Observe your system
with perf, ftrace, eBPF and systemtap

Giovanni Gherdovich
gggherdevich@suse.cz
Before we begin

SUSE is hiring!
http://www.suse.com/jobs
Agenda

- Motivations
- Tracepoints
- Perf
- ftrace
- eBPF
- SystemTap
Motivations

• system administrators vs application developers
Motivations

- These tools requires root (often)
- Allows injection of code directly in kernel space
- Debugging the kernel
- Troubleshooting systems in production
- Performance analysis
Tracepoints

- Tracepoints, or static event. "Almost" kernel ABI
  - Stable.
- Kprobes, aka attach handler to an arbitrary kernel function.
  - Said function could be removed, or renamed… Scripts may not last long and need continuous fixing/porting.
Tracepoints

- **block:block_rq_insert**: who is creating I/O, what type, how much
- **block:block_rq_complete**: to correlate insert w/ completions
- **block:block_dirty_buffer**: in case of excessive amount of writeback
- **block:block_plug/block:block:block_unplug**: why requests merge or not
- **compaction:mm_compaction_try_to_compact_pages**: trying to allocate continuous ranges
- **compaction:mm_compaction_begin/compaction:mm_compaction_end**: for how long compaction lasted
- **filemap:mm_filemap_add_to_page_cache/filemap:mm_filemap_delete_from_page_cache**: track what pages are resident in memory and for how long
- **kmem:*alloc*/kmem:*free***: slab allocations and frees
Tracepoints

- **migrate:mm_migrate_pages**: numa balancing
- **net:** network latencies
- **sched:sched_process_exec**: a process starts execution
- **sched:sched_stat**: a process stalled
- **sched:sched_migrate_task/sched:sched_move_numa**: process migration
- **syscalls:**
- **vmscan:mm_vmscan_kswapd_wake**: model page behaviour
- **vmscan:mm_vmscan_direct_reclaim_begin/vmscan:mm_vmscan_direct_reclaim_end**: heavy memory pressure
Tracepoints

- `writeback:balance_dirty_pages`: generate stalls
- `writeback:writeback_congestion_wait/writeback:writeback_wait_iff_congested`: excessive amount of writeback
perf & perf_events

- perf_events is the kernel infrastructure, perf is the userspace tool
- Lives in tree at tools/perf
- The primary developers are Ingo Molnar and Peter Zijlstra
- Supports CPU performance counters, software events, tracepoints, kprobes and uprobes
- Manpages are provided, see man -k perf
- Require kernel built with appropriate configuration options for some functionalities
perf & perf_events

bench           General framework for benchmark suites
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 properties
kvm             Tool to trace/measure kvm guest os
list            List all symbolic event types
lock            Analyze lock events
mem             Profile memory accesses
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
top             System profiling tool.
trace           strace inspired tool
probe           Define new dynamic tracepoints
perf & perf_events

$ perf record -a sleep 5
$ perf report

# Samples: 2K of event 'cycles:ppp'
# Event count (approx.): 1043962924

<table>
<thead>
<tr>
<th>Overhead</th>
<th>Command</th>
<th>Shared Object</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>24.05%</td>
<td>swapper</td>
<td>[kernel.kallsyms]</td>
<td>[k] 0xffffffff813da762</td>
</tr>
<tr>
<td>1.16%</td>
<td>mplayer</td>
<td>mplayer</td>
<td>[. ] 0x00000000000260028</td>
</tr>
<tr>
<td>1.03%</td>
<td>soffice.bin</td>
<td>libc-2.22.so</td>
<td>[. ] __memmove_avx_unaligned</td>
</tr>
<tr>
<td>0.55%</td>
<td>emacs-gtk</td>
<td>emacs-gtk</td>
<td>[. ] 0x000000000015c976</td>
</tr>
<tr>
<td>0.55%</td>
<td>mplayer</td>
<td>mplayer</td>
<td>[. ] 0x0000000000260015</td>
</tr>
<tr>
<td>0.55%</td>
<td>gnome-terminal-</td>
<td>libc-2.22.so</td>
<td>[. ] __memset_sse2</td>
</tr>
<tr>
<td>0.50%</td>
<td>mplayer</td>
<td>libavcodec.so.57.48.101</td>
<td>[. ] 0x00000000006a48d0</td>
</tr>
<tr>
<td>0.46%</td>
<td>soffice.bin</td>
<td>libpixman-1.so.0.34.0</td>
<td>[. ] 0x00000000008dc3a</td>
</tr>
<tr>
<td>0.40%</td>
<td>mplayer</td>
<td>mplayer</td>
<td>[. ] 0x0000000000260012</td>
</tr>
</tbody>
</table>
perf & perf_events

$ perf script | head

perf 9449 [000] 41595.445170: 1 cycles:ppp: ffffffff8118165e event_function ([kernel.kallsyms])
perf 9449 [000] 41595.445174: 1 cycles:ppp: ffffffff8100c13a intel_pmu_handle_irq ([kernel.kallsyms])
perf 9449 [000] 41595.445175: 9 cycles:ppp: ffffffff81036980 native_sched_clock ([kernel.kallsyms])
perf 9449 [000] 41595.445177: 180 cycles:ppp: ffffffff81036980 native_sched_clock ([kernel.kallsyms])
perf 9449 [000] 41595.445178: 3931 cycles:ppp: ffffffff81036980 native_sched_clock ([kernel.kallsyms])
perf 9449 [000] 41595.445180: 84205 cycles:ppp: ffffffff8123333b do_vfs_ioctl ([kernel.kallsyms])
swapper 0 [001] 41595.445191: 1 cycles:ppp: ffffffff8100b350 intel_pmu_enable_all ([kernel.kallsyms])
swapper 0 [001] 41595.445193: 1 cycles:ppp: ffffffff81058eb0 native_apic_mem_write ([kernel.kallsyms])
swapper 0 [001] 41595.445195: 13 cycles:ppp: ffffffff8103694d native_sched_clock ([kernel.kallsyms])
swapper 0 [001] 41595.445197: 214 cycles:ppp: ffffffff8103694d native_sched_clock ([kernel.kallsyms])
$ perf list | grep sched:
  sched:sched_kthread_stop                           [Tracepoint event]
  sched:sched_kthread_stop_ret                       [Tracepoint event]
  sched:sched_migrate_task                           [Tracepoint event]
  sched:sched_move_numa                              [Tracepoint event]
  sched:sched_pi_setprio                             [Tracepoint event]
  sched:sched_process_exec                           [Tracepoint event]
  sched:sched_process_exit                           [Tracepoint event]
  sched:sched_process_fork                           [Tracepoint event]
  sched:sched_process_free                           [Tracepoint event]
  sched:sched_process_hang                           [Tracepoint event]

[...]
perf & perf_events

$ perf record -e sched:sched_stat_sleep -e sched:sched_switch -g
$ perf report
# Samples: 2M of event 'sched:sched_switch'
# Event count (approx.): 4246430127764
#
# Children      Self       Samples  Command          Shared Object         Symbol
  70.67%  0.00%           0  httpd            [kernel.kallsyms]     [k] schedule
          |---schedule
          |    __schedule
  70.67%  70.67%  1629298  httpd            [kernel.kallsyms]     [k] __schedule
          |--67.92%-- pthread_cond_wait@@GLIBC_2.3.2
          |          entry_SYSCALL_64_fastpath
          |          sys_futex
          |          do_futex
          |          futex_wait
perf & perf_events

$ perf top

PerfTop: 644 irqs/sec  kernel:45.2%  exact: 100.0% [4000Hz cycles:ppp], (all, 4 CPUs)

        4.74%  libz.so.1.2.8         [.] 0x0000000000002f36
        3.88%  [kernel]              [k] module_get_kallsym
        2.43%  [kernel]              [k] format_decode
        2.17%  perf                   [.] symbols__insert
        1.73%  perf                   [.] rb_next
        1.65%  [kernel]              [k] number
        1.43%  [kernel]              [k] kallsyms_expand_symbol.constprop.1
        1.27%  perf                   [.] hex2u64
        1.12%  [kernel]              [k] string
        1.10%  perf                   [.] rb_insert_color
$ perf stat -a

^C

Performance counter stats for 'system wide':

    7786.772339  task-clock (msec)  #  4.001 CPUs utilized
       742  context-switches  #  0.095 K/sec
       18  cpu-migrations  #  0.002 K/sec
        7  page-faults  #  0.001 K/sec
   81,834,321  cycles  #  0.011 GHz
     <not supported>  stalled-cycles-backend
     <not supported>  stalled-cycles-backend
  31,909,569  instructions  #  0.39 insns per cycle
  6,976,025  branches  #  0.896 M/sec
  326,023  branch-misses  #  4.67% of all branches

1.946330553 seconds time elapsed
perf & perf_events

$ perf sched record -a
$ perf sched map

    *.            41428.284823 secs .  => swapper:0
*A0      .            41428.287425 secs A0 => rcu_sched:7
*.       .            41428.287429 secs
  .      .  *B0       41428.289945 secs B0 => gnome-terminal-:2127
  .      .  *.        41428.289983 secs
*A0      .   .        41428.291418 secs
*       .   .        41428.291420 secs
*A0      .   .        41428.295423 secs
*       .   .        41428.295426 secs
  .  *C0  .   .        41428.296136 secs C0 => Timer:8743
  .  *.   .   .        41428.296149 secs
  .  *C0  .   .        41428.297198 secs
perf & perf_events

http://www.brendangregg.com/FlameGraphs/cpufreqmegraphs.html#perf
Off-CPU analysis.

$ perf record -e sched:sched_stat_sleep -e sched:sched_switch \
    -e sched:sched_process_exit -a -g -o perf.data.raw sleep 1
$ perf inject -v -s -i perf.data.raw -o perf.data
perf & perf_events

Examples of analysis carried out with perf (by Ingo Molnar):

- “The problem with prefetch”  [https://lwn.net/Articles/444336/](https://lwn.net/Articles/444336/)
- “Software prefetches considered harmful”  [https://lwn.net/Articles/444346/](https://lwn.net/Articles/444346/)
ftrace

- Kernel tracer, available since ~2008
- Originated from the -rt kernel
- Maintained by Steven Rostedt
- Exposes tracepoints as files under /sys/kernel/debug/tracing
- Programmable via shell and unix tools
- Kernel needs to be compiled with specific options
- Documentation in tree at Documentation/trace/ftrace.txt
- Requires knowledge of kernel tracepoints
- Allows tracing of arbitrary kernel functions with kprobes
- Has CLI (trace-cmd) and GUI (kernelshark)
ftrace: kernel config

$ make allyesconfig
$ grep FTRACE .config
CONFIG_KPROBES_ON_FTRACE=y
CONFIG_HAVE_KPROBES_ON_FTRACE=y
CONFIG_PSTORE_FTRACE=y
CONFIG_HAVE_DYNAMIC_FTRACE=y
CONFIG_HAVE_DYNAMIC_FTRACE_WITH_REGS=y
CONFIG_HAVE_FTRACE_MCOUNT_RECORD=y
CONFIG_FTRACE=y
CONFIG_FTRACE_SYSCALLS=y
CONFIG_DYNAMIC_FTRACE=y
[...]
ftrace: tracepoints

$ mount -t debugfs nodev /sys/kernel/debug

/sys/kernel/debug/tracing # head available_events
kvmmmu:kvm_mmu_pagetable_walk
kvmmmu:kvm_mmu_paging_element
kvmmmu:kvm_mmu_set_accessed_bit
[...]

/sys/kernel/debug/tracing$ cat available_events | wc -l
1832
ftrace: tracepoints

Grouped by system:

608 syscalls
364 xfs
145 cfg80211
114 mac80211
  65 kvm
  48 btrfs
  39 i915
  35 xen
  28 writeback
  24 sched
[...]

26
ftrace example: I/O

$ echo nop > current_tracer
$ echo 1 > events/block/block_rq_issue/enable
$ echo 1 > events/block/block_rq_complete/enable
$ cat trace_pipe

dd-6195  [000] d..2 24591.455048: block_rq_issue: 8,0 W 0 () 414091752 + 1344 [dd]
dd-6195  [000] d..2 24591.455078: block_rq_issue: 8,0 W 0 () 414093096 + 704 [dd]
dd-6195  [000] d..2 24591.455211: block_rq_issue: 8,0 W 0 () 414093800 + 1344 [dd]
dd-6195  [000] d..2 24591.455219: block_rq_issue: 8,0 W 0 () 414095144 + 704 [dd]
dd-6195  [000] d..2 24591.455345: block_rq_issue: 8,0 W 0 () 414095848 + 1368 [dd]
dd-6195  [000] d..2 24591.455353: block_rq_issue: 8,0 W 0 () 414097216 + 680 [dd]
dd-6195  [000] d..2 24591.455476: block_rq_issue: 8,0 W 0 () 414097896 + 1344 [dd]
dd-6195  [000] d..2 24591.455488: block_rq_issue: 8,0 W 0 () 414099240 + 704 [dd]
dd-6195  [000] d..2 24591.455594: block_rq_issue: 8,0 W 0 () 414099944 + 1352 [dd]
dd-6195  [000] d..2 24591.455601: block_rq_issue: 8,0 W 0 () 414101296 + 456 [dd]
ftrace canned scripts: perf-tool

https://github.com/brendangregg/perf-tools

bitesize     - show disk I/O size as a histogram.
cachestat   - Measure page cache hits/misses.
execsnoop   - trace process exec() with arguments.
funccount   - count kernel function calls matching specified wildcards.
iolatency   - summarize block device I/O latency as a histogram.
iosnoop     - trace block I/O events as they occur.
killsnoop   - trace kill() syscalls with process and signal details.
kprobe      - trace a given kprobe definition. Kernel dynamic tracing.
opensnoop   - trace open() syscalls with file details.
tcpretrans  - show TCP retransmits, with address and other details.
ftrace CLI: trace-cmd

$ man -k trace-cmd
trace-cmd (1) - interacts with Ftrace Linux kernel internal tracer
trace-cmd-check-events (1) - parse the event formats on local system
trace-cmd-extract (1) - extract out the data from the Ftrace Linux tracer.
trace-cmd-hist (1) - show histogram of events in trace.dat file
trace-cmd-list (1) - list available plugins, events or options for Ftrace.
trace-cmd-listen (1) - listen for incoming connection to record tracing.
trace-cmd-mem (1) - show memory usage of certain kmem events
trace-cmd-options (1) - list available options from trace-cmd plugins
trace-cmd-profile (1) - profile tasks running live
trace-cmd-record (1) - record a trace from the Ftrace Linux internal tracer
trace-cmd-report (1) - show in ASCII a trace created by trace-cmd record
trace-cmd-reset (1) - turn off all Ftrace tracing to bring back full performance
trace-cmd-restore (1) - restore a failed trace record
trace-cmd-show (1) - show the contents of the Ftrace Linux kernel tracing buffer.
trace-cmd-snapshot (1) - take, reset, free, or show a Ftrace kernel snapshot
trace-cmd-split (1) - split a trace.dat file into smaller files
trace-cmd-stack (1) - read, enable or disable Ftrace Linux kernel stack tracing.
trace-cmd-start (1) - start the Ftrace Linux kernel tracer without recording
trace-cmd-stat (1) - show the status of the tracing (ftrace) system
trace-cmd-stop (1) - stop the Ftrace Linux kernel tracer from writing to the ring buffer.
trace-cmd-stream (1) - stream a trace to stdout as it is happening
trace-cmd.dat (5) - trace-cmd file format
ftrace GUI: kernelshark

http://people.redhat.com/srostedt/kernelshark/HTML/
ftrace: resources

- Documentation/trace/ftrace.txt
- http://lwn.net/Kernel/Index/ (search for “ftrace”, “tracing”)
- https://github.com/brendangregg/perf-tools
- linux-trace-users@vger.kernel.org mailing list
**eBPF**

- extended Berkeley Packet Filter
- BPF is kernel since a long time. Foundation for tcpdump
- eBPF merged in 4.1, ~2015
- Primary developer is Alexei Starovoitov
- In-kernel data aggregation
- Programs are statically verified, “safe” and low overhead
- Programs can be interpreted or Just-In-Time compiled (as opposed to a separate kernel module)
- C syntax, but reduced functionality (no loops, etc)
- Supported as backend by perf and soon SystemTap
eBPF

• An eBPF program can only access data on its stack
• Cannot deref an arbitrary pointer
• ...
• BCC (BPF Compiler Collection) makes this less painful
  • https://github.com/iovisor/bcc
    – https://www.youtube.com/watch?v=eGl bouHkY PU In-Kernel Low Latency Tracing and Networking, Brenden Blanco
• Examples in tree at samples/bpf
• Useful application: frequency-count of stack traces
• http://lwn.net/Kernel/Index/, search for “ebpf”
SystemTap

- Available since 2005
- The most powerful in terms of expressivity
- Out of tree
- Requires kernels debug symbols
- Maintained by Frank Ch. Eigler et al.
- Great care is needed
- Supports tracepoints, kprobes, uprobes
- Can even do live patching!
  - See “Applying band-aids over security wounds with systemtap”
    https://archive.fosdem.org/2016/schedule/event/systemtap/
probepoint: a kernel event
stmt: a statement in the stap language
- see https://sourceware.org/systemtap/langref/langref.html
examples at https://sourceware.org/systemtap/examples/
see also man -k stap
• perf: robust and reliable. Limited flexibility.
• ftrace: available on all kernels, highly programmable. Requires knowledge of tracepoints.
• eBPF: safe, low overhead. Not easy to program. Needs recent kernel.
• SystemTap: best programmability. Overhead can be hard to predict.

Questions!

http://www.suse.com/jobs