Providing Cloud Resiliency:
Fault Localization using Message Flow Reconstruction and Targeted Fault Injection

Zbigniew Kalbarczyk
Cuong Pham, Ravi Iyer
Information Trust Institute and Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign
Outline

• Motivation

• Approximate Fault Localization (AFL)
  - Concept
  - Approach

• Framework overview

• Targeted fault injection to generate failure profiles

• Evaluation

• Conclusions
Motivation

Cloud Computing is becoming mainstream

but its Reliability

and Security

... remain an increasing concerns

Image source: Google Trends for Searches

Achieving Resilient Cloud Computing

- Runtime failures inevitable
  - accidental errors
  - software bugs
  - malicious attacks

- Need efficient monitoring, detection, and recovery from runtime failures

- Need accurate failure diagnosis to enable system and application fixes

- This talk introduces a low-cost approach for approximate fault localization (and hence, failure diagnosis) in a distributed computing infrastructure such as cloud
Fault Localization/Failure Diagnosis

Manual inspection

- Diagnosing a problem starts from certain locations indicated by failure profiles (e.g., error messages, a call stack)
- Trace backward through the execution flows to identify the origin of the problem
- Attempt to identify exact fine-grained locations (e.g., at program statement level) of failure’s root causes
- Prohibitively expensive, particularly in large distributed systems, such as cloud infrastructure

Automated (exact) fault localization

- Manual effort to diagnosing and fixing bug
- Error Messages Stack Dump
- Failure detected
- Manual effort to understand surround context and fix bug
- Automated (exact) fault localization
- Failure detected
Approximate Fault Localization: Concept

Assumption: Similar failure profiles imply similar faults
Approximate Fault Localization: Approach

• Upon a failure in a system collect a failure profile
  - e.g. in terms of sent and received messages

• Process failure profile to reconstruct an end-to-end processing flow corresponding to the failure
  - a sequence of system events across distributed components invoked to process a user/application request

• Use the reconstructed processing flow to query against a pre-constructed failure profiles stored in Failure Profiles Database
  - Use “string edit distance” metric to identify similar flows and “pinpoint” the fault location
Message Flow Reconstruction and Comparison

- Need to represent event flows so to enable fast identification of similar flows

- Event flows translated into event strings
  - an event in a string represented as a letter that corresponds to the source component of this event, e.g., \textit{BBABCABCB}
  - event order based on timestamps

- Compare flows using \textit{String Edit Distance}
  - the minimum number of insertion, deletion, or replacement of a letter required for changing one string into the other
Example: Edit Distance

Run 1 → BBABCABCB → Edit Distance = 5
Run 2 → BBABC CCBAB
Enabling Techniques

- **Distributed Events Tracing**
  
  record system events (e.g., syscall, library call) in distributed systems

- **Message Flow Reconstruction and Comparison**
  
  quantify the dissimilarity between failure profiles

- **Targeted Fault Injection**
  
  deterministically inject faults at exact locations in the execution flow of a distributed system
Framework Overview

Failure Database Construction

- Profiling
- Generating FI Campaign Specification
- Conducting Targeted FI Campaign
- Failure Profile Collection
- Processing of Failure Profiles

Fault Localization Process

- Failure Report
- Collecting Failure Profile
- Processing Failure Profile
- Querying FPDB
- Cluster/Ranking
- Candidate Root Causes

Failure Profile Database

Import to DB
Data Cleanup

• Collected end-to-end flows must be cleaned up to remove non-deterministic events
  - system noise, i.e., periodic messages such as heartbeats
  - message fragmentation,
  - out-of-order messages

• Non-determinisms in the processing flows make trace comparison non-trivial

• In order to automate the diagnosis we assume that
  - processing of a request is deterministic
Targeted Fault Injection

• Allows inserting faults precisely at the intended location
  - Based on the processing flow of each request as the request traverses multiple components

• Minimize side effects to target systems
  - Non-intrusive - no source code modification required
  - Fast and light weigh communication between FI components

• Precise tracking and synchronization of event sequences
  - Catch system level events (e.g., libc function calls)
  - Global synchronization when an event is captured

• Easy to use
  - Compact, reusable specification to define FI experiments
Targeted Fault Injection Approach

Profile the system under workloads to identify injection points and causal event sequences

A specification for defining precise fault injection scenarios

Execution of automated FI experiments

Failure profile database, Reliability assessment

Inserting faults at precisely specified execution points
On creating VM request, before the request state changes to “networking”, inject a crash to the nova-network process.
• **Target Applications:** multiple processes across multiple nodes
• Each node: One *Local Controller*
• Each process: One *Injector (libfi)*, one *Flow Tracer*
• *FI Central Controller* operates in an event-driven fashion to drive the injection
Evaluation

- **Target:** OpenStack, an open source distributed cloud management system

- **Validation**
  - Do similar failure profiles imply similar faults?

- **Evaluation of AFL Accuracy**
  - Identification of fault type and affected component(s)
  - *Fault distance* between the determined fault location and the actual injected fault (Top-K nearest faults)
    - fault distance measured as the number of \textit{libc} calls between the determined approximate fault location and the actual fault location in the end-to-end flow of fault-free execution
### Construction of Failure Profile Database (FPDB)

| Fault Type        | Location Type | \( |F| \) | \( |FP| \) |
|-------------------|---------------|------|------|
| Process Crash     | All monitored libc calls | 23323 | 116589 |
| Message Corruption| All `read, write, send, and recv` libc calls | 18221 | 91092 |
| Deadlock          | All thread and lock related libc calls | 2143  | 10702 |

- The FPDB is constructed for VM Provision (nova boot) requests
- Five failure profiles collected for each fault
- Fault models:
  - Contained faults: *Process crash, deadlock* (within a process)
  - Propagated faults: *Message corruption*
Do Similar Failure Profiles Imply Same faults?

More than 80% of all the injected faults, across all three fault models, result in less than 4% of the failure profile variation.
OpenStack Error Reporting

Error Message Coverage

- Sheduler: 1%
- Conductor: 99%
- Network: 99%
- Compute: 48%
- API: 64%

Error Dection Latency

- Sheduler: 1530
- Conductor: 2210
- Network: 3120
- Compute: 3980
- API: 2250

Crashed Nova components during VM provisioning
**Accuracy of AFL: Determining Fault Type and Affected Component(s)**

<table>
<thead>
<tr>
<th>Fault Type Query Accuracy</th>
<th>Targeted Component Query Accuracy</th>
<th>Both</th>
</tr>
</thead>
<tbody>
<tr>
<td>99.5%</td>
<td>99.1%</td>
<td>99.1%</td>
</tr>
<tr>
<td>94.2%</td>
<td>94.2%</td>
<td>94.2%</td>
</tr>
<tr>
<td>93.1%</td>
<td>93.0%</td>
<td>93.1%</td>
</tr>
<tr>
<td>88.2%</td>
<td>87.5%</td>
<td>87.5%</td>
</tr>
<tr>
<td>88.2%</td>
<td>87.5%</td>
<td>88.2%</td>
</tr>
<tr>
<td>87.5%</td>
<td>98.4%</td>
<td>98.4%</td>
</tr>
<tr>
<td>98.3%</td>
<td>98.4%</td>
<td>98.3%</td>
</tr>
<tr>
<td>100.0%</td>
<td>100.0%</td>
<td>100.0%</td>
</tr>
</tbody>
</table>

**Known fault:** a fault that has at least one failure profile in failure database

**Unknown fault:** a fault that does not have any failure profile in failure database
Accuracy of AFL: Top-K Nearest Faults for Known Faults

50% of the Top-1 query results contain the exact fault locations, i.e., fault distance is zero
Accuracy of AFL: Top-K Nearest Faults for Unknown Faults

Two orders of magnitude better than OpenStack’s error reporting mechanism
Conclusions

• Develop low-cost method for the approximate fault localization
  - reduce the cost of fault diagnostic while providing precision close to the methods used for the exact fault localization
  - support large complex distributed environments such as the cloud computing

• Demonstrate effectiveness of the prototype implementation on the OpenStack
  - effective in determining (approximate) fault/error locations
  - highly accurate in identifying the failure types and the affected components
Sponsors and Other Collaborators

- Long Wang  (IBM T. J. Watson)
- Byung Chul Tak  (IBM T. J. Watson)
- Salman Baset  (IBM T. J. Watson)
- Chunqiang Tang  (Facebook)

- Sponsors: AFRL, NSA, NSF
- IBM