Push-Button Reliability Testing for Cloud-Backed Applications with Rainmaker

Yinfang Chen, Xudong Sun, Suman Nath, Ze Yang, Tianyin Xu
The emerging cloud-based programming model
The emerging cloud-based programming model

- Azure has over 700 million users
- Azure storage SDK (.NET) has ~80K daily downloads
Benefits of cloud-based programming

- Scalability
- Availability
- Flexibility
- Easy deployment
Dark side: new reliability challenges

- Diverse fault domains
Dark side: new reliability challenges

- Diverse fault domains
- A lack of standards
  - No standards such as POSIX
- Inconsistencies
  - E.g., AWS S3 SDKs in different languages treat “limit exceeded” error differently
Dark side: new reliability challenges

It is challenging for application developers to anticipate all faulty scenarios and write comprehensive error-handling code.
Does retry solve all the problems?

The request succeeds but it leads to an unexpected error.

SDK API call → CreateBlob → Blob Created

SDK → CreateBlob → Blob Already Exists

Exception → BlobAlreadyExists

Timeout

BotBuilder → Azure Blob Storage
Does retry solve all the problems?

The API semantic is silently violated

The diagram shows the flow of messages between Orleans and Azure Queue Storage. The `GetQueueMessage()` call from Orleans results in an `SDK API call` to Azure Queue Storage, which can result in `Dequeue` operations. If a `timeout` occurs, the message is not dequeued. The API is non-idempotent as indicated by the red warning in the diagram.
How can applications address the emerging reliability challenges of cloud-based programming?
Contribution

• A call for attention of the emerging reliability challenges of cloud based programming

• A taxonomy of error-handling bugs triggered by transient faults

• Rainmaker: Push-button reliability testing for cloud-backed apps
  • Systematically exercise error-handling code under common faults
  • Detected 73 new bugs in 11 cloud-backed apps (51 fixed)
  • Released at https://github.com/xlab-uiuc/rainmaker
Design goals of Rainmaker

• **Effective**: Detect error-handling bugs of different patterns

• **Easy-to-use**: Directly applied to existing testing environment

• **Efficient**: Efficiently finish testing while ensuring coverage
Design goals of Rainmaker

- **Effective**: Detect error-handling bugs of different patterns
- **Easy-to-use**: Directly applied to existing testing environment
- **Efficient**: Efficiently finish testing while ensuring coverage

Fault injection during testing, before production

**What** faults to inject? **When** to inject them?
A taxonomy of error-handling bugs

Only consider realistic transient error(s) that occur during one REST API call interaction
  - Timeout
  - Server-busy error
Throwing unrelated exceptions

Key: Mishandling leads to a new error unrelated to the root cause error

Buggy error handling

Throwing unrelated exceptions

Azure Blob Storage

SDK

BotBuilder

SDK API call

CreateBlob

Blob Created

Blob Already Exists

Exception

409 BlobAlreadyExists

timeout CreateBlob

State divergence
Silent semantic violations

Key: Mishandling causes semantic violations of the application

Buggy error handling

Throwing unrelated exceptions

State divergence

GetQueueMessage()
State divergence

Key: Mishandling causes divergence between the local and the remote state

Buggy error handling

State divergence

Throwing unrelated exceptions

SDK API call

containerSet.add(c)
CreateContainer(c)

SDK

BotBuilder

Exception

catch(ex)

Container Created

Server busy

Container Not Created

Azure Blob Storage

Create Container

Create Container

Server busy
Rainmaker’s fault injection policies

Buggy error handling

- Throwing unrelated exceptions
- State divergence

Silent semantic violations

- Throwing unrelated exceptions
- State divergence

Application
Cloud service

$P_1$: Timeout the first response

200 or 4XX

timeout

$P_2$: Return 5XX to all requests

5XX
5XX

Application
Cloud service

Throwing unrelated exceptions

Silent semantic violations

Rainmaker has more policies to trigger bugs
Design goals of Rainmaker

• Effective: Detect error-handling bugs of different patterns

• Easy-to-use: Directly applied to existing testing environment

• Efficient: Efficiently finish testing while ensuring coverage
Rainmaker performs HTTP layer injection

- Test case
  - Server busy
  - Response timeout

- Rainmaker HTTP proxy
  - Intercept request
  - Intercept response

- Cloud service
Rainmaker reuses existing test oracles

- Naively reusing oracles could lead to false alarms
- Analyze test execution and output to capture only true alarms

```
// test code
fn unit_test() {
    // set up set env
    SDK.CreateBlob();
    ...
    // call app code
    ...
}
```

The test failure does not point to any error-handling bug in application code.

Solution: Rainmaker checks the stack trace of the exception. If the SDK is directly invoked by test code, it does not report an alarm.
Design goals of Rainmaker

• **Effective**: Detect error-handling bugs of different patterns

• **Easy-to-use**: Directly applied to existing testing environment

• **Efficient**: Efficiently finish testing while ensuring coverage
Ensure coverage while being efficient

Injecting to every REST call takes \textbf{588 days} to test Orleans!

\[ \sim 56 \text{ hours} \]

### Coverage Metric

- Cover all combinations of (test case, SDK API*)
- Cover all test cases and SDK API respectively
- Cover all SDK API
- Cover all test cases

<table>
<thead>
<tr>
<th>Default</th>
<th>Weaker</th>
<th>Faster</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Generating test plans

- Rainmaker generates the test plan that achieves the coverage with minimized test running time for each coverage metric.

A linear optimization problem
Variables: Test cases and SDK APIs
Constraints: Coverage requirements
Objectives: Minimized test running time

Linear optimization solver

Test plan
Evaluation

• We applied Rainmaker to 11 popular cloud-backed applications

• Rainmaker finds 73 new bugs with severe consequences

• Rainmaker has a low false-positive rate 1.96%

• Rainmaker reduces on average 64.47% of test runs compared to exhaustively injecting to every REST call
# Finding new bugs

<table>
<thead>
<tr>
<th>Cloud-backed application</th>
<th>No Error Handling</th>
<th>Unrelated Exception</th>
<th>Semantic Violation</th>
<th>State Divergence</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alpakka</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>AttachmentPlugin</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>BotBuilder</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>DistributedLock</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>EF Core</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>FHIR Server</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>Insights</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>IronPigeon</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Orleans</td>
<td>0</td>
<td>5</td>
<td>2</td>
<td>11</td>
<td>18</td>
</tr>
<tr>
<td>Sleet</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Storage.NET</td>
<td>11</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>14</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>29</strong></td>
<td><strong>23</strong></td>
<td><strong>4</strong></td>
<td><strong>17</strong></td>
<td><strong>73</strong></td>
</tr>
</tbody>
</table>

55 confirmed; 51 fixed
Conclusion

• A call for attention of the emerging reliability challenges of cloud based programming

• A taxonomy of error-handling bugs triggered by transient faults

• Rainmaker: Push-button reliability testing for cloud-backed apps
  • Effective, easy-to-use, efficient
  • Released at https://github.com/xlab-uiuc/rainmaker