ReproLite:
A Lightweight Tool to Quickly Reproduce Hard System Bugs

Kaituo Li (U. Massachusetts, Amherst)
Pallavi Joshi (NEC Labs America)
Aarti Gupta (NEC Labs America)
Malay K. Ganai (NEC Labs America)
Outline

Reproduce what bugs? Why IReproLite?

**Motivation**
- e.g. ASPLOS 2012 “Portend”
  - Programmable scheduler
- e.g. PLDI 2013 “Concurrit”
- e.g. static-analysis
- e.g. ASPLOS 2010 “Sherlog”
  - Provides assistance to understand the cause of a bug from logs
  - Focus on system-level concurrency issues

**RE: Reproduction Engine**
External events: cause them to happen
Internal events: wait for them

**Structure**

**Cassandra bug 1477**

**Example Walkthrough**

**Implementation**

**How?**

**Experiments**

We experimented with using ReproLite to debug 11 bugs from

**So what?**

- Instrumentation Overhead
  - Almost nil for half of the bugs
- Usefulness of Logs
  - 70% percent of bug causes hidden in logs
Motivation
Cloud Systems

- Node 1
  - Process 1
  - Disk 1
- Node 2
  - Process 2
  - Disk 2
- Node 3
  - Process 3
  - Disk 3

- Client1
  - Process
- Client2
  - Process
ReproLite’s Approach: 3 Main Elements (RL, RT, RE)

- **RL** (log analyzer)
- **RT** (bug scenario in DSL)
- **RE** (reproduction engine or scheduler)

Cloud System

Observe Events

Control events
Reproduce what bugs? Why ReproLite?

- **System-level bugs involving**
  - Concurrency
  - System component interaction
  - Environment non-determinism

Main new element: DSL for scenarios + scheduler

- **Benefits**
  - RT (DSL)
    - Expressive for specifying bug scenarios in a deterministic manner
  - RL (log analyzer)
    - Provides assistance to understand the cause of a bug from logs
  - RE (scheduler)
    - Repeatedly reproduce
    - Lightweight
Structure
**RL: Parse, extract and diff buggy and non-buggy logs**

<table>
<thead>
<tr>
<th>Log level</th>
<th>Date &amp; Time</th>
<th>Thread name</th>
<th>File name</th>
<th>Line number</th>
<th>Log message</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEBUG [GC inspection]</td>
<td>2014-10-22 16:23:29.160</td>
<td></td>
<td>GCInspector.java (line 131)</td>
<td></td>
<td>GC for ParNew: 5 ms, 266519024 reclaimed leaving 28339384 used; max is 1263271936</td>
</tr>
</tbody>
</table>
RE: Reproduction Engine

RE control external events: cause them to happen

RE observe internal events: wait for them to happen
**RT: DSL for expressing bug scenario**

<table>
<thead>
<tr>
<th>Bug Scenario</th>
<th>Internal Event</th>
<th>External Event</th>
<th>Workload</th>
<th>Scheduling Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Run in sequence**
  - E1 * E2
- **Left and right run in parallel**
  - E1 || E2
Cassandra
Cassandra bug 1477
**DSL example**

W1 = 'create-ks.sh test'
W2 = 'add-cf.sh test cf'
W3 = 'write.sh 20 test cf'
W4 = 'flush.sh'
W5 = 'drop.sh cf'
W6 = 'add-cf.sh test cf'
W7 = 'read.sh test cf'

E1 [ stack ]= 'File.delete'
E2 [ stack ]= 'BufferedSegmentedFile.getSegment'

A workload given by bash script names and arguments

Stack traces to identify an internal event

Bug scenario
Example Walkthrough
Users write code in DSL: First attempt

W1 = 'create-ks.sh test'
W2 = 'add-cf.sh test cf'
W3 = 'write.sh 20 test cf'
W4 = 'flush.sh'
W5 = 'drop.sh cf'
W6 = 'add-cf.sh test cf'
W7 = 'read.sh test cf'

E1 [ stack ]= 'File.delete'
E2 [ stack ]= 'BufferedSegmentedFile.getSegment'

W1 * W2 * W3 * W4 * ( W5 * W6 * W7 || E1 * E2 )
Garbage collection (gc): reclaim memory objects that are no longer used

Log messages related to FileNotFound Exception

+GCInspector.java (line 131)
+CassandraDaemon.java (line 84)
+(No such file /tmp/rp/cass/server1/data/ks/cf-e-1-Data.db)
+Cassandra.java (line 2761)
Second attempt

Client

create
delete
recreate
read

An external event given by action and node ID

Cassandra

FileNotFoundException Exception

collection 'cnode'

W1 * W2 * W3 * W4 * ( W5 * W6 * W7 || X1 * E1 * E2 )
Rely on logs and source code whenever possible

buggy-proc.out

SSTableReader.java

```java
471  logger.debug("Marking " + getFiles(...)
472  try
473  {
474      if (!new File(desc.filenameFor(...))
475          throw new IOException("Unabl...
476      }
477      catch (IOException e)
478      {
479          throw new IOErr...
480      }
481  }phantomReference.deleteOnCleanup();
```
Final attempt

Client

- create
- delete
- recreate
- read

Cassandra Node

FileNotFoundException Exception

E3 [ stack ]= 'SSTableDeletingReference.deleteOnCleanup'

W1 * W2 * W3 * W4 * ( W5 * W6 * W7 || E3 * X1 * E1 * E2 )
How? (Implementation)
Implementation

RL: Filter, concatenate, diff logs → RT → Workload

Event order

Workload Executor
External Events Emulator

RE

System Under Test

Node instrumented... Node instrumented

Python
AspectJ
RPC
Java
Bash
So What? (Experiments)
Experiments

We experimented with using ReproLite to debug 11 bugs from

Cassandra    HBase

**Complexity**  \( \leq 6 \) events involved in each bug

**Performance Overhead**  0\~132, close to 0 for half of the bugs

**Usefulness of Logs**  70\% percent of bug causes hidden in logs.
Conclusion

Users write code in DSL: Final attempt

High-level language specifies bug scenario

RE: Reproduction Engine

Tool enforces scheduling in distributed environment

Experiments

Illustrated benefits with 11 hard system bugs

Usefulness of Logs

20% percent of bug causes hidden in logs.
Thanks