# **systemtap** update & overview

Frank Ch. Eigler
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+



# systemtap talk outline

- what it's for
- how it works
- other tools to consider
- development community



# 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



# version flexibility

- heterogeneous computer network
- running 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



# special case usage scenarios

- by anyone: simple tracing
- by developers: to debug or comprehend code
  - stepping through code, pretty-printing variables
- by analysts: to measure performance
  - measure elapsed time between events
  - attribute statistics to processes
- by sysadmins: to monitor, to patch
  - activity logging, constraining
  - security band-aids
  - remote diagnostics (tech. support)



# developer usage: monitoring statements & vars

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

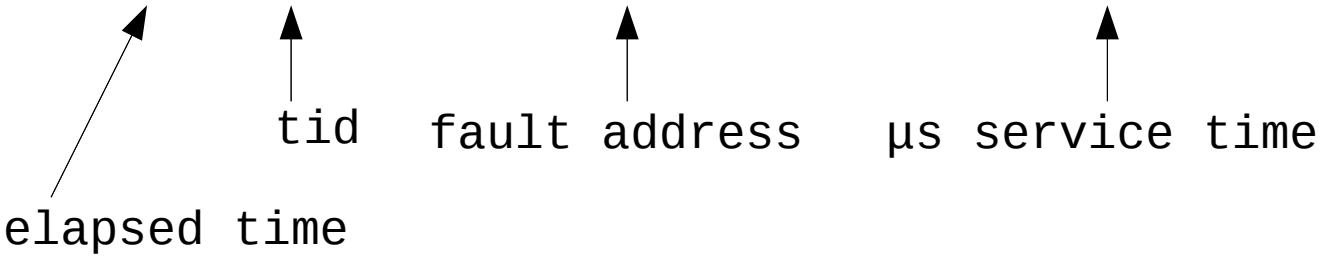
```
open.c:1045 ... $$vars ... thread 9541 from  to
dfd=0xff...ff9c filename=0x3b...bb1 ...
open.c:1049 ... $$vars ... thread 9541 from ... to
dfd=0xff...ff9c filename=? flags=0x8000 mode=0x1 tmp=? fd=?
open.c:1047 ... $$vars ... thread 9541 from ... to
dfd=0xff...ff9c filename=? flags=0x8000 mode=0x1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

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

```
13927:10843:0x7fffffffefec: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 usage: 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,
```



# how it works: 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



# 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



# 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



# 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
  }
}
```



# how it works: 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



# example scripts

- ~80 packaged along with systemtap
- demonstrate common uses and unusual techniques
- a starting point for new users
- http://sourceware.org/systemtap/examples



# perhaps systemtap is not for you if ...

- if offline data analysis is good enough, and ...
- perf?
  - 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 super high performance bulk tracing
- holy grail – a single all-purpose tool?
  - convergence not impending



# 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 work 1

- focus on user-space instrumentation
  - probing processes, shared libraries
  - unwinding-based backtraces
  - java, c++
  - 1-nop-overhead <sys/sdt.h>
  - requires kernel patch utrace
- unprivileged users
  - limited to probing one's own processes
- lots of documentation



# recent work 2

- remote script compilation
  - to centralize installation prerequisites (debuginfo, compilers)
  - to share caches
- remote script execution
  - based on ssh
- event sources shared with perf
  - tracepoints, hardware performance counters
- debuginfo-less probing



# possible future work

- integration with network monitoring tools like PCP (performance co-pilot)
- integration with gdb
- deeper java support
- performance improvements
- android
- new backend not based on kernel modules
- internationalized messages



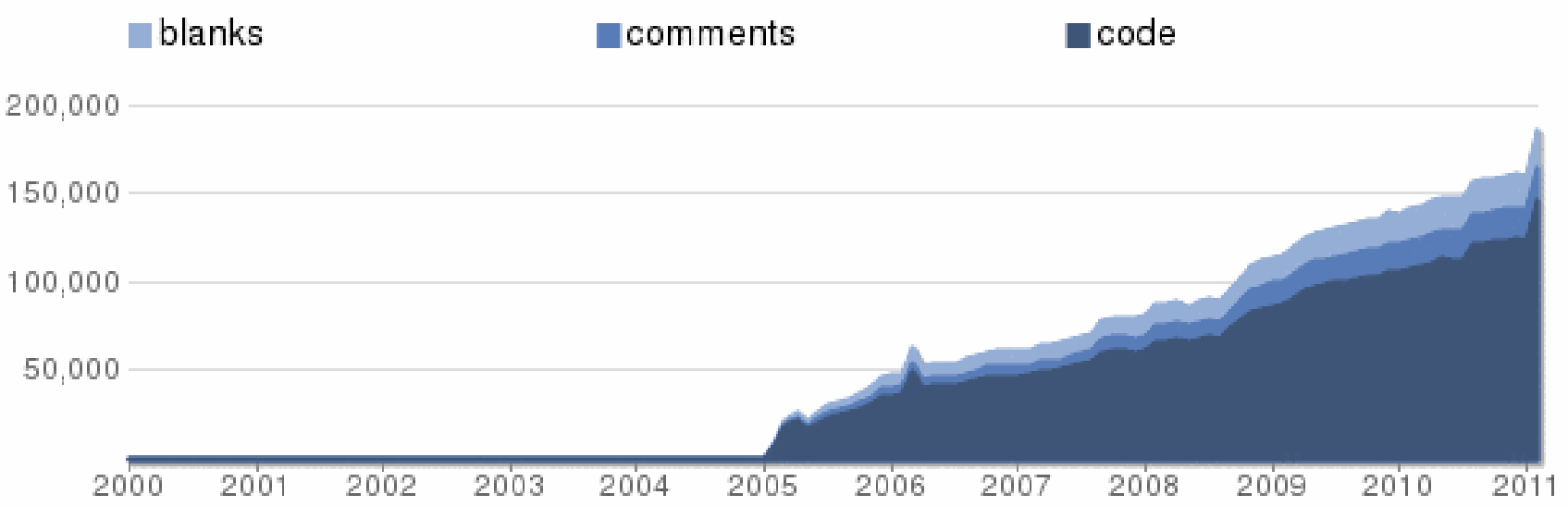
# community outreach

- ongoing porting for upstream kernels
- dwarf debuginfo quality improvements (gcc 4.5 VTA)
- dwarf debuginfo compression (elfutils)
- lkml upstreaming uprobes (IBM)
- widespread dtrace/systemtap <sys/sdt.h> instrumentation



# development numbers

- over history, 77 contributors, 10 companies
- 3-4 releases a 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