial reasons. There is a certain level of educated guess and correlation across cases about causes of failures.
- Fifth, the study is based mostly on forum entries. The study by design covers only the problems that are reported and may not reflect the problems that actually occur. There are effects that can skew the results significantly: i) transient errors that only occur once but then vanish are less likely to be reported in a forum; ii) if a problem occurs often, but an entry in the forum already provides a solution, the problem is less likely to be reported again; iii) the probability of writing an entry in the forum depends on whether the user can find a solution by himself/herself and whether the user wants to contact customer service directly rather than asking questions publicly in the forum.

3. DATA COLLECTED FROM AMAZON EC2 FORUM
As shown in Table 1, the keywords searched for this study are: “describe instance” (in one word), “start instance”, “stop instance”, “detach volume”, and “associate elastic IP”. In this study, we only analyzed the cases posted between 2010 and 2012 since they take significant portions of records from inception to 2012. There are 1109 cases from 2010 to 2012 returned by the keywords.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Number of records from inception to 2012</th>
<th>Number of records from 2010 to 2012</th>
</tr>
</thead>
<tbody>
<tr>
<td>describe instance</td>
<td>283</td>
<td>150</td>
</tr>
<tr>
<td>start instance</td>
<td>227</td>
<td>204</td>
</tr>
<tr>
<td>stop instance</td>
<td>349</td>
<td>348</td>
</tr>
<tr>
<td>detach volume</td>
<td>235</td>
<td>203</td>
</tr>
<tr>
<td>associate elastic IP</td>
<td>264</td>
<td>204</td>
</tr>
<tr>
<td>Total</td>
<td>1358</td>
<td>1109</td>
</tr>
</tbody>
</table>

We classified the cases into three categories – API related failures experienced by the user, general enquiries on how to perform particular actions and suggestions for adding features to the existing APIs. As shown in Table 2, 922 cases are about API failures, 83% of all 1109 cases; 125 cases are enquiries, 11% of all customer cases; 62 cases are about API enhancements, 6% of all customer cases.

<table>
<thead>
<tr>
<th>Case type</th>
<th>Case number</th>
<th>Percentage of all cases from 2010-2012</th>
</tr>
</thead>
<tbody>
<tr>
<td>API failures</td>
<td>922</td>
<td>83%</td>
</tr>
<tr>
<td>Enquiries</td>
<td>125</td>
<td>11%</td>
</tr>
<tr>
<td>API enhancements</td>
<td>62</td>
<td>6%</td>
</tr>
</tbody>
</table>

Enquiries are mainly about unknown API calls (e.g. how to start a reserved Windows instance), problematic results after API calls (e.g. data lost after stopping the instance) not due to API failures, and the difference between different API calls (e.g. the difference between stopping an instance and terminating an instance). Most of the enquiries are due to unclear API documentation, unclear alert messages, or a lack of knowledge.

API feature enhancements are mainly from users who are unhappy with the current feature or expecting new features to facilitate their development or management. Here are some example cases:

- EC2 API is requested to support automatic snapshots of instances based on some schedule.
- Available instance type is requested to be part of the reply of ec2-describe-regions call since one may want to offer their customers dynamic choices of which region and which instance types their operations run on.
- A new API call is requested to support upgrading instance type for the selected instance, e.g. upgrade an instance from t1.micro to m2.xlarge. (Note, this request has already been implemented.)

We now discuss API related failures in detail.
4. CLASSIFICATION OF API FAILURES
As discussed in Section 3, there are 922 cases related to API failures. Based on the taxonomy discussed in [13], we classify the reported API failures into four major types: content failures, late timing failures, halt failures, and erratic failures.

Fig. 2 illustrates the distribution among reported EC2 API failures in the EC2 forum. 60% of API failures (553 cases) are halt failures; 19% of API failures (175 cases) are content failures; 12% of API failures (111 cases) are late timing failures; and 9% of API failures (83 cases) are erratic failures.

![Figure 2. Distribution among API failures in EC2 forum.](image)

4.1 Content Failures
Content failure is defined in [13] “The content of the information delivered at the service interface deviates from implementing the system function”. 19% of the failures fall in this category. We classify content failures into four sub-types: 1. failed calls with error messages (10% of API failures), 2. missing content (3% of API failures), 3. wrong content (3% of API failures), and 4. unexpected content (3% of API failures).

4.1.1 Failed Calls with Error Messages
10% of the cases report that users experience failed calls with error messages. In these cases, 61% of the users understand the problem from the error message but do not know the solutions to the problem while 39% of the users do not understand what caused the problems and how to fix them. Table 3 shows an example where the returned error message does not show the cause of the problem clearly and how to fix the problem. This indicates that clear error messages are essential for diagnosis and repair.

Table 3. An API issue case where the error message is unclear.

<table>
<thead>
<tr>
<th>Posted on Jan 10, 2012 5:42 AM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Symptom:</strong> When a user tried to start an instance, the operation failed with an unclear error message.</td>
</tr>
<tr>
<td><strong>Root cause:</strong> Unknown.</td>
</tr>
<tr>
<td><strong>Solution:</strong> AWS engineers advised detaching the EBS volume from the instance and attaching it to another running instance.</td>
</tr>
<tr>
<td><strong>Error message:</strong> State Transition Reason - Server.InternalError: Internal error on launch</td>
</tr>
</tbody>
</table>

4.1.2 Missing Content
This type of failure captures the cases where some information is missing in the AWS management console or in the returned output after an API call. 3% of the cases are in this category and most of them are related to “describe” type of calls, e.g., ec2-describe-instances, which are used for monitoring.

Table 4. An API issue case where there is missing content.

<table>
<thead>
<tr>
<th>Posted on Apr 9, 2012 8:27 AM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Symptom:</strong> A user tried `ec2-describe-instances --filter &quot;platform==&quot;&quot; using the EC2 API tools and received nothing. In the API document, it says &quot;Use windows if you have Windows based instances; otherwise, leave blank.&quot;</td>
</tr>
<tr>
<td><strong>Root cause:</strong> API bug.</td>
</tr>
<tr>
<td><strong>Solution:</strong> The AWS engineer forwarded this to the EC2 team to investigate.</td>
</tr>
</tbody>
</table>

Table 4 shows an API failure case where there is missing content in the returned output after an API call. This case was caused by a bug. The AWS engineer forwarded this to the EC2 team to investigate. The EC2 team then found there was no easy way to filter Linux instances using ec2-describe-instances and they will be updating API documentation to reflect this.

4.1.3 Wrong Content
There are 3% cases reporting that the output is wrong or the output is inconsistent. This usually happens when a user tries to detach a volume or describe an instance or a volume. Table 5, Table 6, Table 7, and Table 8 show four such API failure cases.

Table 5. An API failure case (1) where the output is wrong.

<table>
<thead>
<tr>
<th>Posted on Aug 28, 2012 4:03 PM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Symptom:</strong> The output of ValidFrom and ValidUntil are swapped in ec2-describe-spot-instances in ec2-api-tools 1.6.1.4.</td>
</tr>
<tr>
<td><strong>Root cause:</strong> API bug.</td>
</tr>
<tr>
<td><strong>Solution:</strong> n/a</td>
</tr>
</tbody>
</table>

In Table 5, the output of ValidFrom and ValidUntil were swapped in ec2-describe-spot-instances which was due to a bug. An AWS engineer recommended using the latest version of the EC2 API tool "ec2-api-tools 1.6.1.4". The user came back and reported that the latest version had the same issue and then AWS did not follow up with the case.

Table 6. An API failure case (2) where the output is wrong.

<table>
<thead>
<tr>
<th>Posted on Jul 21, 2010 8:04 AM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Symptom:</strong> A user was not able to detach the volume from an instance the user did not own.</td>
</tr>
<tr>
<td><strong>Root cause:</strong> n/a</td>
</tr>
<tr>
<td><strong>Solution:</strong> n/a</td>
</tr>
</tbody>
</table>

The case in Table 6 shows a weird output where the user’s volume was attached to an instance the user does not own. The privacy of the volume owner would be disclosed if the instance’s owner can read and write on the volume.
Implementing the team launched an instance in the east region. A user was unable to reach an instance through an elastic IP, but the AWS control panel indicates that the elastic IP is associated with the instance.

**Root cause:** n/a

**Solution:** The instance became reachable after few hours.

Table 7. An API failure case (3) where the output is wrong.

<table>
<thead>
<tr>
<th>Symptom:</th>
<th>A user was not able to reach an instance through an elastic IP, but the AWS control panel indicates that the elastic IP is associated with the instance.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root cause:</td>
<td>n/a</td>
</tr>
<tr>
<td>Solution:</td>
<td>The instance became reachable after few hours.</td>
</tr>
</tbody>
</table>

**Posted on Sep 5, 2012 2:25 PM**

Table 8. An API failure case (4) where the output is wrong.

<table>
<thead>
<tr>
<th>Symptom:</th>
<th>A user found that the output of ec2-describe-instances and ec2-describe-volumes did not match: the output of ec2-describe-instances showed the volume was attached while the output of ec2-describe-volumes showed the volume was attaching.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root cause:</td>
<td>n/a</td>
</tr>
<tr>
<td>Solution:</td>
<td>n/a</td>
</tr>
</tbody>
</table>

**Posted on Jan 20, 2011 9:07 PM**

Table 7 and Table 8 show two cases where the output is inconsistent. The case in Table 8 illustrates a user lost connectivity to an elastic IP which was shown to be associated with the instance in AWS management console. The case in Table 9 shows the output of ec2-describe-instances and ec2-describe-volumes did not match about the volume status.

### 4.1.4 Unexpected Content

3% of the cases report the outputs of the calls are different from the users’ expectation. Most of them are related to APIs describing the status of resources.

Table 9. An API failure where there is an unexpected content.

<table>
<thead>
<tr>
<th>Symptom:</th>
<th>A user launched an instance in the us-west-1 region. When the user ran the ec2-describe-instances command, it shows that the instance did not exist. When the user listed all instances, the output showed all of instances in us-east-1 region.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root cause:</td>
<td>Misconfiguration.</td>
</tr>
<tr>
<td>Solution:</td>
<td>Using the --region option of the command to specify us-west-1.</td>
</tr>
</tbody>
</table>

**Posted on Jul 11, 2012 2:42 PM**

Table 9 shows a very commonly reported API failure. It is caused by user misconfiguration. Many users thought the ec2-describe-instance command returns instances in all EC2 regions.

### 4.2 Late Timing Failures

Late timing failures means that the arrival time of the delivered information at the interface deviates from implementing the system function [13].

There are 12% cases complaining about slow responses to users’ API calls. As explained by AWS support team, slowest API calls usually take several minutes to complete the calls.

Table 10. An API issue case where the API call responses slowly.

<table>
<thead>
<tr>
<th>Symptom:</th>
<th>It took 16 minutes for an instance to stop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root cause:</td>
<td>n/a</td>
</tr>
<tr>
<td>Solution:</td>
<td>The AWS engineer advised to try “force stop” twice if this happens next time.</td>
</tr>
</tbody>
</table>

**Posted on Aug 27, 2012 11:57 AM**

Table 10 shows a reported late timing failure that it took 16 minutes for the instance to stop. Many users experience longer time to complete, e.g. half hour. For some users they even need a couple of hours to complete. However, in the API document, it does not specify the normal time to complete each API call.

![Figure 3. Percentages of late timing failures for different intended calls.](image)

As shown in Fig. 3, a majority of late timing failures happens when a user tries to stop an instance. Users often stop an instance when they are unable to connect to the instance due to the underlying hardware failures.

### 4.3 Halt Failures

Halt failures refer to the case where the external state becomes constant [13]. A special case of halt failures is silent failures, which means no delivered service at all at the system interface. 60% of the failures were in this category.

#### 4.3.1 General Halt Failures

The most frequent API issue is that an API call is stuck at a certain state. There are 391 cases out of 922 cases reporting an API call is hanging at a state, which is 42% of the reported API issues.

As shown in Fig 4., in these 391 cases, 63% cases are stuck at the “stopping” state when trying to stop an instance, 31% cases are stuck at the “detaching” state when trying to detach a volume from an instance, 6% cases are hanging at the “initializing” or “pending” state when trying to start an instance.
The second most frequent API failure category is unresponsive calls. There are 159 cases out of 922 cases reporting that there is no response when a user tries to connect to the instance or call API. It takes 18% of all API cases.

Table 12 shows an API issue case where the user was not able to operate the instance or create a snapshot of it. The case was caused by an AWS outage.

### 4.4 Erratic Failures

Erratic failures refer to the case when the delivered service is erratic [13]. There are 9% cases reporting erratic failures. This type of call includes two subtypes: 1. the call was pending for a certain time and then returned to the original state; 2. the call was successfully executed first but failed eventually;

The first sub-type of erratic failures occurs when a user tries to start an instance, which takes 68% of erratic failures. Table 13 shows an example case. An instance stayed at “pending” state for a few minutes and then changes to “stopped”. The user had attempted to start the instance several times with the same result. This case was due to the EBS volume issue.

Table 13. An API issue case where there is an erratic failure with “start instance”.

<table>
<thead>
<tr>
<th>Posted on Jan 14, 2011 1:43 PM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Symptom:</strong> A user tried to start the instance several times. It indicated that the status is pending and it goes back to stop.</td>
</tr>
<tr>
<td><strong>Root cause:</strong> n/a.</td>
</tr>
<tr>
<td><strong>Solution:</strong> The AWS engineer returned the user’s EBS volume to the available state and believed this would resolve the user’s problem.</td>
</tr>
</tbody>
</table>

The second sub-type of erratic failures happens to elastic IP association, which is 32% of erratic failures. Table 14 shows an example case of such type of failures.

Table 14. An API issue case where there is an erratic failure with elastic IP association.

<table>
<thead>
<tr>
<th>Posted on Feb 1, 2012 8:15 AM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Symptom:</strong> A user associated an elastic IP with an instance and could SSH into the instance with the elastic IP. After a few minutes, the elastic IP was silently disassociated from the instance.</td>
</tr>
<tr>
<td><strong>Root cause:</strong> An issue with the underlying host.</td>
</tr>
<tr>
<td><strong>Solution:</strong> The AWS engineer advised that the quickest fix was to stop and then start the instance to relocate to a different host.</td>
</tr>
</tbody>
</table>

5. **CAUSES OF API FAILURES**

Failures are caused by faults [13]. We analyze the causes from the following sources: API cases in the Amazon EC2 forum [4], Netflix open source center [12], Netflix presentations channel on SlideShare [14], Netflix Tech Blog [11] and Yuruware Bolt [2] development experience. Based on the fault taxonomy discussed in [13], we classify the causes into three major classes which partially overlap with each other:
• Development faults: all faults occurring during development of the system. Here we mainly discuss software bugs.
• Physical faults: all faults affecting hardware. Here we talk about hardware issues, network overload and settings, as well as natural disasters.
• Interaction faults: all external faults that are caused by incorrect interaction between the system and the environment. Here, we mainly discuss misconfigurations by API users.

From a cloud user’s point of view, the faults leading to an API failure might not be important because users cannot do anything. This is largely true for development faults or physical faults. Users can avoid interaction faults to some degree and the nature of the all faults can help them deal with the associated failures better. For example, Netflix’s way to deal with failures is to have cascading timeouts. Instead, they try to use historical timing information to detect failures early and fail “fast” to reduce cascading timeouts.

5.1 Development Faults
Software bugs are a significant cause of API issues. In the reported API cases from the Amazon EC2 forum, the majority (83%) of reported software bugs come from EC2 API (other bugs are from third party software). Here are several representatives of API bugs (development faults) which have caused failures:

- the output format of ec2-describe-reserved-instances does not comply with the specification (Jan 1, 2012 4:50 AM);
- the volume states in the outputs of ec2-describe-instances and ec2-describe-volumes do not match (Jan 20, 2011 9:07 PM);
- as-describe-auto-scaling-groups reports wrong launch configuration name for instances which are in ‘Terminating’ state (Apr 17, 2012 9:45 AM);
- the output of --valid-from, --valid-until swapped in ec2-describe-spot-instances (Aug 28, 2012 4:03 PM);
- ec2-describe-instances returns an extra undocumented column after the Kernel ID column (posted on Oct 11, 2011 6:57 AM);
- ec2-describe-spot-price-history does not reporting spot price for Linux m1.large instance type (posted on Oct 25, 2010 9:12 PM).

For most of the EC2 API bugs reported by users, Amazon fixes the reported bugs and releases a new API version. However, the correction process normally takes several weeks.

5.2 Physical Faults
A significant percentage of the causes of the reported cases in the Amazon EC2 forum are because of the degradations faults causing failures of the underlying hardware upon which the instances reside. Amazon usually sends an email to the owner of the instance with degraded hardware notifying the owner of the scheduled date of retirement. The user has to stop and start the instance in order to get new physical machine. The common solution to hardware failures is to move the instance to a new physical machine by stopping and starting the instance.

However, users often experience failures when they try to stop the instance and detach the volume due to the failed or degraded underlying hardware. AWS support team normally advises them to try force stopping or force detachment more than twice to make the operations work. It is normal that a force stopping takes a few minutes to complete. According to comments in API Tool Reference [15], both force stopping and force detachment operations may cause serious negative impact:

- Force stopping: the instance will not be able to flush file system caches or file system metadata. After force stopping, the user must conduct file system check and repair procedures. This option is not recommended for Windows instances.
- Force detachment: force detachment can cause data loss or a corrupted file system. The instance will not have a chance to flush file system caches or metadata. After force detachment, the user also must conduct file system check and repair procedures.

Many users even fail with several tries of force stopping or force detachment. AWS support team has to help users fix the issues on AWS side.

7% of the reported issues in the Amazon EC2 forum are due to problems with EBS volume. An API can fail due to corrupted volumes. Some users even lost all the data from a volume. AWS often advises to make more snapshots of the volume and copy the data to the new instance.

2% of reported issues in the EC2 forum are caused by upgrade of AWS infrastructure, which includes hardware upgrades, volume upgrades, and network upgrades. Users are unable to start, or stop, or connect to the instance after upgrades.

1% issues reported in the Amazon EC2 forum are caused by network issues. Instances are not responsive because of overloaded network or wrong network settings.

5.3 Interaction Faults
Misconfiguration is the most frequent causes (33%) of the reported API cases in the Amazon EC2 forum [4]. We classify misconfigurations into two sub-types: accidental misconfiguration faults and purposeful misconfiguration faults. Accidental misconfiguration faults refer to the inadvertently introduced misconfigurations while purposeful misconfiguration faults means the user configures incorrectly due to lack of knowledge. This terminology mirrors that of the Human Computer Interaction (HCI) community that distinguishes between “slips” –errors where the user knows the correct action but enters an incorrect one– and “mistakes” –errors where the user does not know the correct action [16].

Purposeful misconfiguration faults are often caused because of unclear API documentation. For example, there are many cases in the EC2 forum discussing the API issues due to the misconfiguration of the parameter “-- region REGION”. A user reported he/she solved problem by reading “Common Options for API Tools” [17], which says that region us-east-1 is assumed if not indicated otherwise. So either give the --region option, or set the EC2_URL environment variable. However, the API Tool Reference [17] does not mention how to set EC2_URL other than us-east-1, although it's easy to guess. Good documentation should not require user to guess.
Accidental misconfiguration faults are often due to unclear/no alert messages. For example, in the reported EC2 forum cases, many users terminate the instances without knowing they are unable to restart the instances and many users do not know elastic IPs are available to other users once they are released.

2% of the reported cases in the EC2 forum are that the users are not able to start new instances with returned error message saying new customers are not allowed to launch instances in a certain availability zone. AWS suggests try it again later or creating an Amazon Machine Image (AMI) for the instance and launch into another available zone which should work fine. However, it will have impact on the availability of the user’s application.

Amazon EC2 releases new API versions very frequently [18], e.g. it released two versions in June 2012 and two versions in December 2011. 2% of reported issues in the EC2 forum happen because users used the old API version. Many users meet problems when they use older version of API tools but read the latest API reference documentation. For example, ec2-describe-instance-status is only available since EC2 API tools version 1.5.3. Therefore, it would be helpful, if AWS could point out the available version for each API command in the documentation.

6. TOLERATING API FAILURES
In this section, we discuss how an in-cloud application might tolerate various API failures. Our basic approach is to intercept, within the application, the calls we have identified as the most problematic. That is, we are wrapping those calls with a wrapper designed to deal with the types of failures we have identified. This work is ongoing and what we present here is a proposal that we are continuing to investigate and refine. We describe our thoughts based on the two types of failures: content failures and timing failures. Timing failures subsumes the categories of late timing failures, halt failures, and erratic failures (Note that most of the erratic failures are mainly timing failures).

In general, some failures can only be detected earlier than otherwise, and some can be tolerated.

6.1 Tolerating Content Failures
The types of content failures we visualize tolerating are those that occur when the API returns incorrect or missing information. One approach is to perform reasonableness checks on the information returned. Another approach is to look for API calls that contain the information desired, perhaps in a different context and issue both API calls. In general, our approach is to treat the AWS as unreliable and verify that their responses are reasonable and appropriate.

We presented several examples above of cases where the content returned by the API call was either missing or incorrect. Our examples were:

- There is missing content in the returned output after an API call. This could be detected by a reasonableness check on the API response and could be tolerated by issuing a different API call that might return desired information.
- The output of ValidFrom and ValidUntil were swapped in ec2-describe-spot-instances. Both of these values are times and a failure can be detected since the ValidFrom time should be smaller than the ValidUntil time.
- A user’s volume was attached to an instance the user does not own. The test in this case is that any attached volume must be owned by an appropriate user.
- There were two cases where the state of one API return does not match the state of another. This can be detected by issuing both API calls where one would be sufficient if AWS were treated as trusted.

6.2 Tolerating Timing Failures
Our approach with respect to timing failures is to adapt workflow patterns of [19, 20]. The adapted patterns assume there are six states of an API call: requested, cancelled, allocated, started, failed and completed. Fig. 5, adapted from workflow exception patterns [19], shows API related failures/faults tolerance patterns overlaid on these six states to form a state machine. Below we describe how we utilize these patterns.

![Figure 5. API failure/fault tolerance patterns.](attachment:image.png)
7. RELATED WORK

Deploying applications in cloud environments will introduce uncertainties for operations that have traditionally been under the direct control of an enterprise. Enterprises will be dependent on cloud infrastructure and will need to use indirect means through APIs to guarantee the achievement of their quality goals such as availability and performance [22]. In cloud systems, runtime failures are due to different reasons. For example, one availability zone may go down; a physical machine may experience a hardware error; a database might be overloaded; an operating system may crash [23]; software may have bugs. Cloud infrastructure providers may not fully disclose the causes of outages or cloud infrastructure design for competitive reasons, which makes the study of API issues more important.

Misconfiguration is another major source that causes system failures, including in-cloud application failures. Yin et. al. conducted an empirical study on system misconfiguration issues [24]. They studied 546 misconfiguration issues and found that a majority of misconfigurations are due to parameter setting mistakes while a significant number of misconfigurations are due to compatibility issues and component misconfigurations. Configuration changes in cloud are mostly done through API calls. Our empirical study focuses on the processes of changing configurations and the associated failures and faults.

Microsoft researchers analysed cloud hardware failure and faults [25]. There are several interesting findings: hard disks are the most frequent failed hardware due to its frequent usage and unreliability; 8% of servers in the data center can experience at least one hardware incident a year; if a failure happens, the occurrence rate of another failure in the same server is high. Gill et. al. [26] found that networks in data centers are highly reliable. However, load balancers experience many software faults and network redundancy is not entirely effective. Many of these failures are reflected differently at the API level where the users may not know the underlying causes. Our empirical study and suggested architecture decisions help tolerate the issues at the API failure level.

As discussed in Section 4, a significant portion (12%) of the reported cases is related to slow API responses. Dean from Google summarized the reasons of slow response API responses [27]: 1) different applications may reside upon one machine and share resources (e.g. memory and CPU cores); 2) applications running on different machines may share global resources (e.g. shared file systems and network switches); 3) background programs may generate latency; 4) various queuing in network switches and intermediate servers may cause latency. Dean believes that resource overprovisioning, real-time engineering of software, and improved reliability can help reduce the causes of API call latency. However, it is impossible to eliminate all API call latency. Therefore, Google proposes two tail-tolerant techniques to deal with API call latency [27]: 1) Within-request immediate-response technique that is to issue the request to multiple replicas and use the first responded results; 2) cross-request long-term adaptation technique that is to issue different requests to different partitioned data. Our empirical study offers a more detailed classification of API failures and adopts an operation-process viewpoint for handling failures. This effort may expand the range of architecture-level tolerance techniques.

At the application deployment level, approaches like [28] were proposed to optimize reliability, latency and energy when
application components are deployed onto physical machines. However, the deployment platform involves physical machines where one has full control/visibility rather than infrastructures with specific auto-scaling facilities and failures ranging from individual nodes to entire region.

A common complaint is that the AWS documentation is out of date or unclear. This is also a finding of Yin et al’s work on configuration errors [24]. On the one hand, there are a half century of complaints about documentation and it would seem that this should be a solved problem. On the other, AWS has a very rapid upgrade cycle (on the order of a new release every several week) and this pace lends itself to documentation problems.

8. CONCLUSIONS
Cloud API is important for in-cloud applications since the availability of in-cloud application heavily relies on the reliability of both cloud infrastructure and API calls. In this paper, we have performed a comprehensive study on 922 API issues reported in the EC2 forum, classified the failures and causes. We also discussed initial ways of tolerating API failures by reasonableness checking responses and adapting workflow exception patterns.

9. ACKNOWLEDGMENTS
NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

10. REFERENCES