Profiling with the Linux perf tools

Introduction to the perf tools

Namhyung Kim

Overview

- Modern, actively-developed performance analysis tool
- Monitoring H/W PMU counter as well as S/W events
- Counting, sampling or tracing
- GIT-like architecture
- NO daemons required
- Callchains (stack backtrace) supported
- perf.data file per session
- Support for perl/python scripting

List of commands

```
usage: perf [--version] [--help] COMMAND [ARGS]

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
  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.
  trace           strace inspired tool
  probe           Define new dynamic tracepoints

See 'perf help COMMAND' for more information on a specific command.
```

How to use the perf?

- TWO things you need to know (at least)
- event
  - performance events what you want to measure
  - ex. cpu cycles, cache misses, page faults
- target
  - where those events come from
  - ex. speficic process or thread, system-wide

Specifying events

- ‘perf list‘ to get full list of supported events
- Pre-defined (architectural) H/W events
- H/W Raw events (hex numbers)
- S/W and tracepoint events

```
List of pre-defined events (to be used in -e):
  cpu-cycles OR cycles                                [Hardware event]
  instructions                                        [Hardware event]
  cache-references                                    [Hardware event]
  cache-misses                                        [Hardware event]
  branch-instructions OR branches                     [Hardware event]
  branch-misses                                       [Hardware event]

  cpu-clock                                           [Software event]
  task-clock                                          [Software event]
  page-faults OR faults                               [Software event]
  context-switches OR cs                              [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]
```

Specifying targets

- Processes (-p *pids*)
- Threads (-t *tids*)
- User (-u *uid*)
- System-wide (-a)
- Cpus (-C *cpus*)
- Cgroups (-G *cgroup*)
- Workload (*command line*)

Basic usage

- Counting number of events
  - perf stat [*events*] *target*
- Collecting event samples
  - perf record [*events*] *target*
- Analyzing the samples
  - perf report
- Live Profiling
  - perf top [*events*] [*target*]

Stat: Counting events

```
$ perf stat ls > /dev/null

 Performance counter stats for 'ls':

         0.657039 task-clock                #    0.713 CPUs utilized
                0 context-switches          #    0.000 K/sec
                0 cpu-migrations            #    0.000 K/sec
              262 page-faults               #    0.399 M/sec
        1,761,029 cycles                    #    3.280 GHz
        1,092,764 stalled-cycles-frontend   #   62.05% frontend cycles idle
          858,094 stalled-cycles-backend    #   48.73% backend  cycles idle
        1,403,788 instructions              #    0.80  insns per cycle
                                            #    0.78  stalled cycles per insn
          386,793 branches                  #  588.691 M/sec
           14,343 branch-misses             #    3.71% of all branches         [80.59%]

      0.000921599 seconds time elapsed
```

Stat: Run workload multiple times

- To reduce variability

```
$ perf stat -e task-clock,cycles,context-switches,page-faults,instructions -r 10 -- noploop 1

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

       1000.308987 task-clock                #    1.000 CPUs utilized            ( +-  0.00% )
     3,729,474,643 cycles                    #    3.728 GHz                      ( +-  0.11% )
                 2 context-switches          #    0.002 K/sec                    ( +-  8.55% )
               110 page-faults               #    0.110 K/sec                    ( +-  0.09% )
     3,594,555,795 instructions              #    0.96  insns per cycle          ( +-  0.39% )

       1.000582770 seconds time elapsed                                          ( +-  0.00% )
```

Stat: Various options

```
$ perf stat -h

 usage: perf stat [<options>] [<command>]

    -e, --event <event>   event selector. use 'perf list' to list available events
        --filter <filter>
                          event filter
    -i, --no-inherit      child tasks do not inherit counters
    -p, --pid <pid>       stat events on existing process id
    -t, --tid <tid>       stat events on existing thread id
    -a, --all-cpus        system-wide collection from all CPUs
    -g, --group           put the counters into a counter group
    -c, --scale           scale/normalize counters
    -v, --verbose         be more verbose (show counter open errors, etc)
    -r, --repeat <n>      repeat command and print average + stddev (max: 100)
    -n, --null            null run - dont start any counters
    -d, --detailed        detailed run - start a lot of events
    -S, --sync            call sync() before starting a run
    -B, --big-num         print large numbers with thousands' separators
    -C, --cpu <cpu>       list of cpus to monitor in system-wide
    -A, --no-aggr         disable CPU count aggregation
    -x, --field-separator <separator>
                          print counts with custom separator
    -G, --cgroup <name>   monitor event in cgroup name only
    -o, --output <file>   output file name
        --append          append to the output file
        --log-fd <n>      log output to fd, instead of stderr
```

Sampling

- Default: cpu cycles event
- Default: 4 KHz frequency (estimated)
- No daemons
- Record + report
- Records system information
- Records build-ids of DSOs with samples
- Support call-stack recording (fp or cfi)

Recording event samples

```
# perf record -h
 usage: perf record [<options>] [<command>]
    or: perf record [<options>] -- <command> [<options>]

    -e, --event <event>   event selector. use 'perf list' to list available events
        --filter <filter>
                          event filter
    -p, --pid <pid>       record events on existing process id
    -t, --tid <tid>       record events on existing thread id
    -a, --all-cpus        system-wide collection from all CPUs
    -C, --cpu <cpu>       list of cpus to monitor
    -c, --count <n>       event period to sample
    -o, --output <file>   output file name
    -i, --no-inherit      child tasks do not inherit counters
    -F, --freq <n>        profile at this frequency
    -m, --mmap-pages <n>  number of mmap data pages
        --group           put the counters into a counter group
    -g, --call-graph <mode[,dump_size]>
                          do call-graph (stack chain/backtrace) recording: [fp] dwarf
    -N, --no-buildid-cache
                          do not update the buildid cache
    -B, --no-buildid      do not collect buildids in perf.data
    -G, --cgroup <name>   monitor event in cgroup name only
    -u, --uid <user>      user to profile
    ...

# perf record -a sleep 1
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.012 MB perf.data (~508 samples) ]
```

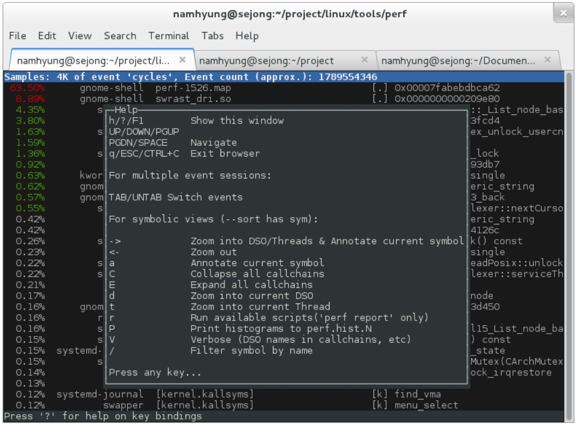
Report

- Supports 3 kind of UI frontend
  - stdio
  - TUI - interactive (using newt/slang)
  - GUI - basic browsing only (using GTK2)
- Supports various sort keys
  - pid, comm, dso, symbol, cpu, ...
- Filtering by thread, dso, symbol
- TUI interactivity
  - Integrated annotation
  - foldable callchain
  - changing filter

Report: stdio output

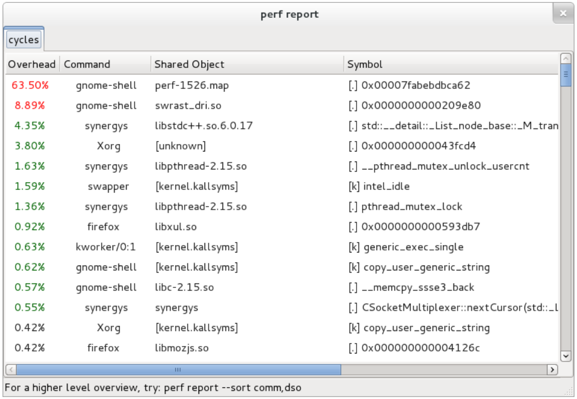
```
# perf report --stdio
# ========
# captured on: Fri Nov 30 20:14:59 2012
# hostname : sejong.aot.lge.com
# os release : 3.7.0-rc6
# perf version : 3.7.rc7.g662355
# arch : x86_64
# nrcpus online : 12
# nrcpus avail : 12
# cpudesc : Intel(R) Core(TM) i7-3930K CPU @ 3.20GHz
# cpuid : GenuineIntel,6,45,7
# total memory : 32851944 kB
# cmdline : /home/namhyung/project/linux/tools/perf/perf record -a sleep 1
# event : name = cycles, type = 0, config = 0x0, config1 = 0x0, config2 = 0x0, excl_usr = 0, excl_kern = 0, excl_host
# HEADER_CPU_TOPOLOGY info available, use -I to display
# pmu mappings: cpu = 4, software = 1, uncore_pcu = 13, tracepoint = 2, uncore_imc_0 = 15, uncore_imc_1 = 16, uncore_
# ========
#
# Samples: 4K of event 'cycles'
# Event count (approx.): 1789554346
#
# Overhead          Command       Shared Object                                                         Symbol
# ........  ...............  ..................  .............................................................
#
    63.50%      gnome-shell  perf-1526.map       [.] 0x00007fabebdbca62
     8.89%      gnome-shell  swrast_dri.so       [.] 0x0000000000209e80
     4.35%         synergys  libstdc++.so.6.0.17  [.] std::__detail::_List_node_base::_M_transfer(std::__detail:
     3.80%             Xorg  [unknown]           [.] 0x000000000043fcd4
     1.63%         synergys  libpthread-2.15.so  [.] __pthread_mutex_unlock_usercnt
     1.59%          swapper  [kernel.kallsyms]   [k] intel_idle
     1.36%         synergys  libpthread-2.15.so  [.] pthread_mutex_lock
     0.92%          firefox  libxul.so           [.] 0x0000000000593db7
...
```

Report: TUI output


namhyung@sejong:~/project/linux/tools/perf
File Edit View Search Terminal Tabs Help
namhyung@sejong:~/project/li...
namhyung@sejong:~/project
namhyung@sejong:~/Documen...

```
Samples: 4K of event 'cycles', Event count (approx.): 1789554346
 63.50%      gnome-shell  perf-1526.map                        [.] 0x00007fabebdbca62
  8.89%      gnome-shell  swrast_dri.so                        [.] 0x0000000000209e80
  4.35%         s┌─Help────────────────────────────────────────────────────────┐::_List_node_bas
  3.80%          │h/?/F1        Show this window                               │3fcd4
  1.63%         s│UP/DOWN/PGUP                                                 │ex_unlock_usercn
  1.59%          │PGDN/SPACE    Navigate                                       │
  1.36%         s│q/ESC/CTRL+C  Exit browser                                   │_lock
  0.92%          │                                                             │93db7
  0.63%      kwor│For multiple event sessions:                                 │single
  0.62%      gnom│                                                             │eric_string
  0.57%      gnom│TAB/UNTAB Switch events                                      │3_back
  0.55%         s│                                                             │lexer::nextCurso
  0.42%          │For symbolic views (--sort has sym):                         │eric_string
  0.42%          │                                                             │4126c
  0.26%         s│->            Zoom into DSO/Threads & Annotate current symbol│k() const
  0.23%          │<-            Zoom out                                       │single
  0.22%         s│a             Annotate current symbol                        │eadPosix::unlock
  0.22%         s│C             Collapse all callchains                        │lexer::serviceTh
  0.21%          │E             Expand all callchains                          │
  0.17%          │d             Zoom into current DSO                          │node
  0.16%      gnom│t             Zoom into current Thread                       │3d450
  0.16%         r│r             Run available scripts('perf report' only)      │
  0.16%         s│P             Print histograms to perf.hist.N                │l15_List_node_ba
  0.15%         s│V             Verbose (DSO names in callchains, etc)         │) const
  0.15%  systemd-│/             Filter symbol by name                          │_state
  0.15%         s│                                                             │Mutex(CArchMutex
  0.14%          │Press any key...                                             │ock_irqrestore
  0.13%          └─────────────────────────────────────────────────────────────┘
  0.12%  systemd-journal  [kernel.kallsyms]                    [k] find_vma
  0.12%          swapper  [kernel.kallsyms]                    [k] menu_select
Press '?' for help on key bindings
```

Report: GTK2 output



```
Overhead  Command      Shared Object           Symbol
63.50%    gnome-shell  perf-1526.map           [.] 0x00007fabebdbca62
8.89%     gnome-shell  swrast_dri.so           [.] 0x0000000000209e80
4.35%     synergys     libstdc++.so.6.0.17     [.] std::__detail::_List_node_base::_M_tran
3.80%     Xorg         [unknown]               [.] 0x000000000043fcd4
1.63%     synergys     libpthread-2.15.so      [.] __pthread_mutex_unlock_usercnt
1.59%     swapper      [kernel.kallsyms]       [k] intel_idle
1.36%     synergys     libpthread-2.15.so      [.] pthread_mutex_lock
0.92%     firefox      libxul.so               [.] 0x0000000000593db7
0.63%     kworker/0:1  [kernel.kallsyms]       [k] generic_exec_single
0.62%     gnome-shell  [kernel.kallsyms]       [k] copy_user_generic_string
0.57%     gnome-shell  libc-2.15.so            [.] __memcpy_ssse3_back
0.55%     synergys     synergys                [.] CSocketMultiplexer::nextCursor(std::_l
0.42%     Xorg         [kernel.kallsyms]       [k] copy_user_generic_string
0.42%     firefox      libmozjs.so             [.] 0x000000000004126c

For a higher level overview, try: perf report --sort comm,dso
```

Report: Customize output using sort keys

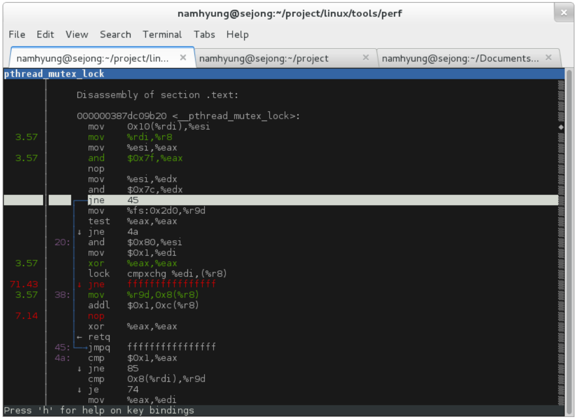
- Collect/sort sample histograms based on:
  - cpu - cpu number
  - pid - process ID
  - comm - process name
  - dso - name of executable or library
  - symbol - function name
  - parent - caller of this function (requires callchain)
- Default: comm,dso,symbol
- Default parent regex: ^sys_ |^do_page_fault
- Filtering by thread, dso, symbol

Callchain example

Needs to be recorded with -g option

```
# cat perf.hist.5
-  100.00%  sleep  libc-2.12.so  [.] malloc
   - malloc
      - 45.16% __strdup
         + 85.71% setlocale
         + 7.14% _nl_load_locale_from_archive
         + 7.14% __textdomain
      + 38.71% _nl_intern_locale_data
      + 6.45% _nl_normalize_codeset
      + 3.23% _nl_load_locale_from_archive
      - 3.23% new_composite_name
           setlocale
           0x4014ec
           __libc_start_main
           0x4011f9
      + 3.23% set_binding_values
```

Annotation


namhyung@sejong:~/project/linux/tools/perf
File Edit View Search Terminal Tabs Help
namhyung@sejong:~/project/lin...
namhyung@sejong:~/project
namhyung@sejong:~/Documents...

```
pthread_mutex_lock

              Disassembly of section .text:

              000000387dc09b20 <__pthread_mutex_lock>:
                mov    0x10(%rdi),%esi
  3.57          mov    %rdi,%r8
                mov    %esi,%eax
  3.57          and    $0x7f,%eax
                nop
                mov    %esi,%edx
                and    $0x7c,%edx
                jne    45
                mov    %fs:0x2d0,%r9d
                test   %eax,%eax
              ↓ jne    4a
         20:    and    $0x80,%esi
                mov    $0x1,%edi
  3.57          xor    %eax,%eax
                lock   cmpxchg %edi,(%r8)
 71.43        ↓ jne    ffffffffffffffff
  3.57   38:    mov    %r9d,0x8(%r8)
                addl   $0x1,0xc(%r8)
  7.14          nop
                xor    %eax,%eax
              ← retq
         45:    jmpq   ffffffffffffffff
         4a:    cmp    $0x1,%eax
              ↓ jne    85
                cmp    0x8(%rdi),%r9d
              ↓ je     74
                mov    %eax,%edi
Press 'h' for help on key bindings
```

Live profiling

- perf top
- Record + report at the same time
- No perf.data file generated
- Reuses report TUI interactive interface
- Live decaying histograms
- System-wide monitoring by default

Tracing kernel events

- There are so many tracepoints in kernel
  - needs CONFIG_EVENT_TRACING enabled
  - 960 (= perf list |grep Tracepoint |wc -l)
- They can be used as perf events
- They contain useful information
- It can trace every occurrence of such events
- Usual perf record + report works greatly
- Run perf script if you want to see the raw event information

Advanced features

- diff - Differential profiling
- probe - Dynamic probe for kernel/user-space
- script - Run or generate script for recorded events
- sched - Trace/measure scheduler properties (latency)
- lock - Analyze kernel locking events
- kvm - Trace/measure kvm guest os properties
- trace - ’strace’ inspired tool
- timechart - Generate a SVG image of system behavior

Differential Profiling

- 'perf diff' for comparing performance results (perf.data)
  - perf record
  - *do your job...*
  - perf record
  - perf diff
- Differential profiling
  - paper from Paul McKenney
  - Delta profiling (default)
    - new_percent - old_percent
  - Ratio differential profiling
    - new_period / old_period
  - Weighted differential profilng
    - new_period * w2 - old_period * w1

Dynamic Probes

- use kprobes/uprobes to add tracepoints
  - self-modifying code
  - kernel provides a way to add/remove probes
    - /sys/kernel/debug/tracing/[ku]probe_events
  - support arguments
    - register, local variables, return value
- use ELF/DWARF information for function/variable name
  - requires libelf, libdw, libdwfl from elfutils
  - can apply filter using wildcard

Kprobes

- Listing probeable lines in function

```
# perf probe -L __schedule
<__schedule@/home/namhyung/project/linux/kernel/sched/core.c:0>
      0 static void __sched __schedule(void)
      1 {
                 struct task_struct *prev, *next;
                 unsigned long *switch_count;
                 struct rq *rq;
                 int cpu;

         need_resched:
       8         preempt_disable();
       9         cpu = smp_processor_id();
      10         rq = cpu_rq(cpu);
      11         rcu_note_context_switch(cpu);
```

- Show avaiable variables on each line

```
# perf probe -V __schedule:11
Available variables at __schedule:11
        @<__schedule+94>
                int     cpu
                struct rq*     rq
                struct task_struct    next;
```

Uprobes

- Listing probeable functions in userspace DSO

```
# perf probe -F -x /lib64/libc-2.15.so | grep ^m | head -5
madvise
malloc
malloc@plt
malloc_info
mblen
```

- Adding probes

```
# perf probe -x /lib64/libc-2.15.so -a malloc
Added new event:
  probe_libc:malloc    (on 0x79b80)

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

        perf record -e probe_libc:malloc -aR sleep 1
```

Trace

- New ’perf trace’ command for tracing system call events
- Aim at strace-like easy to use tool
- Support live mode only currently
- Will include more information like page faults
- Will be able to add some kernel events also
- Initial work by Thomas Gleixner and Ingo Molnar
    - `http://lwn.net/Articles/415728/`

Tracing with perf trace

```
# perf trace usleep 1

     0.000 ( 0.000 ms):  ... [continued]: read()) = 0
Problems reading syscall 59(execve) information
     0.027 ( 0.000 ms):  ... [continued]: execve()) = -2
     0.043 ( 0.015 ms): execve(arg0: 140733994181707, arg1: 140733994192272, arg2: 26938192, arg3: 242620
     0.047 ( 0.003 ms): execve(arg0: 140733994181714, arg1: 140733994192272, arg2: 26938192, arg3: 242620
     0.052 ( 0.004 ms): execve(arg0: 140733994181718, arg1: 140733994192272, arg2: 26938192, arg3: 242620
     0.388 ( 0.335 ms): execve(arg0: 140733994181724, arg1: 140733994192272, arg2: 26938192, arg3: 242620
     0.406 ( 0.001 ms): brk(brk: 0                                                        ) = 2156134
     0.421 ( 0.003 ms): mmap(addr: 0, len: 4096, prot: 3, flags: 34, fd: 4294967295, off: 0   ) = -779063
     0.434 ( 0.005 ms): access(filename: 242615439840, mode: 4                            ) = -1 ENOE
     0.501 ( 0.004 ms): open(filename: 139903485625561, flags: 524288, mode: 242617553576     ) = 4
     0.504 ( 0.001 ms): read(fd: 4, buf: 140733881537160, count: 832                      ) = 832
     0.506 ( 0.001 ms): fstat(fd: 4, statbuf: 140733881536816                             ) = 0
     0.511 ( 0.003 ms): mmap(addr: 242619514880, len: 3892376, prot: 5, flags: 2050, fd: 4, off: 0) = 210
     0.515 ( 0.003 ms): mprotect(start: 242621267968, len: 2097152, prot: 0               ) = 0
     0.520 ( 0.003 ms): mmap(addr: 242623365120, len: 24576, prot: 3, flags: 2066, fd: 4, off: 1753088) =
     0.526 ( 0.003 ms): mmap(addr: 242623389696, len: 17560, prot: 3, flags: 50, fd: 4294967295, off: 0) =
     0.529 ( 0.001 ms): close(fd: 4                                                       ) = 0
     0.534 ( 0.002 ms): mmap(addr: 0, len: 4096, prot: 3, flags: 34, fd: 4294967295, off: 0   ) = -779169
     0.544 ( 0.002 ms): mmap(addr: 0, len: 8192, prot: 3, flags: 34, fd: 4294967295, off: 0   ) = -779177
     0.548 ( 0.001 ms): arch_prctl(option: 4098, arg2: 139903485523776, arg3: 139903485526096, arg4: 34, a
     0.569 ( 0.003 ms): mprotect(start: 242680365056, len: 4096, prot: 1                  ) = 0
     0.573 ( 0.003 ms): mprotect(start: 242623365120, len: 16384, prot: 1                 ) = 0
     0.576 ( 0.003 ms): mprotect(start: 242617544704, len: 4096, prot: 1                  ) = 0
     0.582 ( 0.004 ms): munmap(addr: 139903485534208, len: 98465                          ) = 0
     0.635 ( 0.001 ms): brk(brk: 0                                                        ) = 2156134
     0.638 ( 0.002 ms): brk(brk: 21696512                                                 ) = 2169651
     0.640 ( 0.001 ms): brk(brk: 0                                                        ) = 2169651
     0.708 ( 0.056 ms): nanosleep(rqtp: 140733881538896, rmtp: 0                          ) = 0
     0.713 ( 0.000 ms): exit_group(error_code: 0
```

Scripting

- Show each sample in (chronological) order by timestamp
- Use script languages to process events
- Currently perl and python are supported
    - Require development packages installed
- Several scripts are available
- Run existing script to perf.data
- Generate script from perf.data
- Support non-tracepoint event samples
- Script browser patch submitted

Available scripts

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

Generating script

```
# perf record -a -e sched:sched_wakeup sleep 1
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.351 MB perf.data (~15347 samples) ]
#
# perf script -g python
generated Python script: perf-script.py
#
# cat perf-script.py
def trace_begin():
        print "in trace_begin"

def trace_end():
        print "in trace_end"

def sched__sched_wakeup(event_name, context, common_cpu,
        common_secs, common_nsecs, common_pid, common_comm,
        comm, pid, prio, success,
        target_cpu):
                print_header(event_name, common_cpu, common_secs, common_nsecs,
                        common_pid, common_comm)

                print "comm=%s, pid=%u, prio=%u, " \
                "success=%u, target_cpu=%u\n" % \
                (comm, pid, prio, success,
                target_cpu),

def print_header(event_name, cpu, secs, nsecs, pid, comm):
        print "%-20s %5u %05u.%09u %8u %-20s " % \
        (event_name, cpu, secs, nsecs, pid, comm),
```

Running script

```
# perf script -s perf-script.py
in trace_begin
sched__sched_wakeup   3 24794.823739899  13044 perf          comm=perf, pid=13045, prio=120, success=1, ta
sched__sched_wakeup   9 24794.823800642  13045 perf          comm=migration/9, pid=50, prio=0, success=1,
sched__sched_wakeup   0 24794.824777438   1412 gnome-shell   comm=kworker/0:1, pid=75, prio=120, success=
sched__sched_wakeup   5 24794.826330919   1413 gnome-shell   comm=gnome-shell, pid=1411, prio=120, success
sched__sched_wakeup   5 24794.826332752   1413 gnome-shell   comm=gnome-shell, pid=1414, prio=120, success
sched__sched_wakeup   5 24794.826333622   1413 gnome-shell   comm=gnome-shell, pid=1409, prio=120, success
sched__sched_wakeup   5 24794.826334675   1413 gnome-shell   comm=gnome-shell, pid=1416, prio=120, success
sched__sched_wakeup   5 24794.826335456   1413 gnome-shell   comm=gnome-shell, pid=1415, prio=120, success
sched__sched_wakeup   5 24794.826336199   1413 gnome-shell   comm=gnome-shell, pid=1412, prio=120, success
sched__sched_wakeup   5 24794.826336899   1413 gnome-shell   comm=gnome-shell, pid=1410, prio=120, success
sched__sched_wakeup  10 24794.826376847   1409 gnome-shell   comm=gnome-shell, pid=1397, prio=120, success
sched__sched_wakeup   1 24794.826475446   1397 gnome-shell   comm=Xorg, pid=940, prio=120, success=1, targ
sched__sched_wakeup   2 24794.826895617    940 Xorg          comm=gnome-shell, pid=1397, prio=120, success
sched__sched_wakeup   3 24794.826929411   1397 gnome-shell   comm=Xorg, pid=940, prio=120, success=1, targ
sched__sched_wakeup   4 24794.826970603    940 Xorg          comm=gnome-shell, pid=1397, prio=120, success
sched__sched_wakeup   4 24794.826973342    940 Xorg          comm=firefox, pid=10371, prio=120, success=1
sched__sched_wakeup   0 24794.827043688  10371 firefox       comm=firefox, pid=10342, prio=120, success=1
sched__sched_wakeup   4 24794.827087472    940 Xorg          comm=gnome-shell, pid=1397, prio=120, success
sched__sched_wakeup   9 24794.827138617  10342 firefox       comm=firefox, pid=10371, prio=120, success=1
...
in trace_end
```

It'd be wonderful if you submit your great script to upstream!

KVM Support

- Collect guest OS statistics from host
- Use –pid to specify specific guest
- Need to specify guest vmlinux, kallsyms or modules info
- Or –guestmount directory with sshfs mounted per pid subdirs
- ’perf kvm (top/record/report/diff/buildid-list)’
    - New ’stat’ subcommand added recently
    - Currently vmexit, mmio, ioport events are supported

perf kvm stat

```
# pgrep qemu
12027
#
# perf kvm stat record -p 12027
^C[ perf record: Woken up 6 times to write data ]
[ perf record: Captured and wrote 12.837 MB perf.data.guest (~560846 samples) ]
#
# perf kvm stat report --event=vmexit


Analyze events for all VCPUs:

             VM-EXIT    Samples  Samples%     Time%         Avg time

      IO_INSTRUCTION      17094    30.12%     0.20%       7.24us ( +-   5.05% )
         APIC_ACCESS      13546    23.87%     0.10%       4.47us ( +-   4.58% )
       EPT_VIOLATION      10146    17.88%     0.02%       1.29us ( +-   4.54% )
   PAUSE_INSTRUCTION       6660    11.73%     0.01%       0.97us ( +-   1.89% )
                 HLT       3380     5.96%    99.63%  18632.56us ( +-   7.74% )
  EXTERNAL_INTERRUPT       2006     3.53%     0.03%       8.92us ( +-   8.10% )
           CR_ACCESS       1482     2.61%     0.01%       2.36us ( +-   1.83% )
   PENDING_INTERRUPT        827     1.46%     0.00%       1.37us ( +-   3.31% )
       EXCEPTION_NMI        773     1.36%     0.00%       2.54us ( +-   2.54% )
       EPT_MISCONFIG        587     1.03%     0.01%      10.32us ( +-   3.47% )
               CPUID        253     0.45%     0.00%       1.14us ( +-   4.04% )

Total Samples:56754, Total events handled time:63212778.18us.
```

Thanks!

- Any questions?