

News from tools/perf land

What has been brewing in the Linux observability tools

Arnaldo Carvalho de Melo

Red Hat Inc.

November 11, 2017

Tackling Growing Complexity

- Performance analysis is hard
- Lots of systems
- Each with lots of metrics
- Some only the processor can provide
- Break the complexity with multiple tools
- Let users combine them

Lots of news

- perf trace
- Vendor events
- eBPF integration
- Intel PT
- JIT profiling
- python scripts profiling, others follow
- perf c2c
- In the kernel source tree
- Reusing kernel code and coding style
- tools/lib
- container tests
- perf test
- More than we have time to talk here

perf trace

- strace like
- low overhead, no ptrace
- more targets
- perf features: more events, callchains, etc
- syscall arg beautifiers

```
# perf trace -e futex --duration 1
200.139 (2999.515 ms): virt-manager/3384 futex(uaddr: 0x7fb9280039d0,
      op: WAIT_BITSET|PRIV|CLKRT,
      val3: 4294967295) = 0
3202.723 (3000.439 ms): virt-manager/3384 futex(uaddr: 0x7fb928001950,
      op: WAIT_BITSET|PRIV|CLKRT,
      val3: 4294967295) = 0
196.247 (6009.648 ms): libvirtd/1084 futex(uaddr: 0x5573466d9f60,
      op: WAIT|PRIV) = 0
2029.069 (4999.934 ms): docker-contain/3259 futex(uaddr: 0xd7a718,
      utime: 0xc42001fed8) = -1 ETIMEDOUT Connection
2029.136 (4999.990 ms): docker-contain/2912 futex(uaddr: 0xc420027510) =
2029.166 (4999.912 ms): docker-contain/2895 futex(uaddr: 0xd7acf8,
      utime: 0x7fcc5761adb8) = 0
2029.156 (4999.940 ms): docker-contain/2898 futex(uaddr: 0xc420058110) =
6207.427 (2999.572 ms): virt-manager/3384 futex(uaddr: 0x7fb928001f80,
      op: WAIT_BITSET|PRIV|CLKRT,
      val3: 4294967295) = 0

^C#
```

pid, no children

```
# trace -p 3384 --no-inherit -e futex --duration 1 --max-stack 8
2.538 (3001.384 ms): virt-manager/3384 futex(uaddr: 0x7fb928005ee0,
                                           op: WAIT_BITSET|PRIV|CLKRT,
                                           val3: 4294967295) = 0
do_futex_wait.constprop.1 (/usr/lib64/libpthread-2.25.so)
__new_sem_wait_slow.constprop.0 (/usr/lib64/libpthread-2.25.so)
PyThread_acquire_lock (/usr/lib64/libpython2.7.so.1.0)
[0xffff8046a2c02fd1] (/usr/lib64/libpython2.7.so.1.0)
PyEval_EvalFrameEx (/usr/lib64/libpython2.7.so.1.0)
PyEval_EvalCodeEx (/usr/lib64/libpython2.7.so.1.0)
PyEval_EvalFrameEx (/usr/lib64/libpython2.7.so.1.0)
PyEval_EvalCodeEx (/usr/lib64/libpython2.7.so.1.0)
^C[root@jouet ~]#
```

Running 'perf trace' as non-root

```
$ trace usleep 1  
Error: No permission to read events/raw_syscalls/sys_(enter|exit)  
Hint: Try 'sudo mount -o remount,mode=755 /sys/kernel/tracing'  
$
```

Running 'perf trace' as non-root: 2

```
$ sudo mount -o remount,mode=755 /sys/kernel/tracing
```

```
$ trace usleep 1
```

```
Error: Permission denied.
```

```
Hint: Check /proc/sys/kernel/perf_event_paranoid setting.
```

```
Hint: For your workloads it needs to be <= 1
```

```
Hint: For system wide tracing it needs to be set to -1.
```

```
Hint: Try: 'sudo sh -c "echo -1 > /proc/sys/kernel/perf_event_paranoid"
```

```
Hint: The current value is 2.
```


Running 'perf trace' as non-root: 2

```
$ sudo sh -c "echo -1 > /proc/sys/kernel/perf_event_paranoid"  
$ trace usleep 1  
<SNIP>  
0.357 (0.001ms): usleep/18517 brk() = 0x55e44d210000  
0.373 (0.058ms): usleep/18517 nanosleep(rqtp: 0x7ffca6c566c0) = 0  
<SNIP>  
$
```

Specifying syscalls

```
$ trace -e nanosleep sleep 1  
0.000 (1000.195ms): sleep/22896 nanosleep(rqtp: 0x7ffd03270f10) =
```

With backtraces!

```
$ trace -e nanosleep --call-graph dwarf sleep 1
0.000 (1000.143ms): sleep/22906 nanosleep(rqtp: 0x7ffe91fed9c0) = 0
    __nanosleep_nocancel (/usr/lib64/libc-2.25.so)
    rpl_nanosleep (/usr/bin/sleep)
    xnanosleep (/usr/bin/sleep)
    main (/usr/bin/sleep)
    __libc_start_main (/usr/lib64/libc-2.25.so)
    _start (/usr/bin/sleep)

$
```

- fp: cheap but not always available
- dwarf: expensive, but works
- dwarf: Collects user stack + registers, per sample
- To reduce the cost: use just in some events
- Reduce the stack sampled
- LBR last branch record can be used as well

Reducing stack sampled: drastically

```
$ trace -e nanosleep --call-graph dwarf,128 sleep 1
0.000 (1000.135ms): sleep/23073 nanosleep(rqtp: 0x7ffc90169e70) = 0
    __nanosleep_nocancel (/usr/lib64/libc-2.25.so)
    rpl_nanosleep (/usr/bin/sleep)
    xnanosleep (/usr/bin/sleep)
    [0] ([unknown])

$
```

Reducing stack sampled: 2

```
$ trace -e nanosleep --call-graph dwarf,256 sleep 1
0.000 (1000.192ms): sleep/23133 nanosleep(rqtp: 0x7fff7fb7e9e0) = 0
    __nanosleep_nocancel (/usr/lib64/libc-2.25.so)
    rpl_nanosleep (/usr/bin/sleep)
    xnanosleep (/usr/bin/sleep)
    main (/usr/bin/sleep)
    __libc_start_main (/usr/lib64/libc-2.25.so)
[0] ([unknown])

$
```

Reducing stack sampled: righsized

```
$ trace -e nanosleep --call-graph dwarf,512 sleep 1
0.000 (1000.125ms): sleep/23146 nanosleep(rqtp: 0x7ffd896eea50) = 0
    __nanosleep_nocancel (/usr/lib64/libc-2.25.so)
    rpl_nanosleep (/usr/bin/sleep)
    xnanosleep (/usr/bin/sleep)
    main (/usr/bin/sleep)
    __libc_start_main (/usr/lib64/libc-2.25.so)
    _start (/usr/bin/sleep)

$
```

Features present in various tools

- We saw callchains in 'perf trace'
- It is also present in other tools
- As lots of other concepts, as we'll see

Measuring DWARF callgraphs cost

```
$ perf record -a --call dwarf sleep 5s
[ perf record: Woken up 30 times to write data ]
[ perf record: Captured and wrote 11.043 MB perf.data (1955 samples) ]
$ perf evlist -v
cycles:ppp: sample_freq: 4000,
sample_type: IP|TID|TIME|ADDR|CALLCHAIN|CPU|PERIOD|REGS_USER|STACK_US
mmap: 1, comm: 1, task: 1, precise_ip: 3, exclude_guest: 1,
exclude_callchain_user: 1, comm_exec: 1, sample_regs_user: 0xff0fff,
sample_stack_user: 8192
$
```

Measuring DWARF callgraphs cost

```
$ perf record -a --call dwarf,128 sleep 5s  
[ perf record: Woken up 1 times to write data ]  
[ perf record: Captured and wrote 4.033 MB perf.data (1821 samples) ]  
$ perf evlist -v  
cycles:ppp: ..., sample_stack_user: 128  
$
```

using max-stack

- Use max-stack to reduce sample size and postprocessing time
- Ask for callgraphs for just some of the events
- And mix syscalls with other events, say tracepoints
- Use wildcards to type less

```
$ trace -e *sleep -e sched:*wakeup,sched:*switch/call-graph=fp,max-st
1.195 (0.017ms): sleep/23575 nanosleep(rqtp: 0x7ffcf19a17d0) ...
1.211 (      ): sched:sched_switch:sleep:23575 [120] S ==> swapp
    __schedule ([kernel])
    __schedule ([kernel])
    schedule ([kernel])
    do_nanosleep ([kernel])
    hrtimer_nanosleep ([kernel])
    sys_nanosleep ([kernel])
    do_syscall_64 ([kernel])
    return_from_SYSCALL_64 ([kernel])
1001.369 (      ): d_wakeup:sleep:23575 [120] success=1 CPU:002)
1.195 (1000.226ms): sleep/23575 ... [continued]: nanosleep()) = 0
$
```

eBPF Integration

```
# cat filter.c
#include <uapi/linux/bpf.h>
#define SEC(NAME) __attribute__((section(NAME), used))

SEC("func=hrtimer_nanosleep rqtp->tv_nsec")
int func(void *ctx, int err, long nsec)
{
    return nsec > 1000;
}
char _license[] SEC("license") = "GPL";
int _version SEC("version") = LINUX_VERSION_CODE;
```

1000ns

```
# trace -e *sleep,filter.c usleep 1
0.000 (0.096ms): usleep/7978 nanosleep(rqtp: 0x7fff3ac3cb00) = 0
#
```

2000ns

```
# trace -e nanosleep,filter.c  usleep 2
0.000 (0.012ms): usleep/7564 nanosleep(rqtp: 0x7ffec11bbd20) ...
0.012 (      ): perf_bpf_probe:func:(ffffffff9f10cb30) tv_nsec=2000)
0.000 (0.071ms): usleep/7564  ... [continued]: nanosleep()) = 0
#
```

With a backtrace

```
# trace -e nanosleep,filter.c/max-stack=5/ usleep 1001
0.000 (0.018ms): usleep/963 nanosleep(rqtp: 0x7ffdc6421410) ...
0.018 (          ): perf_bpf_probe:func:(ffffffff9f10cb30) tv_nsec=10010
                    hrtimer_nanosleep ([kernel])
                    SyS_nanosleep ([kernel])
                    do_syscall_64 ([kernel])
                    return_from_SYSCALL_64 ([kernel])
                    __GI_nanosleep (/usr/lib64/libc-2.25.so)
0.000 (1.095ms): usleep/963 ... [continued]: nanosleep()) = 0
[root@jouet bpf]#
```

- clang + llvm
- sys_bpf()
- Attach to kprobes, uprobes, tracepoints
- Work as filters
- Should be used for filtering by syscall args
- Also to copy pointer payloads for beautification

syscall args: pointers

```
# trace -e open touch /tmp/bla
0.000 (0.020ms): touch open(filename: 0x8992e37, flags: CLOEXEC) = 3
0.051 (0.012ms): touch open(filename: 0x8b96640, flags: CLOEXEC) = 3
0.440 (0.018ms): touch open(filename: 0x873cc70, flags: CLOEXEC) = 3
0.532 (0.311ms): touch open(filename: 0x66a9340,
                        flags: CREAT|NOCTTY|NONBLOCK|WRONLY,
                        mode: IRUGO|IWUGO) = 3
#
```

Copying that pointer contents

```
# perf probe "vfs_getname=getname_flags:72 \  
            pathname=result->name:string"
```

Added new event:

```
probe:vfs_getname (on getname_flags:72 with \  
                  pathname=result->name:string)
```

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

```
perf record -e probe:vfs_getname -aR sleep 1
```

```
#
```

Integration of 'perf trace' with 'perf probe'

```
# trace -e open touch /tmp/bla
0.038 (0.028ms): touch open(filename: /etc/ld.so.cache,
                        flags: CLOEXEC) = 3
0.106 (0.016ms): touch open(filename: /lib64/libc.so.6,
                        flags: CLOEXEC) = 3
0.710 (0.030ms): touch open(filename: /usr/lib/locale/locale-archive,
                        flags: CLOEXEC) = 3
0.834 (0.026ms): touch open(filename: /tmp/bla,
                        flags: CREAT|NOCTTY|NONBLOCK|WRONLY,
                        mode: IRUGO|IWUGO) = 3
#
```

- Do not use headers directly
- Keep a copy
- Check for drift: `check_headers.sh`
- Use to automatically create string tables
- For beautifying args
- Allow building the tools on older distros
- Where recent defines/structs/etc are not present

Reuse kernel code

- rbtree, refcount_t, atomic_t, listh, etc
- Facilitate hacking on tools/ by kernel folks
- Write kernel facility
- Show how it is used in tools/
- Not just perf, objtool, etc

- JSON file
- Vendor defined names
- oprofile users used to them
- Intel first: Broadwell, Skylake, etc
- ARM, PowerPC too

On a Broadwell based ThinkPad t450s

```
[root@jouet ~]# perf list l1d_miss
```

List of pre-defined events (to be used in -e):

pipeline:

```
cycle_activity.cycles_l1d_miss
```

```
  [Cycles while L1 cache miss demand load is outstanding]
```

```
cycle_activity.stalls_l1d_miss
```

```
  [Execution stalls while L1 cache miss demand load is outstanding]
```

```
[root@jouet ~]#
```

Testing with containers

```
# dm
 1 alpine:3.4:           Ok gcc (Alpine 5.3.0) 5.3.0
 2 alpine:3.5:           Ok gcc (Alpine 6.2.1) 6.2.1 20160822
 3 alpine:3.6:           FAIL gcc (Alpine 6.3.0) 6.3.0
 4 alpine:edge:          FAIL gcc (Alpine 6.4.0) 6.4.0
 5 android-ndk:r12b-arm: Ok gcc 6.3.1 20161221 (Red Hat)
 6 android-ndk:r15c-arm: Ok gcc 7.2.1 20170915 (Red Hat)
 7 centos:5:             Ok gcc 4.1.2 20080704 (Red Hat)
 8 centos:6:             Ok gcc 4.4.7 20120313 (Red Hat)
 9 centos:7:             Ok gcc 4.8.5 20150623 (Red Hat)
10 debian:7:             Ok gcc (Debian 4.7.2-5) 4.7.2
11 debian:8:             Ok gcc (Debian 4.9.2-10) 4.9.2
12 debian:9:             Ok gcc (Debian 6.3.0-18) 6.3.0 20170516
13 debian:experimental: Ok gcc (Debian 7.2.0-11) 7.2.0
^C
```


Testing with Containers

- Make sure it continues building with older systems
- Test with upcoming distros and tools
- flex 2.6.4 on alpine 3.6 and edge: FAIL
- flex 2.6.1 on fedora 26: Ok
- After fixing it: gentoo user reports same problem

Testing with Containers

- Fetches tarball from https server
- make perf-tarxz-src-pkg
- builds with gcc and clang
- Already goes in each pull request I send upstream
- Next step: run with kubernetes
- Next step: run 'perf test' as well

Full Dockerfile

```
# docker.io/acmel/linux-perf-tools-build-fedora:rawhide
FROM docker.io/fedora:rawhide
MAINTAINER Arnaldo Carvalho de Melo <acme@kernel.org>
# Remove NO_LIBPERL=1 and fix perl breakage
RUN dnf -y install make gcc flex bison elfutils-libelf-devel \
    elfutils-devel libunwind-devel xz-devel \
    numactl-devel openssl-devel slang-devel \
    gtk2-devel perl-ExtUtils-Embed python-devel \
    binutils-devel audit-libs-devel && \
    dnf -y clean all && \
    rm -rf /usr/share/doc /usr/share/gtk-doc && \
    mkdir -m 777 -p /tmp/build/perf /tmp/build/objtool && \
    groupadd -r perfbuilder && \
    useradd -r -g perfbuilder perfbuilder
USER perfbuilder
ENTRYPOINT make -C /git/linux/tools/objtool \
O=/tmp/build/objtool && \
    make NO_LIBPERL=1 \
-C /git/linux/tools/perf O=/tmp/build/perf
```

Run the build from inside the container

- Replace the ENTRYPOINT with /bin/bash
- Using the full Dockerfile, its not root anymore
- Just call make
- Outside the container, do changes to the source
- Rebuild inside and outside

```
# docker run --entrypoint=/bin/bash \  
    -v /home/acme/git:/git:Z \  
    -ti docker.io/acmel/linux-perf-tools-build-fedora:rawhide  
bash-4.3$ cd /git/linux  
bash-4.3$ make -C tools/perf O=/tmp  
make: Entering directory '/git/linux/tools/perf'  
BUILD:   Doing 'make -j4' parallel build
```

Dockerfile for Debian 8

```
# docker.io/acmel/linux-perf-tools-build-debian:8
FROM docker.io/debian:8
MAINTAINER Arnaldo Carvalho de Melo <acme@kernel.org>
ENV DEBIAN_FRONTEND noninteractive
RUN apt-get update && \
    apt-get install -y apt-utils && \
    apt-get install -y make gcc flex bison libelf-dev libdw-dev \
        libunwind-dev libaudit-dev libssl-dev \
        libslang2-dev libgtk2.0-dev libperl-dev \
        python-dev libiberty-dev binutils-dev \
        liblzma-dev libnuma-dev && \
    apt-get clean -y && \
    rm -rf /usr/share/doc /usr/share/gtk-doc && \
    mkdir -m 777 -p /tmp/build/perf /tmp/build/objtool && \
    groupadd -r perfbuilder && \
    useradd -r -g perfbuilder perfbuilder
USER perfbuilder
ENTRYPOINT make -C /git/linux/tools/perf O=/tmp/build/perf && \
    make -C /git/linux/tools/objtool O=/tmp/build/objtool
```

Another Dockerfile: Android NDK

```
# cat Dockerfile
# docker.io/acmel/linux-perf-tools-build-android-ndk:r12b
FROM docker.io/fedora:24
MAINTAINER Arnaldo Carvalho de Melo <acme@kernel.org>
ENV SOURCEFILE=android-ndk-r12b-linux-x86_64.zip
RUN dnf -y install make bison flex unzip && \
    dnf -y clean all && \
    mkdir -m 777 -p /tmp/build/perf && \
    curl -OL http://dl.google.com/android/repository/${SOURCEFILE} && \
    unzip -d /opt ${SOURCEFILE} && \
    rm -f ${SOURCEFILE} && \
    rm -rf /opt/android-ndk-r12b/sources \
        /opt/android-ndk-r12b/platforms/android-[19]* \
        /opt/android-ndk-r12b/platforms/android-2[0-3]* \
        /opt/android-ndk-r12b/platforms/android-24/arch-mips* \
        /opt/android-ndk-r12b/platforms/android-24/arch-x86* \
        /opt/android-ndk-r12b/toolchains/x86* \
        /opt/android-ndk-r12b/toolchains/mips* \
        /opt/android-ndk-r12b/toolchains/llvm* \
        /opt/android-ndk-r12b/prebuilt/ \
        /opt/android-ndk-r12b/python* \
        /opt/android-ndk-r12b/shader-tools/ &&\
    groupadd -r perfbuilder && \
    useradd -r -g perfbuilder perfbuilder
USER perfbuilder
ENV NDK=/opt/android-ndk-r12b/
ENV NDK_TOOLCHAIN=${NDK}/toolchains/....bin/arm-linux-androideabi-
ENV NDK_SYSROOT=${NDK}/platforms/android-24/arch-arm
ENTRYPOINT make -C /git/linux/tools/perf O=/tmp/build/perf \
    ARCH=arm CROSS_COMPILE=${NDK_TOOLCHAIN} \
    EXTRA_CFLAGS="-pie --sysroot=${NDK_SYSROOT}"

#
```

perf record

- `-switch-time`: perf.data file rotation
- by size, time or by receiving a SIGUSR2
- External app decides when to take a snapshot
- When a packet is received, mouse moves over some area, etc

Thanks!

Arnaldo Carvalho de Melo

acme@kernel.org

acme@redhat.com

<http://vger.kernel.org/~acme/perf/>