Pip: Detecting the Unexpected in Distributed Systems

Patrick Reynolds, Charles Killian, Janet Wiener, Jeffrey Mogul, Mehul Shah

Presented by:
Aaron Schulman

Some slides and images were borrowed from NSDI Talk
Introduction

- Distributed systems are complex
- Harder to debug than centralized systems
- Often, deviations from expected behavior indicate bugs
- Characterize system behavior
Distributed Systems (DS) are hard to test

- Distributed systems are subject to:
  - Independent node failures
  - Incorrect synchronization of parallel tasks
  - Network errors
  - Security Breaches
Goal

Develop tool to aid DS programmers by identifying bugs.
DS testing tools
<table>
<thead>
<tr>
<th>Approach</th>
<th>Scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>gdb and gprof</td>
<td>low level bugs at a single node / core dumps</td>
</tr>
<tr>
<td>black box testing</td>
<td>enough consistency to do statistical analysis</td>
</tr>
<tr>
<td>model checking</td>
<td>small system with hard to reproduce bug</td>
</tr>
<tr>
<td>printf</td>
<td>bugs detected with local log</td>
</tr>
</tbody>
</table>
## DS testing tools

<table>
<thead>
<tr>
<th>Approach</th>
<th>Scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>gdb and gprof</td>
<td>low level bugs at a single node / core dumps</td>
</tr>
</tbody>
</table>

- **gdb and gprof**: These tools are used for debugging and profiling. `gdb` is a general-purpose debugger, and `gprof` is a Unix/Linux command-line tool for profiling.

- **Scenario**: This scenario involves analyzing low-level bugs that occur at a single node or core dump. The approach involves using `gdb` and `gprof` to identify and resolve these bugs.

- **Approach**: The approach in this scenario is to use black box testing to ensure enough consistency to do statistical analysis. Additionally, `printf` is used to detect bugs related to local log files.

- **Scenario**: The scenario for this approach is suitable for small systems with hard-to-reproduce bugs. It involves using causal path analysis to understand the distribution of bugs across the system.
# DS testing tools

<table>
<thead>
<tr>
<th>Approach</th>
<th>Scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>gdb and gprof</td>
<td>low level bugs at a single node / core dumps</td>
</tr>
<tr>
<td>black box testing</td>
<td>enough consistency to do statistical analysis</td>
</tr>
</tbody>
</table>
## DS testing tools

<table>
<thead>
<tr>
<th>Approach</th>
<th>Scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>gdb and gprof</td>
<td>low level bugs at a single node / core dumps</td>
</tr>
<tr>
<td>black box testing</td>
<td>enough consistency to do statistical analysis</td>
</tr>
<tr>
<td>model checking</td>
<td>small system with hard to reproduce bug</td>
</tr>
</tbody>
</table>
## DS testing tools

<table>
<thead>
<tr>
<th>Approach</th>
<th>Scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>gdb and gprof</td>
<td>low level bugs at a single node / core dumps</td>
</tr>
<tr>
<td>black box testing</td>
<td>enough consistency to do statistical analysis</td>
</tr>
<tr>
<td>model checking</td>
<td>small system with hard to reproduce bug</td>
</tr>
<tr>
<td>printf</td>
<td>bugs detected with local log</td>
</tr>
</tbody>
</table>
## DS testing tools

<table>
<thead>
<tr>
<th>Approach</th>
<th>Scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>gdb and gprof</td>
<td>low level bugs at a single node / core dumps</td>
</tr>
<tr>
<td>black box testing</td>
<td>enough consistency to do statistical analysis</td>
</tr>
<tr>
<td>model checking</td>
<td>small system with hard to reproduce bug</td>
</tr>
<tr>
<td>printf</td>
<td>bugs detected with local log</td>
</tr>
<tr>
<td>Causal Path Analysis</td>
<td>distributed system does not behave as expected</td>
</tr>
</tbody>
</table>
Causal Path Analysis

Example:
Web service system

Web server
App server
Database
Causal Path Analysis

- Path is caused by input to system
- e.g. user request from web service

Example:
Web service system

- Web server
- App server
- Database
Causal Path Analysis

- Path is caused by input to system
  - e.g. user request from web service
- Components delay

Example:
Web service system

- Web server
- App server
- Database

500ms
Causal Path Analysis

- Path is caused by input to system
  - e.g. user request from web service
- Components delay
- Attribution of resource consumption

Example:
Web service system

- Web server
- App server: 500ms
- Database: 2000 page faults
Path Instances

"Request = /cgi/…"

"2096 bytes in response"

"done with request 12"

WWW

Parse HTTP

Send response

App srv

Run application

DB

Query

time
Path Instances

- Within paths are tasks, messages, and notices.

```
“Request = /cgi/...”
“2096 bytes in response”
“done with request 12”
```

```
WWW
<table>
<thead>
<tr>
<th>Parse HTTP</th>
</tr>
</thead>
<tbody>
<tr>
<td>App srv</td>
</tr>
<tr>
<td>DB</td>
</tr>
<tr>
<td>Query</td>
</tr>
<tr>
<td>Run application</td>
</tr>
<tr>
<td>Send response</td>
</tr>
</tbody>
</table>
```

(time)
Path Instances

- Within paths are *tasks*, *messages*, and *notices*
  - **Tasks**: processing with start and end points
Path Instances

• Within paths are *tasks*, *messages*, and *notices*
  – **Tasks**: processing with start and end points
  – **Messages**: send and receive events for any communication
Within paths are *tasks*, *messages*, and *notices*

- **Tasks**: processing with start and end points
- **Messages**: send and receive events for any communication

*Includes network, synchronization (lock/unlock), and timers*
Path Instances

- Within paths are tasks, messages, and notices
  - Tasks: processing with start and end points
  - Messages: send and receive events for any communication
    - Includes network, synchronization (lock/unlock), and timers
  - Notices: time-stamped strings; essentially log entries

WWW → Parse HTTP → Run application → Query → Send response

“Request = /cgi/…” → “done with request 12” → “2096 bytes in response”
Pip workflow
Pip workflow
Pip workflow

1. Captures events from an instrumented system
Pip workflow

1. Captures events from an instrumented system
2. Generate paths from events
1. Captures events from an instrumented system
2. Generate paths from events
3. Checks behavior against expectations
1. Captures events from an instrumented system
2. Generate paths from events
3. Checks behavior against expectations
4. Displays unexpected behavior
Main Contribution: Expectations

Declarative language to describe a DS’s expected path.
Describing Expected Behavior

Recognizers
Description of Behavior
Structural or Performance

Aggregates
Assertions about sets of path instances
Expectation: Recognizers

- Validator, invalidator, building block
- Can match a complete path or fragment
  - Invalid paths are often represented as fragments
Expectation: Recognizers

- Validator, invalidator, building block
- Can match a complete path or fragment
- Invalid paths are often represented as fragments
validator CGIRequest
    thread WebServer(*, 1)
    task("Parse HTTP") limit(CPU_TIME, 100ms);
    notice(m/Request URL: .*/);
    send(AppServer);
    recv(AppServer);

invalidator DatabaseError
    notice(m/Database error: .*/);

“Request = /cgi/...”
“2096 bytes in response”
“done with request 12”

WWW
App srv
DB

Parse HTTP
Run application
Query
Send response
Other Statements

- **repeat**: matches $a \leq n \leq b$ copies of a block

- **xor**: matches any one of several blocks

```plaintext
xor {
    branch: ...
    branch: ...
}
```

- **future**: lets a block match now or later
  - **done**: forces the named block to match
Expectation: Aggregate Paths

- Recognizers categorize paths into sets
- Aggregates make assertions about sets of paths
  - Instances, unique instances, resource constraints
  - Simple math and set operators

```plaintext
assert(instances(CGIRequest) > 4);
assert(max(CPU_TIME, CGIRequest) < 500ms);
assert(max(REAL_TIME, CGIRequest) <=
       3*avg(REAL_TIME, CGIRequest));
```
Results

• Applied Pip to several distributed systems:
  – **FAB**: distributed block store
  – **SplitStream**: DHT-based multicast protocol
  – Others: RanSub, Bullet, SWORD, Oracle of Bacon

• We have found unexpected behavior in each system

• We have fixed bugs in some systems
  … and used Pip to verify that the behavior was fixed
Conclusions

• Causal paths are a useful abstraction of distributed system behavior
• Expectations serve as a high-level description
  – Summary of inter-component behavior and timing
  – Regression test for structure and performance
• Finding unexpected behavior can help us find bugs
  – Both structure and performance bugs