Fault Replication and the Distributed Systems Group at INESC ID Lisboa

João Garcia
Valencia, 2010-06-25
Summary

1. Distributed Systems Group presentation
2. Fault Replication: Replicating System Errors
   - Techniques and examples
3. Case Study I: “Better Bug Reporting with Better Privacy”
4. Case Study II: PRES
INESC ID Lisboa’s Distributed Systems Group

- 7 Ph. D., ~20 Ph.D. Students, ~35 M.Sc. students
- Current research areas:
  - Adaptive systems
  - Ad hoc and wireless computing
  - Data management
  - Grid computing
  - Overlay construction and maintenance
  - Security
- Several European Projects and many tens of FCT National Research Projects.
- **Fault Replication in FastFix**
Fault Replication: **What?**

- **Definition**: reproducing application errors which happened in a client device in a development environment.
Fault Replication: *Who?*

**Luis Rodrigues:**
- Two books on distributed computing
- Appia Java-based *group communication* toolkit
- Gorda *database replication* architecture (European project)

**Paolo Romano:**
*Replication* schemes for streaming apps.:
- APART: A posteriori active replication [NCA08 – best paper]
  - Currently applied to SNORT
Fault Replication: Who?

Paolo Romano:
Software Transactional Memories:
• Dependable distributed STM [PRDC09]
• Innovative transactional replication schemes [ongoing]

João Garcia:
• PerDiS transactional model, European Project [2000-2002]
• Ph.D. thesis on task-awareness
Fault Replication: **Who?**

- Distributed Fault Tolerance
- Transactions, DSM and STM
- Execution Environments and VMs

**Luís Veiga:**
- Virtual Machines and Garbage Collection, e.g. using DGC-Consistent Cuts[...], IET Software 2007
Fault Replication: *Who?*

- Distributed Fault Tolerance
- Transactions, DSM and STM
- Execution Environments and VMs
- Security

**Carlos Ribeiro:**
- *e-voting*, e.g. VeryVote, VoteID 2009
- The *Fenix* FOSS Univ. Manag. System
Applications Crashes are Ugly

The following process has performed an illegal operation.
F:\Technical\Develop\Public\Cpp\Apps\Windows\CrashDoctor\TestApp\obj\chk_wxp_x86\386\testCrash.exe

CrashDoctor can attempt to recover the process from the crashing state. Alternatively, you can choose to terminate the faulty process or debug it using one of the debuggers listed below.

Debuggers Available
- C:\Debuggers\ntsd.exe -p %Id -e %Id
- C:\Debuggers\windbg.exe -p %Id -e %Id
- C:\Program Files\Common Files\Microsoft Shared\...n.exe -p %Id -e %Id
- C:\Program Files\Common Files\Microsoft Shared\...n.exe -p %Id -e %Id
- C:\WINDOWS\system32\drvtmsm32.exe -p %Id -e %Id
-...

Please tell Microsoft about this problem.

We have created an error report that you can send to us. We will treat this report as confidential and anonymous.

To see what data this error report contains, click here.
Fault Replication: Why?

- **Reduce cost of software maintenance.**
- Self-healing SW at the client is heavy/hard! There are better SW maintenance resources on the development side.
  - The growth of devices (smartphones, tablet PCs) with limited resources but complex apps increases these needs.
- Execution environments are not controlled. User descriptions are nearly useless. ("What were you doing?"). There is a need for:
  - **Precise** error capture at the client. What happened?
  - **Automatic** and transparent error reporting. What am I sending?
Basic Idea

- Simplistic approach: crime scene photograph e.g. Windows Error Reporting and Mozilla Talkback:
  - Send home a dump of the state after the crash and some additional user information.
- But the dump is after the crash. How did we get here?
  - Record application execution!
Fault Replication: *How?*

- **Interesting requirements:**
  - **precision:** replay should allow for *error correction*.
  - **transparency:** least possible *changes to the application* code.
  - **lightweight:** the *overhead* introduced shall be *minimal* so to ensure negligible performance degradation during production runs.
  - **security:** the execution *traces will be obfuscated* to preserve users' privacy, without compromising the determinism of the execution replay.
What are the steps in fault replication?

1. Control of the Execution Environment
2. Application Instrumentation
3. Monitoring
4. Trace Analysis and Reporting
5. Replay Environment
FastFix: a Broader Context

- Fast Fix: Monitoring Control for Remote Software Maintenance
Execution Environment: Challenges

- Application crashes:
  - Require recording sources of **non-determinism**.

- Single processor environment:
  - User Input
  - System calls: Files, cache state, buses, devices, network, clocking, random number generation.

- Thread scheduling:
  - How can we deterministically replay scheduling?
  - Solution: record logical time of thread scheduling.
Execution Environment: Challenges

- Multi-core environments:
  - How to correct shared memory (and cache) races.
  - Solution: record and replay memory races.

- Virtual Machines:
  - Simpler but heavier environment.
  - Raises the issue of the interaction between VMs on different cores.
Execution vs. Replay

Execution

Common practice
- 0% overhead

Replay
- >1000 replay attempts*

Existing research proposals
- 10-100 X slowdown
- Impractical!

the 1st replay attempt
Application Instrumentation

• **Log** non-deterministic operations:
  • 1: log every operation => easy replay. VERY heavy!
  • 2: very lightweight (and vague) logging => long diagnosis investigation.
  • Other points of the spectrum: see PRES ahead.

• Or, save state periodically with **checkpointing**.

• Or, SMP-Revirt[Dunl08]:
  • Force page fault protection and interception (and log) protection changes to detect memory races.
  • Problem: Overhead acceptable only for coarse grained sharing.

• **Hardware modifications** like Flight Data Recorder [Xu09], a specific (simulated) architecture for race detection. (There are other simulated proposals)
Monitoring

- Monitoring is the consequence of running a instrumented application in a controlled execution environment.
Trace Analysis and Reporting

- **Trace Privacy:**
  - How much information disclosure is acceptable for the user?
  - Does the user understand the process and the risks?
  - How much liability is acceptable for the developer?
  - How anonymous can the trace be? See Castro 2008 ahead.

- **Secure Data Transmission:**
  - No obvious need to improve on classical solution.
Replay Environment

- The complexity of the replay environment is inverse to the complexity of the execution environment and the anonymity of the trace.

- May include:
  - Storing, reasoning and pattern matching over logged events and traces.
  - Patch generation and distribution.
Case Studies

- 2 Interesting Contributions:
  - PRES [Park SOSP’09]: Configurable monitoring overhead.
“Better Bug Reporting with Better Privacy”

- Castro et al. @ ASPLOS ’08

- Idea:
  - Complete execution path replication.
  - Generation of alternative input.
  - A quantified bound on leaked information.
Instrumentation and Monitoring: Microsoft Nirvana and iDNA
Path Conditions

- Calculate logical clause that expresses input restrictions.
- Example:
  
  ```
  Read(x); // x’s bytes (x_0-x_3) can be ambiguous
  Read(y); // y’s bytes (y_0-y_3) can be ambiguous
  if (x > 0 && even(y))
      //if branch is taken: (x_3>0 ∧ bit(y_0,0)==1)
  ```
Measuring Anonymization

• Anonymize the error report:
  • alternative input generated as a solution to the path conditions using Satisfiability Modulo Theories (SMT) solver.

• Give an estimate on number of leaked input bytes:
  • Calculate entropy measurement of the leaked information based on number of alternative inputs that solve path conditions.
"Better…": Results

Anonymous and highly compressable!!
“Better…”: Results

Percentage of input bits revealed

Report size
Open Issues

- Heavy monitoring.
- Heavy preprocessing: client-side replay with additional calculations.
- Multi-core memory races.
- Does the user understand the entropy bound?
- Are the leaked bytes critical?
• Probabilistic Replay with Execution Sketching of Multiprocessors.

• Idea:
  • Code is instrumented using Intel Pin.
  • Generate partial execution information: a sketch.
  • Multiple runs of smart replayer.
Execution vs. Replay

Execution

Common practice

0% overhead

Error

Existing research proposals

10-100X slowdown

Error

Replay

>1000 replay attempts*

Error

Impractical!

the 1st replay attempt
A sketch is a partial log of application execution.

**Loggable events:**
- lock operations (SYNC)
- system calls (SYS)
- Function calls (FUNC)
- Basic code blocks or groups of basic blocks (BB and BB-n)
`Thread 1
(parent thread)`

`Thread 2
(child thread)`

```c
int gid = 0;
/* create child thread */

worker () {
...
lock (L);
myid = gid;
gid = myid+1;
unlock (L);
...
/* work */
...
}

worker () {
...
lock (L);
myid = gid;
gid = myid+1;
unlock (L);
...
/* work */
...
if (myid == 0) {
    result = data;
}
}

tmp = result;
printf("%d\n", tmp);
```

*Wrong output*
**PRES: Tracing costs**

<table>
<thead>
<tr>
<th>Applications</th>
<th>Pin</th>
<th>SYNC</th>
<th>SYS</th>
<th>FUNC</th>
<th>BB-5</th>
<th>BB-2</th>
<th>BB</th>
<th>RW</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Server Applications (throughput degradation)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Apache</td>
<td>4.23%</td>
<td>7.05%</td>
<td>10.91%</td>
<td>15.55%</td>
<td>18.18%</td>
<td>19.70%</td>
<td>36.32%</td>
<td>53.24%</td>
</tr>
<tr>
<td>MySQL</td>
<td>13.48%</td>
<td>15.48%</td>
<td>15.87%</td>
<td>18.06%</td>
<td>32.59%</td>
<td>37.16%</td>
<td>43.93%</td>
<td>75.35%</td>
</tr>
<tr>
<td>Cherokee</td>
<td>7.04%</td>
<td>7.24%</td>
<td>7.39%</td>
<td>7.87%</td>
<td>7.88%</td>
<td>8.06%</td>
<td>8.99%</td>
<td>28.45%</td>
</tr>
<tr>
<td>OpenLDAP</td>
<td>19.35%</td>
<td>33.03%</td>
<td>33.58%</td>
<td>38.29%</td>
<td>48.01%</td>
<td>52.64%</td>
<td>59.55%</td>
<td>76.20%</td>
</tr>
<tr>
<td><strong>Desktop Applications (overhead)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mozilla</td>
<td>32.55%</td>
<td>59.58%</td>
<td>60.63%</td>
<td>83.89%</td>
<td>598.01%</td>
<td>858.86%</td>
<td>1213.90%</td>
<td>3093.59%</td>
</tr>
<tr>
<td>PBZip2</td>
<td>17.46%</td>
<td>18.00%</td>
<td>18.07%</td>
<td>18.43%</td>
<td>595.43%</td>
<td>1066.64%</td>
<td>1977.96%</td>
<td>27009.79%</td>
</tr>
<tr>
<td>Transmission</td>
<td>5.9%</td>
<td>14.50%</td>
<td>21.33%</td>
<td>32.75%</td>
<td>30.34%</td>
<td>33.98%</td>
<td>41.92%</td>
<td>71.80%</td>
</tr>
<tr>
<td><strong>Scientific Applications (overhead)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Barnes</td>
<td>2.53%</td>
<td>6.50%</td>
<td>6.83%</td>
<td>427.76%</td>
<td>424.27%</td>
<td>1122.343%</td>
<td>2351.02%</td>
<td>28702.26%</td>
</tr>
<tr>
<td>Radiosity</td>
<td>16.01%</td>
<td>16.11%</td>
<td>16.19%</td>
<td>779.05%</td>
<td>480.35%</td>
<td>1181.73%</td>
<td>2425.53%</td>
<td>27209.67%</td>
</tr>
<tr>
<td>FMM</td>
<td>8.87%</td>
<td>17.50%</td>
<td>17.50%</td>
<td>47.75%</td>
<td>659.41%</td>
<td>1366.94%</td>
<td>2697.55%</td>
<td>31736.43%</td>
</tr>
<tr>
<td>LU</td>
<td>9.23%</td>
<td>11.01%</td>
<td>12.06%</td>
<td>86.75%</td>
<td>668.80%</td>
<td>1315.89%</td>
<td>2633.13%</td>
<td>31018.02%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Application</th>
<th>SYNC</th>
<th>SYS</th>
<th>FUNC</th>
<th>BB-5</th>
<th>BB-2</th>
<th>BB</th>
<th>RW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache</td>
<td>1.03</td>
<td>1.71</td>
<td>190.10</td>
<td>128.52</td>
<td>317.17</td>
<td>768.92</td>
<td>2076.38</td>
</tr>
<tr>
<td></td>
<td>Log size(KB/req)</td>
<td>2.00</td>
<td>3.34</td>
<td>371.30</td>
<td>251.01</td>
<td>619.47</td>
<td>1501.81</td>
</tr>
<tr>
<td>MySQL</td>
<td>0.65</td>
<td>0.65</td>
<td>27.00</td>
<td>52.21</td>
<td>129.54</td>
<td>258.83</td>
<td>861.50</td>
</tr>
<tr>
<td></td>
<td>Log size(KB/req)</td>
<td>1.27</td>
<td>1.27</td>
<td>52.73</td>
<td>101.97</td>
<td>253.00</td>
<td>505.65</td>
</tr>
<tr>
<td>Cherokee</td>
<td>0.12</td>
<td>0.12</td>
<td>0.68</td>
<td>0.42</td>
<td>0.77</td>
<td>1.31</td>
<td>2.62</td>
</tr>
<tr>
<td></td>
<td>Log size(KB/req)</td>
<td>0.95</td>
<td>0.95</td>
<td>5.42</td>
<td>3.32</td>
<td>6.17</td>
<td>10.44</td>
</tr>
<tr>
<td>OpenLDAP</td>
<td>64.79</td>
<td>64.79</td>
<td>1784.64</td>
<td>2157.55</td>
<td>3764.65</td>
<td>6880.73</td>
<td>14838.75</td>
</tr>
<tr>
<td></td>
<td>Log size(KB/req)</td>
<td>126.50</td>
<td>126.50</td>
<td>3485.63</td>
<td>4214.00</td>
<td>7352.86</td>
<td>13439.00</td>
</tr>
</tbody>
</table>
Partial information replay involves some deduction:

- Replay is **retried** until application reaches point of crash.
- Replay is **interrupted** if recording points are not passed/reached.
PRES: Replay Effort

<table>
<thead>
<tr>
<th>Application</th>
<th>Bug Id.</th>
<th>Bug Type</th>
<th>Base</th>
<th>SYNC</th>
<th>SYS</th>
<th>FUNC</th>
<th>BB-5</th>
<th>BB-2</th>
<th>BB</th>
<th>RW</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Server</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Applications</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Apache#1</td>
<td>NO</td>
<td>atomicity</td>
<td>96th</td>
<td>28th</td>
<td>7th</td>
<td>8th</td>
<td>7th</td>
<td>7th</td>
<td>1st</td>
<td>1st</td>
</tr>
<tr>
<td>Apache#2</td>
<td>NO</td>
<td>deadlock</td>
<td>1st</td>
<td>1st</td>
<td>1st</td>
<td>1st</td>
<td>1st</td>
<td>1st</td>
<td>1st</td>
<td>1st</td>
</tr>
<tr>
<td>MySQL#1</td>
<td>NO</td>
<td>atomicity</td>
<td>3rd</td>
<td>3rd</td>
<td>4th</td>
<td>3rd</td>
<td>2nd</td>
<td>1st</td>
<td>1st</td>
<td>1st</td>
</tr>
<tr>
<td>MySQL#2</td>
<td>NO</td>
<td>deadlock</td>
<td>1st</td>
<td>1st</td>
<td>1st</td>
<td>1st</td>
<td>1st</td>
<td>1st</td>
<td>1st</td>
<td>1st</td>
</tr>
<tr>
<td>Cherokee</td>
<td>NO</td>
<td>atomicity</td>
<td>33th</td>
<td>33th</td>
<td>33th</td>
<td>33th</td>
<td>33th</td>
<td>33th</td>
<td>33th</td>
<td>33th</td>
</tr>
<tr>
<td>OpenLDAP</td>
<td>NO</td>
<td>deadlock</td>
<td>1st</td>
<td>1st</td>
<td>1st</td>
<td>1st</td>
<td>1st</td>
<td>1st</td>
<td>1st</td>
<td>1st</td>
</tr>
<tr>
<td><strong>Desktop</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Applications</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mozilla#1</td>
<td>NO</td>
<td>atomicity</td>
<td>1st</td>
<td>1st</td>
<td>1st</td>
<td>1st</td>
<td>1st</td>
<td>1st</td>
<td>1st</td>
<td>1st</td>
</tr>
<tr>
<td>Mozilla#2</td>
<td>NO</td>
<td>multi-var</td>
<td>3rd</td>
<td>3rd</td>
<td>3rd</td>
<td>4th</td>
<td>4th</td>
<td>3rd</td>
<td>3rd</td>
<td>3rd</td>
</tr>
<tr>
<td>PBZip2</td>
<td>NO</td>
<td>atomicity</td>
<td>3rd</td>
<td>3rd</td>
<td>2nd</td>
<td>4th</td>
<td>3rd</td>
<td>3rd</td>
<td>3rd</td>
<td>3rd</td>
</tr>
<tr>
<td>Transmission</td>
<td>NO</td>
<td>order</td>
<td>2nd</td>
<td>2nd</td>
<td>2nd</td>
<td>2nd</td>
<td>2nd</td>
<td>2nd</td>
<td>2nd</td>
<td>2nd</td>
</tr>
<tr>
<td><strong>Scientific</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Applications</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Barnes</td>
<td>NO</td>
<td>order</td>
<td>10th</td>
<td>10th</td>
<td>1st</td>
<td>74th</td>
<td>19th</td>
<td>1st</td>
<td>1st</td>
<td>1st</td>
</tr>
<tr>
<td>Radiocity</td>
<td>NO</td>
<td>order</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>152th</td>
<td>5th</td>
<td>1st</td>
<td>1st</td>
<td>1st</td>
</tr>
<tr>
<td>LU</td>
<td>NO</td>
<td>order</td>
<td>2nd</td>
<td>2nd</td>
<td>2nd</td>
<td>2nd</td>
<td>2nd</td>
<td>1st</td>
<td>1st</td>
<td>1st</td>
</tr>
</tbody>
</table>

**Light tracing** => slow diagnosis

**Heavy tracing** => quick diagnosis
What Next?  
Open Questions in Fault Replication

- **System:**
  - How low can we the **monitoring overhead** become?
  - How to explain **privacy issues** clearly to the user?
  - Can we record/replay **multi-core VM apps** correctly?
  - Can we **combine partial traces** from multiple users?

- **General:**
  - Can we monitor user behaviour in **small devices** with acceptable overhead?
  - Can we **abstract execution traces** to descriptions that developers understand?
  - Can the **sketches** be used to inform **patch generation**?
Conclusions

- **Automatic software maintenance** is still a very tough problem.
- Moving the problem to the development side is a good strategy.
- **Fault replication** techniques are an essential tool for self-healing.
- The hurdles of **monitoring overheads** and **user privacy** questions have not been removed yet.
- **FastFix** is going to tackle these and other issues: check [www.fastfixproject.eu](http://www.fastfixproject.eu) for news/events!
Thank You!
Questions?

jog@gsd.inesc-id.pt
www.gsd.inesc-id.pt