Automated Diagnosis of Software Misconfigurations Based on Static Analysis

Zhen Dong
Mohammadreza Ghanavati
Artur Andrzejak
Heidelberg University
artur@uni-hd.de
http://pvs.ifi.uni-heidelberg.de/
Complex Software Needs Customization

- Non-trivial applications need adaptation to a particular deployment scenario
  - Locations of Files, Functionality, Working Modes ...
- Configuration achieves this without recompiling

Customization is critical for software to run efficiently and correctly
What is a Configuration?

- **Configuration**: a set of constants stored in a user-editable file or accessible via GUI
- Typical schemas
  - *Key Value* and *Structured XML*
- We focus on the key value schema

```
chord.main.class = foo.Main
```

**Configuration option**

**Configuration option value**
Complexity => Misconfiguration

• High complexity (in terms of number of options):
  – Hadoop: 200+, JChord: 70+, Derby: 69, Hbase: 64, ...
• Traps: value inconsistencies (with environment or other options); cryptic names; formatting; typos; ...

Configuration defects  ➔  Misconfiguration!

Configuration options

Wrong inputs

Inputs

Outputs

Bugs
Impact of Miconfigurations

- Distribution of problem causes on cases with high severity
  - Commercial storage systems app with 1000s of users
  - From: An Empirical Study on Configuration Errors in Commercial and Open Source Systems by Zuoning Yin et al. (SOSP’11)

“From 5:00 a.m. to 8:00 a.m. PT, some users received errors when trying to access Gmail, Drive, Talk, Google Sync, the Admin panel, and the Cloud Console, and ... Groups, Sites, and Contacts.”

A misconfiguration at Google in April 17, 2013 (http://goo.gl/2iC9Tm)
Selected Related Work

• Andrew Whitaker et al. (OSDI ’04): Configuration debugging as search
• YaYunn Su et al. (SOSP'07): AutoBash
• Mona Attariyan, Jason Flinn (OSDI'10): Dynamic Information flow analysis
• Ariel Rabkin, Randy Katz (ASE 2011): ConfAnalyzer - Precomputing Configuration Error Diagnoses
• S. Zhang, M. D. Ernst (ICSE 2013): ConfDiagnoser - static analysis, dynamic profiling & statistical analysis
Our **ConfDebugger** Approach

- Automatically debugs crashing errors due to a misconfiguration based on:
  - Source code of application
  - Stack trace of the crashing error
  - Entry statements of each configuration option in application code
- We use **only static analysis** techniques (essentially, *thin slicing*)
- We do **not** need any profiles of the application with correct configuration
Static Analysis: Forward & Backward Slicing

**Forward Slicing**

\[ S^F (A) = \{ B | A \rightarrow^{\ast} B \} \]

- \( S^F (A) \) = essentially, all statements reachable from a seed statement A

**Backward Slicing**

\[ S^B (B) = \{ A | A \rightarrow^{\ast} B \} \]

- \( S^B (B) \) = essentially, all statements which could have reached seed B
Workflow of the ConfDebugger Approach

Configuration options -> Propagation analysis

Forward slices -> Filtering

Backward slices -> Filtering

Intersection -> Diagnosis analysis -> Error report

Prg. source -> Stack trace analysis

Failure stack trace -> Prg. source
Configuration Propagation Analysis

- Extract the **option value entry statements** of all $n$ configuration options from the source code
- For each one: **compute forward slices** for seeds = option value entry statements
Workflow of the ConfDebugger Approach

- Configuration options
- Prg. source

Propagation analysis

Forward slices

Filtering

Intersection

Diagnosis analysis

Error report

Stack trace analysis

Prg. source

Failure stack trace

Backward slices

Filtering
Stack Trace Analysis

- Retrieve **failure sites** from the stack trace, i.e. code locations pointed to by stack trace entries
- For each failure site $t$, compute its backward slice $BS(t)$
- Later: merge all backward slices, i.e. get $M = \bigcup_t BS(t)$
Workflow of the ConfDebugger Approach

- Configuration options
- Prg. source
  - Propagation analysis
  - Stack trace analysis
  - Forward slices
  - Backward slices
  - Filtering
  - Inter-section
  - Diagnosis analysis
  - Error report
Filtering: Reducing Slice Sizes

• Despite of thin slicing, slices are still large => induce false positives

• Intuitively, statements closer to the seed are more relevant
  – **FS**: statements closer to option entry are better
  – **BS**: statements closer to the failure site are better
Filtering: Reducing Slice Sizes /2

• The estimate relevance, we compute the (method) call graph / „reversed“ call graph for each FS and BS
• All statements in a method with breadth-first search distance $k$ from seed get relevance $1/k$
• All statements with relevance below a threshold are removed from a FS / BS

Significantly decreases # false positives in the final result!
Workflow of the ConfDebugger Approach

Configuration options → Propagation analysis → Filtering → Intersection → Option entry statement

Prg. source → Stack trace analysis → Filtering

Prg. source → Failure stack trace

Forward slices → Intersection

Backward slices → Intersection

Option entry statement → Failure site
Intersecting FSs and BSs

Forward slices of entry statements

Union of backward slices of failure sites

FS for Conf_1

Affected statements

FS for Conf_2

Affected statements

... 

FS for Conf_n

Affected statements

Intersect each FS (left) with union of BSs (right)

Intersection ∅

Intersection ∅

A non-empty intersection indicates suspects

A possible exec. chain from entry statement to failure site
Workflow of the ConfDebugger Approach

1. Configuration options
2. Prg. source
3. Prg. source
4. Failure stack trace

- Propagation analysis
- Stack trace analysis
- Forward slices
- Backward slices
- Filtering
- Filtering
- Intersection
- Diagnosis analysis
- Error report
Diagnosis Analysis

• Analysis options depending on number of the suspect sets
  – One (1) suspect ✔
    • Immediate result, report to users directly
  – Two (2) or more suspects
    • Filter suspects
  – No suspect
    • Use approximate intersection
Two or More Suspects

- Heuristic approach to filter suspects: keep suspect only if at least one of $C_1$, $C_2$ applies:
  - $C_1$: FS (option entry statement) contains at least one failure site
  - $C_2$: BS(failure site) contains the entry point of suspected option
No Suspect Set: Use Approx. Intersection

• All intersections empty: use approximate intersection
  • Statements $S_f$ (from FS) and $S_b$ (from BS) are both included in approximate intersection if ..
    – Both are in the same method of the same class, and difference of their line numbers is at most $\epsilon$
  • We increase parameter $\epsilon$ until one or more approximate intersections become non-empty
    – Then we have 1+ suspects
Evaluation

• Target application: JChord
  – A Java dataflow analysis framework
• Eight (8) configuration errors used in the experiment

<table>
<thead>
<tr>
<th>Error</th>
<th>Configuration crashing errors in JChord</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>No main class is specified</td>
</tr>
<tr>
<td>2</td>
<td>No main method in the specified class</td>
</tr>
<tr>
<td>3</td>
<td>Running a nonexistent analysis</td>
</tr>
<tr>
<td>4</td>
<td>Invalid context-sensitive analysis name</td>
</tr>
<tr>
<td>5</td>
<td>Printing nonexistent relations</td>
</tr>
<tr>
<td>6</td>
<td>Disassembling nonexistent classes</td>
</tr>
<tr>
<td>7</td>
<td>Invalid reflection kind</td>
</tr>
<tr>
<td>8</td>
<td>Wrong classpath</td>
</tr>
</tbody>
</table>
# Evaluation Results

<table>
<thead>
<tr>
<th>Error ID</th>
<th>Erroneous Configuration Option</th>
<th>False Positives</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>chord.main.class</td>
<td>1</td>
<td>Y</td>
</tr>
<tr>
<td>2</td>
<td>chord.main.class</td>
<td>1</td>
<td>Y</td>
</tr>
<tr>
<td>3</td>
<td>chord.run.analyses</td>
<td>0</td>
<td>Y</td>
</tr>
<tr>
<td>4</td>
<td>chord.ctxt.kind</td>
<td>0</td>
<td>Y</td>
</tr>
<tr>
<td>5</td>
<td>chord.print.rels</td>
<td>0</td>
<td>Y</td>
</tr>
<tr>
<td>6</td>
<td>chord.print.classes</td>
<td>0</td>
<td>Y</td>
</tr>
<tr>
<td>7</td>
<td>chord.reflect.kind</td>
<td>0</td>
<td>Y</td>
</tr>
<tr>
<td>8</td>
<td>chord.class.path</td>
<td>2</td>
<td>N</td>
</tr>
</tbody>
</table>

False positives rate | Success ratio in %
--- | ---
0.5 | 87.5%

- False positive rate is very low
Comparison With Related Techniques

• ConfAnalyzer [Rabkin ’11]
  – Uses static data flow analysis to trace values of configuration options
  – Focus on configuration crashing errors

• ConfDiagnoser [Sai Zhang ’13]
  – Combine static analysis, dynamic profiling and statistical analysis
  – Identify abnormal behavior of configuration option
  – Need correct profiles databases
Comparison With Related Techniques

False positives rate:
- ConfDebugger: 0.5
- ConfDeb. - No Filtering: 5.5
- ConfAnalyzer: 1.7
- ConfDiagnoser: 5.7

Success rate in %:
- ConfDebugger: 87.5%
- ConfDeb. - No Filtering: 75%
- ConfAnalyzer: 88.9%
- ConfDiagnoser: 100%
## Execution Time for Eight Errors

<table>
<thead>
<tr>
<th>Error ID</th>
<th>Forward Slicing (sec)</th>
<th>Backward Slicing (sec)</th>
<th>Analysis (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>21</td>
<td>35</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>21</td>
<td>35</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>21</td>
<td>18</td>
<td>&lt;1</td>
</tr>
<tr>
<td>4</td>
<td>21</td>
<td>21</td>
<td>&lt;1</td>
</tr>
<tr>
<td>5</td>
<td>21</td>
<td>14</td>
<td>&lt;1</td>
</tr>
<tr>
<td>6</td>
<td>21</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>21</td>
<td>13</td>
<td>&lt;1</td>
</tr>
<tr>
<td>8</td>
<td>21</td>
<td>35</td>
<td>&lt;1</td>
</tr>
</tbody>
</table>

Note: Maximum time is less than 1 minute
Future Work

• Apply ConfDebugger to other application programs and evaluate how the approach generalizes
• Improve scalability of static analysis to cope with RAM problems (now: Wala tool)
• Explore diagnosing inconsistencies in configuration settings
• Long term: extend to cover non-crashing errors (e.g. empty outputs)
Summary

• We proposed a misconfiguration debugging approach
• Implemented a prototype
• Multiple advantages over recent works
  – Very low rate of false positives (ave# = 0.5) for our data
  – Short execution time
  – Uses only static analysis
  – No need for profiles of correct runs
    • Input are only: failure stack trace, source code and configuration options
Thank you.

QUESTIONS ARE WELCOME!