Failure Sketches: A Better Way to Debug

Baris Kasikci, Cristiano Pereira, Gilles Pokam, Benjamin Schubert, Madan Musuvathy, George Candea
Failure and Root Cause

• Failure
  • Violation of a program specification
  • Memory errors, hangs, etc

• Root cause
  • “The real reason” behind the failure
  • When removed from the program, the failure does not recur
Debugging In-Production Software Failures Today
Debugging In-Production Software Failures Today
Debugging In-Production Software Failures Today
Debugging In-Production Software Failures Today

Understand root cause
Debugging In-Production Software Failures Today

Understand root cause

#0 0x00007f51abae820b in raise (sig=11) at ../nptl/sysdeps/unix/sysv/linux/pt-raise.c:37
#1 0x000000000042d289 in ap_buffered_log_writer (r=0x7f51a40053d0, handle=0x20eeba0, _strs=0x7f51a4003578, strsl=0x7f51a40035e8, nelts=14, len=82) at mod_log_config.c:1368
#2 0x000000000042b10d in ap_buffered_log_writer (r=0x7f51a40053d0, handle=0x20eeba0, _strs=0x7f51a4003578, strsl=0x7f51a40035e8, nelts=14, len=82) at mod_log_config.c:1368
#3 0x000000000042b10d in ap_buffered_log_writer (r=0x7f51a40053d0) at mod_log_config.c:930
#4 0x000000000042b10d in ap_buffered_log_writer (r=0x7f51a40053d0) at mod_log_config.c:930
#5 0x000000000042b10d in ap_buffered_log_writer (r=0x7f51a40053d0) at mod_log_config.c:930
#6 0x000000000042b10d in ap_buffered_log_writer (r=0x7f51a40053d0) at mod_log_config.c:930
#7 0x000000000042b10d in ap_buffered_log_writer (r=0x7f51a40053d0) at mod_log_config.c:930
#8 0x000000000042b10d in ap_buffered_log_writer (r=0x7f51a40053d0) at mod_log_config.c:930
#9 0x000000000042b10d in ap_buffered_log_writer (r=0x7f51a40053d0) at mod_log_config.c:930
#10 0x000000000042b10d in ap_buffered_log_writer (r=0x7f51a40053d0) at mod_log_config.c:930
#11 0x000000000042b10d in ap_buffered_log_writer (r=0x7f51a40053d0) at mod_log_config.c:930
#12 0x000000000042b10d in ap_buffered_log_writer (r=0x7f51a40053d0) at mod_log_config.c:930
#13 0x000000000042b10d in ap_buffered_log_writer (r=0x7f51a40053d0) at mod_log_config.c:930

Reproduce the problem
Debugging In-Production Software Failures Today

Understand root cause

Reproduce the problem
Tackling the Debugging Challenge

• Record/replay

• Special runtime support\(^1\)
  • VM checkpointing

• Custom hardware\(^2\)
  • Not widely available

\(^1\) J. Tuček et al., Triage: Diagnosing Production Run Failures at the User's Site, SOSP 2007
\(^2\) G. Pokam et al., QuickRec: prototyping an intel architecture extension for record and replay of multithreaded programs, ISCA 2013
Tackling the Debugging Challenge

• Record/replay
• Special runtime support\(^1\)
  • VM checkpointing
• Custom hardware\(^2\)
  • Not widely available

Existing tools don’t help debugging in-production failures\(^3\)

\(^1\) J. Tucek et al., Triage: Diagnosing Production Run Failures at the User's Site, SOSP 2007
\(^2\) G. Pokam et al., QuickRec: prototyping an intel architecture extension for record and replay of multithreaded programs, ISCA 2013
\(^3\) C. Sadowski et al., How developers use data race detection tools, Workshop on Evaluation and Usability of Programming Languages and Tools 2014
Debugging In-Production Software Failures Today

Understand root cause

Reproduce the problem

```c
#0 0x00007f51abae820b in raise (sig=11) at ../nptl/sysdeps/unix/sysv/linux/pt-raise.c:37
#1 0x000000000042d289 in ap_buffered_log_writer (r=0x7f51a40053d0, handle=0x282eeba0, strs=0x7f51a4003570, strl=0x7f51a40035e8, neltz=14, len=82) at mod_log_config.c:1368
#2 0x000000000042b10d in config_log_transaction (r=0x7f51a40053d0, cls=0x20b9d50, default_format=0x208ee370) at mod_log_config.c:950
#3 0x000000000042aad6 in multi_log_transaction (r=0x7f51a40053d0) at mod_log_config.c:950
#4 0x000000000042b10d in ap_run_log_transaction (r=0x7f51a40053d0) at protocol.c:1563
#5 0x0000000000436e81 in ap_process_request (r=0x7f51a40053d0) at http_request.c:312
#6 0x000000000042aad6 in ap_run_process_connection (c=0x7f519c000b68) at connection.c:85
#7 0x00000000004461f5 in ap_process_connection (c=0x7f519c000b68) at connection.c:211
#8 0x0000000000451ba0 in process_socket (p=0x7f519c000b68, sock=0x7f519c000b68, my_child_num=0, my_thread_num=0, bucketalloca=0x7f519c001348) at worker.c:496
#9 0x0000000000451ba0 in worker_thread (thd=0x2820fa90, dummy=0x7f51a40058c0) at worker.c:496
#11 0x0000000000451ba0 in dummy_worker (opaque=0x2820fa90) at thread.c:117
#12 0x0000000000451ba0 in start_thread (argv=0x7f51a4001348) at pthread_create.c:312
#13 0x0000000000451ba0 in clone () at ../sysdeps/unix/sysv/linux/clone.c:111
```

Understand

Understand the root cause of the software failure. This involves analyzing the stack trace and identifying the underlying issue.

Reproduce

Reproduce the problem by carefully isolating the conditions under which the failure occurs. This could involve setting up a test environment that mirrors the production setup.

Understand root cause

Understand root cause

Reproduce

Reproduce the problem

Understand root cause

Understand root cause

Reproduce

Reproduce the problem

Understand root cause

Understand root cause

Reproduce

Reproduce the problem

Understand root cause

Understand root cause

Reproduce

Reproduce the problem

Understand root cause

Understand root cause

Reproduce

Reproduce the problem

Understand root cause

Understand root cause

Reproduce

Reproduce the problem
Failure Sketch

Time

Thread 1

1 main() {
2    queue* f = init(size);
3    create_thread(cons, f);
4    ...
5    free(f->mut);
6    f->mut = NULL;
7    ...
8 }

Thread 2

1 cons(queue* f) {
2    ...
3    mutex_unlock(f->mut);
4    ...
5    ...

```c
main() {
    queue* f = init(size);
    create_thread(cons, f);
    ...
    free(f->mut);
    f->mut = NULL;
    ...
}
```

```c
cons(queue* f) {
    ...
    mutex_unlock(f->mut);
    ...
}
```

**Thread 1**

- Line 1: Main function starts
- Line 2: Allocate a queue
- Line 3: Create a thread
- Line 4: ... (remaining lines)

**Thread 2**

- Line 1: Thread function starts
- Line 4: ... (remaining lines)

**Root cause:**
- Line 7: `f->mut = NULL;`
- Line 7: `mutex_unlock(f->mut);`

**Failure:** segmentation fault
Failure Sketch

Thread 1

```
1 main() {
2     queue* f = init(size);
3     create_thread(cons, f);
4     ...
5     free(f->mut);
6     f->mut = NULL;
7     ...
8 }
```

Thread 2

```
1 cons(queue* f) {
2     ...
3     mutex_unlock(f->mut);
4     ...
5 }
```

Failure: segmentation fault
Failure Sketch Use Case

Understand root cause

Reproduce the problem

```c
#0 0x00007f51abae820b in raise (sig=11) at ../npt1/sysdeps/unix/sysv/linux/pt-raise.c:37
#1 0x0000000000042ad89 in ap_buffered_log_writer (r=0x7f51a40053d0, handle=0x20eaa6, 
    strs=0x7f51a4003358, ptrl=0x7f51a40035e8, nelts=14, 
    len=82) at mod_log_config.c:1368
#2 0x0000000000042b10d in ap_buffered_log_writer (r=0x7f51a40053d0, handle=0x20eaa6, 
    strs=0x7f51a4003578, strl=0x7f51a40035e8, nelts=14, 
    len=82) at mod_log_config.c:1368
#3 0x0000000000042aad6 in multi_log_transaction (r=0x7f51a40053d0) at mod_log_config.c:1950
#4 0x0000000000042b10d in ap_buffered_log_writer (r=0x7f51a40053d0, handle=0x20eaa6, 
    strs=0x7f51a4003578, strl=0x7f51a40035e8, nelts=14, 
    len=82) at mod_log_config.c:1368
#5 0x0000000000042aad6 in multi_log_transaction (r=0x7f51a40053d0) at mod_log_config.c:1950
#6 0x0000000000042aad6 in multi_log_transaction (r=0x7f51a40053d0) at mod_log_config.c:1950
#7 0x00000000000465bd in ap_run_log_transaction (r=0x7f51a40053d0) at protocol.c:1563
#8 0x00000000000465bd in ap_run_log_transaction (r=0x7f51a40053d0) at protocol.c:1563
#9 0x00000000000465bd in ap_run_log_transaction (r=0x7f51a40053d0) at protocol.c:1563
#10 0x00000000000465bd in ap_run_log_transaction (r=0x7f51a40053d0) at protocol.c:1563
#11 0x00000000000465bd in ap_run_log_transaction (r=0x7f51a40053d0) at protocol.c:1563
#12 0x00000000000465bd in ap_run_log_transaction (r=0x7f51a40053d0) at protocol.c:1563
#13 0x00000000000465bd in ap_run_log_transaction (r=0x7f51a40053d0) at protocol.c:1563
```
Failure Sketch Use Case

Understand root cause

Reproduce the problem

```c
#0 0x00007f51abae820b in raise (sig=11) at ../nptl/sysdeps/unix/sysv/linux/pt-raise.c:37
#1 0x0000000000042a0d in ap_buffered_log_writer (r=0x7f51a40053d0, handle=0x200e9ba0, strs=0x7f51a4003558, ptrl=0x7f51a40035e8, nelt=14, len=2) at mod_log_config.c:1308
#2 0x0000000000042a0d in config_log_transaction (r=0x7f51a40053d0, cls=0x200b9d50, default_format=0x200e9de70) at mod_log_config.c:930
#3 0x0000000000042a0d in multi_log_transaction (r=0x7f51a40053d0) at mod_log_config.c:950
#4 0x0000000000042a0d in ap_run_log_transaction (r=0x7f51a40053d0) at protocol.c:1563
#5 0x0000000000042a0d in ap_process_request (r=0x7f51a40053d0) at http_request.c:312
#6 0x0000000000042a0d in process_socket (p=0x7f519c0009b8, sock=0x7f519c000a20, my_child_num=0, my_thread_num=0, bucket_alloc=0x7f51a4001348) at worker.c:632
#7 0x0000000000042a0d in worker_thread (thd=0x210fa90, dummy=0x7f51a40058c0) at worker.c:946
#8 0x0000000000042a0d in dummy_worker (opaque=0x210fa90) at thread.c:112
#9 0x0000000000042a0d in start_thread (arg=0x7f51a40058c0) at pthread_create.c:312
#10 0x0000000000042a0d in clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.S:111
```
Failure Sketch Use Case

Understand root cause

Reproduce the problem

Runtime traces

```
#0 0x00007f51abae820b in raise (sig=11) at ../nptl/sysdeps/unix/sysv/linux/pt-raise.c:37
#1 0x000000000004242d89 in ap_buffered_log_writer (r=0x7f51a40053d0, handle=0x2026e0a0, strs=0x7f51a4003578, strl=0x7f51a40035e8, nelts=14, len=82) at mod_log_config.c:1368
#2 0x0000000000042b10d in ap_buffered_log_writer (r=0x7f51a40053d0, handle=0x20eeba0, strs=0x7f51a4003578, strl=0x7f51a40035e8, nelts=14, len=82) at mod_log_config.c:1368
#3 0x0000000000042aad6 in multi_log_transaction (r=0x7f51a40053d0) at mod_log_config.c:950
#4 0x0000000000046cb2d in ap_run_log_transaction (r=0x7f51a40053d0) at protocol.c:1563
#5 0x0000000000046aad6 in ap_process_request (r=0x7f51a40053d0) at http_request.c:312
#6 0x0000000000046aad6 in ap_process_request (r=0x7f51a40053d0) at http_request.c:312
#7 0x0000000000046aad6 in ap_process_request (r=0x7f51a40053d0) at http_request.c:312
#8 0x0000000000046aad6 in ap_process_request (r=0x7f51a40053d0) at http_request.c:312
#9 0x0000000000046aad6 in ap_process_request (r=0x7f51a40053d0) at http_request.c:312
#10 0x0000000000046aad6 in ap_process_request (r=0x7f51a40053d0) at http_request.c:312
#11 0x0000000000046aad6 in ap_process_request (r=0x7f51a40053d0) at http_request.c:312
#12 0x0000000000046aad6 in ap_process_request (r=0x7f51a40053d0) at http_request.c:312
#13 0x0000000000046aad6 in ap_process_request (r=0x7f51a40053d0) at http_request.c:312
```

Understand root cause

Reproduce the problem

Runtime traces

```
Understand
root cause

Reproduce
the problem

Runtime traces

Failure Sketch Use Case
```
Failure Sketch Use Case

Understand root cause

Reproduce the problem

main() {
    queue* f = init(size);
    create_thread(cons, f);
    ...
    free(f->mut);
    f->mut = NULL;
    ...
}

cons(queue* f) {
    ...
    mutex_unlock(f->mut);
    ...
}

Failure: segmentation fault

Runtime traces
Failure Sketch Use Case

Understand root cause

```c
main() { 
  queue* f = init(size);
  create_thread(cons, f);
  ...
  free(f->mut);
  f->mut = NULL;
  ...
}
```

Thread 1

```c
cons(queue* f) {
  ...
  mutex_unlock(f->mut);
  Failure: segmentation fault
}
```

Thread 2

Runtime traces
Failure Sketch Use Case

main() {
    queue* f = init(size);
    create_thread(cons, f);
    ...
    free(f->mut);
    f->mut = NULL;
    ...
}

Thread 1
Time

Thread 2

1 1
2 2
3 3
4 4
5 5
6 6
7 7
8 8

cons(queue* f) {
    ...
    mutex_unlock(f->mut);
    ...
}

Failure: segmentation fault

Runtime traces
Failure Sketch Use Case

main() {
  queue* f = init(size);
  create_thread(cons, f);
  ...
  free(f->mut);
  f->mut = NULL;
  ...
}

cons(queue* f) {
  ...
  mutex_unlock(f->mut);
  ...
}

Time

Thread 1

1 main() {
  queue* f = init(size);
  create_thread(cons, f);
  ...
  free(f->mut);
  f->mut = NULL;
  ...
}

Thread 2

1 cons(queue* f) {
  ...
  mutex_unlock(f->mut);
  ...
}

Runtime traces

Failure: segmentation fault
Research Challenges

- Hard-to-reproduce failures
  - Recur only a few times in production
- Accuracy of failure sketches
  - No extraneous elements in the failure sketch
- Latency of failure sketch computation
  - Developers can’t wait forever for failure sketches
System Architecture

```c
main() {
  queue* f = init(size);
  create_thread(cons, f);
  ...
  free(f->mut);
  f->mut = NULL;
  ...
}

cons(queue* f) {
  ...
  mutex_unlock(f->mut);
  ...
}
```

Time

Thread 1

1 main() {
2   queue* f = init(size);
3   create_thread(cons, f);
4   ...
5   free(f->mut);
6   f->mut = NULL;
7   ...
8 }

Thread 2

1 cons(queue* f) {
2   ...
3   mutex_unlock(f->mut);
4   ...
5   ...
6   ...
7   Failure: segmentation fault
8 }

Runtime traces
Program P (source)

Failure report (core dump, stack trace, etc)
Program P (source)

Failure report (core dump, stack trace, etc)

Server

Static Analyzer

- queue* f = init(size);
- create_thread(cons, f);
- free(f->mut);
- f->mut = NULL;
- mutex_unlock(f->mut);

Static slice
System Architecture

1. Program P (source)
2. Failure report (core dump, stack trace, etc)

Client

Tracking control and data flow

Server

Static Analyzer

Instrumentation
System Architecture

1. **Program P (source)**
2. **Static Analyzer**
3. **Instrumentation**

- **Server**
- **Client**

- Tracking control and data flow

- Failure report (core dump, stack trace, etc)

- Refinement with runtime traces
System Architecture

Failure Sketch

Program P (source)
Failure report (core dump, stack trace, etc)

Static Analyzer

Failure Sketch Computation Engine

Server

Client
Tracking control and data flow

Refinement with runtime traces
Instrumentation

Time

main() {
  queue* f = init(size);
  create_thread(cons, f);
  ...
  cons(queue* f) {
    ...
    free(f->mut);
    f->mut = NULL;
  }
}

Thread 1

mutex_unlock(f->mut);

Failure: segmentation fault

Thread 2
Intel Processor Trace (Intel PT)

- Control flow information
  - Compressed trace of branches taken (~1 bit per instruction)
  - Low overhead (~40% **full** tracing overhead)
Tracking Control Flow Using Intel PT
Tracking Control Flow Using Intel PT

Static Slice

Root cause

Failure
Tracking Control Flow Using Intel PT

Static Slice

Root cause

Failure

Tracking 1st iteration
## Tracking Control Flow Using Intel PT

<table>
<thead>
<tr>
<th></th>
<th>Static Slice</th>
<th>Tracking 1(^{st}) iteration</th>
<th>Tracking 2(^{nd}) iteration</th>
<th>Tracking 3(^{rd}) iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root cause</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Failure</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. **Failure Root cause**
2. **Tracking iteration**
3. **Static Slice**
Tracking Control Flow Using Intel PT

<table>
<thead>
<tr>
<th>Root cause</th>
<th>Static Slice</th>
<th>Tracking 1st iteration</th>
<th>Tracking 2nd iteration</th>
<th>Tracking 3rd iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Monitoring small portions of a slice works well because most failures have nearby root causes\textsuperscript{1,2}

\textsuperscript{1}W. Zhang et al., ConSeq: Detecting concurrency bugs through sequential errors. ASPLOS 2011
\textsuperscript{2}F. Qin et al., Rx: Treating bugs as allergies a safe method to survive software failures. SOSP 2005
Discussion

• Intrusiveness
  • Currently, we do static instrumentation
  • Dynamic instrumentation is less intrusive

• Privacy
  • Use anonymization
  • Forgo data monitoring when privacy requirements are very strict
Future Work

- Diagnosing performance problems
  - Correlating control flow with slowdowns
- Speeding up program analysis
  - Use control flow information to tackle path explosion
- Using failure sketches for test case generation
• Failure sketches
  • Summary explaining failure root causes
• Application of hardware-based monitoring
  • Enabler for building failure sketches
  • Many potential use cases