Failure Sketching: A Technique for Automated Root Cause Diagnosis of In-Production Failures

Baris Kasikci, Benjamin Schubert, Cristiano Pereira, Gilles Pokam, George Candea
Debugging In-Production Software Failures Today
Debugging In-Production Software
Failures Today
Debugging In-Production Software Failures Today

```c
#0  0x0000000000002000 in raise (sig=11) at /lib64/.
    raise_log_writer
    ap_buffered_log_writer
    #1  0x0000000000000004 in ap_buffered_log_writer
        (r=0x7f51a40053d0, handle=0x20eeba8, _
        str=0x7f51a40053d0, strl=0x7f51a40053d0, _
        nelts=14, len=82) at mod_log_config.c:1368
#2  0x0000000000000000 in config_log_transaction
    (r=0x7f51a40053d0, cls=0x20b9d50, 
    default_format=0x20ee370, ) at mod_log_config.c:1390
#3  0x0000000000000000 in multi_log_transaction
    (r=0x7f51a40053d0) at mod_log_config.c:930
#4  0x0000000000000000 in ap_run_log_transaction
    (r=0x7f51a40053d0) at protocol.c:1563
#5  0x0000000000000000 in ap_process_request
    (r=0x7f51a40053d0) at http_request.c:312
#6  0x0000000000000000 in ap_process_http_connection
    (r=0x7f51a40053d0) at http_core.c:293
#7  0x0000000000000000 in ap_run_process_connection
    (r=0x7f51a40053d0) at connection.c:85
#8  0x0000000000000000 in ap_process_connection
    (c=0x7f51a40053d0, csd=0x7f51a40053d0) at 
    connection.c:211
#9  0x0000000000000000 in process_socket
    (p=0x7f51a40053d0, sock=0x7f51a40053d0, 
    my_child_num=0, my_thread_num=0, 
    bucket_alloc=0x7f51a40053d0) at worker.c:632
#10 0x0000000000000000 in worker_thread
    (thd=0x7f51a40053d0, dummy=0x7f51a40053d0) at worker.c:946
#11 0x0000000000000000 in dummy_worker
    (opaque=0x7f51a40053d0) at thread.c:227
#12 0x0000000000000000 in start_thread
    (arg=0x7f51a40053d0) at pthread_create.c:211
#13 0x0000000000000000 in clone () at /lib64/.
    thread.c:127
```
Debugging In-Production Software Failures Today

Understand root cause
Debugging In-Production Software

Failures Today

Understand root cause

Reproduce the failure

0: 0x00007f51abae820b in raise (sig=11) at ../nptl/sysdeps/unix/sysv/linux/pt-raise.c:37
#1 0x000000000004250289 in ap_buffered_log_writer (r=0x7f51a0053d0, handle=0x20eeba8, _strs=0x7f51a403578, strl=0x7f51a4035e8, nelts=14, _len=82) at mod_log_config.c:1368
#2 0x0000000000042b10d4 in config_log_transaction (r=0x7f51a0053d0, cls=0x20b9d50, default_format=0x20ee370) at mod_log_config.c:1368
#3 0x00000000000423a4 in multi_log_transaction (r=0x7f51a0053d0) at mod_log_config.c:930
#4 0x00000000000423a4 in ap_run_log_transaction (r=0x7f51a0053d0) at protocol.c:1368
#5 0x00000000000423a4 in ap_process_request (r=0x7f51a0053d0) at http_request.c:312
#6 0x00000000000423a4 in ap_process_http_connection (c=0x7f519c000b68) at http_core.c:293
#7 0x00000000000423a4 in ap_run_process_connection (c=0x7f519c000b68) at connection.c:211
#8 0x00000000000423a4 in process_socket (p=0x7f519c0009b8, sock=0x7f519c000a20, my_child_num=0, my_thread_num=0, bucket_alloc=0x7f51a001348) at worker.c:632
#9 0x00000000000423a4 in worker_thread (thd=0x210fa90, dummy=0x7f51a000a20) at worker.c:946
#10 0x00000000000423a4 in process_socket (p=0x7f519c0009b8, sock=0x7f519c000a20, my_child_num=0, my_thread_num=0, bucket_alloc=0x7f51a001348) at worker.c:632
#11 0x00000000000423a4 in dummy_worker (opaque=0x210fa90) at thread.c:327
#12 0x00000000000423a4 in start_thread (arg=0x7f51a001348) at pthread_create.c:312
#13 0x00000000000423a4 in clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.c:111
Debugging In-Production Software Failures Today

Understand root cause

Reproduce the failure

#0 0x00007f51abae820b in raise (sig=11) at ../nptl/sysdeps/unix/sysv/linux/pt-raise.c:37
#1 0x0000000000042d289 in ap_buffered_log_writer (r=0x7f51a40053d0, handle=0x20eeba0, _strs=0x7f51a4003578, strl=0x7f51a40035e8, nelts=14, is_locked=0x2000000000000000) at mod_log_config.c:1368
#2 0x0000000000042b10d in ap_buffered_log_writer (r=0x7f51a40053d0, handle=0x20eeba0, _strs=0x7f51a4003578, strl=0x7f51a40035e8, nelts=14, is_locked=0x2000000000000000) at mod_log_config.c:1368
#3 0x0000000000042aad6 in multi_log_transaction (r=0x7f51a40053d0) at mod_log_config.c:930
#4 0x0000000000042aad6 in multi_log_transaction (r=0x7f51a40053d0) at mod_log_config.c:930
#5 0x0000000000042aad6 in multi_log_transaction (r=0x7f51a40053d0) at mod_log_config.c:930
#6 0x0000000000042aad6 in multi_log_transaction (r=0x7f51a40053d0) at mod_log_config.c:930
Related Work

• Collaborative approaches
  • *WER [SOSP’09], CBI [PLDI’05], CCI [OOPSLA’10]*

• Identifying differences of failing and successful runs
  • *Delta debugging [TSE’02], Symbiosis [PLDI’15]*

• Record & replay, checkpointing
  • *ODR [SOSP’09], Triage [SOSP’07]*

• Hardware support
  • *PBI [ASPLOS’13], LBRA/LCRA [ASPLOS’14]*
Related Work

• Collaborative approaches
  • WER [SOSP’09], CBI [PLDI’05], CCI [OOPSLA’10]

• Identifying differences of failing and successful runs
  • Delta debugging [TSE’02], Symbiosis [PLDI’15]

• Record & replay, checkpointing
  • ODR [SOSP’09], Triage [SOSP’07]

• Hardware support
  • PBI [ASPLOS’13], LBRA/LCRA [ASPLOS’14]
Contributions
Contributions

Goal: automate the manual detective work of debugging
Contributions

Goal: automate the manual detective work of debugging

Failure sketching

Complements in-house static analysis with in-production dynamic analysis

Automatically and efficiently builds accurate failure sketches that show root causes of failures
main() {
    queue* f = init(size);
    create_thread(cons, f);
    ...
    free(f->mut);
    ...
}

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

Thread 1

Thread 2

1 cons(queue* f) {
2 ...
3 }
4 mutex_unlock(f->mut);
5 ...
6 }

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

Failure Sketch

Time

Thread 1

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

Thread 2

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

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

Thread 1

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

Thread 2

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

Time

Thread 2

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

Thread 1

Thread 2

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

Failure Sketch

Time

Thread 2

Thread 1

1
2
3
4
5
6
7
8

Segfault

1
2
3
4
5
6
7
8

Thread 2

Thread 1

1
2
3
4
5
6
7
8

Segfault

Thread 2

Thread 1

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

failure sketch

1
2
3
4
5
6
7
8

Thread 1

1
2
3
4
5
6
7
8

Thread 2

1
2
3
4
5
6
7
8

mutex_unlock(f->mut);

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

Failure Sketch

Time

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

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

Root cause

Segfault
Failure Sketch Usage Model

Understand root cause

Reproduce the failure
Failure Sketch Usage Model

Understand root cause

Reproduce the failure

#0 0x00007f51abae820b in raise (sig=11) at ../opt1/sysdeps/unix/sysv/linux/pt-raise.c:37
#1 0x0000000000524f39 in _ap_buffered_log_writer (r=0x7f51a00553d0, handle=0x20e9ea0a8, strs=0x7f51a400335f8, strl=0x7f51a00335f8, neltz=14, len=82) at mod_log_config.c:1368
#2 0x0000000000524f39 in _ap_buffered_log_writer (r=0x7f51a00553d0, handle=0x20e9ea0a8, strs=0x7f51a400335f8, strl=0x7f51a00335f8, neltz=14, len=82) at mod_log_config.c:1368
#3 0x0000000000524f39 in _ap_buffered_log_writer (r=0x7f51a00553d0, handle=0x20e9ea0a8, strs=0x7f51a400335f8, strl=0x7f51a00335f8, neltz=14, len=82) at mod_log_config.c:1368
#4 0x0000000000524f39 in _ap_buffered_log_writer (r=0x7f51a00553d0, handle=0x20e9ea0a8, strs=0x7f51a400335f8, strl=0x7f51a00335f8, neltz=14, len=82) at mod_log_config.c:1368
#5 0x0000000000524f39 in _ap_buffered_log_writer (r=0x7f51a00553d0, handle=0x20e9ea0a8, strs=0x7f51a400335f8, strl=0x7f51a00335f8, neltz=14, len=82) at mod_log_config.c:1368
#6 0x0000000000524f39 in _ap_buffered_log_writer (r=0x7f51a00553d0, handle=0x20e9ea0a8, strs=0x7f51a400335f8, strl=0x7f51a00335f8, neltz=14, len=82) at mod_log_config.c:1368
#7 0x0000000000524f39 in _ap_buffered_log_writer (r=0x7f51a00553d0, handle=0x20e9ea0a8, strs=0x7f51a400335f8, strl=0x7f51a00335f8, neltz=14, len=82) at mod_log_config.c:1368
#8 0x0000000000524f39 in _ap_buffered_log_writer (r=0x7f51a00553d0, handle=0x20e9ea0a8, strs=0x7f51a400335f8, strl=0x7f51a00335f8, neltz=14, len=82) at mod_log_config.c:1368
#9 0x0000000000524f39 in _ap_buffered_log_writer (r=0x7f51a00553d0, handle=0x20e9ea0a8, strs=0x7f51a400335f8, strl=0x7f51a00335f8, neltz=14, len=82) at mod_log_config.c:1368
#10 0x0000000000524f39 in _ap_buffered_log_writer (r=0x7f51a00553d0, handle=0x20e9ea0a8, strs=0x7f51a400335f8, strl=0x7f51a00335f8, neltz=14, len=82) at mod_log_config.c:1368
#11 0x0000000000524f39 in _ap_buffered_log_writer (r=0x7f51a00553d0, handle=0x20e9ea0a8, strs=0x7f51a400335f8, strl=0x7f51a00335f8, neltz=14, len=82) at mod_log_config.c:1368
#12 0x0000000000524f39 in _ap_buffered_log_writer (r=0x7f51a00553d0, handle=0x20e9ea0a8, strs=0x7f51a400335f8, strl=0x7f51a00335f8, neltz=14, len=82) at mod_log_config.c:1368
#13 0x0000000000524f39 in _ap_buffered_log_writer (r=0x7f51a00553d0, handle=0x20e9ea0a8, strs=0x7f51a400335f8, strl=0x7f51a00335f8, neltz=14, len=82) at mod_log_config.c:1368

#0 0x00000000042e820b in raise (sig=11) at ../opt1/sysdeps/unix/sysv/linux/pt-raise.c:37
#1 0x00000000042e820b in raise (sig=11) at ../nptl/sysdeps/unix/sysv/linux/pt-raise.c:37
#2 0x00000000042e820b in ap_buffered_log_writer (r=0x7f51a000530d, handle=0x202ea8a0, strs=0x7f51a0003578, strl=0x7f51a00035e8, nelts=14, len=82) at mod_log_config.c:1368
#3 0x00000000042e820b in config_log_transaction (r=0x7f51a000530d, cls=0x20b9d50, default_format=0x206eae37d) at mod_log_config.c:930
#4 0x00000000042e820b in multi_log_transaction (r=0x7f51a000530d) at mod_log_config.c:950
#5 0x00000000042e820b in ap_process_request (r=0x7f51a000530d) at http_request.c:312
#6 0x00000000042e820b in ap_process_request (r=0x7f51a000530d) at http_request.c:312
#7 0x00000000042e820b in ap_run_log_transaction (r=0x7f51a000530d) at protocol.c:1563
#8 0x00000000042e820b in ap_process_connection (c=0x7f51a0009e82) at connection.c:211
#9 0x00000000042e820b in process_socket (p=0x7f51a0009e82, sock=0x7f51a0009e82) at worker.c:632
#10 0x00000000042e820b in worker_thread (thd=0x210fe19a0, dummy=0x7f51a0009e82) at worker.c:946
#11 0x00000000042e820b in dummy_worker (opaque=0x210fe19a0) at thread.c:132
#12 0x00000000042e820b in start_thread (ary=0x7f51a0009e82) at pthread_create.c:312
#13 0x00000000042e820b in clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.c:111

Understand root cause

Reproduce the failure

Runtime traces

Failure Sketch Usage Model
Failure Sketch Usage Model

Understand root cause

Time
1 main() {
    queue* f = init(size);
    create_thread(cons, f);
    ...
7 ...
8 }

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

Thread 2
1
2 cons(queue* f) {
3 ...
4
5 ...
6 mutex unlock(f->mut);
7 ...
8 }

Runtime traces

Reproduce the failure
Failure Sketch Usage Model

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

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

Thread 1

Thread 2

Runtime traces
Failure Sketch Usage Model

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

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

Thread 1

Thread 2

Time

Runtime traces
Outline

- Challenges
- Design
- Evaluation
Outline

- Challenges
- Design
- Evaluation
Challenges of Building Failure Sketches

• Accuracy
  • *Exclude all irrelevant information, preserve all relevant one*

• Recurrence
  • *Gathering enough execution information from rare failures*

• Latency
  • *Achieve high accuracy after just a few recurrences*
Outline

- Challenges
- Design
- Evaluation
Outline

• Challenges

• Design

• Evaluation
main() {
 queue* f = init(size);
 create_thread(cons, f);
 ...
 free(f->mut);
 ...
}
Gist System Architecture
Gist System Architecture

Program P (source) → Server

Failure report (core dump, stack trace, etc) → Server

Client
Gist System Architecture

Program P (source)

Failure report (core dump, stack trace, etc)

1

Static Analyzer

- queue* f = init(size);
- init_vars();
- free(f->mut);
- print("Done");
- f->mut = NULL;

Static slice
Gist System Architecture

Program P (source)

Failure report (core dump, stack trace, etc)

1. Static Analyzer

- queue* f = init(size);
- init_vars();
- free(f->mut);
- print(“Done”);
- f->mut = NULL;

2. Instrumentation

Client

Tracking control and data flow

Static slice
Gist System Architecture

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

- queue* f = init(size);
- init_vars();
- free(f->mut);
- print("Done");
- f->mut = NULL;

Static slice
Gist System Architecture

1. Program P (source)
2. Static Analyzer
3. Client
   - Refinement with runtime traces
   - Tracking control and data flow

Failure report (core dump, stack trace, etc)

• queue* f = init(size);
  ```
  init_vars();
  ```
• free(f->mut);
  ```
  print("Done");
  ```
• f->mut = NULL;

Refined static slice
Gist System Architecture

Client

Server

Failure Sketch

4

Failure Sketch Computation Engine

Failure Sketch

Refinement with runtime traces

3

Static Analyzer

2

Instrumentation

1

Program P (source)

Failure report (core dump, stack trace, etc)

Thread 1

Thread 2

main()

create_thread(cons, f);

cons(queue* f) {

free(f->mut);

mutex_unlock(f->mut);

f->mut = NULL;

print("Done");

f->mut = NULL;

...}

...}

...}

...}

create_thread(cons, f);

cons(queue* f) {

free(f->mut);

mutex_unlock(f->mut);

f->mut = NULL;

print("Done");

f->mut = NULL;

...}

...}

...}

...}

create_thread(cons, f);

cons(queue* f) {

free(f->mut);

mutex_unlock(f->mut);

f->mut = NULL;

print("Done");

f->mut = NULL;

...}

...}

...}

...}

create_thread(cons, f);

cons(queue* f) {

free(f->mut);

mutex_unlock(f->mut);

f->mut = NULL;

print("Done");

f->mut = NULL;

...}

...}

...}

...}

create_thread(cons, f);

cons(queue* f) {

free(f->mut);

mutex_unlock(f->mut);

f->mut = NULL;

print("Done");

f->mut = NULL;

...}

...}

...}

...}

create_thread(cons, f);

cons(queue* f) {

free(f->mut);

mutex_unlock(f->mut);

f->mut = NULL;

print("Done");

f->mut = NULL;

...}

...}

...}

...}

create_thread(cons, f);

cons(queue* f) {

free(f->mut);

mutex_unlock(f->mut);

f->mut = NULL;

print("Done");

f->mut = NULL;

...}

...}

...}

...}

create_thread(cons, f);

cons(queue* f) {

free(f->mut);

mutex_unlock(f->mut);

f->mut = NULL;

print("Done");

f->mut = NULL;

...}

...}

...}

...}

create_thread(cons, f);

cons(queue* f) {

free(f->mut);

mutex_unlock(f->mut);

f->mut = NULL;

print("Done");

f->mut = NULL;

...}

...}

...}

...}

create_thread(cons, f);

cons(queue* f) {

free(f->mut);

mutex_unlock(f->mut);

f->mut = NULL;

print("Done");

f->mut = NULL;

...}

...}

...}

...}

create_thread(cons, f);

cons(queue* f) {

free(f->mut);

mutex_unlock(f->mut);

f->mut = NULL;

print("Done");

f->mut = NULL;

...}

...}

...}

...}

create_thread(cons, f);

cons(queue* f) {

free(f->mut);

mutex_unlock(f->mut);

f->mut = NULL;

print("Done");

f->mut = NULL;

...}

...}

...}

...}

create_thread(cons, f);

cons(queue* f) {

free(f->mut);

mutex_unlock(f->mut);

f->mut = NULL;

print("Done");

f->mut = NULL;

...}

...}

...}

...}

create_thread(cons, f);

cons(queue* f) {

free(f->mut);

mutex_unlock(f->mut);

f->mut = NULL;

print("Done");

f->mut = NULL;

...}
Outline

- Challenges

- Design

- Evaluation
Outline

- Challenges
- Design
- Evaluation
Static Analysis to Reduce the Overhead

• Computes backward slices
  • *Includes statements with dependencies to the failure*
  • *Excludes all other statements*

• Inter-procedural
  • *Identify dependencies across functions*
Static Analysis to Reduce the Overhead

- Computes backward slices
  - Includes statements with dependencies to the failure
  - Excludes all other statements
- Inter-procedural
  - Identify dependencies across functions

Static analysis reduces subsequent runtime tracking (20x)
Example

void cleanup(State* s) {
    log(“Func:cleanup”);
    if(Verbose)
        log(“Cleaning up %p”, s);
    delete s;
}

void display_size(State* s) {
    log(“Func:display_size”);
    log(“State: %u”, s->size);
}
Example

```c
void cleanup(State* s) {
    log("Func:cleanup");
    if(verbose)
        log("Cleaning up %p", s);
    delete s;
}

void display_size(State* s) {
    log("Func:display_size");
    log("State: %u", s->size);
}
```
Example

```c
void cleanup(State* s) {
    log("Func:cleanup");
    if(verbos)
        log("Cleaning up %p", s);
    delete s;
}

void display_size(State* s) {
    log("Func:display_size");
    log("State: %u", s->size);
}
```
Example

```c
void cleanup(State* s) {
    log("Func:cleanup");
    if(verb)
        log("Cleaning up %p", s);
    delete s;
}
```

```c
void display_size(State* s) {
    log("Func:display_size");
    log("State: %u", s->size);
}
```
Example

void cleanup(State* s) {
    log("Func:cleanup");
    if( verbose )
        log("Cleaning up %p", s);
    delete s;
}

void display_size(State* s) {
    log("Func:display_size");
    log("State: %u", s->size);
}
Example

```c
void cleanup(State* s) {
    log("Func:cleanup");
    if(verbose)
        log("Cleaning up %p", s);
    delete s;
}

void display_size(State* s) {
    log("Func:display_size");
    log("State: %u", s->size);
}
```

Segfault
void cleanup(State* s) {
    log("Func:cleanup");
    if(verbose)
        log("Cleaning up %p", s);
    delete s;
}

void display_size(State* s) {
    log("Func:display size");
    log("State: %u", s->size);
}
Example: Static Backward Slicing

```c
void cleanup(State* s) {
    log("Func:cleanup");
    if(verbose)
        log("Cleaning up %p", s);
    delete s;
}

void display_size(State* s) {
    log("Func:display_size");
    log("State: %u", s->size);
}
```
Outline

- Challenges
- Design
- Evaluation

Static Analysis  Control and Data Flow Tracking  Statistical Analysis  Failure sketch
Outline

• Challenges

• Design

• Evaluation
Low-Overhead Control Flow Tracking

• Software-based tracking is expensive (up to 15x)

• Hardware-based tracking is more efficient
  • *Intel PT: new feature in Intel CPUs (~40%)

• Gist combines static analysis and hardware-based control flow tracking
  • *Low overhead (~2%)
Low-Overhead Control Flow Tracking

- Software-based tracking is expensive (up to 15x)
- Hardware-based tracking is more efficient
  - *Intel PT: new feature in Intel CPUs (~40%)*
- Gist combines static analysis and hardware-based control flow tracking
  - *Low overhead (~2%)*

Static analysis → Low-overhead control flow tracking
Example: Control Flow Tracking (Step 1)

```c
void cleanup(State* s) {
    log("Func:cleanup");
    if (verbose)
        log("Cleaning up %p", s);
    delete s;
}

void display_size(State* s) {
    log("Func:display_size");
    log("State: %u", s->size);
}
```

Segfault
void cleanup(State* s) {
    log("Func:cleanup");
    if (verbose)
        log("Cleaning up %p", s);
    delete s;
}

void display_size(State* s) {
    log("Func:display_size");
    log("State: %u", s->size);
}

Segfault
Example: Control Flow Tracking (Step 2)

```c
void cleanup(State* s) {
    log("Func:cleanup");
    if(VERBOSE)
        log("Cleaning up %p", s);
    delete s;
}

void display_size(State* s) {
    log("Func:display_size");
    log("State: %u", s->size);
}
```

Static analysis + control flow tracking shorten the sketch.
Data Flow Tracking to Increase Accuracy
Data Flow Tracking to Increase Accuracy

- Data flow information
  - Variable values & total order of memory accesses
Data Flow Tracking to Increase Accuracy

- Data flow information
  - Variable values & total order of memory accesses
- Hardware watchpoints
  - Allow tracking reads and writes with low overhead
  - Allow tracking the total order of accesses
Data Flow Tracking to Increase Accuracy

• Data flow information
  • Variable values & total order of memory accesses

• Hardware watchpoints
  • Allow tracking reads and writes with low overhead
  • Allow tracking the total order of accesses

• Monitor multiple clients when run out of watchpoints
Data Flow Tracking to Increase Accuracy

• Data flow information
  • *Variable values & total order of memory accesses*

• Hardware watchpoints
  • *Allow tracking reads and writes with low overhead*
  • *Allow tracking the total order of accesses*

• Monitor multiple clients when run out of watchpoints

Precise ordering information → High accuracy
Example: Data Flow Tracking

```c
void cleanup(State* s) {
    log(“Func:cleanup”);
    if(verbose)
        log(“Cleaning up %p”, s);
    delete s;
}

void display_size(State* s) {
    log(“Func:display_size”);
    log(“State: %u”, s->size);
}

Segfault
```
Example: Data Flow Tracking

```c
void cleanup(State* s) {
    log("Func:cleanup");
    if(Verbose)
        log("Cleaning up %p", s);
    delete s;
}

void display_size(State* s) {
    log("Func:display_size");
    log("State: %u", s->size);
}
```
Example: Data Flow Tracking

```c
void cleanup(State* s) {
    log("Func:cleanup");
    if(verbose)
        log("Cleaning up %p", s);
    delete s;
}

void display_size(State* s) {
    log("Func:display_size");
    log("State: %u", s->size);
}
```

Watch &s

Segfault
Example: Data Flow Tracking

Thread 1

```c
void display_size(State* s) {
    log("Func:display_size");
    log("State: %u", s->size);
}
```

Thread 2

```c
void cleanup(State* s) {
    log("Func:cleanup");
    if (verbose)
        log("Cleaning up %p", s);
    delete s;
}
```

Success
Example: Data Flow Tracking

```c
void cleanup(State* s) {
    log("Func:cleanup");
    if(verbose)
        log("Cleaning up %p", s);
    delete s;
}

void display_size(State* s) {
    log("Func:display_size");
    log("State: %u", s->size);
}
```

**Success**

**Failure**

```
void cleanup(State* s) {
    log("Func:cleanup");
    if(verbose)
        log("Cleaning up %p", s);
    delete s;
}

void display_size(State* s) {
    log("Func:display_size");
    log("State: %u", s->size);
}```
Outline

• Challenges

• Design

  ![Diagram](image)
  - Static Analysis
  - Control and Data Flow Tracking
  - Statistical Analysis
  - Failure sketch

• Evaluation
Outline

• Challenges
• Design
• Evaluation
Statistical Analysis
Statistical Analysis

- Identification of failure predictors
  - A good predictor portends a failure with high probability
    (e.g., data races, atomicity violations)

\(^1\) Liblit, B. et al. Scalable statistical bug isolation. PLDI 2005
Statistical Analysis

- Identification of failure predictors¹
  - A good predictor portends a failure with high probability (e.g., data races, atomicity violations)
  - Example: data races

¹ Liblit, B. et al. Scalable statistical bug isolation. PLDI 2005
Identification of failure predictors

- A good predictor portends a failure with high probability (e.g., data races, atomicity violations)

- Example: data races

Failure predictors across multiple executions

---

1 Liblit, B. et al. Scalable statistical bug isolation. PLDI 2005
Example: Statistical Analysis

```c
void display_size(State* s) {
    log("Func:display_size");
    log("State: %u", s->size);
}

void cleanup(State* s) {
    log("Func:cleanup");
    if(verbose)
        log("Cleaning up %p", s);
    delete s;
}
```

Success

```c
void cleanup(State* s) {
    log("Func:cleanup");
    if(verbose)
        log("Cleaning up %p", s);
    delete s;
}
```

Failure

```c
void display_size(State* s) {
    log("Func:display_size");
    log("State: %u", s->size);
}
```
Example: Statistical Analysis

```c
void cleanup(State* s) {
    log("Func:cleanup");
    if(verbose)
        log("Cleaning up %p", s);
    delete s;
}

void display_size(State* s) {
    log("Func:display_size");
    log("State: %u", s->size);
}
```

Success

```c
void cleanup(State* s) {
    log("Func:cleanup");
    if(verbose)
        log("Cleaning up %p", s);
    delete s;
}

void display_size(State* s) {
    log("Func:display_size");
    log("State: %u", s->size);
}
```

Success

```c
void cleanup(State* s) {
    log("Func:cleanup");
    if(verbose)
        log("Cleaning up %p", s);
    delete s;
}

void display_size(State* s) {
    log("Func:display_size");
    log("State: %u", s->size);
}
```

Success

```c
void cleanup(State* s) {
    log("Func:cleanup");
    if(verbose)
        log("Cleaning up %p", s);
    delete s;
}

void display_size(State* s) {
    log("Func:display_size");
    log("State: %u", s->size);
}
```

Success

```c
void cleanup(State* s) {
    log("Func:cleanup");
    if(verbose)
        log("Cleaning up %p", s);
    delete s;
}
```

Success

```c
void cleanup(State* s) {
    log("Func:cleanup");
    if(verbose)
        log("Cleaning up %p", s);
    delete s;
}

void display_size(State* s) {
    log("Func:display_size");
    log("State: %u", s->size);
}
```

Success
Example: Statistical Analysis

```c
void cleanup(State* s) {
    log("Func:cleanup");
    if (verbose)
        log("Cleaning up \%p", s);
    delete s;
}

void display_size(State* s) {
    log("State: \%u", s->size);
}
```
Example: Statistical Analysis

```c
void cleanup(State* s) {
    log("Func:cleanup");
    if(verbose)
        log("Cleaning up %p", s);
    delete s;
}

void display_size(State* s) {
    log("State: %u", s->size);
}
```

```c
Failure
```
Example: Statistical Analysis

void cleanup(State* s) {
    log("Func:cleanup");
    if(verbose)
        log("Cleaning up %p", s);
    delete s;
}

void display_size(State* s) {
    log("State: %u", s->size);
}

void cleanup(State* s) {
    log("Func:cleanup");
    if(verbose)
        log("Cleaning up %p", s);
    delete s;
}

void display_size(State* s) {
    log("State: %u", s->size);
}

Success

void cleanup(State* s) {
    log("Func:cleanup");
    if(verbose)
        log("Cleaning up %p", s);
    delete s;
}

void display_size(State* s) {
    log("State: %u", s->size);
}

Failure

void cleanup(State* s) {
    log("Func:cleanup");
    if(verbose)
        log("Cleaning up %p", s);
    delete s;
}

void display_size(State* s) {
    log("State: %u", s->size);
}

Success

void cleanup(State* s) {
    log("Func:cleanup");
    if(verbose)
        log("Cleaning up %p", s);
    delete s;
}

void display_size(State* s) {
    log("State: %u", s->size);
}

Success

void cleanup(State* s) {
    log("Func:cleanup");
    if(verbose)
        log("Cleaning up %p", s);
    delete s;
}

void display_size(State* s) {
    log("State: %u", s->size);
}

Success

void cleanup(State* s) {
    log("Func:cleanup");
    if(verbose)
        log("Cleaning up %p", s);
    delete s;
}

void display_size(State* s) {
    log("State: %u", s->size);
}

Success

Static analysis + cooperative dynamic analysis
Outline

• Challenges

• Design

• Evaluation
Outline

• Challenges

• Design

• Evaluation
  
  • Does Gist help developers do root cause diagnosis?
  
  • Is Gist efficient?

  • Is Gist accurate?
Experimental Setup

• Client side executions are analyzed in the lab
• Real world server and desktop programs
Do Failure Sketches Help Developers?

• We manually analyzed the usefulness of Gist for 11 failures

• Gist-identified failure predictors point to root causes
  • *Developers eliminated those root causes to fix the bugs*
  • *Average number of statements to look at: 7*
Do Failure Sketches Help Developers?

- We manually analyzed the usefulness of Gist for 11 failures
- Gist-identified failure predictors point to root causes
  - *Developers eliminated those root causes to fix the bugs*
  - *Average number of statements to look at: 7*
Efficiency

(Control & data flow tracking)
Gist has low average overhead (always below 5%)
Accuracy

Accuracy [%]

Apache-1
Apache-2
Apache-3
Apache-4
Cppcheck-1
Cppcheck-2
Curl
Transmission
SQLite
memcached
Pbzip2
Accuracy

- Static slicing
- Control flow tracking
- Data flow tracking

Accuracy [%]

- Apache-1
- Apache-2
- Apache-3
- Apache-4
- Cppcheck-1
- Cppcheck-2
- Curl
- Transmission
- SQLite
- memcached
- Pbzip2
Accuracy

- Static slicing
- Control flow tracking
- Data flow tracking

Average accuracy is 96%
Accuracy

Each technique is needed for accuracy

Average accuracy is 96%
Conclusion

• Failure sketching
  • Combination of static and dynamic program analysis
  • Failure sketches are summaries explaining failure root causes
  • Accurate, efficient, improves developer productivity

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

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

- Failure sketching
  - Combination of static and dynamic program analysis
  - Failure sketches are summaries explaining failure root causes
  - Accurate, efficient, improves developer productivity

http://dslab.epfl.ch/proj/gist

Baris  Ben  Cristiano  Gilles  George