Methods for testing fault tolerant systems

FAULT INJECTION
Overview

- Introduction
- Hardware Fault Injection
- Software Fault Injection
- Protocol Fault Injection

- Conclusions
- Questions
Introduction

- Test robust systems

- Run two copies
  - One is run normally
  - One is run with injected faults

  - Check that fault is detected
  - Compare behavior
HARDWARE FAULT INJECTION
Hardware Fault Injection

- Tests both hardware and software
- Physically cause faults
  - Heavy-ion radiation
  - Pin level injection
  - EMI

- Focused on hardware testing

Hardware Fault Injection

- Could be used to test software
- Software based techniques work
  - Software doesn’t know where fault came from
  - Can be used to test hardware
  - Tends not to trigger hardware fault detection

SOFTWARE FAULT INJECTION
Software Fault Injection

- Modify system state programmatically
- Only tests software
- Repeatable
- Possible to run on a larger scale
- Vary from full virtualization to none at all
Software Fault Injection

- Simulation based fault injection
  - Can trace the flow of an error
  - May need fewer test runs
  - Test runs are slower
  - Virtual environment not like real one

- Software implemented fault injection

- Many things fall in between
Software Fault Injection

- Full Simulation
- V-System VHDL-Simulator
- JVM exception injector
- RUGRAT
- Holodeck
- Xception
- FERRARI
- No simulation
FERRARI

- Use known models of hardware faults
  - Simulate bit errors

- Load target program into memory
- Inserts a trap to the injection process
- At runtime trap is executed
- Modifies program state

<table>
<thead>
<tr>
<th>Model</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AddIF</td>
</tr>
<tr>
<td>2</td>
<td>AddIF2</td>
</tr>
<tr>
<td>3</td>
<td>AddOF</td>
</tr>
<tr>
<td>4</td>
<td>AddOS</td>
</tr>
<tr>
<td>5</td>
<td>DataIF</td>
</tr>
<tr>
<td>6</td>
<td>DataOF</td>
</tr>
<tr>
<td>7</td>
<td>DataOS</td>
</tr>
<tr>
<td>8</td>
<td>CndCR</td>
</tr>
</tbody>
</table>

**Diagram:**

- **Trap:**
  - `0x100`
  - `PC` -> `Inst @ 0x100`
  - `Inst @ 0x104`
  - `Inst @ 0x110`
  - Marked as `Wrong instruction to be executed`.

- **Modify:**
  - `0x100`
  - `PC` -> `0x100`
  - Marked as `Faulty execution`.

- **Trap:**
  - `0x100` again
  - `Inst @ 0x110`
  - `Inst @ 0x104` repeated.

- **Restore:**
  - `Inst @ 0x100`
  - `Inst @ 0x104` repeated.
FERRARI

- Injected faults can be transient
- Can remove the trap after it is used

- Some faults are caught by the hardware
  - Illegal instructions, memory addresses, etc.

- Most faults are detected by software
FERRARI

- Dependent on system configuration
  - Has been ported to other platforms

- Validation given in terms of coverage

- Experiments run a large number of times
  - Only two different targets used
**Xception**

- Avoids modifying the target program at all
- Use processor features to inject faults

- Allows testing of run-time sanity checks
  - Run-time checksums of memory
  - Dynamic generation of function pointers

Xception

- Target program runs a full speed
- Use processor to jump to Xception code
- Generate hardware exception
  - Count a certain number of instructions
  - Set a memory address as inaccessible
  - Etc.
- Exception is handled by Xception code
Xception

- Has inject exception handlers into target
- Target may actually try to handle itself
- Needs to be tailored to the architecture
- Some exceptions are triggered frequently
PROTOCOL FAULT INJECTION
Protocol Fault Injection

- Distributed fault tolerant system
  - Needs to use some sort of protocol
  - Inject faults in the communications
  - Test that the system doesn’t fail

- Only need protocol description
  - Rest can be black box
ORCHESTRA

- Protocol injection platform
- Can be used to test any system
  - Needs code for intercepting messages
  - Needs code for relaying messages
  - Needs code for mutating messages
- Only really implemented for TCP

ORCHESTRA

- Manipulates messages by
  - Dropping
  - Delaying
  - Reordering
  - Duplicating
  - Modifying contents
  - Generating extra messages
ORCHESTRA

- Has a nice GUI for writing scripts
  - State machine model
  - Control how messages are modified
  - Generation of messages

- Can simulate both faulty node or network
Validation of target depends on system
  ◦ Quorum negotiates reaches consensus
  ◦ Database is consistent and correct

Can test only one kind of fault at a time
CONCLUSIONS
Conclusions

- Hardware fault injection is good for testing hardware, but not needed for testing software
- Software fault injection has many forms
  - Different advantages and disadvantages
- Protocol fault injection
  - Generally grouped with software fault injection
  - Relatively straightforward
Conclusions

- Case by case validation
  - Golden system
  - Assume handles are correct if they are triggered
  - Verify that system remains in a valid state
QUESTIONS?