Systemtap times

April 2009

Frank Ch. Eigler <fche@redhat.com>
systemtap lead

why trace/probe

- to monitor future
  - background monitoring, flight recording
  - programmed response
- to debug present
  - symbolic, source-level exploration
  - unforseen problems
- to analyze past
  - collect traces
  - analyze dumps

rich capabilities

- system-wide (kernel + userspace) programmable tracing/probing
- compatible with a wide range of kernels, distributions
- operates on live system, no patch/reconfigure/recompile/reboot
- measure time, access any data, explore control flow, correlate events, inject faults
- integrated access to multiple tracing facilities

consider alternatives

- ftrace
  - hard-coded, kernel-only, single-user
  - we share instrumentation hooks, some infrastructure
- ksplice
  - unprotected, kernel-only, x86
  - maybe let's share code recompilation process
- dtrace
  - not available on linux
  - we share ambitions

examples

- http://sourceware.org/systemtap/examples/
- http://sourceware.org/systemtap/wiki/WarStories
- ordinary
  - log events, filtered + correlated + summarized
  - call graphs with variables
  - measure times/values, indexed by anything
  - graph cpu/net/disk utilization, act upon thresholds
- esoteric
  - kernel-enforced file naming policy filters
  - security bug band-aids

operation part 1

- compile probe script foo.stp:
  - parse script
  - combine it with tapset (library of scripts by experts)
  - combine it with debugging information, probe catalogues, event source metadata
  - generate C code with safety checks
  - compile into kernel module with kbuild
  - result: vanilla kernel module

operation part 2

- run probe module foo.ko:
  - load into kernel
  - detach (flight-recorder mode) or consume trace live
  - unload
- probe module may be cached, reused, shared with other machines running same kernel
- sysadmins can authorize others to run precompiled modules

the “upstream” question

- but it already works on your machine
  - not a driver; not a filesystem
  - uses vanilla module APIs
  - a little like X.org or glibc or kgdb
  - or even latencytop ... but with ~no kernel prereqs
- has large userspace component
- few novel kernel-side fixed pieces with likely non-stap in-kernel usage
  - some have been & more will be submitted

community: inward

- contributors: dozens per release
- open project since inception
- user groups: university students, sysadmins, support engineers, kernel developers, userspace developers, data center customers
- distributions shipping systemtap: rhel, debian, fedora, suse, ubuntu, windows, mandriva, maemo, solaris, oracle, gentoo, centos, ...

community: outward

- OLS presence since 2005, regular LKML presence since 2006
- responding to kernel developer requests
  - kernel build tree targeting
  - debuginfo-less operation
  - http://sourceware.org/systemtap/wiki/Myths
- promote kernel “dual use” technologies
  - markers, tracepoints, kprobes, relayfs
  - utrace merging goalposts
  - motivating tracing area

debuginfo

- bountiful gcc byproduct
- ease his pain:
  - on-the-fly debuginfo generation, compression
  - remote compilation server
  - but: is it faster to repeatedly recompile w/ printk?
- they will come:
  - statement-level, source-level symbolic access
  - local variables, arbitrary expressions
  - full type information
- but still “go some distance” without it

recent developments

- probing user-space programs
- attaching to user + kernel markers, tracepoints
- organizing more samples, documentation
- easing deployment: compile server
- easing usability by kernel developers: testing linux-next etc., kernel trees
- better error messages

kernel markers/tracepoints

- statically compiled into kernel/programs
- supplements dynamic instrumentation
- higher performance, reliable data
- shared hook sites between tracing tools
- programmable handling of events

user-space probing

- finally, system-wide, seamless, symbolic
- based upon dwarf debugging data (gcc -g)
- dynamically instrument binaries, shared libraries, potentially at the statement level
- easily trace variables
- attach to sys/sdt.h dtrace markers too, as compiled into postgres, java, ...

user-space probing

- measure average dbms query execution times

```
function time() { return gettimeofday_us() }
probe process("psql").function("SendQuery").call
{
      entry[tid()]=time()
}
probe process("psql").function("SendQuery").return
{
      tid=tid()
      if (! ([tid] in entry)) next
      query=user_string($query)
      queries[query] <<< time() - entry[tid]
      delete entry[tid]
}
/* and an “end” probe to format report */
```

user-space probing

```
probe end,error,timer.s(5) {
      foreach ([q] in queries limit 1)
        { any = 1 }
      if (any) {
        printf("%2s %6s %-40s\n",
               "#", "uS", "query");
        foreach ([q] in queries- limit 10)
          printf("%2d %6d %-40s\n",
                 @count(queries[q]),
                 @avg(queries[q]), q)
        printf("\n");
        delete queries
      }
}
```

user-space probing

```
 #     uS query
12    990 DELETE FROM num_result;
 6   3909 COMMIT TRANSACTION;
 6    132 BEGIN TRANSACTION;
 6    143 SELECT date '1999-01-08';
 4   3651 insert into toasttest
values(decode(repeat('1234567890',10000),'escape'));
 4   3786 insert into toasttest
values(repeat('1234567890',10000));
 4   1218 SELECT '' AS five, * FROM FLOAT8_TBL;
 3    804 END;
 3    295 BEGIN;
 3   1032 INSERT INTO TIMESTAMPTZ_TBL VALUES ('now');
```

under construction

- system-wide backtracing for deep profiling
- java probing & backtracing
- unprivileged user support: “masochism” mode
- more debuginfo-less operation
- gui-controlled integrated general monitoring
- better quality and smaller quantity of debuginfo
- interface to other kernel event sources: perfctr, ftrace, kmmiotrace

samples/documentation

- samples installed, categorized, also online
  - http://sourceware.org/systemtap/examples
- “beginner's guide”
  - http://tinyurl.com/ar8wat
- wiki
  - http://sourceware.org/systemtap/wiki

systemtap

http://sourceware.org/systemtap