Linux Perf Tools

Overview and Current Developments

Arnaldo Carvalho de Melo

Red Hat Inc.

The Linux Foundation Collaboration Summit, San Francisco

April, 2012

Overview

1. Architecture
2. Counting
3. Sampling
4. Code Annotation
5. Dynamic Probes
6. Scripting

Architecture

1. One tool doesn't do it all
2. git like
3. Combine steps to achieve multiple results
4. Allows spreading work flows over multiple machines

List of commands

```
# perf help
The most commonly used perf commands are:
  annotate        Read perf.data (created by perf record) and display annotated code
  archive         Create archive with object files with build-ids found in perf.data file
  bench           General framework for benchmark suites
  buildid-cache   Manage build-id cache.
  buildid-list    List the buildids in a perf.data file
  diff            Read two perf.data files and display the differential profile
  evlist          List the event names in a perf.data file
  inject          Filter to augment the events stream with additional information
  kmem            Tool to trace/measure kernel memory(slab) properties
  kvm             Tool to trace/measure kvm guest os
  list            List all symbolic event types
  lock            Analyze lock events
  probe           Define new dynamic tracepoints
  record          Run a command and record its profile into perf.data
  report          Read perf.data (created by perf record) and display the profile
  sched           Tool to trace/measure scheduler properties (latencies)
  script          Read perf.data (created by perf record) and display trace output
  stat            Run a command and gather performance counter statistics
  test            Runs sanity tests.
  timechart       Tool to visualize total system behavior during a workload
  top             System profiling tool.
```

Counting

1. Most common events counted
2. Supports cgroups
3. Multiplexes counters when more than hw supports is asked for

Default perf stat output

```
# perf stat ls
anaconda-ks.cfg  bin  install.log  install.log.syslog  perf.data  perf.data.old

 Performance counter stats for 'ls':

          2.035241 task-clock                #    0.649 CPUs utilized
                 2 context-switches          #    0.983 K/sec
                 0 CPU-migrations            #    0.000 K/sec
               246 page-faults               #    0.121 M/sec
         1,558,753 cycles                    #    0.766 GHz                      [59.38%]
           971,655 stalled-cycles-frontend   #   62.34% frontend cycles idle     [53.15%]
           684,257 stalled-cycles-backend    #   43.90% backend  cycles idle
         1,105,923 instructions              #    0.71  insns per cycle
                                             #    0.88  stalled cycles per insn
           210,716 branches                  #  103.534 M/sec
            10,632 branch-misses             #    5.05% of all branches

       0.003138057 seconds time elapsed

#
```

Specifying event lists

```
# perf stat -e cycles,cache-misses sleep 2

 Performance counter stats for 'sleep 2':

           790,540 cycles                    #    0.000 GHz
               929 cache-misses

       2.002180353 seconds time elapsed

#
```

List of Suported Events

```
# perf list

List of pre-defined events (to be used in -e):
  cpu-cycles OR cycles                               [Hardware event]
  stalled-cycles-frontend OR idle-cycles-frontend    [Hardware event]
  stalled-cycles-backend OR idle-cycles-backend      [Hardware event]
  instructions                                       [Hardware event]
  cache-references                                   [Hardware event]
  cache-misses                                       [Hardware event]
  branch-instructions OR branches                    [Hardware event]
  branch-misses                                      [Hardware event]
  bus-cycles                                         [Hardware event]
  ref-cycles                                         [Hardware event]

  cpu-clock                                          [Software event]
  task-clock                                         [Software event]
  page-faults OR faults                              [Software event]
  minor-faults                                       [Software event]
  major-faults                                       [Software event]
  context-switches OR cs                             [Software event]
  cpu-migrations OR migrations                       [Software event]
  alignment-faults                                   [Software event]
  emulation-faults                                   [Software event]

  L1-dcache-loads                                    [Hardware cache event]
  L1-dcache-load-misses                              [Hardware cache event]
  L1-dcache-stores                                   [Hardware cache event]
  L1-dcache-store-misses                             [Hardware cache event]
  L1-dcache-prefetches                               [Hardware cache event]
  L1-dcache-prefetch-misses                          [Hardware cache event]
  L1-icache-loads                                    [Hardware cache event]
  L1-icache-load-misses                              [Hardware cache event]
  L1-icache-prefetches                               [Hardware cache event]
  L1-icache-prefetch-misses                          [Hardware cache event]
  LLC-loads                                          [Hardware cache event]
  LLC-load-misses                                    [Hardware cache event]
```

List of Suported Events

```
node-prefetches                                  [Hardware cache event]
node-prefetch-misses                             [Hardware cache event]

rNNN                                             [Raw hardware event descriptor]
cpu/t1=v1[,t2=v2,t3 ...]/modifier                [Raw hardware event descriptor]
 (see 'perf list --help' on how to encode it)

mem:<addr>[:access]                              [Hardware breakpoint]

kvmmmu:kvm_mmu_pagetable_walk                    [Tracepoint event]
kvmmmu:kvm_mmu_paging_element                    [Tracepoint event]
kvmmmu:kvm_mmu_set_dirty_bit                     [Tracepoint event]
kvmmmu:kvm_mmu_walker_error                      [Tracepoint event]
kvmmmu:kvm_mmu_sync_page                         [Tracepoint event]
kvmmmu:kvm_mmu_unsync_page                       [Tracepoint event]
kvmmmu:mark_mmio_spte                            [Tracepoint event]
kvmmmu:handle_mmio_page_fault                    [Tracepoint event]
kvm:kvm_entry                                    [Tracepoint event]
kvm:kvm_hypercall                                [Tracepoint event]
kvm:kvm_pio                                      [Tracepoint event]
kvm:kvm_cpuid                                    [Tracepoint event]
kvm:kvm_inj_virq                                 [Tracepoint event]
kvm:kvm_inj_exception                            [Tracepoint event]
kvm:kvm_page_fault                               [Tracepoint event]
kvm:kvm_msr                                      [Tracepoint event]
kvm:kvm_cr                                       [Tracepoint event]
kvm:kvm_pic_set_irq                              [Tracepoint event]
kvm:kvm_apic_ipi                                 [Tracepoint event]
kvm:kvm_apic_accept_irq                          [Tracepoint event]
kvm:kvm_nested_intercepts                        [Tracepoint event]
kvm:kvm_nested_vmexit                            [Tracepoint event]
kvm:kvm_nested_intr_vmexit                       [Tracepoint event]
kvm:kvm_invlpga                                  [Tracepoint event]
kvm:kvm_skinit                                   [Tracepoint event]
kvm:vcpu_match_mmio                              [Tracepoint event]
kvm:kvm_userspace_exit                           [Tracepoint event]
```

List of events: wildcards

```
# perf list sched:*
  sched:sched_kthread_stop                   [Tracepoint event]
  sched:sched_kthread_stop_ret               [Tracepoint event]
  sched:sched_wakeup                         [Tracepoint event]
  sched:sched_switch                         [Tracepoint event]
  sched:sched_migrate_task                   [Tracepoint event]
  sched:sched_process_free                   [Tracepoint event]
  sched:sched_process_exit                   [Tracepoint event]
  sched:sched_wait_task                      [Tracepoint event]
  sched:sched_process_wait                   [Tracepoint event]
  sched:sched_process_fork                   [Tracepoint event]
  sched:sched_process_exec                   [Tracepoint event]
  sched:sched_stat_wait                      [Tracepoint event]
  sched:sched_stat_iowait                    [Tracepoint event]
  sched:sched_stat_blocked                   [Tracepoint event]
  sched:sched_stat_runtime                   [Tracepoint event]
#
```

Detailed perf stat output

```
[acme@felicio ~]$ perf stat -d ls > /dev/null

 Performance counter stats for 'ls':

          0.930393 task-clock                #    0.793 CPUs utilized
                 0 context-switches          #    0.000 K/sec
                 0 CPU-migrations            #    0.000 K/sec
               296 page-faults               #    0.318 M/sec
         1,830,263 cycles                    #    1.967 GHz
         1,542,403 stalled-cycles-frontend   #   84.27% frontend cycles idle
         1,165,732 stalled-cycles-backend    #   63.69% backend  cycles idle
         2,701,723 instructions              #    1.48  insns per cycle
                                             #    0.57  stalled cycles per insn
           541,169 branches                  #  581.656 M/sec
            19,298 branch-misses             #    3.57% of all branches
                 0 L1-dcache-loads           #    0.000 K/sec
            28,559 L1-dcache-load-misses     #    0.00% of all L1-dcache hits
                 0 LLC-loads                 #    0.000 K/sec
                 8 LLC-load-misses           #    0.00% of all LL-cache hits    [37.57%]

       0.001172696 seconds time elapsed

[acme@felicio ~]$
```

Run workload multiple times to reduce variability

```
# perf stat -r 10 -e cycles,page-faults,branches sleep 1

 Performance counter stats for 'sleep 1' (10 runs):

          816,122 cycles                    #    0.000 GHz
              160 page-faults
          101,688 branches

       1.002034845 seconds time elapsed

#
```

Sampling

1. Default: cpu cycles
2. Default: 1 kHz frequency
3. No daemons
4. Targets: system wide, workload, pid, tid, cpu, cgroup, uid
5. perf.data file per session
6. Records build ids of DSOs with samples
7. Records system information

record example

```
# perf record -a sleep 15s
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.494 MB perf.data (~21597 samples
#
```

Report

1. stdio
2. TUI
3. TUI - Interactivity
4. Filtering by DSO, thread, more to come
5. Integrated with Annotation
6. Foldable callchains

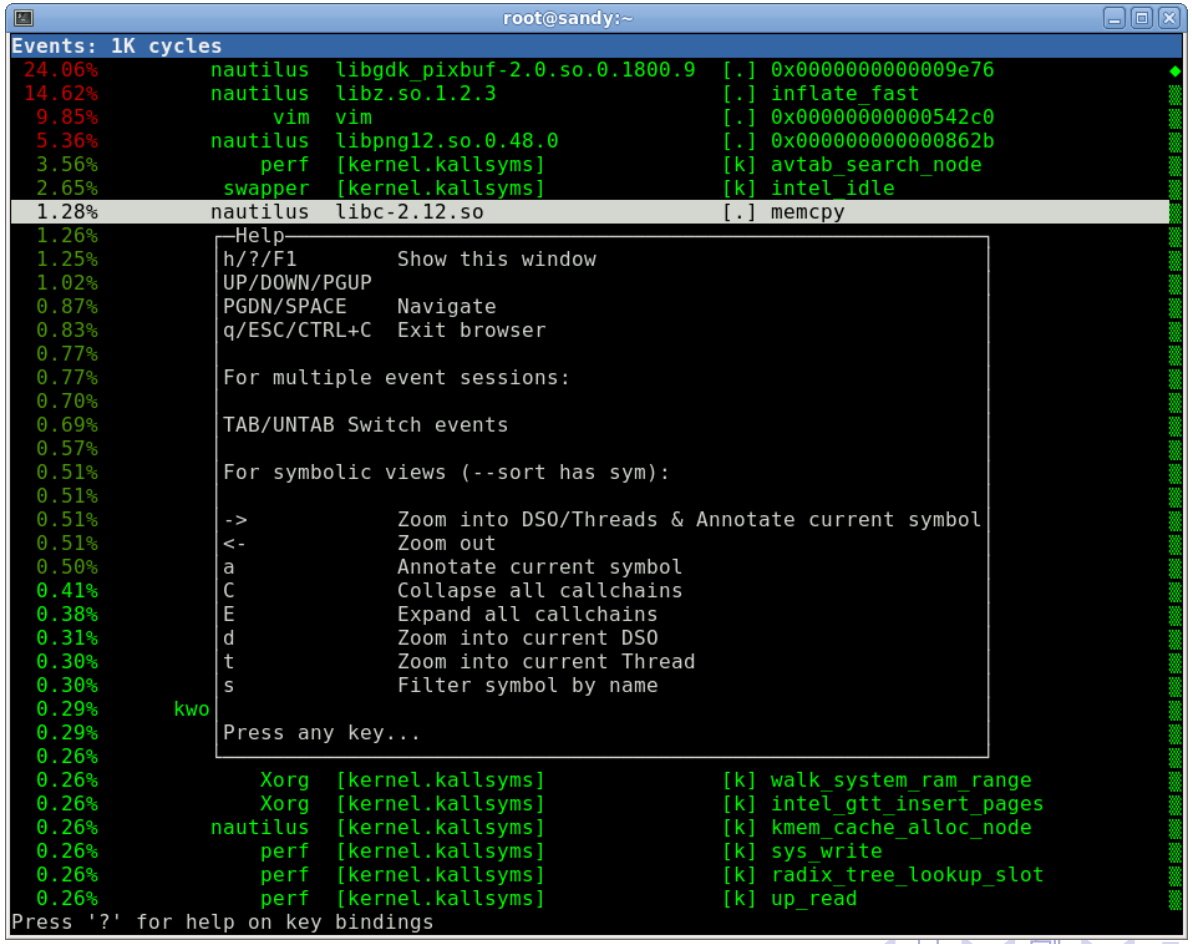
report: stdio output

```
# perf report --stdio
# ========
# captured on: Tue Mar 27 18:43:42 2012
# hostname : sandy.ghostprotocols.net
# os release : 3.3.0+
# perf version : 3.3.316.gebb914
# arch : x86_64
# nrcpus online : 8
# nrcpus avail : 8
# cpudesc : Intel(R) Core(TM) i7-2920XM CPU @ 2.50GHz
# cpuid : GenuineIntel,6,42,7
# total memory : 16220544 kB
# cmdline : /home/acme/bin/perf record -a sleep 15s
# event : name = cycles, type = 0, config = 0x0, config1 = 0x0, config2 = 0x0, excl_usr = 0, e
# HEADER_CPU_TOPOLOGY info available, use -I to display
# HEADER_NUMA_TOPOLOGY info available, use -I to display
# ========
#
# Events: 1K cycles
#
# Overhead          Command                 Shared Object
# ........  ...............  ............................  ................................
#
    24.06%         nautilus  libgdk_pixbuf-2.0.so.0.1800.9  [.] 0x0000000000009e76
    14.62%         nautilus  libz.so.1.2.3                 [.] inflate_fast
     9.85%              vim  vim                           [.] 0x00000000000542c0
     5.36%         nautilus  libpng12.so.0.48.0            [.] 0x000000000000862b
     3.56%             perf  [kernel.kallsyms]             [k] avtab_search_node
     2.65%          swapper  [kernel.kallsyms]             [k] intel_idle
     1.28%         nautilus  libc-2.12.so                  [.] memcpy
     1.26%             perf  [kernel.kallsyms]             [k] number
     1.25%         nautilus  [kernel.kallsyms]             [k] clear_page
     1.02%         nautilus  libz.so.1.2.3                 [.] crc32
     0.87%         nautilus  libgdk-x11-2.0.so.0.1800.9    [.] 0x0000000000035798
     0.83%              vim  vim                           [.] utfc_ptr2len
     0.77%         nautilus  [kernel.kallsyms]             [k] avtab_search_node
     0.77%         nautilus  libz.so.1.2.3                 [.] inflate
     0.70%               ps  libc-2.12.so                  [.] _IO_vfscanf
     0.69%              vim  vim                           [.] vim_strchr
```

Text Based Interface

1. mutt like interface
2. report to annotate fast path
3. Zoom in/out DSOs/threads
4. Keys used: arrows + ENTER mostly, TAB sometimes
5. Still don't like it? Use –stdio
6. Initial GTK+ Browser contributed by Pekka Enberg

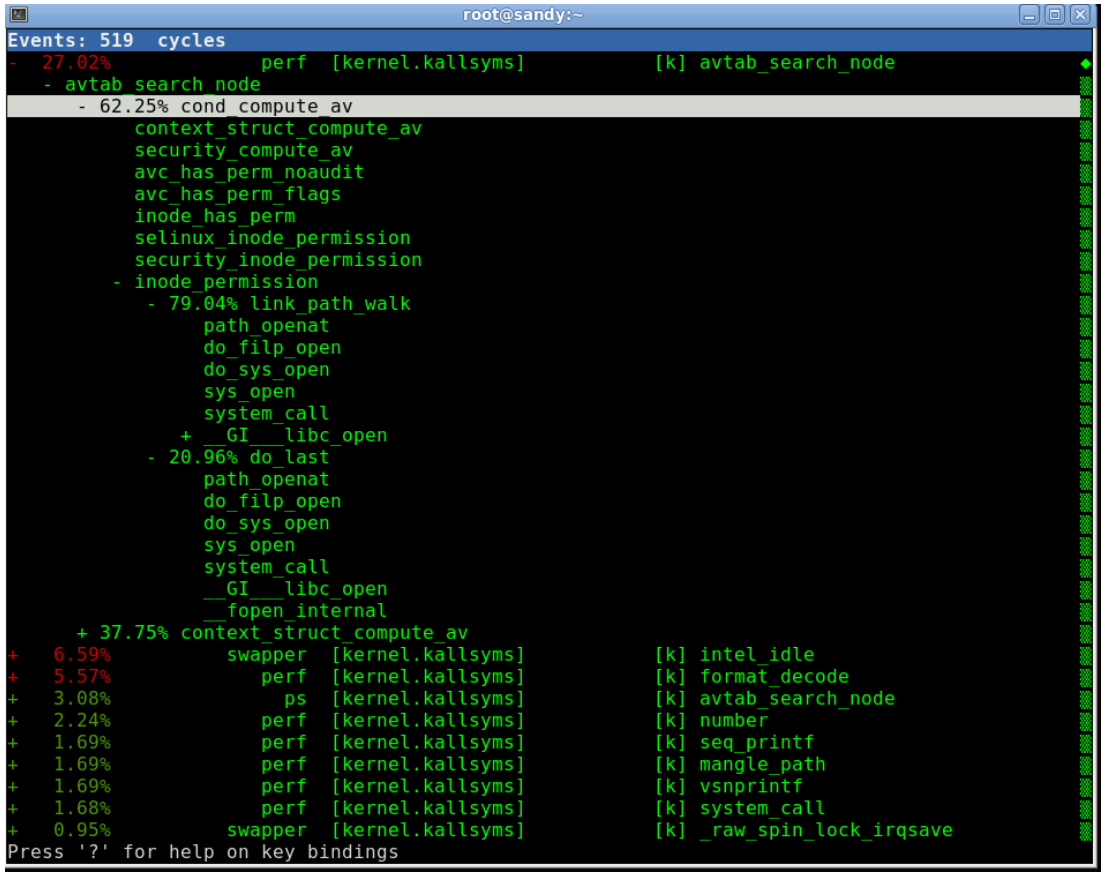
perf report TUI



record callchains example

```
# perf record -a -g sleep 15s
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.509 MB perf.data (~22231 samples
#
```

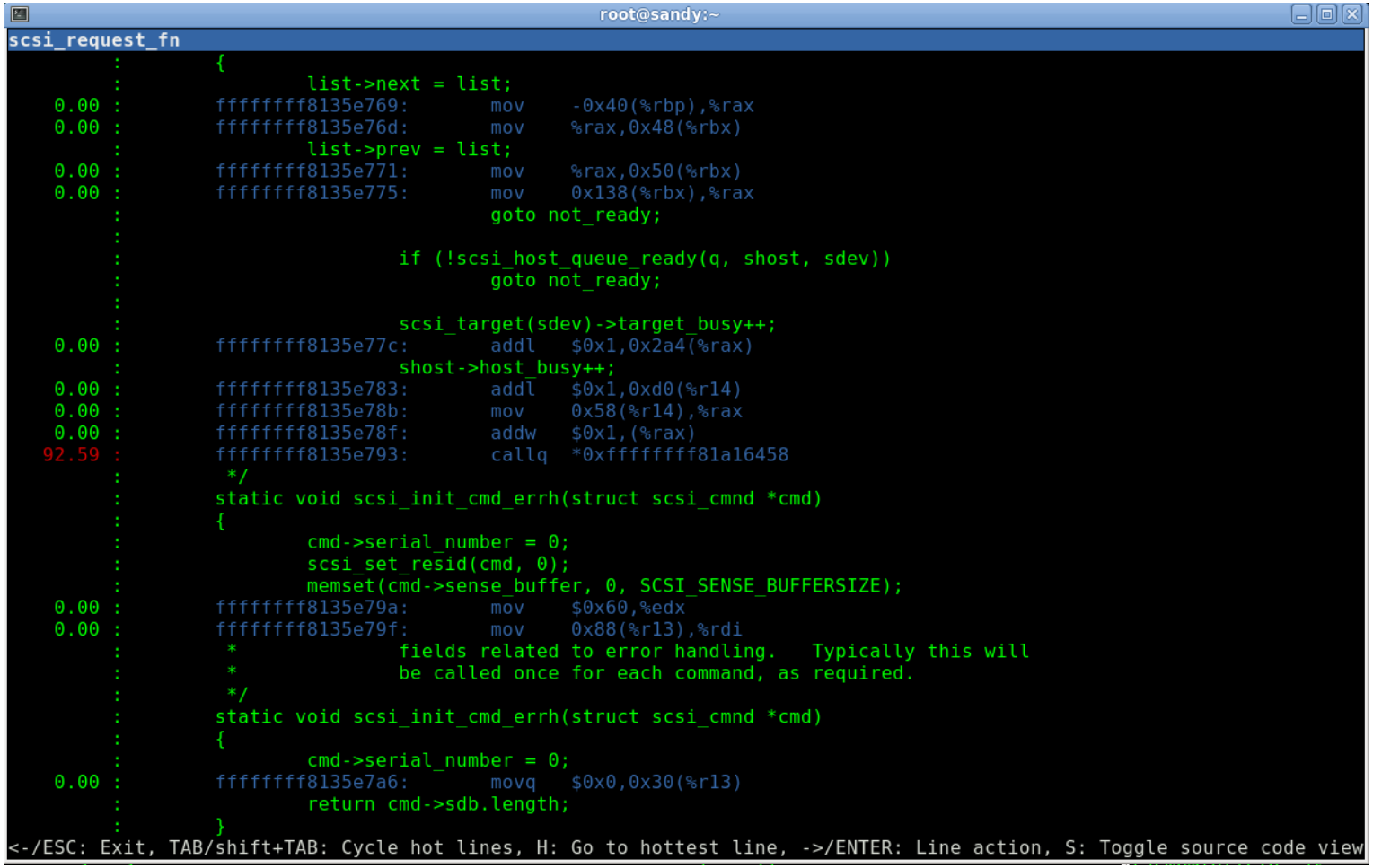
TUI report callchains example



Code Annotation

1. Shows which lines had hits
2. Starts at the line with most hits
3. Tabs through ordered list of hot lines
4. Allows toggling source code lines
5. Navigate thru call sites

perf annotate TUI



perf top TUI

1. Reuses report TUI
2. Integrated live annotation
3. Live decaying callchains

perf top stdio

```
PerfTop: 155 irqs/sec kernel:83.9% [1000Hz cycles], (all, 2 CPUs)

 samples  pcnt function                                DSO
 _______ _____ ____________________________________ ___________________

  119.00 12.0% read_hpet                              [kernel]
   43.00  4.4% __strstr_ia32                          /lib/libc-2.12.1.so
   28.00  2.8% system_call                            [kernel]
   25.00  2.5% unix_poll                              [kernel]
   24.00  2.4% aes_enc_blk                            [aes_i586]
   21.00  2.1% schedule                               [kernel]
   21.00  2.1% _raw_spin_lock_irqsave                 [kernel]
   19.00  1.9% _raw_spin_unlock_irqrestore            [kernel]
   19.00  1.9% aes_dec_blk                            [aes_i586]
   18.00  1.8% probe_workqueue_insertion              [kernel]
   17.00  1.7% hpet_next_event                        [kernel]
   13.00  1.3% fget_light                             [kernel]
   13.00  1.3% do_select                              [kernel]
   12.00  1.2% audit_syscall_entry                    [kernel]
   12.00  1.2% ktime_get                              [kernel]
   11.00  1.1% test_ti_thread_flag                    [kernel]
   11.00  1.1% std::_List_node_base::transfer(std::_L libstdc++.so.6.0.13
   11.00  1.1% native_sched_clock                     [kernel]
   11.00  1.1% vsnprintf                              [kernel]
   11.00  1.1% format_decode                          [kernel]
   10.00  1.0% index                                  /lib/libc-2.12.1.so
```

Offsite Analysis

1. perf archive
2. Looks at perf.data files for DSOs with hits
3. Creates tarball
4. Transfer to another machine
5. Populate the cache
6. Use report and annotate
7. Handles endianness

perf probe

1. Inserts dynamic probes
2. Doesn't necessarily requires debuginfo
3. Can collect variables
4. Struct members can be specified to any level
5. Works with callchains
6. Works on the core kernel and on modules
7. Supports wildcards in probe names
8. Together with perf script == systemtap subset
9. Example of use together with scripting later in this presentation
10. Contributed by Masami Hiramatsu, go to his Talk!

Scripting

1. Use scripting languages to process events
2. Python and Perl
3. Allows tapping into tons of language libraries
4. Several scripts available
5. Generate scripts from perf.data

Available Scripts

```
# perf script --list
List of available trace scripts:
  sched-migration                         sched migration overview
  failed-syscalls-by-pid [comm]           system-wide failed syscalls, by pid
  netdev-times [tx] [rx] [dev=] [debug] display a process of packet and processing t
  net_dropmonitor                         display a table of dropped frames
  syscall-counts [comm]                   system-wide syscall counts
  syscall-counts-by-pid [comm]            system-wide syscall counts, by pid
  futex-contention                        futext contention measurement
  sctop [comm] [interval]                 syscall top
  failed-syscalls [comm]                  system-wide failed syscalls
  workqueue-stats                         workqueue stats (ins/exe/create/destroy)
  wakeup-latency                          system-wide min/max/avg wakeup latency
  rw-by-file <comm>                       r/w activity for a program, by file
  rwtop [interval]                        system-wide r/w top
  rw-by-pid                               system-wide r/w activity
[root@sandy ~]#
```

Generate Scripts

1. From the events found in perf.data file
2. Quickly start writing event handling
3. Creates function skeletons for each trace event
4. With a common set of parameters
5. Plus event specific parameters
6. Calls methods at init, exit and for unhandled events
7. Comes with library of tracing specific methods

Listing Possible probe points

```
[root@ana icmp]# perf probe -L icmp_rcv
<icmp_rcv:0>
      0  int icmp_rcv(struct sk_buff *skb)
      1  {

     59          if (rt->rt_flags & (RTCF_BROADCAST | RTCF_MULTICAST)) {
                         /*
                          * RFC 1122: 3.2.2.6 An ICMP_ECHO to broadcast MAY be
                          * silently ignored (we let user decide with a sysctl).
                          * RFC 1122: 3.2.2.8 An ICMP_TIMESTAMP MAY be silently
                          * discarded if to broadcast/multicast.
                          */
     66                   if ((icmph->type == ICMP_ECHO ||
                               icmph->type == ICMP_TIMESTAMP) &&
                              net->ipv4.sysctl_icmp_echo_ignore_broadcasts) {
                                  goto error;
                         }
     71                   if (icmph->type != ICMP_ECHO &&
```

Listing variables that can be collected

```
[root@ana ~]# perf probe -V icmp_rcv:66
Available variables at icmp_rcv:66
        @<icmp_rcv+343>
                struct icmphdr* icmph
                struct net*     net
                struct rtable*  rt
                struct sk_buff* skb
[root@ana ~]#
```

Adding a probe

```
[root@ana icmp]# perf probe icmp_rcv:66 'type=icmph->type'
Add new event:
  probe:icmp_rcv     (on icmp_rcv:66 with type=icmph->type)

You can now use it on all perf tools, such as:

perf record -e probe:icmp_rcv -aR sleep 1

[root@ana ~]# perf probe --list
  probe:icmp_rcv (on icmp_rcv:66@net/ipv4/icmp.c with type)

[root@ana icmp]# perf record -a -g -e probe:icmp_rcv
^C[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.324 MB perf.data ]
```

Generating a python script from perf.data

```
[root@ana icmp]# perf script -g python
generated Python script: perf-trace.py

[root@ana icmp]# cat perf-trace.py

def trace_begin():
        print "in trace_begin"

def trace_end():
        print "in trace_end"

def probe__icmp_rcv(evname, cpu, secs, nsecs, pid, comm,
                    probe_ip, type):
        print "%s %u.%u type=%u" % (evname, secs, nsecs, type)
```

Running python script

```
[root@ana icmp]# perf script -s perf-trace.py
in trace_begin
probe__icmp_rcv 71171.964568380 type=8
probe__icmp_rcv 71177.792382154 type=8
probe__icmp_rcv 71178.792236953 type=8
in trace_end
[root@ana icmp]#
```

Backtraces from probes

```
[root@ana ~]# perf report --stdio
# Events: 2
#
# Overhead  Command      Shared Object     Symbol
# ........  .......  .................  ........
#
   100.00%     ping  [kernel.kallsyms]  [k] icmp_rcv
               |
               --- icmp_rcv
                   ip_local_deliver_finish
                   NF_HOOK.clone.1
                   ip_local_deliver
                   ip_rcv_finish
                   NF_HOOK.clone.1
                   ip_rcv
                   __netif_receive_skb
                   process_backlog
                   net_rx_action
                   __do_softirq
                   0xb7707424

[root@ana ~]#
```

Language bindings

1. Python so far
2. Another approach for scripting
3. Not driven from perf
4. Normal way to program python
5. twatch.py canonical example
6. More internals need to be exposed

twatch.py example

```
[root@sandy linux]# ./tools/perf/python/twatch.py
cpu:  3, { type: fork, pid: 11481, ppid: 11480, tid: 11481, ptid: 11480, time:
cpu:  7, { type: comm, pid: 11481, tid: 11481, comm: awk }
cpu:  7, { type: exit, pid: 11481, ppid: 11481, tid: 11481, ptid: 11481, time:
cpu:  4, { type: fork, pid: 11482, ppid: 17762, tid: 11482, ptid: 17762, time:
cpu:  7, { type: fork, pid: 11483, ppid: 11482, tid: 11483, ptid: 11482, time:
cpu:  7, { type: fork, pid: 11484, ppid: 11482, tid: 11484, ptid: 11482, time:
cpu:  7, { type: comm, pid: 11483, tid: 11483, comm: ps }
cpu:  3, { type: comm, pid: 11484, tid: 11484, comm: awk }
^CTraceback (most recent call last):
  File "./tools/perf/python/twatch.py", line 41, in <module>
    main()
  File "./tools/perf/python/twatch.py", line 30, in main
    evlist.poll(timeout = -1)
KeyboardInterrupt
[root@sandy linux]#
```

title

```
import perf

def main():
cpus = perf.cpu\_map()
threads = perf.thread\_map()
evsel = perf.evsel(task = 1, comm = 1,
   wakeup\_events = 1,
   sample\_type = perf.SAMPLE\_TID | perf.SAMPLE\_CPU)
evsel.open(cpus = cpus, threads = threads);
evlist = perf.evlist(cpus, threads)
evlist.add(evsel)
evlist.mmap()
while True:
evlist.poll(timeout = -1)
for cpu in cpus:
event = evlist.read\_on\_cpu(cpu)
if not event:
continue
print "cpu: %2d, pid: %4d, tid: %4d" % (event.sample\_cpu,
event.sample\_pid,
event.sample\_tid),
print event
```

KVM Support

1. Collect guest OS statistics from host side.
2. top, record, report, diff, buildid-list
3. Need to specify guest vmlinux or kallsyms, /proc/modules
4. Or –guestmount directory with sshfs mounted per pid subdirs
5. Use –pid to specify specific guest
6. Contributed by Zhang, Yanmin.

perf top kvm example

```
# perf kvm --host --guest --guestkallsyms=guest/kallsyms \
          --guestmodules=guest/modules top

 PerfTop: 16010 irqs/sec kernel:59.1% us: 1.5% guest
          kernel:31.9% guest us:7.5% [+1000Hz cycles]

  samples  pcnt function                    DSO
  _______ _____ ________________________ _____________

 38770.00 20.4% __ticket_spin_lock         [guest.kernel]
 22560.00 11.9% ftrace_likely_update       [kernel]
  9208.00  4.8% __lock_acquire             [kernel]
  5473.00  2.9% trace_hardirqs_off_caller  [kernel]
  5222.00  2.7% copy_user_generic_string   [guest.kernel]
  4450.00  2.3% validate_chain             [kernel]
  4262.00  2.2% trace_hardirqs_on_caller   [kernel]
  4239.00  2.2% do_raw_spin_lock           [kernel]
  3548.00  1.9% do_raw_spin_unlock         [kernel]
  2487.00  1.3% lock_release               [kernel]
  2165.00  1.1% __local_bh_disable         [kernel]
  1905.00  1.0% check_chain_key            [kernel]
```

Work in Progress

1. perfviz
2. uprobes to probe user space
3. GTK+ report browser
4. CFI/DWARF based postprocessing userspace callchains

Thanks!

Arnaldo Carvalho de Melo

acme@infradead.org

acme@redhat.com

linux-perf-users@vger.kernel.org