Minimizing Faulty Executions of Distributed Systems

Colin Scott, Aurojit Panda, Vjekoslav Brajkovic, George Necula, Arvind Krishnamurthy, Scott Shenker
Software Developer
49% of developers’ time spent on debugging!\(^1\)

\(^1\) LaToza, Venolia, DeLine, ICSE’ 06
49% of developers’ time spent on debugging!\(^1\)

Understanding How Bug Is Triggered

Fixing Problematic Code

\(^1\) LaToza, Venolia, DeLine, ICSE’ 06
Our Goal

Allow Developers To Focus on Fixing the Underlying Bug
Problem Statement

Identify a **minimal causal sequence** of events that triggers the bug.
Outline

<table>
<thead>
<tr>
<th>Introduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Background</td>
</tr>
<tr>
<td>Randomized Testing with DEMi</td>
</tr>
<tr>
<td>Minimization</td>
</tr>
<tr>
<td>Evaluation</td>
</tr>
<tr>
<td>Conclusion</td>
</tr>
</tbody>
</table>

**Randomized Testing with DEMi**

Diagram of Node 1 to Node N connected through arrows, indicating a network with a Test Harness and Test Coordinator for Software Under Test.
Outline

Introduction

Background

Randomized Testing with DEMi

Minimization

Evaluation

Conclusion
Outline

Introduction

Background

Randomized Testing with DEMi

Minimization

Evaluation

Conclusion
Outline

Introduction

Background

Randomized Testing with DEMi

Minimization

Evaluation

Conclusion
Randomized Testing with DEMi

App
RPC lib
OS
Randomized Testing with DEMi

Diagram:
- App
- RPC lib
  - AspectJ
- OS
Randomized Testing with DEMi

![Diagram showing the app, RPC library, and OS layers with messages and aspects.]

- App
  - msg dst: b
  - RPC lib
  - AspectJ
  - OS

- App
  - RPC lib
  - AspectJ
  - OS

- App
  - RPC lib
  - AspectJ
  - OS

- Additional panels for further layers or components.
Randomized Testing with DEMi

App

RPC lib
AspectJ

msg
dst. b

OS

App

RPC lib
AspectJ

OS

App

RPC lib
AspectJ

OS
Randomized Testing with DEMi

Diagram showing three tiers: App, RPC lib, AspectJ, and OS, with an arrow labeled "msg dst: b".
Randomized Testing with DEMi

Diagram showing three instances of an application (App), with each instance having a Remote Procedure Call (RPC) library (RPC lib), an AspectJ library, and an Operating System (OS). There is a message (msg) with destination (dst: b) shown at the bottom.
Randomized Testing with DEMi
Randomized Testing with DEMi

msg

dst: b

message
delivery

App

RPC lib

AspectJ

OS

App

RPC lib

AspectJ

OS

App

RPC lib

AspectJ

OS
Randomized Testing with DEMi

Diagram showing the interaction between applications, RPC libraries, and operating systems.
Randomized Testing with DEMi

```
App
  | RPC lib
  |  AspectJ
  |  OS
App
  | RPC lib
  |  AspectJ
  |  OS
App
  | RPC lib
  |  AspectJ
  |  OS
```

message delivery

```
msg
dst: a
timer
dst: b
```
Randomized Testing with DEMi

External events (events outside system's control):
- Crash-recovery
- Process creation
- External message

- App
  - RPC lib
  - AspectJ
  - OS

msg
dst: a
timer
dst: b
Randomized Testing with DEMi

External events (events outside system’s control):
- Crash-recovery
- Process creation
- External message
Randomized Testing with DEMi

External events (events outside system's control):
- Crash-recovery
- Process creation
- External message

```plaintext
App
  RPC lib
    AspectJ
  OS
```

```
msg dst: a
timer dst: b
```

message delivery

crash recovery
Randomized Testing with DEMi

- App
  - RPC lib
  - AspectJ
  - OS

- App
  - RPC lib
  - AspectJ
  - OS

- App
  - RPC lib
  - AspectJ
  - OS

message delivery → crash recovery

dst: a  dst: b

timer

msg
An invariant is a predicate $P$ over the state of all processes.
An invariant is a predicate $P$ over the state of all processes. A faulty execution is one that ends in an invariant violation.
Formal Problem Statement

Given: schedule $\tau$ that results in violation of $P$

Find: locally minimal reproducing sequence $\tau'$:

- $\tau'$ violates $P$, $|\tau'| \leq |\tau|
- $\tau'$ contains a subsequence of the external events of $\tau$
- if we remove any external event $e$ from $\tau'$,
- $\not\exists \tau''$ containing same external events - $e$, s.t. $\tau''$ violates $P$
After finding $\tau'$, minimize internal events:

- remove extraneous message deliveries from $\tau'$
Sending to raft-member-4: RequestVote(Ter
Sending to raft-member-2: BeginElection
 Received message from raft-member-3: Requ
Received timer: Timer(election-timer,Elec
Received message from raft-member-2: Begi
Sending to raft-member-3: VoteCandidate(T
Sending to raft-member-3: BeginElection
Sending to raft-member-1: RequestVote(Ter
 Received message from raft-member-4: Vote
Received message from raft-member-2: Requ
Received message from raft-member-3: Begi
Sending to raft-member-2: VoteCandidate(T
Sending to raft-member-4: RequestVote(Ter
Received message from raft-member-1: Vote
Received timer: Timer(election-timer,Elec
Received message from raft-member-3: Requ
Received message from raft-member-2: Begi
Received timer: Timer(election-timer,Elec
Sending to raft-member-3: VoteCandidate(T
Sending to raft-member-3: RequestVote(Ter
Received message from raft-member-4: Vote
Received message from raft-member-2: Requ
Sending to raft-member-2: VoteCandidate(T
Received message from raft-member-1: VoteCandidate(term(2)))
Minimization

Given

τ : e_1 → i_1 → i_2 → e_2 → i_4 → ... → i_m → e_n
Minimization

Given

\[ \tau \]

Straightforward approach:
- Enumerate all schedules \(|\tau'| \leq |\tau|\),
- Pick shortest sequence that reproduces \(x\)

\[ \tau : e_1 \rightarrow i_1 \rightarrow i_2 \rightarrow e_2 \rightarrow i_4 \rightarrow \ldots \rightarrow i_m \rightarrow e_n \]
Minimization

Given

Straightforward approach:

- Enumerate all schedules $|\tau'| \leq |\tau|$
- Pick shortest sequence that reproduces $\tau$
O(n!)
Observation #1: many schedules are commutative

Adopt DPOR:
Dynamic Partial Order Reduction

$O\left(\frac{n!}{k^k}\right)$
Approach: prioritize schedule space exploration
Approach: prioritize schedule space exploration

Assume: fixed time budget
Objective: quickly find small failing schedules
Observation #2: selectively mask original events
Observation #2: selectively mask original events

\[ \tau : e_1 \rightarrow i_1 \rightarrow i_2 \rightarrow e_2 \rightarrow i_4 \rightarrow \ldots \rightarrow i_m \rightarrow e_n \]

\[ \text{ext: } e_1, e_2, e_3, e_4, e_5, e_n \]
Observation #2: selectively mask original events

\[ \tau : e_1 \rightarrow i_1 \rightarrow i_2 \rightarrow e_2 \rightarrow i_4 \rightarrow \ldots \rightarrow i_m \rightarrow e_n X \]

\[ \text{ext: } e_1 \rightarrow e_2 \rightarrow e_3 \rightarrow e_4 \rightarrow e_5 \rightarrow e_n \]
Observation #2: selectively mask original events

τ : e₁ → i₁ → i₂ → e₂ → i₄ → ... → iₘ → eₙ

ext: e₁ e₂ e₃ e₄ e₅ eₙ
Observation #2: selectively mask original events

$$\tau : e_1 \rightarrow i_1 \rightarrow i_2 \rightarrow e_2 \rightarrow i_4 \rightarrow \ldots \rightarrow i_m \rightarrow e_n \quad \times$$

ext: $e_1$, $e_2$, $e_3$, $e_4$, $e_5$, $e_n$

(Apply Delta Debugging\(^1\))

\(^1\)A Zeller, R. Hildebrandt, “Simplifying and Isolating Failure-Inducing Input”, IEEE ‘02
Observation #2: selectively mask original events

$\tau : e_1 \rightarrow i_1 \rightarrow i_2 \rightarrow e_2 \rightarrow i_4 \rightarrow \ldots \rightarrow i_m \rightarrow e_n \times$

$\text{ext: } e_1 \rightarrow e_2 \rightarrow e_3 \rightarrow e_4 \rightarrow e_5 \rightarrow e_n$

$\text{sub}_1: e_4 \rightarrow \ldots \rightarrow e_5 \rightarrow e_n$

(Apply Delta Debugging\(^1\))

\(^1\)A Zeller, R. Hildebrandt, “Simplifying and Isolating Failure-Inducing Input”, IEEE ‘02
Observation #2: selectively mask original events

\[ \tau : \quad e_1 \xrightarrow{\text{i}_1} i_2 \xrightarrow{\text{i}_2} e_2 \xrightarrow{\text{i}_4} \ldots \xrightarrow{\text{i}_m} e_n \]

ext: \quad e_1 \rightarrow e_2 \rightarrow e_3 \rightarrow e_4 \rightarrow e_5 \rightarrow e_n

sub_1: \quad e_4 \rightarrow \ldots \rightarrow e_5 \rightarrow e_n

\text{foreach i in } \tau : \quad \text{if i is pending: deliver i}
\text{# ignore unexpected events}
Observation #2: selectively mask original events

∀ i ∈ τ:
  if i is pending:
    deliver i
# ignore unexpected
Observation #2: selectively mask original events

\[ \tau : e_1 \rightarrow i_1 \rightarrow i_2 \rightarrow e_2 \rightarrow i_4 \rightarrow \ldots \rightarrow i_m \rightarrow e_n \]

ext: \[ e_1 \rightarrow e_2 \rightarrow e_3 \rightarrow e_4 \rightarrow e_5 \rightarrow e_n \]

sub\(_1\): \[ i_1 \rightarrow e_4 \rightarrow \ldots \rightarrow e_5 \rightarrow e_n \]

foreach i in \( \tau \):
  if i is pending:
    deliver i
# ignore unexpected
Observation #2: selectively mask original events

\[ \tau : e_1 \rightarrow i_1 \rightarrow i_2 \rightarrow e_2 \rightarrow i_4 \rightarrow \ldots \rightarrow i_m \rightarrow e_n \]

\[ \text{ext: } e_1 \rightarrow e_2 \rightarrow e_3 \rightarrow e_4 \rightarrow e_5 \rightarrow e_n \]

\[ \text{sub}_1: \]

\[ \text{foreach } i \text{ in } \tau : \]
\[ \text{if } i \text{ is pending:} \]
\[ \text{deliver } i \]
\[ \# \text{ ignore unexpected} \]
Observation #2: selectively mask original events

foreach i in $\tau$:
  if i is pending:
    deliver i
# ignore unexpected

$\tau$: $e_1 \rightarrow i_1 \rightarrow i_2 \rightarrow e_2 \rightarrow i_4 \rightarrow \ldots \rightarrow i_m \rightarrow e_n$

ext: $e_1 \rightarrow e_2 \rightarrow e_3 \rightarrow e_4 \rightarrow e_5 \rightarrow e_n$

sub$_1$: $i_1 \rightarrow \underline{i_4} \rightarrow e_4 \rightarrow \ldots \rightarrow e_5 \rightarrow i_m \rightarrow e_n$
Observation #2: selectively mask original events

∀ \tau \in \tau:
if i is pending:
   deliver i
# ignore unexpected

foreach i in \tau:
Observation #2: selectively mask original events

\[ \text{foreach i in } \tau: \]
if i is pending:
  deliver i
# ignore unexpected

for \( \tau : e_1 \rightarrow i_1 \rightarrow i_2 \rightarrow e_2 \rightarrow i_4 \rightarrow \ldots \rightarrow i_m \rightarrow e_n \) with \( \text{ext}: e_1 \rightarrow e_2 \rightarrow e_3 \rightarrow e_4 \rightarrow e_5 \rightarrow e_n \) and \( \text{sub}_1 : i_1 \rightarrow i_4 \rightarrow \ldots \rightarrow e_5 \rightarrow i_m \rightarrow e_n \)
Observation #2: selectively mask original events
Observation #2: selectively mask original events

\[ \tau: e_1 \rightarrow i_1 \rightarrow i_2 \rightarrow e_2 \rightarrow i_4 \rightarrow \ldots \rightarrow i_m \rightarrow e_n \]

\[ \text{ext:} \quad e_4 \quad e_5 \quad e_n \]

\[ \text{sub}_1: \quad i_1 \quad i_4 \quad e_4 \quad \ldots \quad e_5 \quad i_m \quad e_n \]
Observation #2: selectively mask original events

$\tau: e_1 \rightarrow i_1 \rightarrow i_2 \rightarrow e_2 \rightarrow i_4 \rightarrow \ldots \rightarrow i_m \rightarrow e_n \times$

ext:

sub$_1$: $i_1$

sub$_2$: $e_5 \rightarrow e_4 \rightarrow \ldots \rightarrow e_5 \rightarrow i_m \rightarrow e_n \times$
Observation #2: selectively mask original events

\[ \tau : e_1 \rightarrow i_1 \rightarrow i_2 \rightarrow e_2 \rightarrow i_4 \rightarrow \ldots \rightarrow i_m \rightarrow e_n \]

\[ \text{ext:} \quad e_4 \quad e_5 \quad e_n \]

\[ \text{sub}_1 : \quad i_1 \quad i_4 \quad e_4 \quad \ldots \quad e_5 \quad i_m \quad e_n \]

\[ \text{sub}_2 : \quad i_1 \quad i_4 \quad \ldots \quad e_5 \quad i_m \quad e_n \]
Observation #2: selectively mask original events

\[ \begin{align*}
T: & \quad \textcolor{green}{e_1} \rightarrow \textcolor{blue}{i_1} \rightarrow \textcolor{green}{i_2} \rightarrow \textcolor{green}{e_2} \rightarrow \textcolor{blue}{i_4} \rightarrow \ldots \rightarrow \textcolor{blue}{i_m} \rightarrow \textcolor{green}{e_n} \quad \text{✗} \\
\text{ext:} & \quad \textcolor{green}{e_4} \quad \textcolor{green}{e_5} \quad \textcolor{green}{e_n} \\
\text{sub}_1: & \quad \textcolor{blue}{i_1} \quad \textcolor{blue}{i_4} \quad \textcolor{green}{e_4} \quad \ldots \quad \textcolor{green}{e_5} \quad \textcolor{blue}{i_m} \quad \textcolor{green}{e_n} \quad \text{✗} \\
\text{sub}_2: & \quad \textcolor{blue}{i_1} \quad \textcolor{blue}{i_4} \quad \ldots \quad \textcolor{green}{e_5} \quad \textcolor{blue}{i_m} \quad \textcolor{green}{e_n} \quad \checkmark
\end{align*} \]
Observation #2: selectively mask original events

$\tau: \quad e_1 \rightarrow i_1 \rightarrow i_2 \rightarrow e_2 \rightarrow i_4 \rightarrow \ldots \rightarrow i_m \rightarrow e_n \times$

$\text{ext:} \quad i_1 \rightarrow e_4 \rightarrow e_5 \rightarrow e_n$

$\text{sub}_1: \quad i_1 \rightarrow i_4 \rightarrow e_4 \rightarrow \ldots \rightarrow e_5 \rightarrow i_m \rightarrow e_n \times$

$\text{sub}_2: \quad i_1 \rightarrow i_4 \rightarrow \ldots \rightarrow e_5 \rightarrow i_m \rightarrow e_n \checkmark$

Explore backtrack points until (i) $\times$ or (ii) time budget for sub$_2$ expired

$\ldots$
Goal: find minimal schedule that produces violation

Observation #1: many schedules are commutative

Approach: prioritize schedule space exploration

Observation #2: selectively mask original events

Observation #3: some contents should be masked

Observation #4: shrink external message contents

Minimize internal events after externals minimized
Outline

Introduction

Background

Randomized Testing with DEMi

Minimization

Evaluation

Conclusion
Target Systems

[Logos of Raft and Spark]
How well does DEMi work?

<table>
<thead>
<tr>
<th>Case Study</th>
<th>Initial Execution</th>
<th>After Minimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>raft-45</td>
<td>2160</td>
<td>23</td>
</tr>
<tr>
<td>raft-46</td>
<td>1250</td>
<td>35</td>
</tr>
<tr>
<td>raft-56</td>
<td>2380</td>
<td>82</td>
</tr>
<tr>
<td>raft-58a</td>
<td>2850</td>
<td>226</td>
</tr>
<tr>
<td>raft-58b</td>
<td>1500</td>
<td>40</td>
</tr>
<tr>
<td>raft-42</td>
<td>1710</td>
<td>180</td>
</tr>
<tr>
<td>raft-66</td>
<td>1000</td>
<td>40</td>
</tr>
<tr>
<td>spark-2294</td>
<td>40</td>
<td>14</td>
</tr>
<tr>
<td>spark-3150</td>
<td>600</td>
<td>300</td>
</tr>
<tr>
<td>spark-9256</td>
<td>300</td>
<td>11</td>
</tr>
</tbody>
</table>

Total Events: 0, 300, 600, 900, 1200, 1500, 1800, 2100, 2400, 2700, 3000
How well does DEMi work?

80% - 97% Reduction!
How well does DEMi work?

Total Events

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>raft-45</td>
<td>23</td>
<td>35</td>
<td>23</td>
<td>21</td>
<td>51</td>
<td>40</td>
<td>28</td>
<td>39</td>
<td>29</td>
<td>40</td>
</tr>
<tr>
<td>raft-46</td>
<td>22</td>
<td>23</td>
<td>21</td>
<td>21</td>
<td>28</td>
<td>28</td>
<td>21</td>
<td>23</td>
<td>14</td>
<td>11</td>
</tr>
<tr>
<td>raft-56</td>
<td>22</td>
<td>23</td>
<td>23</td>
<td>22</td>
<td>29</td>
<td>40</td>
<td>25</td>
<td>14</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>raft-58a</td>
<td>22</td>
<td>23</td>
<td>23</td>
<td>22</td>
<td>29</td>
<td>40</td>
<td>25</td>
<td>14</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>raft-58b</td>
<td>22</td>
<td>23</td>
<td>23</td>
<td>22</td>
<td>29</td>
<td>40</td>
<td>25</td>
<td>14</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>raft-42</td>
<td>22</td>
<td>23</td>
<td>23</td>
<td>22</td>
<td>29</td>
<td>40</td>
<td>25</td>
<td>14</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>raft-66</td>
<td>22</td>
<td>23</td>
<td>23</td>
<td>22</td>
<td>29</td>
<td>40</td>
<td>25</td>
<td>14</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>spark-2294</td>
<td>22</td>
<td>23</td>
<td>23</td>
<td>22</td>
<td>29</td>
<td>40</td>
<td>25</td>
<td>14</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>spark-3150</td>
<td>22</td>
<td>23</td>
<td>23</td>
<td>22</td>
<td>29</td>
<td>40</td>
<td>25</td>
<td>14</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>spark-9256</td>
<td>22</td>
<td>23</td>
<td>23</td>
<td>22</td>
<td>29</td>
<td>40</td>
<td>25</td>
<td>14</td>
<td>11</td>
<td>11</td>
</tr>
</tbody>
</table>
How well does DEMi work?

Total Events

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>23</td>
<td>35</td>
<td>82</td>
<td>226</td>
<td>51</td>
<td>40</td>
<td>29</td>
<td>40</td>
<td>25</td>
<td>14</td>
</tr>
<tr>
<td>Factor of</td>
<td>1x - 5x</td>
<td>from hand-crafted</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
How quickly does DEMi work?

<table>
<thead>
<tr>
<th>Case Study</th>
<th>Runtime in Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>raft-45</td>
<td>170</td>
</tr>
<tr>
<td>raft-46</td>
<td>282</td>
</tr>
<tr>
<td>raft-56</td>
<td>2132 (≈35 minutes)</td>
</tr>
<tr>
<td>raft-58a</td>
<td>43482 (~12 hours)</td>
</tr>
<tr>
<td>raft-58b</td>
<td>66</td>
</tr>
<tr>
<td>raft-42</td>
<td>10676 (~3 hours)</td>
</tr>
<tr>
<td>raft-66</td>
<td>348</td>
</tr>
<tr>
<td>spark-2294</td>
<td>427</td>
</tr>
<tr>
<td>spark-3150</td>
<td>245</td>
</tr>
<tr>
<td>spark-9256</td>
<td>210</td>
</tr>
</tbody>
</table>

Overall Minimization:
- (~12 hours)
- (~3 hours)
How quickly does DEMi work?

Runtime in Seconds

<table>
<thead>
<tr>
<th>Case Study</th>
<th>Overall Minimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>raft-45</td>
<td>170</td>
</tr>
<tr>
<td>raft-46</td>
<td>282</td>
</tr>
<tr>
<td>raft-56</td>
<td>2132 (~35 minutes)</td>
</tr>
<tr>
<td>raft-58a</td>
<td>43482 (~12 hours)</td>
</tr>
<tr>
<td>raft-58b</td>
<td>69</td>
</tr>
<tr>
<td>raft-42</td>
<td>10676 (~3 hours)</td>
</tr>
<tr>
<td>raft-66</td>
<td>348</td>
</tr>
<tr>
<td>spark-2294</td>
<td>427</td>
</tr>
<tr>
<td>spark-3150</td>
<td>245</td>
</tr>
<tr>
<td>spark-9256</td>
<td>210</td>
</tr>
</tbody>
</table>

<10 minutes except 3 cases
See the paper for...

- How we handle non-determinism
- Handling multithreaded processes
- Supporting other RPC libraries
- Sketch for minimizing production traces
- More in-depth evaluation
- Related work
- ...

Conclusion

Optimistic that these techniques can be successfully applied more broadly

Open source tool: github.com/NetSys/demi

Read our paper!
eecs.berkeley.edu/~rcs/research/nsdi16.pdf

Contact me! cs@cs.berkeley.edu

Thanks for your time!
Attributions

Inspiration for slide design: Jay Lorch’s IronFleet slides

Graphic Icons: thenounproject.org
logfile: mantisshrimpdesign
magnifying glass: Ricardo Moreira
disk: Anton Outkine
hook: Seb Cornelius
bug report: Lemon Liu
devil: Mourad Mokrane
Putin: Remi Mercier
Production Traces

Model: feed partially ordered log into single machine DEMi

Require:
- Partial ordering of all message deliveries
- All crash-recoveries logged to disk
## Instrumentation Complexity

<table>
<thead>
<tr>
<th></th>
<th>akka-raft</th>
<th>Spark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message Fingerprint</td>
<td>59</td>
<td>56</td>
</tr>
<tr>
<td>Non-Determinism</td>
<td>2</td>
<td>~250</td>
</tr>
</tbody>
</table>

**Table 4:** Instrumentation complexity (lines of code) needed to define message fingerprints, and to mitigate non-determinism.
Related Work

- **Thread Schedule Minimization**
  - Isolating Failure-Inducing Thread Schedules. SIGSOFT ’02.
  - A Trace Simplification Technique for Effective Debugging of Concurrent Programs. FSE ’10.

- **Program Flow Analysis**
  - Enabling Tracing of Long-Running Multithreaded Programs via Dynamic Execution Reduction. ISSTA ’07.
  - Toward Generating Reducible Replay Logs. PLDI ’11.

- **Best-Effort Replay of Field Failures**
  - A Technique for Enabling and Supporting Debugging of Field Failures. ICSE ’07.
  - Triage: Diagnosing Production Run Failures at the User’s Site. SOSP ’07.
Input: \( T_x \) s.t. \( T_x \) is a trace and \( \text{test}(T_x) = \bullet \). Output: \( T'_x = \text{ddmin}(T_x) \) s.t. \( T'_x \subseteq T_x \), \( \text{test}(T'_x) = \bullet \), and \( T'_x \) is minimal.

\[
\text{ddmin}(T_x) = \text{ddmin}_2(T_x, \emptyset)
\]

where

\[
\text{ddmin}_2(T'_x, R) = \begin{cases} 
T'_x & \text{if } |T'_x| = 1 \text{ ("base case")}
\text{ddmin}_2(T_1, R) & \text{else if } \text{test}(T_1 \cup R) = \bullet \text{ ("in } T_1\text{")}
\text{ddmin}_2(T_2, R) & \text{else if } \text{test}(T_2 \cup R) = \bullet \text{ ("in } T_2\text{")}
\text{ddmin}_2(T_1, T_2 \cup R) \cup \text{ddmin}_2(T_2, T_1 \cup R) & \text{otherwise ("interference")}
\end{cases}
\]

where \( \text{test}(T) \) denotes the state of the system after executing the trace \( T \), \( \bullet \) denotes an invariant violation, \( T_1 \subset T'_x, T_2 \subset T'_x, T_1 \cup T_2 = T'_x, T_1 \cap T_2 = \emptyset \), and \( |T_1| \approx |T_2| \approx |T'_x|/2 \) hold.
- **Monotonic:**
  \[ P \oplus C = \chi \Rightarrow P \oplus (C \cup C') \neq \checkmark \]

- **Unambiguous:**
  \[ P \oplus C = \chi \land P \oplus C' = \chi \Rightarrow P \oplus (C \cap C') \neq \checkmark \]

- **Consistent**
  \[ P \oplus C \neq ? \]
Local vs. global minima

Definition 8 (Global minimum). A set $c \subseteq c_\chi$ is called the global minimum of $c_\chi$ if: $\forall c' \subseteq c_\chi \cdot (|c'| < |c| \Rightarrow \text{test}(c') \neq \chi)$ holds.

Definition 10 ($n$-minimal test case). A test case $c \subseteq c_\chi$ is $n$-minimal if: $\forall c' \subset c \cdot |c| - |c'| \leq n \Rightarrow (\text{test}(c') \neq \chi)$ holds. Consequently, $c$ is 1-minimal if $\forall \delta_i \in c \cdot \text{test}(c - \{\delta_i\}) \neq \chi$ holds.
Minimization Pace

**Figure 2:** Minimization pace for raft-58b. Significant progress is made early on, then progress becomes rare.
Dealing With Threads

If you’re lucky: threads are largely independent (Spark)
If you’re unlucky: key insight:
A write to shared memory is equivalent to a message delivery

Approach:
• interpose on virtual memory, thread scheduler
• pause a thread whenever it writes to shared memory / disk

Cf. “Enabling Tracing Of Long-Running Multithreaded Programs Via Dynamic Execution Reduction”, ISSTA ‘07
Dealing With Non-Determinism

Interpose on:
- Timers
- Random number generators
- Unordered hash values
- ID allocation

Stop-gap: replay each schedule multiple times
## Complete Results

<table>
<thead>
<tr>
<th>Bug Name</th>
<th>Bug Type</th>
<th>Initial</th>
<th>Provenance</th>
<th>STSSched</th>
<th>TFB</th>
<th>Optimal</th>
<th>NoDiverge</th>
</tr>
</thead>
<tbody>
<tr>
<td>raft-45</td>
<td>Akka-FIFO, reproduced</td>
<td>2160 (E:108)</td>
<td>2138 (E:108)</td>
<td>1183 (E:8)</td>
<td>23 (E:8)</td>
<td>22 (E:8)</td>
<td>1826 (E:11)</td>
</tr>
<tr>
<td>raft-46</td>
<td>Akka-FIFO, reproduced</td>
<td>1250 (E:108)</td>
<td>1243 (E:108)</td>
<td>674 (E:8)</td>
<td>35 (E:8)</td>
<td>23 (E:6)</td>
<td>896 (E:9)</td>
</tr>
<tr>
<td>raft-56</td>
<td>Akka-FIFO, found</td>
<td>2380 (E:108)</td>
<td>2376 (E:108)</td>
<td>1427 (E:8)</td>
<td>82 (E:8)</td>
<td>21 (E:8)</td>
<td>2064 (E:9)</td>
</tr>
<tr>
<td>raft-58a</td>
<td>Akka-FIFO, found</td>
<td>2850 (E:108)</td>
<td>2824 (E:108)</td>
<td>953 (E:32)</td>
<td>226 (E:31)</td>
<td>51 (E:11)</td>
<td>2368 (E:35)</td>
</tr>
<tr>
<td>raft-58b</td>
<td>Akka-FIFO, found</td>
<td>1500 (E:208)</td>
<td>1496 (E:208)</td>
<td>164 (E:13)</td>
<td>40 (E:8)</td>
<td>28 (E:8)</td>
<td>1103 (E:13)</td>
</tr>
<tr>
<td>raft-42</td>
<td>Akka-FIFO, reproduced</td>
<td>1710 (E:208)</td>
<td>1695 (E:208)</td>
<td>1093 (E:39)</td>
<td>180 (E:21)</td>
<td>39 (E:16)</td>
<td>1264 (E:43)</td>
</tr>
<tr>
<td>raft-66</td>
<td>Akka-UDP, found</td>
<td>400 (E:68)</td>
<td>392 (E:68)</td>
<td>262 (E:23)</td>
<td>77 (E:15)</td>
<td>29 (E:10)</td>
<td>279 (E:23)</td>
</tr>
<tr>
<td>spark-2294</td>
<td>Akka-FIFO, reproduced</td>
<td>1000 (E:30)</td>
<td>886 (E:30)</td>
<td>43 (E:3)</td>
<td>40 (E:3)</td>
<td>25 (E:1)</td>
<td>43 (E:3)</td>
</tr>
<tr>
<td>spark-3150</td>
<td>Akka-FIFO, reproduced</td>
<td>600 (E:20)</td>
<td>536 (E:20)</td>
<td>18 (E:3)</td>
<td>14 (E:3)</td>
<td>11 (E:3 )</td>
<td>18 (E:3)</td>
</tr>
<tr>
<td>spark-9256</td>
<td>Akka-FIFO, found (rare)</td>
<td>300 (E:20)</td>
<td>256 (E:20)</td>
<td>11 (E:1)</td>
<td>11 (E:1)</td>
<td>11 (E:1)</td>
<td>11 (E:1)</td>
</tr>
</tbody>
</table>
## Runtime Breakdown

<table>
<thead>
<tr>
<th>Bug Name</th>
<th>STSSched</th>
<th>TFB</th>
</tr>
</thead>
<tbody>
<tr>
<td>raft-45</td>
<td>56s (594)</td>
<td>114s (2854)</td>
</tr>
<tr>
<td>raft-46</td>
<td>73s (384)</td>
<td>209s (4518)</td>
</tr>
<tr>
<td>raft-56</td>
<td>54s (524)</td>
<td>2078s (31149)</td>
</tr>
<tr>
<td>raft-58a</td>
<td>137s (624)</td>
<td>43345s (834972)</td>
</tr>
<tr>
<td>raft-58b</td>
<td>23s (340)</td>
<td>31s (1747)</td>
</tr>
<tr>
<td>raft-42</td>
<td>118s (568)</td>
<td>10558s (176517)</td>
</tr>
<tr>
<td>raft-66</td>
<td>14s (192)</td>
<td>334s (10334)</td>
</tr>
<tr>
<td>spark-2294</td>
<td>330s (248)</td>
<td>97s (78)</td>
</tr>
<tr>
<td>spark-3150</td>
<td>219s (174)</td>
<td>26s (21)</td>
</tr>
<tr>
<td>spark-9256</td>
<td>96s (73)</td>
<td>0s (0)</td>
</tr>
</tbody>
</table>
Integrating with other RPC libs