USCOPE: A SCALABLE UNIFIED TRACER FROM KERNEL TO USER SPACE

Junghwan Rhee, Hui Zhang, Nipun Arora, Guofei Jiang, Kenji Yoshihira

NEC Laboratories America

www.nec-labs.com
Motivation

• Complex IT services face diverse functional and non-functional issues due to complexity of software and usage of underlying components.

• OS kernel event tracing is a convenient method to monitor and debug system operations without hard dependency on application layers (e.g., Libraries, program binaries).
  • Example: System call trace

• However, OS events can be triggered by diverse programs and code. Therefore there is semantic gap to understand application program behavior from OS events.
Unified Tracing

- Trace logs across the boundary of kernel and user space
  - Examples: Dtrace, Windows ETW, System Tap

- Two types of Unified Tracing

<table>
<thead>
<tr>
<th>Trigger</th>
<th>Traced Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type 1</td>
<td>User code</td>
</tr>
<tr>
<td></td>
<td>Trace the execution of known user code</td>
</tr>
<tr>
<td>Type 2</td>
<td>Kernel code</td>
</tr>
<tr>
<td></td>
<td>Trace unknown user code triggering kernel code</td>
</tr>
</tbody>
</table>
Type 2 Unified Tracing

- Service problems can be caused from any program/layers. Type 2 unified tracers can cover such unknown cases.
- A typically used technique to collect user space code information is stack walking.
- Tracer finds the user process stack in the current context and scan stack frames from the stack pointer address.
- Examples
  - Ustack of Dtrace, Stackwalking of Microsoft ETW
- These solutions have been generally used for debugging scenarios. How we can lower overhead?
Challenges: Tracing Focus

- **Tracing all programs?**

  Figure: A hierarchy of live processes in an idle desktop machine

- There are numerous processes in typical desktop and server systems at runtime for various purposes (e.g., multitasking, administration, accounting, updating software, users’ daemons).
- Unless the user does not know which program to diagnose, tracing all processes is not ideal.
Challenges: Tracing Focus

- **Tracing an application software?**

  Child of the Apache Controller becomes the Apache daemon.

  Apache daemon forks children on demand.

- Programs create and kill many sub processes dynamically.
- Some processes change their identity (execve system calls).
- How to systematically track all processes from their start? (instead of giving PIDs to tracers)
Challenges: Tracing Focus

- **Tracing the whole stack?**
  - Programs may have deep stacks. ECLIPSE project reported that the collected stack trace ranged from 1 ~ 1024 stack frames.
  - A stack includes function call information of multiple software layers (programs, libraries, middleware, and kernel etc.)
  - Not every stack layer may be in users’ interest.
Uscope: Systematic Unified Tracer

- **Flexible and configurable tracing scopes**
  - Efficient per-application tracing
  - Systematic tracking of dynamic processes
  - A highly configurable focus within the call stack
Uscope Architecture

**Input:**
1. Kernel Tracing Target: the kernel events that generate log events
2. User Tracing Target: the application software to be traced
3. Tracing Mode: specification on the call stack focus to be traced

**Output:**
- Unified Trace for the user tracing target
Per-Application Tracing Logic

- This diagram shows the logic how Uscope performs per-application tracing and systematic tracking of dynamic processes.
- Trace map maintains the sets of processes in three states: (1) unknown, (2) to be traced, and (3) not to be traced.
- Kernel events making dynamic changes of processes (e.g., fork, exit, execve) trigger corresponding changes on the trace map.
Flexible Call Stack Scope

- Uscope provides flexible call stack scopes in tracing.
- Maximum budge $S$. Further fine control is available.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Binary config</th>
<th>In-Binary config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mode 1</td>
<td>App binary</td>
<td>The last stack frame</td>
</tr>
<tr>
<td>Mode 2</td>
<td>App binary</td>
<td>All stack frames</td>
</tr>
<tr>
<td>Mode 3</td>
<td>All binaries, libraries</td>
<td>Last stack frames</td>
</tr>
<tr>
<td>Mode 4</td>
<td>All binaries, libraries</td>
<td>All stack frames</td>
</tr>
</tbody>
</table>
Implementation

• Tracer
  • Implemented by extending SystemTap.
  • SystemTap hooks system calls to generate log events.
  • Trace map and tracing logic is implemented as a kernel module.
  • Redhat Enterprise Linux 5 is supported.

• Input:
  1. Kernel Tracing Target : System call events
  2. User Tracing Target : Apache webserver (Server workload), MySQL database (Server workload), Nbench (computation)

• Uscope can be dynamically attached and detached to the kernel at runtime. When it is detached, there is no overhead.
Runtime Overhead

- Workload
  - Apache: Apache HTTP Benchmark tool (ab), 100 concurrency, $10^6$ requests
  - MySQL: MySQL Benchmark suite (alter-table, ATIS, big-tables, connect, create, insert, select, transactions, and wisconsin)
  - Nbench: Linux/Unix of BYTE's Native Mode Benchmarks (version 2.2.3). “Memory Index”, “Integer Index”, and “FP Index” are used.

- Tracing Modes:
  1. Mode 1: application call stack layer, the last stack frame
  2. Mode 2: application call stack layer, 3 or 5 last stack frames
  3. Mode 3: all layers, the last stack frames up to 5

- Less than 6% overhead in three benchmarking cases
Case Study Application 1

- **Testbed**
  - Three tier PetStore system (Apache, Jboss, MySQL)

- **Symptom**
  - Web requests failed.

- **Tracing**: Mode 2 (S=3)

- **Dual Space Analysis**
  - X axis shows different types of system calls and Y axis shows application code (i.e., triggers).
  - Unique events in normal case
    - Read: `my_read`
    - Accept: `handle_connections_socket`
    - More..
  - Unique events in abnormal case
    - **Stat**: `archive_discover`
  - => Problem in accessing the database file
Case Study Application 2

- **Testbed**
  - Apache Webserver

- **Symptom**
  - Concurrency error that threads are in a deadlock condition (Case number: Apache #42031)

- **Tracing**: Mode 2 (S=5)

- **Call Stack Analysis**
  - Call stacks on `futex` system calls are captured and analyzed.
  - Worker Thread
    - `apr_thread_mutex_call` (a wrapper of `pthread_mutex_call`)
  - Listener Thread
    - `apr_thread_cond_wait` (a wrapper of `pthread_cond_wait`)
  - => Deadlock conditions are identified.
Conclusion

- Uscope provides efficient type 2 unified tracing for kernel and unknown user code.
- Uscope provides per-application tracing, systematic tracking of dynamic processes, and flexible specification on call stack scopes to be traced.
- Our prototype has 6% overhead compared to native execution in several benchmarks.
- Also we showed two case studies illustrating how unified tracers can be used for diagnosing service systems.
Thank you