SystemTap update & overview

Josh Stone <jistone@redhat.com>
Software Engineer, Red Hat

Introduction

- SystemTap: a tool for system-wide instrumentation
- Inspired by Sun DTrace, IBM dprobes, etc.
- GPL license, open project since 2005
- Current release 1.4, for kernels 2.6.9 ... 2.6.37+
  - Release 1.5 coming Real Soon Now™

http://sourceware.org/systemtap

Coming up:

- Overview of SystemTap
- Development update

System-wide instrumentation

- The most general case:
  - Look into a live, unmodified system
  - Examine what's going on
  - Take action as appropriate
  - Operate in the background

- Tracing, Debugging, Manipulation

Version flexibility

- Heterogeneous computer network
  - different versions of the OS and/or applications
- Patching or upgrading not always practical
- Sometimes need a tool that works across the spectrum
- SystemTap has several mechanisms to adapt/abstract

Example usage scenarios

- **Anyone**: simple tracing
- **Developers**: to debug or comprehend code
  - stepping through code, pretty-printing variables
- **Analysts**: to measure performance
  - measure elapsed time between events
  - attribute statistics to processes
- **Sysadmins**: to monitor, to patch
  - activity logging, constraining
  - security band-aids
  - remote diagnostics (tech. support)

Developer: monitoring statements & vars

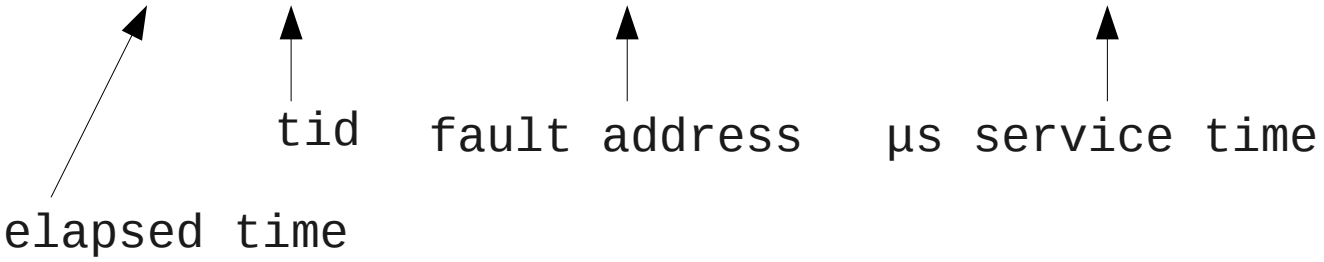
- # stap .../examples/general/varwatch.stp \
  'kernel.statement("do_sys_open@fs/open.c:*")' '$$vars'

```
open.c:1045 ... $$vars ... thread 9541 from  to
dfd=0xff...ff9c filename=0x3b...bb1 ...
open.c:1049 ... $$vars ... thread 9541 from ... to
dfd=0xff...ff9c filename=? flags=0x8000 mode=0x1 tmp=? fd=?
open.c:1047 ... $$vars ... thread 9541 from ... to
dfd=0xff...ff9c filename=? flags=0x8000 mode=0x1
tmp=0xffff8803c8d0a000 fd=0xffffffffc8d0a000
open.c:1052 ... $$vars ... thread 9541 ...
open.c:1057 ... $$vars ... thread 9541 ...
open.c:1058 ... $$vars ... thread 9541 ...
open.c:1061 ... $$vars ... thread 9541 ...
open.c:1045 ... $$vars ... thread 9541 ...
open.c:1049 ... $$vars ... thread 9541 ...
```


Sysadmin: page faults

- # stap ... examples/memory/pfaults.stp

```
13927:10843:0x7ffffffefec:w:minor:16
14106:10843:0x7ffff822ed29:w:minor:3
14193:10843:0x3b0e81f0d0:w:minor:4
14643:10843:0x607348:r:major:418
14655:10843:0x7ffff8359038:r:minor:2
14683:10843:0x2b6bf2ea0018:w:minor:9
15250:10843:0x7ffff822a74c:w:minor:2
24565:10843:0x7ffff822b77f:w:minor:4
24575:10843:0x7ffff822c77f:w:minor:3
60976:10843:0x2b6bf2f20270:r:major:21323
83819:10843:0x7ffff8229ed8:w:minor:4
83866:10843:0x2b6bf8d65000:w:minor:3
```



Sysadmin: monitoring ttys

- # stap ... examples/io/ttyspy.stp

```
(maj,min, pgrp,  uid)
(128,  1,    0,   99) \244\263\377}#\300!})\314} }5} }(\352d\\:i\353
(136,  8, 8331,  500) ls -al\necho hello world\n
(128,  8,    0,    0) Nov 23  2002 \033[01;34m.netscape6\033[0m\rd\be
(128,  2,    0,    0) \033[1;1H\033[J(maj,min, pgrp,  uid)\r\n(128,
```


Conceptual model

- probe points: “events when to do something”
- probe handlers: “what to do then”
- script: a collection of probe points & handlers, plus utility functions
- many scripts can run concurrently, independently

Probe points – low level

- Provide an operational definition of the events:
  - kernel.function(“vfs_*”)
  - process(“a.out”).function(“*”).return
  - module(“foo”).statement(“*@file.c:2323”)
  - kernel.trace(“timer_*”)
  - timer.s(1)
  - perf.type(0).config(3).sample(2000)
- and context variables to probe handlers
  - $arg4, $ptr->field[5], $$vars

Probe points – high level

- Defined as aliases in the standard tapset library
  - syscall.open = kernel.function(“sys_open”) { ... }
  - perf.hw.bus_cycles
  - hotspot.thread_start
  - python.function.entry
- Provide salient values to probe handlers
- Wildcards, metavariables widely available
- Also support add-on tapsets

Utility functions

- also defined in standard tapset library
- provide information not specific to context of probe point
  - tid(), cpu(), get_cycles(), execname(): obvious
  - tz_ctime: formatted timestamp in local time zone
  - indent: formatting aid for per-thread nested reports
  - backtrace, ubacktrace: unwound stack frames
  - symdata, usymdata: symbol table lookup by address

Probe handlers: where magic happens

- Variables and metavariables available from context of probe point
- Developer chooses:
  - trace some values
  - filter, summarize, aggregate
  - or traverse data structures
  - or collect statistics via global variables
  - or compose report
  - or change state (guru mode)

Probe handlers

- small safe domain-specific language
- loops, conditionals, functions
- inferred strong data typing for temporary and context variables
- arrays, global variables
- structured error handling (cleanup, try/catch)
- automatic concurrency protection
- automatic resource limits (time & space)
- optional escape hatch from safety constraints

Sample script fragments

```
probe begin { printf("hello world\n") }   # say hello

function gtod() { return gettimeofday_us() } # helper

probe syscall.*.return {      # after every syscall
  errno = $return             # check return value
  if (errno < 0) {            # if it's negative
  e = gtod()-@entry(gtod())   # measure time
                              # print a line
  printf("tid %d %s errno %d %s after %d us \n",
         tid(), name, errno, errno_str(errno), elapsed)
}

probe syscall.ptrace {        # every syscall
  if (target()==pid()) {      # if this is the target
                              # print a line
    printf("noptrace(%s) from pid %s\n",argstr,pid())
    $request=0xbeef           # clobber code
  }
}
```


Example scripts

- ~80 packaged along with SystemTap
- Demonstrate common uses and unusual techniques
- Starting point for new users

http://sourceware.org/systemtap/examples

Current implementation

- Compile: (local or remote)
  - translate script to constrained C code
  - compile into a kernel module using ordinary system compiler
- Run: (local or remote)
  - loads module
  - attach to kernel instrumentation callbacks (kprobes, perf, uprobes, ...)
  - at conclusion, detach, unload, clean up

Perhaps SystemTap is not for you if ...

- If offline data analysis is good enough, and ...
- perf? ftrace? kernelshark?
  - if you only run recent upstream kernels
  - for relatively simple kernel-only tracing/analysis
  - for easiest deployment
- lttng?
  - if you can run patched kernels
  - if you need high performance bulk tracing
- holy grail – a single all-purpose tool?
  - convergence not impending

SystemTap release history

- First release with RHEL4U2 (October 2005)
- Recent release 1.4, January 2011

- It still works with RHEL4
  - and RHEL5, RHEL6, Fedoras
  - and several distributions (suse, debian and derivatives)
  - and many upstream kernels

Recent SystemTap releases

- 1.2 March 22, 2010
  - Support for perf events (notably PMU)
  - Support for hardware breakpoints
  - New syntax: @defined(), try-catch

- 1.3 July 21, 2010
  - NOP optimization for uprobes
  - Improved backtracing
  - Integrated compile-server client
  - New syntax: @entry(), C-expr, var pretty-printing

Recent SystemTap releases (2)

- 1.4 January 17, 2011
  - SDT v3 (fewer relocations, better args, no DWARF req)
  - Other userspace-focused improvements
  - Prototype remote execution
  - New policy for deprecation & compatibility

- 1.5 (impending)
  - Better remoting – more robust; multiple hosts
  - Improved compile-server
  - Powerful new option: --version

Userspace probing

- Probe processes & shared libraries
- System-wide or focused
- Major support for C, C++, Java
- Limited support for Perl, Python, TCL

- Out-of-tree uprobes module, based on utrace
- Upstream utrace-free uprobes getting closer...
  - Zeno's paradox resolved?

Compilation servers & unprivileged users

- Compilation server
  - Centralize the analysis dependencies (e.g. debuginfo)
  - Share cached results

- Unprivileged mode
  - May only probe one's own processes
  - Restricted functionality and memory access
  - Module must be server-compiled and signed
    - Trust database is root-managed

Remote execution

- Specify where to run, --remote user@host
  - Target is connected and identified via ssh
  - Script module is prepared locally
  - Module is transferred to the remote and loaded
  - Output is collected back home

- Fan out for fun and profit!
  - Specify multiple hosts to target
  - Heterogeneity is seemless

Statically Defined Tracing (SDT)

- Probe points compiled into userspace applications
- Source-compatible with DTrace SDT

  DTRACE_PROBE2 (provider, name, arg1, arg2)

- SystemTap's ABI:
  - 1-byte NOP at probe site
  - .note.stapsdt section for metadata
  - “Semaphore” variable for predicating probe setup

- Collaborative use desired, in API if not also ABI

SDT-enabled packages in Fedora rawhide

- gcc – libgcc unwind
- glibc – setjmp/longjmp, pthreads (pending)
- glib2 – memory allocation, gobject lifetime, signals
- libmemcached – start/end on many operations
- libvirt – client connect/auth
- MySQL – start/done on many operations
- PostgreSQL – start/done on many operations

SDT-enabled packages in Fedora rawhide (2)

- OpenJDK – method entry/exit, VM, JNI
  - Also backtracing support at runtime
- Perl – sub entry/return
- Python – function entry/return
- TCL – cmd/proc entry/return, object lifetime

- SystemTap – start/end of each pass, caching, child processes

GDB support for static probes

- Patchset enabling GDB to use SDT probes: http://sourceware.org/ml/gdb-patches/2011-04/msg00036.html
- Users can set breakpoints, access arguments

  (gdb) break probe:objfile:provider:name

  (gdb) print $_probe_arg0

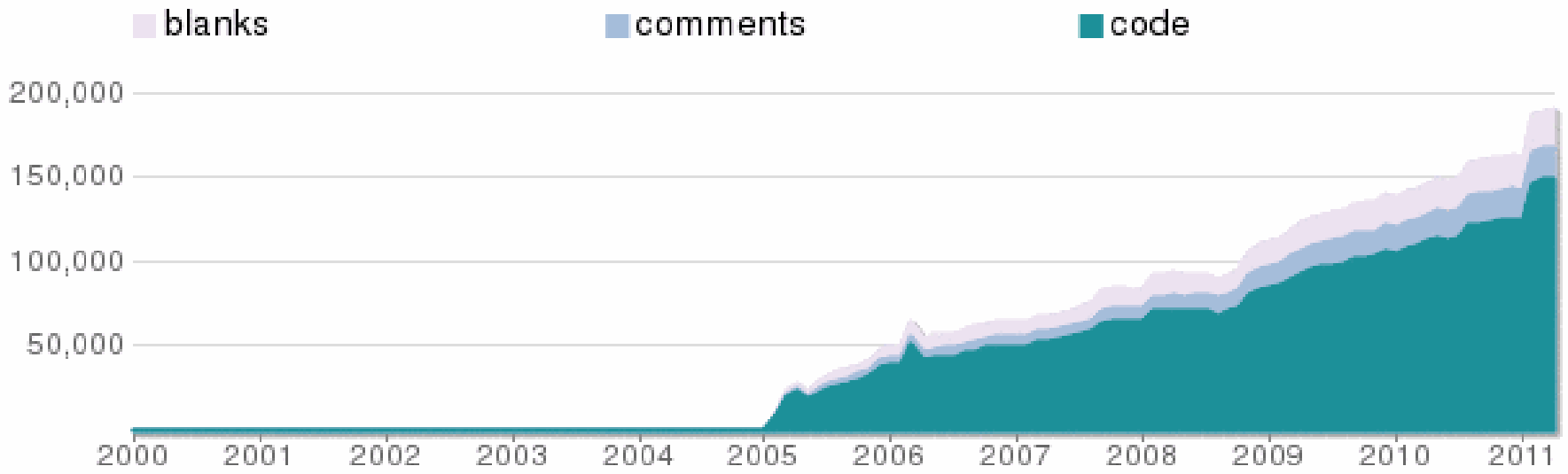
- GDB will use SDT internally as well
  - libgcc “unwind” for exception handling
  - glibc setjmp/longjmp
  - Avoids the need for gcc/glibc debuginfo

Possible futures

- Integration with network monitoring tools like PCP (performance co-pilot)
- Further integration with gdb
- Deeper java support
- Performance improvements
- Support for Android targets
- New backend not based on kernel modules
- Internationalized messages

Development numbers

- 79 contributors, 10 companies
- 3-4 releases per year
- Code growth, according to ohloh:



Conclusions

- Project on steady trajectory
- Unique combination of features
- Tool of choice for complex kernel+userspace instrumentation
- Suitable for many simple tracing tasks
- wiki, mailing lists, bugs at:

  http://sourceware.org/systemtap

instrumentation for the unforeseen

Extracurricular

- SystemTap Games by Masami Hiramatsu

  http://sourceforge.net/projects/stapgames/

  - Tetris, Minesweeper, Live, and more!

- BOFH meets SystemTap by Adrien Kunysz

  http://stapbofh.krunch.be/

  - Who says users deserve respect...