PreFail: Programmable and Efficient Failure Testing Framework

Pallavi Joshi, Haryadi S. Gunawi, Koushik Sen
UC Berkeley

Motivation

- Large scale distributed systems face frequent, multiple, and diverse hardware failures
- Recovery protocols are often buggy
- Most of the previous work on failure testing focuses on single failures
- For multiple failures, brute force has to explore huge (e.g. >40,000) number of failure scenarios
- Thus new challenge: combinatorial explosion of multiple failures

Failure Testing

Example Program

Node A

<table>
<thead>
<tr>
<th>Exp</th>
<th>Node A</th>
<th>Node B</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1</td>
<td>write(B, msg)</td>
<td>write(A, msg)</td>
</tr>
<tr>
<td>L2</td>
<td>read(B, header)</td>
<td>read(A, header)</td>
</tr>
<tr>
<td>L3</td>
<td>read(B, body)</td>
<td>read(A, body)</td>
</tr>
<tr>
<td>L4</td>
<td>write(B, msg)</td>
<td>write(A, msg)</td>
</tr>
<tr>
<td>L5</td>
<td>write(Disk, buf)</td>
<td>write(Disk, buf)</td>
</tr>
</tbody>
</table>

Failure ID (FID)

<table>
<thead>
<tr>
<th>I/O ID Fields</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td></td>
</tr>
<tr>
<td>src loc</td>
<td>Write.java (line L1)</td>
</tr>
<tr>
<td>Dynamic</td>
<td></td>
</tr>
<tr>
<td>node</td>
<td>A</td>
</tr>
<tr>
<td>target</td>
<td>B</td>
</tr>
<tr>
<td>stack</td>
<td>(the stack trace)</td>
</tr>
<tr>
<td>Domain Specific</td>
<td>network msg.</td>
</tr>
<tr>
<td></td>
<td>“Heartbeat Msg”</td>
</tr>
</tbody>
</table>

Failure ID = hash (I/O ID + Crash) = 2849067135

Programmable Failure Testing

- Testers write policies to indicate the subset of failure space to explore
- Policies can be of varying complexities
- Policies can help achieve different coverage criteria
- Filter policy
  - Express which failure sequences to exercise
- Cluster policy
  - Express the equivalence of two failure sequences
  - Only one failure sequence from an equivalence class is exercised

Optimizations

- Crashes before writes
- Read failures/corruption on first reads
- No crashes/network failures for dead nodes

Efficient Failure Testing

- Triaging
  - Cluster according to root cause (bug)
  - Sort according to bug type

Parallelization

- Experiments with failure sequences of a particular length i are distributed across m machines

Policies

- Filter policy
  - Express which failure sequences to exercise
- Cluster policy
  - Express the equivalence of two failure sequences
  - Only one failure sequence from an equivalence class is exercised

Recovery Coverage

- Equivalence classes

Evaluation

- Target systems: HDFS, Zookeeper, Cassandra
- 6 new, 16 old bugs found
- Data loss, unavailability
- Reduction of experiments with policies: 1 to 3 orders of magnitude
- Optimizations: 5 times (average)

Testing workflow

<table>
<thead>
<tr>
<th>FID</th>
<th>Seqs</th>
<th>Policies</th>
<th>FID</th>
<th>Seqs</th>
<th>Policies</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>failure/exp</td>
<td>A B C AD AE BE</td>
<td>2</td>
<td>failures/exp</td>
<td>A D E AD F G</td>
</tr>
</tbody>
</table>

Efficient Failure Testing

- Optimizations
  - Crashes before writes
  - Read failures/corruption on first reads
- No crashes/network failures for dead nodes

Triaging

- Cluster according to root cause (bug)
- Sort according to bug type

Parallelization

- Experiments with failure sequences of a particular length i are distributed across m machines