Introduction to Perf

Beijing Trace Training 2017

Gary Lin
Software Engineer, SUSE Labs
glin@suse.com

Before using perf...

- dmesg/syslog
- top
- iostat/vmstat
- iotop
- pidstat
- strace
- latencytop

perf

- A collection of performance analysis tools
- Sampling and profiling the system
- Providing some tracing mechanisms

perf events

Events

```
# perf list
List of pre-defined events (to be used in -e):

  branch-instructions OR branches       [Hardware event]
  branch-misses                         [Hardware event]
  bus-cycles                            [Hardware event]
...
  alignment-faults                      [Software event]
  bpf-output                            [Software event]
  context-switches OR cs                [Software event]
...
  block:block_bio_backmerge           [Tracepoint event]
  block:block_bio_bounce              [Tracepoint event]
  block:block_bio_complete            [Tracepoint event]
...
```

Hardware Events

“**hardware performance counters** … are a set of special-purpose registers … to store the counts of hardware-related activities”

– Wikipedia

Hardware Events

- Hardware events are CPU-specific.
  Check the CPU manual for more details.
- The common events:
  - branch-instructions
  - bus-cycles
  - cache-misses
  - cache-references
  - cpu-cycles
  - Instructions

Multiplexing and Scaling Events

- The hardware resources are limited.
- When there are more enabled events than the available counters, the events are managed in round-robin fashion.
- The event count may be scaled.

  final_count = raw_count * time_enabled / time_running

Raw PMU

- Show the available PMUs

```
# showevtinfo |less
IDX       : 23068702
PMU name  : core (Intel Core)
Name      : L2_LINES_IN
Desc      : L2 cache misses
```

- Look up the raw number from the PMU name

```
# evt2raw L2_LINES_IN
r537024
```

- Use the event

```
# perf stat -e r537024 -a sleep 10
```

Event Modifiers

u - user-space counting
k - kernel counting
h - hypervisor counting
I - non idle counting
G - guest counting (in KVM guests)
H - host counting (not in KVM guests)
p - precise level
P - use maximum detected precise level
S - read sample value (PERF_SAMPLE_READ)
D - pin the event to the PMU

Event Modifiers – Examples

- Request 0 skid on cpu-cycles

```
# perf stat -e cpu-cycles:pp -a sleep 10
```

- Only count user cache misses

```
# perf record -e cache-misses:u -a sleep 10
```

Software Events

Software Events

- Special “software” counters provided by the kernel, even if the hardware does not support performance conters.
- The software events:
  - cpu-clock
  - task-clock
  - context-switches
  - page-faults, major-faults, and minor-faults
  - alignment-faults
- The available SW events are defined in “`perf_sw_ids`” in `include/uapi/linux/perf_event.h`.

Page Fault

- Major Fault

  The page is not loaded at the time the fault is generated. The additional disk latency is expected.

- Minor Fault

  The page is already in the physical memory but not marked in MMU at the time the fault is generated. No disk latency occurs.

Tracepoint Events

Tracepoint Events

- In-kernel **static** trace events
- Tracepoint events depend on the kernel and loaded modules.
- The tracepoint event list:

```
# perf list tracepoint
# less /sys/kernel/debug/tracing/avaliable_events
```

Dynamic Tracing

perf probe

- Define new dynamic tracepoints
- kprobe
  - The kernel symbols are listed in /proc/kallsyms.
- uprobe
  - The symbols can be read with “`readelf`”.
- Define a new kprobe event

```
# perf probe <kprobe event>
```

- Delete a kprobe event

```
# perf probe --del=<kprobe event>
```

- Use the kprobe event

```
# perf trace --event probe:<kprobe event>
# perf record -e probe:<kprobe event>
```

perf probe (cont'd)

- The following commands need kernel debuginfo and debugsource.
- Show the arguments of the probed function

```
# perf probe -V <func>
```

- Show the source code of the probed function

```
# perf probe -L <func>
```

- Probe the line 12 of a function

```
# perf probe <func>:12
```

- Probe a member of a struct in the arguments (`ifindex` in `struct net` of `icmp_out_count()`)

```
# perf probe 'icmp_out_count net->ifindex'
```

- Do a verbose dry run

```
# perf probe -nv <probe>
```


NOTE: You may encounter build-id mismatching. See bsc#964063.

Perf Commands

Basic perf Commands

- Show the supported events
  `perf list`
- Real-time monitoring
  `perf top`
- Offline profiling
  `perf record`, `perf report`, `perf script`
- Event counting
  `perf stat`
- System event tracing
  `perf trace`

perf top – A real-time system profiling tool

```
Samples: 973  of event 'cycles:p', Event count (approx.): 2310200324878
Overhead  Shared Object                     Symbol
  43.32%  [kernel]                          [k] acpi_processor_ffh_cstate_enter
  13.77%  [vdso]                            [.] __vdso_clock_gettime
   5.55%  [kernel]                          [k] read_hpet
   5.20%  libpthread-2.22.so                [.] __pthread_mutex_unlock_usercnt
   4.55%  libxul.so                         [.] 0x0000000002be54d8
   3.05%  ld-2.22.so                        [.] check_match
   2.67%  libgmodule-2.0.so.0.4800.2        [.] g_module_symbol
   2.04%  liblua.so.5.2                     [.] 0x000000000001763f
   1.79%  libxul.so                         [.] 0x00000000011b0b76
   1.56%  [kernel]                          [k] unix_stream_read_generic
   1.56%  liblua.so.5.2                     [.] 0x0000000000018353
   1.40%  libxul.so                         [.] 0x00000000011894d3
   1.23%  libxcb.so.1.1.0                   [.] xcb_send_request64
   1.20%  libxul.so                         [.] 0x0000000001ddf05b
   0.94%  libxul.so                         [.] 0x00000000009ffe05
   0.94%  libxul.so                         [.] 0x000000000097f4ca
   0.70%  libxul.so                         [.] 0x000000000114351a
   0.70%  [unknown]                         [.] 0x00007fc5cc822d7b
   0.63%  intel_drv.so                      [.] 0x00000000000127dd
   0.61%  libxul.so                         [.] 0x0000000000987cee
   0.54%  awesome                           [.] client_focus_refresh
   0.51%  firefox                           [.] 0x000000000001350e
   0.47%  libxul.so                         [.] 0x0000000002747dcb
   0.47%  corelgilua51.so                   [.] lgi_marshal_access
   0.31%  libxul.so                         [.] 0x000000000103c3fd
   0.31%  firefox                           [.] 0x0000000000011652
   0.28%  libc-2.22.so                      [.] __memcpy_sse2_unaligned
   0.21%  [unknown]                         [.] 0x00007fc5cc528053
   0.18%  [kernel]                          [k] copy_msghdr_from_user
   0.16%  libxul.so                         [.] 0x00000000009ae604
Failed to open /tmp/perf-3181.map, continuing without symbols
```

perf top – Options

-C, --cpu

Focuses on specific CPUs

-p, --pid, or -t, --tid

Focuses on specific process or thread IDs

-d, --delay

Number of seconds to delay between refreshes

-n

Show the number of samples

--dsos, --comms

Limits the information to particular DSO’s and commands

-s, --sort

Sorts the samples

-g

enable call-graph (stack chain/backtrace) recording

perf top – Tip

Press ‘z’ to zero all counts for a few seconds to avoid recording activity from “perf top” itself.

perf record – Record perf profile into perf.data

perf record – Options

-d, --delay N

Wait N msecs before the profiling starts

-F N

Sample every N times a second

-c N

Sample every N events

-s, --stat

Record per-thread event counts

-i, --no-inhert

Child tasks do not inherit counters

-a, --all-cpus

System-wide collection from all CPUs

perf record – Options (cont'd)

-g

Enable call-graph recording

--call-graph <type>

Setup and enable call-graph recording.

"fp" (frame pointer) is the default for "-g". Try "dwarf" if "--fomit-frame-pointer" is used

-e, --event=<event>

Select the event

--filter=<filter>

Use the event filter

-m

Specify the buffer size

perf record – Examples

Sample on-CPU functions for the specified PID, until Ctrl-C:

```
# perf record -p PID
```

Sample CPU stack traces for the PID, using dwarf (dbg info) to unwind stacks, for 10 seconds

```
# perf record -p PID --call-graph dwarf sleep 10
```

Sample CPU stack traces for the entire system, for 10 seconds:

```
# perf record -ag -- sleep 10
```

Trace all block completions, of size at least 100 Kbytes

```
# perf record -e block:block_rq_complete \
              --filter 'nr_sector > 200'
```

perf report – Display the profile

```
Samples: 32  of event 'cycles:p', Event count (approx.): 2120672040374
  Children      Self  Command  Shared Object      Symbol
+   58.47%     0.00%  swapper  [kernel.kallsyms]  [k] acpi_idle_enter
+   58.47%     0.00%  swapper  [kernel.kallsyms]  [k] cpuidle_enter_state
+   58.47%     0.00%  swapper  [kernel.kallsyms]  [k] cpu_startup_entry
+   58.47%    58.47%  swapper  [kernel.kallsyms]  [k] acpi_processor_ffh_cstate_enter
+   31.93%     0.00%  swapper  [kernel.kallsyms]  [k] start_kernel
+   31.93%     0.00%  swapper  [kernel.kallsyms]  [k] x86_64_start_kernel
+   27.79%     0.00%  swapper  [kernel.kallsyms]  [k] acpi_idle_enter_bm
+   26.54%     0.00%  swapper  [kernel.kallsyms]  [k] start_secondary
+   12.94%     0.00%  swapper  [kernel.kallsyms]  [k] handle_irq_event_percpu
+   12.94%     0.00%  swapper  [kernel.kallsyms]  [k] handle_irq_event
+   12.94%     0.00%  swapper  [kernel.kallsyms]  [k] handle_edge_irq
+   12.94%     0.00%  swapper  [kernel.kallsyms]  [k] handle_irq
+   12.94%     0.00%  swapper  [kernel.kallsyms]  [k] do_IRQ
+   12.94%     0.00%  swapper  [kernel.kallsyms]  [k] ret_from_intr
+   12.94%    12.94%  swapper  [kernel.kallsyms]  [k] native_write_msr_safe
+   12.94%     0.00%  swapper  [kernel.kallsyms]  [k] __intel_pmu_enable_all.isra.9
+   12.94%     0.00%  swapper  [kernel.kallsyms]  [k] perf_event_task_tick
+   12.94%     0.00%  swapper  [kernel.kallsyms]  [k] scheduler_tick
+   12.94%     0.00%  swapper  [kernel.kallsyms]  [k] update_process_times
+   12.94%     0.00%  swapper  [kernel.kallsyms]  [k] tick_sched_handle.isra.17
+   12.94%     0.00%  swapper  [kernel.kallsyms]  [k] tick_sched_timer
+   12.94%     0.00%  swapper  [kernel.kallsyms]  [k] __hrtimer_run_queues
+   12.94%     0.00%  swapper  [kernel.kallsyms]  [k] hrtimer_interrupt
+   12.94%     0.00%  swapper  [kernel.kallsyms]  [k] timer_interrupt
+   12.94%     0.00%  swapper  [kernel.kallsyms]  [k] __handle_irq_event_percpu
+   12.94%     0.00%  perf     libc-2.22.so       [.] __GI___ioctl
+   12.94%     0.00%  perf     perf               [.] cmd_record
+   12.94%     0.00%  perf     perf               [.] 0xfffffffffc94ff5
+   12.94%     0.00%  perf     perf               [.] main
+   12.94%    12.94%  perf     [kernel.kallsyms]  [k] native_write_msr_safe
For a higher level overview, try: perf report --sort comm,dso
```

perf script – Display the trace output

```
Process        PID    CPU    Timestamp          CPU Cycles
audioPipe:src  31884  [000]  620396.493554:      141988 cycles:
                      3ed40 [unknown] (/usr/lib64/gstreamer-1.0/libgstcoreeleme
                         8f [unknown] ([unknown])
                55b5f048f100 [unknown] ([unknown])
                          0 [unknown] ([unknown])

offlineimap 32210 [000] 620396.507109:     156215 cycles:
                       2529 [unknown] (/usr/lib64/python2.7/lib-dynload/time.so
            c87200016b0004 [unknown] ([unknown])

alsa-sink-ALC32  3300 [002] 620396.512810:     419889 cycles:
                       39a17 [unknown] (/usr/lib64/pulseaudio/libpulsecommon-10.
              55ba449bb190 [unknown] ([unknown])

threaded-ml 31876 [000] 620396.512898:     156215 cycles:
              7fff8173d763 __sched_text_start ([kernel.kallsyms])
              7fff8173e04d schedule ([kernel.kallsyms])
              7fff81741f9b schedule_hrtimeout_range_clock ([kernel.kallsyms])
              7fff8124bae1 poll_schedule_timeout ([kernel.kallsyms])
              7fff8124cf2a do_sys_poll ([kernel.kallsyms])
                     e418d [unknown] (/lib64/libc-2.25.so)
                         6 [unknown] ([unknown])
```

backtrace

perf stat – Performance counter statistics

“perf stat” is much lighter weight than “perf record”.

perf stat – Output

```
# perf stat -a sleep 10

 Performance counter stats for 'system wide':

   20032.666323  task-clock (msec)         #  2.000 CPUs utilized   (100.00%)
          41074  context-switches          #  0.002 M/sec           (100.00%)
           2152  cpu-migrations            #  0.107 K/sec           (100.00%)
           6252  page-faults               #  0.312 K/sec
     6838201211  cycles                    #  0.341 GHz              (50.01%)
<not supported>  stalled-cycles-frontend
<not supported>  stalled-cycles-backend
     4157856056  instructions              #  0.61  insns per cycle  (74.99%)
      871168471  branches                  # 43.487 M/sec           (75.03%)
       38497517  branch-misses             #  4.42% of all branches  (74.98%)

  10.016014696 seconds time elapsed
```

count

scale

perf stat – Options

--pre, --post

Pre and post measurement hooks

--sync

Call sync every iteration

-d, --detailed

Print more detailed statistics, can be specified up to 3 times

-r, --repeat=<n>

Repeat command and print average + stddev

-I msecs, --interval-print msecs

Print count deltas every N msecs

-e, --event=<event>

Select the event

-a, --all-cpus

System-wide collection from all CPUs

perf stat – Examples

CPU counter statistics for the entire system, for 10 seconds:

```
# perf stat -a sleep 10
```

Various CPU level 1 data cache statistics for the specified command:

```
# perf stat -e L1-dcache-loads,L1-dcache-load-misses,\
L1-dcache-stores command
```

Count scheduler events for the specified PID, until Ctrl-C:

```
# perf stat -e 'sched:*' -p PID
```

Measure “make” 10 times:

```
# perf stat -r 10 --sync --pre make clean -- make
```

perf trace – System event tracing

```
Timestamp   Duration   Process     PID  Syscall
16412.718 ( 0.002 ms): InputThread/2933 epoll_wait(epfd: 35<anon_inode:[event
16412.723 ( 0.003 ms): InputThread/2933 read(fd: 40</dev/input/event1>, buf:
16412.727 ( 0.001 ms): InputThread/2933 read(fd: 40</dev/input/event1>, buf:
16412.733 ( 0.001 ms): InputThread/2933 read(fd: 40</dev/input/event1>, buf:
16412.735 ( 0.001 ms): InputThread/2933 read(fd: 40</dev/input/event1>, buf:
16412.745 ( 0.004 ms): InputThread/2933 write(fd: 31<pipe:[21362]>, buf: 0x7f
16394.158 (18.593 ms): X/2874  ... [continued]: epoll_wait()) = 1
16412.750 ( 0.004 ms): InputThread/2933 epoll_wait(epfd: 34<anon_inode:[event
16412.753 ( 0.002 ms): X/2874 read(fd: 30<pipe:[21362]>, buf: 0x7fffae2c9890,
16412.765 ( 0.005 ms): X/2874 writev(fd: 53, vec: 0x7fffae2c87d0, vlen: 1
16394.152 (18.621 ms): QXcbEventReade/3136  ... [continued]: poll()) = 1
16412.775 ( 0.003 ms): QXcbEventReade/3136 recvmsg(fd: 3<socket:[31188]>, msg
16412.783 ( 0.003 ms): QXcbEventReade/3136 write(fd: 5<anon_inode:[eventfd]>,
16396.068 (16.723 ms): kwin_x11/3131  ... [continued]: ppoll()) = 1
16412.787 ( 0.006 ms): QXcbEventReade/3136 poll(ufds: 0x7f90bc8e6bc8, nfds: 1
16412.793 ( 0.001 ms): kwin_x11/3131 read(fd: 5<anon_inode:[eventfd]>, buf: 0
16412.814 ( 0.004 ms): konsole/13778 poll(ufds: 0x55db78c35440, nfds: 16, tim
16412.820 ( 0.001 ms): konsole/13778 ioctl(fd: 28</dev/ptmx>, cmd: FIONREAD,
16412.822 ( 0.003 ms): konsole/13778 read(fd: 28</dev/ptmx>, buf: 0x55db78afd
```

perf trace vs strace

- perf trace is inspired by strace.
- perf trace uses tracepoints instead of ptrace.
  - perf trace is faster in general.
- perf trace outputs to a buffer first while strace dumps the result directly.
- perf trace may lose events if the ring buffer fills while strace never loses events.
  - It can be moderated by increasing the buffer size.

perf trace – Options

-p, --pid, -t, --tid, or, -u, --uid

Focuses on specific process, thread, or user IDs

-e, --expr

List of syscalls to show

--event=<event>

Trace other events

--no-syscalls

Don’t trace syscalls

-m

Specify the buffer size

--duration

Show only events that had a duration greater than N.M msecs

perf trace – Options (cont'd)

-s, --summary

Show only a summary of syscalls

-S, --with-summary

Show all syscalls followed by a summary

-o, --output=

Output file name

-F=[all|min|maj]

Trace pagefaults

perf trace – Examples

Only display the events had duration longer than 0.2 for the specified command (also allocate 64MB for perf buffer):

```
# perf trace -m 64M --duration 0.2 command
```

Trace all syscalls except write

```
# perf trace -m 64M -e \!write
```

Trace only block_rq_issue and block_rq_complete

```
# perf trace --no-syscalls \
  --event block:block_rq_issue,block:block_rq_complete
```

Choose Your perf command

- Real-time monitoring
  `perf top`
- Offline profiling
  `perf record`, `perf report`, `perf script`
- Event counting
  `perf stat`
- System event tracing
  `perf trace`

Special Commands

- perf **sched** – scheduler properties
- perf **kmem** – kernel memory properties
- perf **mem** – profiling memory accesses
- perf **lock** – analyzing lock events

perf vs ftrace

perf vs ftrace (Events)

	PMU	software event	ftrace filter function	tracepoint	kprobe	uprobe
ftrace			●	●	●	●
perf	●	●	○	●	●	●

* perf ftrace supports “ftrace filter functions”.

perf vs ftrace (Output)

- perf
  - syscall: `__NR_perf_event_open`
  - Allow multiple users
- ftrace
  - `/sys/kernel/debug/tracing/trace_pipe` or `/sys/kernel/debug/tracing/trace`
  - Allow only a single root user

Choose Your Tool

- For CPU-bound issues: **perf**
- For IO-bound issues: **ftrace, perf trace**, or others

Question?


SUSE®
We adapt. You succeed.

References

- Linux perf Examples
  http://www.brendangregg.com/perf.html
- Linux kernel profiling with perf
  https://perf.wiki.kernel.org/index.php/Tutorial