The New Linux ’perf’ tools

Linux Kongress
September, 2010

Arnaldo Carvalho de Melo
acme@redhat.com
Presentation Flow

- Motivation
- Focus on the tools
- But some kernel details will be mentioned
- Some demos hopefully at the end
How did I get involved?

- I am no specialist on performance counters
- pahole & the dwarves
- ELF, DWARF, symtabs, dynsym, relocations, etc
- ftrace
How did I get involved? Part two

- Part of the Red Hat Real Time team
- We need to discover why 100us deadlines are not being met
- Why this is slower on your RT kernel than on the RHEL one?
- Observability tooling!
- Huge educational value, use it!
Renewed interest in profiling tools

- Complexity of systems growing
- Pervasiveness of multithreading
- Hardware assists
Performance Counters

- Performance counters are special hardware registers
- Available on most modern CPUs
- Count the number of some hw events
  - instructions executed
  - cache-misses suffered
  - branches mispredicted
- Without slowing down the kernel or applications
- Can trigger interrupts when a number of events have passed
Limited resource:

Some are programmable, some are for specific events.

Processor:

- UltraSparc 2
- Pentium III 2
- Athlon 4
- IA-64 4
- POWER4 8
- Pentium IV 18
- Nehalem 7
Tracepoints

Static probe points that are put in place by subsystem maintainers and that can be enabled later.
Dynamic probe points

Dynamicly inserted probe points using hardware breakpoints.
The oprofile development problem

-Disconnected kernel & userspace development
-Linus problem with Atom and Nehalem support
-Less of the "2 broken pieces" approach -> one working piece
-http://lwn.net/Articles/339406/
The perf user interface approach

- git like
- Many subcommands
- Per thread/per workload/per CPU/system wide
- No daemons
The perf development approach

- Tools hosted in the kernel sources: tools/perf/
- Subcommands can be developed largely independently
- Developers expected to touch both sides (kernel/user)
- Written in the C idiom used in the kernel
- Shares code with the kernel (rbtree, list, more to come)
The new implementation approach

- Just one new syscall: sys_perf_counter_open
  - Returns a file descriptor
  - read/write/mmap/close/fcntl/ioctl/poll work as usual
  - Per thread/cpu/whole system
  - Transparent inheritance support
    - Full workloads can be measured
    - Without using ptrace methods to follow forks & clones
  - Events mapped to closest per arch hw counter
  - Possible to use raw events
  - Supports tracepoints
  - Software counters (hrtimer based or not)
  - Dynamic probes (kprobes, uprobes)
sys_perf_counter_open - The syscall

- event type attributes for monitoring/sampling
- target pid
- target cpu
- group_fd
- flags
sys_perf_counter_open - event type

- PERF_TYPE_HARDWARE
- PERF_TYPE_SOFTWARE
- PERF_TYPE_TRACEPOINT
- PERF_TYPE_HW_CACHE
- PERF_TYPE_RAW (for raw tracepoint data)
sys_perf_counter_open - attr.sample_type

- bitmask
  - PERF_SAMPLE_IP
  - PERF_SAMPLE_TID
  - PERF_SAMPLE_TIME
  - PERF_SAMPLE_CALLCHAIN
  - PERF_SAMPLE_ID
  - PERF_SAMPLE_CPU
sys_perf_counter_open - attr config bitfield

- disabled: off by default
- inherit: children inherit it
- exclude_{user,kernel,hv,idle}: don’t count these
- mmap: include mmap data
- comm: include comm data
- inherit_stat: per task counts
- enable_on_exec: next exec enables
Architectures already supported

- x86: p6, core+, k7+, p4
- ppc64
- sparc: ultra 3 & 4
- arm: v5 (xscale), v6, v7 (Cortex A8 & A9)
- alpha: EV56 and later
- sh: 4A
- Others supporting just software/ftrace events
Tools

.git like: subcomands

$ perf help
  annotate  Read perf.data and display annotated code
  archive   Create archive with object files with build-ids
  diff      Read perf.data files and display differential profile
  kmem      Tool to trace/measure kernel memory(slab) properties
  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 and display the profile
  sched     Tool to trace/measure scheduler properties (latencies)
  stat      Run a command and gather performance counter statistics
  top       System profiling tool.
  trace     Read perf.data and display trace output
$ perf list
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]
    bus-cycles           [Hardware event]
perf list - continued

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]
perf list - continued

<table>
<thead>
<tr>
<th>Metric</th>
<th>Event Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1-dcache-loads</td>
<td>[Hardware cache event]</td>
</tr>
<tr>
<td>L1-dcache-load-misses</td>
<td>[Hardware cache event]</td>
</tr>
<tr>
<td>L1-dcache-stores</td>
<td>[Hardware cache event]</td>
</tr>
<tr>
<td>L1-dcache-store-misses</td>
<td>[Hardware cache event]</td>
</tr>
<tr>
<td>L1-dcache-prefetches</td>
<td>[Hardware cache event]</td>
</tr>
<tr>
<td>L1-dcache-prefetch-misses</td>
<td>[Hardware cache event]</td>
</tr>
<tr>
<td>L1-icache-loads</td>
<td>[Hardware cache event]</td>
</tr>
<tr>
<td>L1-icache-load-misses</td>
<td>[Hardware cache event]</td>
</tr>
<tr>
<td>L1-icache-prefetches</td>
<td>[Hardware cache event]</td>
</tr>
<tr>
<td>L1-icache-prefetch-misses</td>
<td>[Hardware cache event]</td>
</tr>
</tbody>
</table>
perf list - continued

<table>
<thead>
<tr>
<th>Event</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>LLC-loads</td>
<td>[Hardware cache event]</td>
</tr>
<tr>
<td>LLC-load-misses</td>
<td>[Hardware cache event]</td>
</tr>
<tr>
<td>LLC-stores</td>
<td>[Hardware cache event]</td>
</tr>
<tr>
<td>LLC-store-misses</td>
<td>[Hardware cache event]</td>
</tr>
<tr>
<td>LLC-prefetches</td>
<td>[Hardware cache event]</td>
</tr>
<tr>
<td>LLC-prefetch-misses</td>
<td>[Hardware cache event]</td>
</tr>
</tbody>
</table>
perf list - continued

dTLB-loads [Hardware cache event]
dTLB-load-misses [Hardware cache event]
dTLB-stores [Hardware cache event]
dTLB-store-misses [Hardware cache event]
dTLB-prefetches [Hardware cache event]
dTLB-prefetch-misses [Hardware cache event]
iTLB-loads [Hardware cache event]
iTLB-load-misses [Hardware cache event]
branch-loads [Hardware cache event]
branch-load-misses [Hardware cache event]
rNNN [raw hardware event descriptor]
perf list - example of tracepoints

block:block_rq_insert [Tracepoint event]
jbd2:jbd2_start_commit [Tracepoint event]
ext4:ext4Allocate_inode [Tracepoint event]
kmem:kmalloc [Tracepoint event]
module:module_load [Tracepoint event]
workqueue:workqueue_execution [Tracepoint event]
timer:timer_expire{entry,exit} [Tracepoint event]
timer:hrtimer_start [Tracepoint event]
irq:irq_handler{entry,exit} [Tracepoint event]
irq:softirq{entry,exit} [Tracepoint event]
sched:sched{wakeup,switch} [Tracepoint event]
syscalls:sys{enter,exit}_epoll_wait [Tracepoint event]
$ perf stat ls Makefile
Makefile

Performance counter stats for 'ls Makefile':

2.204554 task-clock-msecs  # 0.842 CPUs
  0 context-switches  # 0.000 M/sec
  0 CPU-migrations  # 0.000 M/sec
  240 page-faults  # 0.109 M/sec
2176584 cycles  # 987.313 M/sec
1224357 instructions  # 0.563 IPC
  60577 cache-references  # 27.478 M/sec
  1788 cache-misses  # 0.811 M/sec

0.002618700  seconds time elapsed
$

$
$ perf stat -r 5 sleep 5

Performance counter stats for 'sleep 5' (5 runs):

1.411021 task-clock-msecs  # 0.000 CPUs  ( +-  0.829% )
  1 context-switches  # 0.001 M/sec ( +-  0.000% )
  0 CPU-migrations   # 0.000 M/sec ( +-   nan% )
  176 page-faults    # 0.125 M/sec ( +-  0.000% )
1378625 cycles       # 977.041 M/sec ( +-  0.796% )
  752343 instructions # 0.546 IPC ( +-  0.362% )
  30534 cache-references # 21.639 M/sec ( +-  0.763% )
  2074 cache-misses   # 1.470 M/sec ( +-  4.879% )

5.001883846 seconds time elapsed  ( +-  0.000% )

$
<table>
<thead>
<tr>
<th>sample</th>
<th>pcnt</th>
<th>function</th>
<th>DSO</th>
</tr>
</thead>
<tbody>
<tr>
<td>303.00</td>
<td>9.2%</td>
<td>read_hpetch</td>
<td>[kernel.kallsyms]</td>
</tr>
<tr>
<td>72.00</td>
<td>2.2%</td>
<td>pthread_mutex_lock</td>
<td>/lib/libpthread-2.12.so</td>
</tr>
<tr>
<td>68.00</td>
<td>2.1%</td>
<td>delay_tsc</td>
<td>[kernel.kallsyms]</td>
</tr>
<tr>
<td>55.00</td>
<td>1.7%</td>
<td>aes_dec_blk</td>
<td>[aes_i586]</td>
</tr>
<tr>
<td>55.00</td>
<td>1.7%</td>
<td>drm_clflush_pages</td>
<td>[drm]</td>
</tr>
<tr>
<td>52.00</td>
<td>1.6%</td>
<td>system_call</td>
<td>[kernel.kallsyms]</td>
</tr>
<tr>
<td>49.00</td>
<td>1.5%</td>
<td>memcpy[]</td>
<td>/lib/libc-2.12.so</td>
</tr>
<tr>
<td>48.00</td>
<td>1.4%</td>
<td>strstr[ia32]</td>
<td>/lib/libc-2.12.so</td>
</tr>
<tr>
<td>46.00</td>
<td>1.4%</td>
<td>unix_poll</td>
<td>[kernel.kallsyms]</td>
</tr>
<tr>
<td>42.00</td>
<td>1.3%</td>
<td>__ieee754_pow</td>
<td>/lib/libm-2.12.so</td>
</tr>
<tr>
<td>41.00</td>
<td>1.2%</td>
<td>do_select</td>
<td>[kernel.kallsyms]</td>
</tr>
<tr>
<td>40.00</td>
<td>1.2%</td>
<td>pixman_rasterize_edges</td>
<td>libpixman-1.so.0.18.0</td>
</tr>
<tr>
<td>37.00</td>
<td>1.1%</td>
<td>_raw_spin_lock_irqsave</td>
<td>[kernel.kallsyms]</td>
</tr>
<tr>
<td>36.00</td>
<td>1.1%</td>
<td>_int_malloc</td>
<td>/lib/libc-2.12.so</td>
</tr>
</tbody>
</table>

^C

$
perf record

- No daemons
- Callchains
- Output to different files
- Feed to other tools
- Outputs just into the regular filesystem
- No separate 'oprofile repository' of sample files
- Files are next to the project you are working on
- Can record events on a task, on a CPU or on the whole system
- Records the build-ids
$ cd firefox.data
$ perf record --pid `pidof firefox`
^C[ perf record: Captured and wrote 1.215 MB perf.data (~53065 samples) ]
$ ls -sh perf.data
 1,3M perf.data
perf report

. Lazy/Late symbol resolution by build-ids, when present
. Picks what is available
. -debuginfo packages, .symtab, .dynsym
. --fractal, --graph
. Supports JATO generated symbol tables for JAVA JIT profiling
. Automatically pick them from the dso name
. Cross platform support/offline report
$ perf report -C firefox --sort comm,dso
# Samples: 52628
# Overhead     Shared Object
# ......           ..........    ...           0.68% /lib64/libc-2.10.1.so                              0.55% /usr/lib64/libsqlite3.so.0.8.6                 
36.37% /usr/lib64/xulrunner-1.9.1/libxul.so
30.29% /usr/lib64/xulrunner-1.9.1/libmozjs.so
19.39% [kernel]
3.69% /usr/lib64/firefox-3.5/firefox
2.48% /lib64/libpthread-2.10.1.so
1.78% /lib64/libnsspr4.so
0.98% /usr/lib64/libjpeg.so.62.0.0
0.87% /lib64/libglib-2.0.so.0.2000.3
0.68% /lib64/libc-2.10.1.so
0.55% /usr/lib64/libsqLite3.so.0.8.6
$
$ perf report
# Samples: 52628
# Overhead     Shared Object     Symbol
# ...... ...................................... ......  ...  
13.17% [kernel] vread_hpet
  7.51% /lib64/xulrunner/libxul.so SelectorMatches(RuleProcessorData&, nsCSSSelecto
  5.82% /lib64/xulrunner/libmozjs.so js_Interpret
  2.90% /lib64/firefox-3.5/firefox 0x00000000000dd26
  1.68% /lib64/xulrunner/libxul.so SelectorMatchesTree(RuleProcessorData&, nsCSSSel
  1.50% /lib64/xulrunner/libmozjs.so js_Invoke
  1.46% /lib64/xulrunner/libmozjs.so js_InternalInvoke
  1.42% /lib64/xulrunner/libmozjs.so js_LookupPropertyWithFlags
  1.31% /lib64/xulrunner/libxul.so nsAttrValue::Contains(nsIAtom*, nsCaseTreatment)
  1.27% /lib64/libpthread-2.10.1.so __pthread_mutex_lock_internal
  1.22% /lib64/xulrunner/libmozjs.so js_GetPropertyHelper
  1.12% /lib64/xulrunner/libmozjs.so js_ExecuteRegExp
  1.10% /lib64/xulrunner/libmozjs.so js_SearchScope
$
perf report -g

- Callchains
- Needs -fno-omit-frame-pointer
- Register pressure on IA32
  . Fedora 13 seems to have it enabled
perf report -g

# Samples: 216342
# Overhead  Command                Shared Object  Symbol
# ........  .......  ...   __dwfl_getmodules_internal                 |          |          |           --36.36%-- die__create_new_lexblock <SNIP>

  15.82%  pahole /usr/lib64/libdw-0.141.so  [] __libdw_find_attr
  --1.85%-- __libdw_findabbrev
  --1.78%-- __die__process_tag
    cus__load_module
    cus__process_dwfilmod
    __dwfl_getmodules_internal
  --1.25%-- Dwarf_Abbrev_Hash_find
  --1.14%-- __die__process_function
   --63.33%-- die__create_new_lexblock
     --57.89%-- __die__process_function
      --63.64%-- __die__process_tag
        cus__load_module
        cus__process_dwfilmod
        __dwfl_getmodules_internal
     --36.36%-- die__create_new_lexblock

<SNIP>
perf report TUI

- Integrates report and annotate
- Zoom operation for thread and DSO
- Will support based on context (hard, soft IRQ, etc)
perf report - TODO

- Really big files take long to load
- Progressive loading, kinda similar to perf top
- Snapshots updated to the screen every N seconds
perf annotate

- similar purpose as opannotate
- colors for hot lines
- still uses objdump
- need to make objdump -S use the source in -debuginfo pkgs
- TUI allows tabbing thru hot spots, starts on hottest line
Another perf report example

```
$ perf record -g pahole vmlinux > /tmp/vmlinux.pahole
[ perf record: Captured and wrote 13.408 MB perf.data (~585799 samples) ]
$ perf report -g none -C pahole -d libdwarves.so.1.0.0
# dso: ./build/libdwarves.so.1.0.0
# comm: pahole
# Samples: 39486
# Overhead Symbol
# ........
12.57%  [.] tag__recode_dwarf_type
10.81%  [.] namespace__recode_dwarf_types
10.49%  [.] die__process_class
10.20%  [.] cu__find_base_type_by_sname_and_size
  6.15%  [.] strings__compare
  4.93%  [.] tag__init
  4.29%  [.] cus__load_module
   3.99%  [.] list__for_all_tags
   3.71%  [.] tag__size
   2.95%  [.] __die__process_tag
   2.38%  [.] cu__table_add_tag
   2.28%  [.] class_member__cache_byte_size
   1.87%  [.] strings__add
   1.86%  [.] dwarf_attr@plt
   1.75%  [.] die__create_new_subroutine_type
```
What is happening in tag__recode_dwarf_type?

Disassembly of section .text:
0000000000007ae0 <cu__table_add_tag>:

```
struct dwarf_tag *tpos;
struct hlist_node *pos;
uint16_t bucket = hashtags__fn(id);
const struct hlist_head *head = hashtable + bucket;

hlist_for_each_entry(tpos, pos, head, hash_node) {
    if (tpos->id == id)
```

```
mov %rdx,%rax
jne 11860 <tag__recode_dwarf_type+0x4e0>
jmpq 11741 <tag__recode_dwarf_type+0x3c1>
```

```
dtype = dwarf_cu__find_type_by_id(cu->priv, dtag->containing_type)
```
Integration with other tools

- ftrace
  - ‘perf ftrace’ proposed as a start

- Oprofile
  - Keep userspace utilities as-is, use perf kernel bits
  - Generic perf backend for oprofile from sh/ARM
  - Counter multiplexing added, first seen in perf land
  - Reduce the feature gap, future merge

- sysprof
  - Converted to the perf events infrastructure

- PAPI
  - Has support since in 3.7.0 version.
Thanks’n’Links

. Thanks to Ingo, Thomas, PeterZ, Rostedt, Frederic, Mike, Paul
. And everybody else contributing and testing these new tools

. tools/perf/Documentation/examples.txt (in the kernel tree)
. tools/perf/design.txt
. git://git.kernel.org/pub/scm/linux/kernel/git/tip/linux-2.6-tip.git
. Performance Counters on Linux: v8: http://lwn.net/Articles/336542
. This presentation: http://vger.kernel.org/~acme/perf/

Arnaldo Carvalho de Melo
acme@redhat.com