Reverse execution with persistent data structures

Omar Roth

Advisor: Joseph Hollingsworth

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Abstract/Introduction

Reversible debuggers are useful tools for developing and deploying modern applications. However, due to their high memory requirements and runtime overhead, their functionality is generally reserved for rare cases (e.g., identifying short-term memory corruption). This paper describes an alternative approach for implementing a low-overhead memory snapshotting mechanism using fully persistent data structures. Memory usage and performance analyses will be presented and compared against alternative implementations.

Background/References

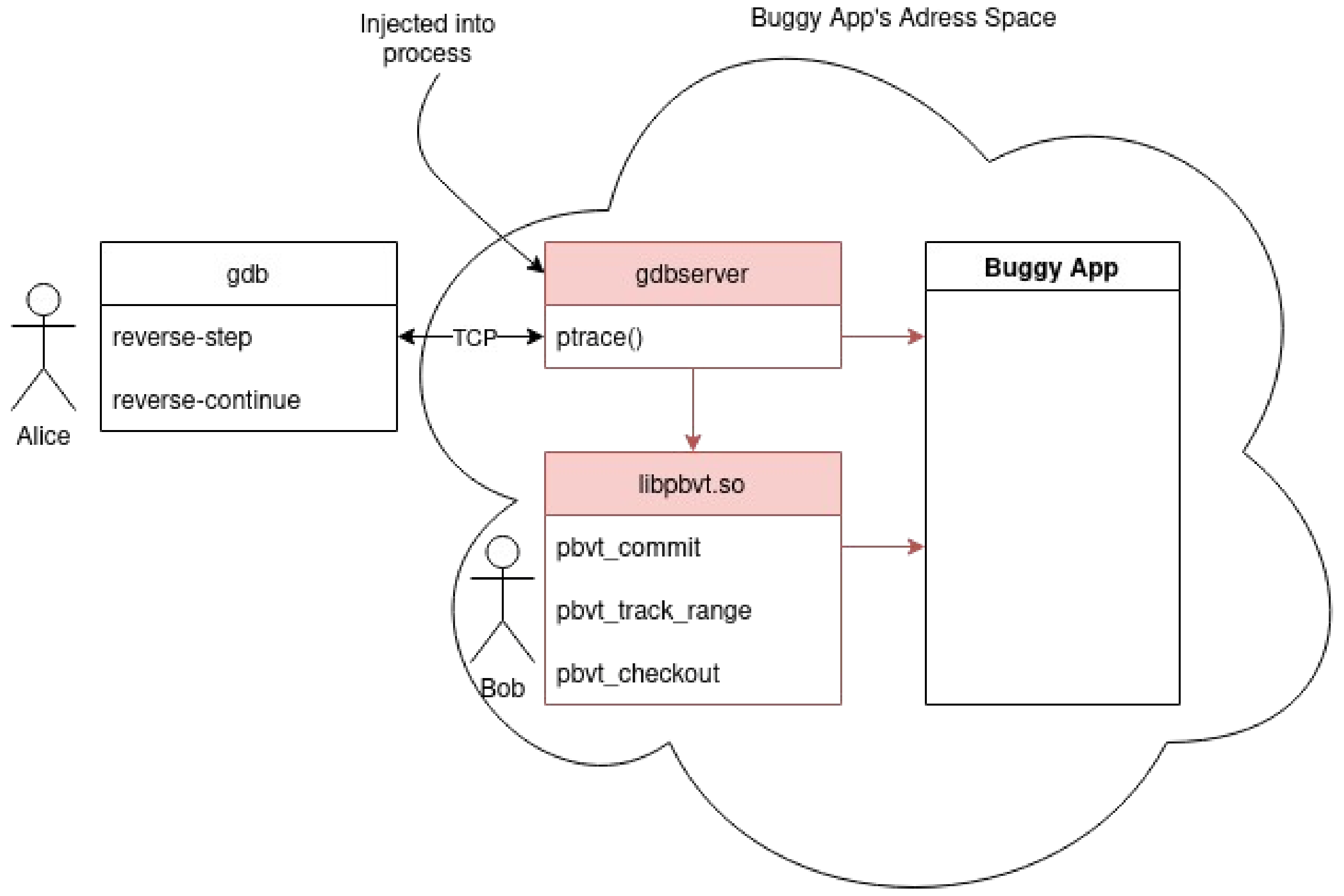
- Fully persistent data structure: Immutable data structure that returns a copy of itself when modified. All copies retain the same algorithmic complexity guarantees (i.e., older versions don't get slower).
- Fully reversible execution: Execution of a program that can move forwards and backwards, can be modified in a previous state, and then run forward (supports divergent re-execution).

Goal(s)/RQs

1. Can overhead for Alice be pushed down to ~20% of wall-clock time while still supporting fully reversible execution for unmodified multi-threaded userspace applications?
2. Can we support fully reversible execution for Alice with less than ~20% space overhead (as compared to uninstrumented execution) for long-running applications?

Methods

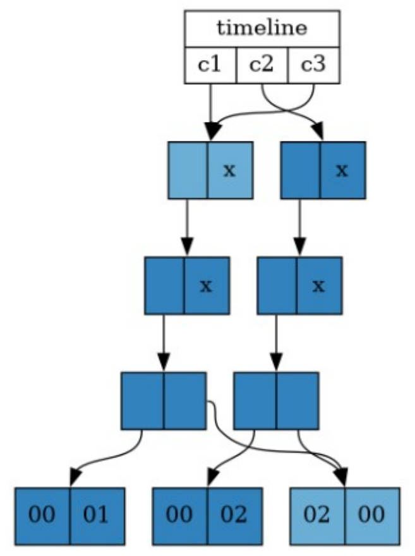
- Created library for tracking memory, provides "version control" for arbitrary memory regions
- Created debugger that injects into existing application, exposes stub that connects to GDB



- Data structure of choice is persistent bit-partitioned hash trie with hash-consing. This allows us to identify duplicate sub-tries, and provies O(log32(n)) updates.
- Writes to memory are tracked through page-faults, "compacted" and set to a transient (writable) state until the next commit.

```
char myArr[0x100];
myArr[0] = 0;
myArr[1] = 1;
myArr[2] = 2;
state_commit(); // c1
myArr[1] = 2;
state_commit(); // c2
myArr[1] = 1;
state_commit(); // c3
```

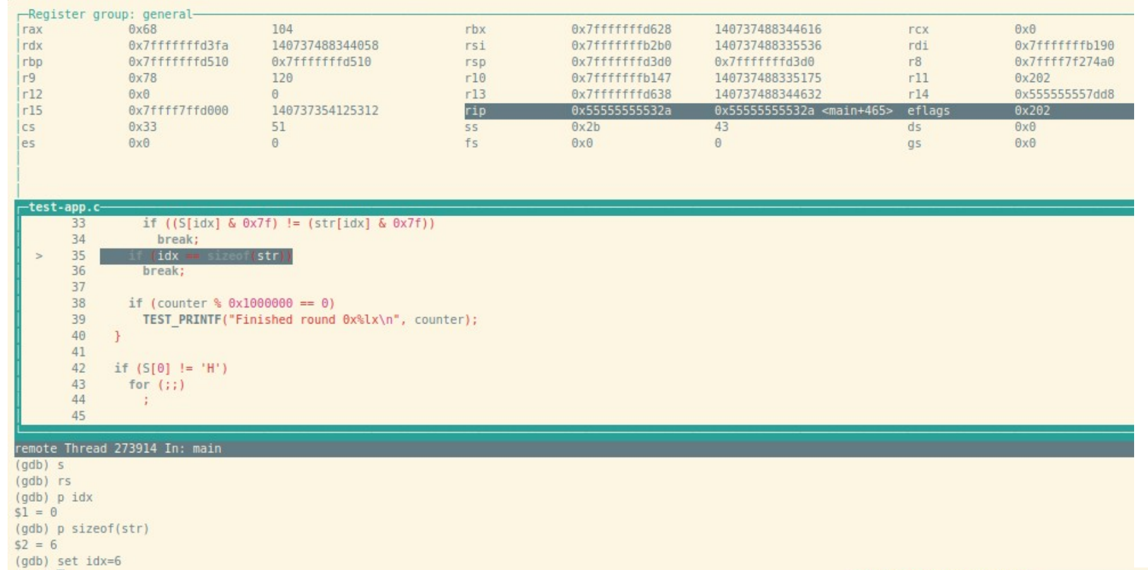
(a) Sample listing of memory writes, where $c_1 = c_3$.



(b) Our implementation recognizes duplicate subtrees and can compact them up to the root.

Results

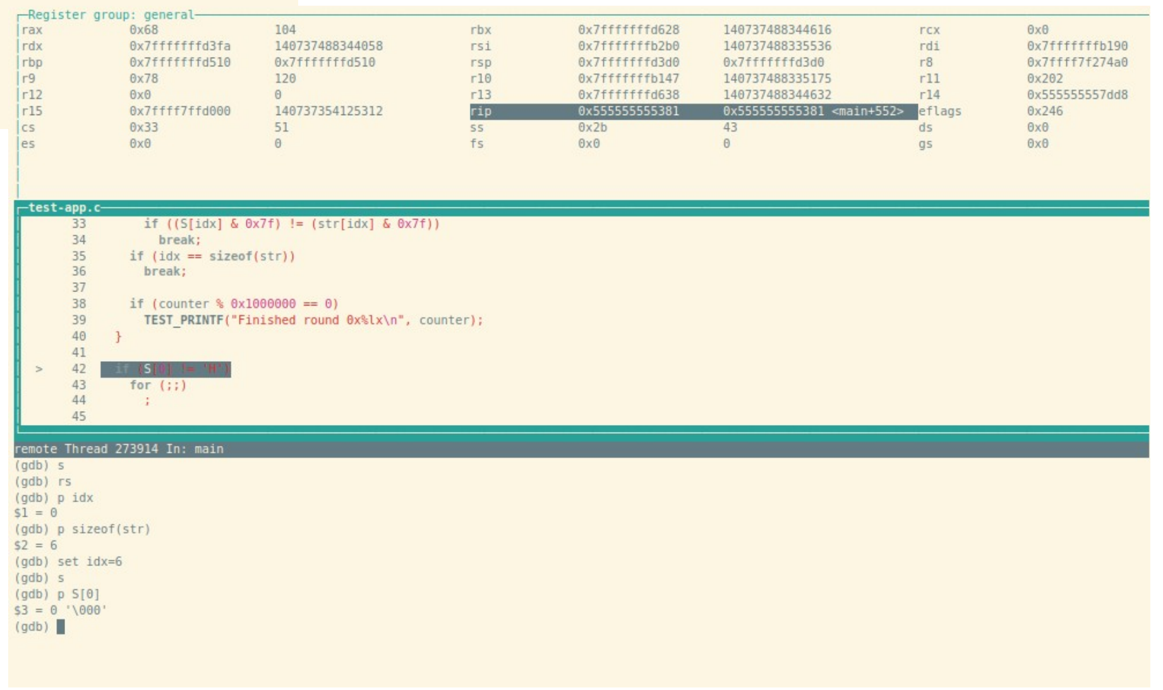
- Our debugger currently supports reverse breakpoints, reverse-step, reverse-continue, as well as normal forward debugging supported by GDB
- After reverse-step, the state of the program can be modified and re-executed.
- Performance results are forthcoming



```
┌Register group: general──────────────────────────────────────────────────────────────
│rax            0x68               104                rbx            0x7fffffffd628     140737488344616    rcx            0x0
│rdx            0x7fffffffd3fa     140737488344058    rsi            0x7ffffffb2b0      140737488335536    rdi            0x7ffffffb190
│rbp            0x7fffffffd510     0x7fffffffd510     rsp            0x7fffffffd3d0     0x7fffffffd3d0     r8             0x7ffff7f274a0
│r9             0x78               120                r10            0x7ffffffb147      140737488335175    r11            0x202
│r12            0x0                0                  r13            0x7fffffffd638     140737488344632    r14            0x555555557dd8
│r15            0x7ffff7ffd000     140737354125312    rip            0x55555555532a     0x55555555532a <main+465>  eflags  0x202
│cs             0x33               51                 ss             0x2b               43                 ds             0x0
│es             0x0                0                  fs             0x0                0                  gs             0x0

┌test-app.c─────────────────────────────────────────────────────────────────────────────
│          if ((S[idx] & 0x7f) != (str[idx] & 0x7f))
│            break;
│  >     if (idx == sizeof(str))
│          break;
│
│        if (counter % 0x1000000 == 0)
│          TEST_PRINTF("Finished round 0x%lx\n", counter);
│      }
│
│      if (S[0] != 'H')
│        for (;;)
│          ;
│
remote Thread 273914 In: main
(gdb) s
(gdb) rs
(gdb) p idx
$1 = 0
(gdb) p sizeof(str)
$2 = 6
(gdb) set idx=6
(gdb) █
```

After reverse-stepping, we can change state and continue



```
┌Register group: general──────────────────────────────────────────────────────────────
│rax            0x68               104                rbx            0x7fffffffd628     140737488344616    rcx            0x0
│rdx            0x7fffffffd3fa     140737488344058    rsi            0x7ffffffb2b0      140737488335536    rdi            0x7ffffffb190
│rbp            0x7fffffffd510     0x7fffffffd510     rsp            0x7fffffffd3d0     0x7fffffffd3d0     r8             0x7ffff7f274a0
│r9             0x78               120                r10            0x7ffffffb147      140737488335175    r11            0x202
│r12            0x0                0                  r13            0x7fffffffd638     140737488344632    r14            0x555555557dd8
│r15            0x7ffff7ffd000     140737354125312    rip            0x555555555381     0x555555555381 <main+552>  eflags  0x246
│cs             0x33               51                 ss             0x2b               43                 ds             0x0
│es             0x0                0                  fs             0x0                0                  gs             0x0

┌test-app.c─────────────────────────────────────────────────────────────────────────────
│          if ((S[idx] & 0x7f) != (str[idx] & 0x7f))
│            break;
│        if (idx == sizeof(str))
│          break;
│
│        if (counter % 0x1000000 == 0)
│          TEST_PRINTF("Finished round 0x%lx\n", counter);
│      }
│
│  >   if (S[0] != 'H')
│        for (;;)
│          ;
│
remote Thread 273914 In: main
(gdb) s
(gdb) rs
(gdb) p idx
$1 = 0
(gdb) p sizeof(str)
$2 = 6
(gdb) set idx=6
(gdb) s
(gdb) p S[0]
$3 = 0 '\000'
(gdb) █
```

Future Work/Acknowledgments

- Extending our debugger to support multithreaded applications
- Can our library be used to implement mult-shot continuations, transactional memory, backtracking search?
- Integrate into QEMU: Would provide full-system reversible debugging