Synthetic events and Histograms

And what's coming in the future

Steven Rostedt
Open Source Engineer

srostedt@vmware.com / rostedt@goodmis.org

Github / twitter

Latency tracing in the kernel

The “latency-tracer” in the preempt-realtime patch 2005

- Recorded interrupt disabled latency
- Recorded preempt off latency
- Recorded preempt off and interrupt disabled combined latency
- Recorded wake up to schedule of highest prio task latency

©2019 VMware, Inc.

Latency tracing in the kernel

The “latency-tracer” today

- irqsoff tracer - Recorded interrupt disabled latency
- preemptoff tracer - Recorded preempt off latency
- preemptirqsoff tracer - Recorded preempt off and interrupt disabled combined latency
- Recorded wake up to schedule of highest prio task latency
  - wakeup_rt - same as preempt-realtime patch (only real-time tasks)
  - wakeup - records all tasks (not just Real Time - FIFO and RR)
  - wakup_dl - records only deadline tasks (SCHED_DEADLINE)

©2019 VMware, Inc.

Latency tracing in the kernel

/sys/kernel/tracing

- current_tracer
  - echo preemptirqsoff > current_tracer

cat trace

- Shows the last snapshot of the max latency (after it is hit)

cat tracing_max_latency

- Holds the currently highest latency (in microseconds) that was hit

cat tracing_threshold

- The minimum latency to trigger a trace

©2019 VMware, Inc.

Problems with the Latency Tracers

They were ported straight from the RT patch

- Made for a specific purpose

Immutable

- Very little to customize what you want to see

Snapshot only

- No continuous tracing
- Breaks from the other tracers
- A different paradigm

©2019 VMware, Inc.

Event Triggers

Triggers are actions attached to events

A “trigger” file exists for each event

Kinds of triggers:

- traceon
- traceoff
- snapshot
- stacktrace
- enable_event
- disable_event

©2019 VMware, Inc.

Event Triggers

More triggers:

- hist
- enable_hist
- disable_hist

©2019 VMware, Inc.

Histograms

Every event has a way to add a histogram

Can take keys and values from any field

Customizable

©2019 VMware, Inc.

Histograms

Every event has a way to add a histogram

Can take keys and values from any field

Customizable

```
hist:keys=<field1[,field2,...]>[:values=<field1[,field2,...]>]
          [:sort=<field1[,field2,...]>][:size=#entries][:pause][:continue]
          [:clear][:name=histname1][:<handler>.<action>] [if <filter>]
```

©2019 VMware, Inc.

Histograms

Every event has a way to add a histogram

Can take keys and values from any field

Customizable

```
hist:keys=<field1[,field2,...]>[:values=<field1[,field2,...]>]
          [:sort=<field1[,field2,...]>][:size=#entries][:pause][:continue]
          [:clear][:name=histname1][:<handler>.<action>] [if <filter>]
```

See Documentation/trace/histogram.rst for more information

©2019 VMware, Inc.

Histograms

```
# cd /sys/kernel/tracing
# echo 'hist:keys=skbaddr.hex:vals=len' events/net/netif_receive_skb/trigger
# cat events/net/netif_receive_skb/hist

# event histogram
#
# trigger info: hist:keys=skbaddr.hex:vals=hitcount,len:sort=hitcount:size=2048 [active]
#

{ skbaddr: ffff888089907a40 } hitcount:          1  len:         88
{ skbaddr: ffff888089907cc0 } hitcount:          1  len:         52
{ skbaddr: ffff888089907680 } hitcount:          1  len:         88
{ skbaddr: ffff88808992a280 } hitcount:          1  len:         88
{ skbaddr: ffff888089906dc0 } hitcount:          1  len:         52
{ skbaddr: ffff8880899063c0 } hitcount:          1  len:         88
{ skbaddr: ffff888089907e00 } hitcount:          1  len:         88
{ skbaddr: ffff888089906780 } hitcount:          1  len:         52
{ skbaddr: ffff888089906140 } hitcount:          1  len:         88
{ skbaddr: ffff88808992a140 } hitcount:          1  len:         52
{ skbaddr: ffff888089907900 } hitcount:          1  len:         52
{ skbaddr: ffff888089906b40 } hitcount:          1  len:         88
{ skbaddr: ffff888089906500 } hitcount:          1  len:         52
{ skbaddr: ffff888089906000 } hitcount:          1  len:         52
{ skbaddr: ffff88808992a500 } hitcount:          1  len:         52
{ skbaddr: ffff888089906640 } hitcount:          1  len:         88
{ skbaddr: ffff888089907180 } hitcount:          1  len:         52
{ skbaddr: ffff8880899068c0 } hitcount:          1  len:         88
{ skbaddr: ffff8880899072c0 } hitcount:          1  len:         88
{ skbaddr: ffff888089906280 } hitcount:          1  len:         52
{ skbaddr: ffff888089907540 } hitcount:          1  len:         52
{ skbaddr: ffff888089906f00 } hitcount:          1  len:         88
{ skbaddr: ffff888089906a00 } hitcount:          1  len:         52

Totals:
    Hits: 23
    Entries: 23
    Dropped: 0
```

©2019 VMware, Inc.

Histograms

Removing a histogram use a “!”

```
# echo '!hist:keys=skbaddr.hex:vals=len' events/net/netif_receive_skb/trigger
```

©2019 VMware, Inc.

kprobe events

Can now be used to read function parameters easily

- $arg1 - first parameter
- $arg2 - second parameter
- [...]

Available since Linux v4.20

Also allows to follow the pointers

and even read user space

©2019 VMware, Inc.

kprobe events

```
# echo 'p:open do_sys_open hex=$arg2:x64 file=+0($arg2):ustring' > kprobe_events
# echo 1 > events/kprobes/open/enable
# cat trace

# tracer: nop
#
# entries-in-buffer/entries-written: 4/4   #P:8
#
#                              _-----=> irqs-off
#                             / _----=> need-resched
#                            | / _---=> hardirq/softirq
#                            || / _--=> preempt-depth
#                            ||| /     delay
#           TASK-PID   CPU#  ||||    TIMESTAMP  FUNCTION
#              | |       |   ||||       |         |
           <...>-1906  [000] ...1  2971.823217: open: (do_sys_open+0x0/0x2b0) hex=0x7f5f7fa788b3 file="/etc/ld.so.cache"
           <...>-1906  [000] ...1  2971.823358: open: (do_sys_open+0x0/0x2b0) hex=0x7f5f7fa82d00 file="/lib64/libc.so.6"
           <...>-1906  [000] ...1  2971.824578: open: (do_sys_open+0x0/0x2b0) hex=0x7f5f7f9f0a20 file="/usr/lib/locale/locale-archive"
           <...>-1906  [000] ...1  2971.824849: open: (do_sys_open+0x0/0x2b0) hex=0x7ffc346e7582 file="trace"
```

©2019 VMware, Inc.

Matching events

Be able to pick events to match

Not just task wake up to scheduling

Or preempt / interrupts off

But any events!

- Like system call latency

©2019 VMware, Inc.

Matching events

```
# trace-cmd list -e raw -F

# trace-cmd list -e raw -F
system: raw_syscalls
name: sys_exit
ID: 21
format:
        field:unsigned short common_type;       offset:0;       size:2; signed:0;
        field:unsigned char common_flags;       offset:2;       size:1; signed:0;
        field:unsigned char common_preempt_count;       offset:3;       size:1; signed:0;
        field:int common_pid;   offset:4;       size:4; signed:1;

        field:long id;  offset:8;       size:8; signed:1;
        field:long ret; offset:16;      size:8; signed:1;

system: raw_syscalls
name: sys_enter
ID: 22
format:
        field:unsigned short common_type;       offset:0;       size:2; signed:0;
        field:unsigned char common_flags;       offset:2;       size:1; signed:0;
        field:unsigned char common_preempt_count;       offset:3;       size:1; signed:0;
        field:int common_pid;   offset:4;       size:4; signed:1;

        field:long id;  offset:8;       size:8; signed:1;
        field:unsigned long args[6];    offset:16;      size:48;        signed:0;
```

©2019 VMware, Inc.

Matching events

```
# echo 'hist:keys=common_pid,id:ts=common_timestamp.usecs' > events/raw_syscalls/sys_enter/trigger
# echo 'syscall long id; u64 time' > synthetic_events
# echo 'hist:keys=common_pid,id:
    time=common_timestamp.usecs-$ts:onmatch(raw_syscalls.sys_enter).trace(syscall,id,$time)'
    > events/raw_syscalls/sys_exit/trigger
# echo 1 > events/synthetic/syscall/enable
# cat trace

# tracer: nop
#
# entries-in-buffer/entries-written: 1485/1485   #P:8
#
#                              _-----=> irqs-off
#                             / _----=> need-resched
#                            | / _---=> hardirq/softirq
#                            || / _--=> preempt-depth
#                            ||| /     delay
#           TASK-PID   CPU#  ||||    TIMESTAMP  FUNCTION
#              | |       |   ||||       |         |
            bash-1512  [000] ...2  7975.394875: syscall: id=257 time=69590
            bash-1512  [000] ...2  7975.394892: syscall: id=72 time=5
            bash-1512  [000] ...2  7975.394903: syscall: id=72 time=6
            bash-1512  [000] ...2  7975.394912: syscall: id=72 time=3
            bash-1512  [000] ...2  7975.394922: syscall: id=72 time=3
            bash-1512  [000] ...2  7975.394935: syscall: id=33 time=5
            bash-1512  [000] ...2  7975.394944: syscall: id=3 time=4
            bash-1512  [000] ...2  7975.394954: syscall: id=33 time=4
            bash-1512  [000] ...2  7975.394971: syscall: id=72 time=3
            bash-1512  [000] ...2  7975.394981: syscall: id=3 time=3
            bash-1512  [000] ...2  7975.395002: syscall: id=13 time=5
            bash-1512  [000] ...2  7975.395032: syscall: id=14 time=6
            bash-1512  [000] ...2  7975.395047: syscall: id=16 time=8
            bash-1512  [000] ...2  7975.395056: syscall: id=14 time=4
            bash-1512  [000] ...2  7975.395065: syscall: id=13 time=3
            bash-1512  [000] ...2  7975.395074: syscall: id=72 time=3
```

©2019 VMware, Inc.

Matching events

1) Define the event that will start the latency measurement
   - May be a kprobe!
2) Define what fields you want to measure
   - id - the system call number
   - ts - start time of the latency measurement
3) Define what you want to use to match with the end event
   - common_pid - the process id of the task that triggered the event
   - id - the system call number

```
# echo 'hist:keys=common_pid,id:ts=common_timestamp.usecs' > events/raw_syscalls/sys_enter/trigger
```

©2019 VMware, Inc.

Matching events

4) Define what you want to record

- id - the system call number
- time - the latency measurement from the start to end event

```
# echo 'syscall long id; u64 time' > synthetic_events
```

©2019 VMware, Inc.

Matching events

5) Define the end event to stop the latency measurment

- sys_exit event

6) Match the end event fields with the start event fields

- common_pid - the process id of the task that triggered the event
- id - the system call number

7) Create the measurement variable

- time - compares the current time with the saved time from the start ($ts)

```
# echo 'hist:keys=common_pid,id:
    time=common_timestamp.usecs-$ts:onmatch(raw_syscalls.sys_enter).trace(syscall,id,$time)'
    > events/raw_syscalls/sys_exit/trigger
```

©2019 VMware, Inc.

Matching events

8) Match the start event

- onmatch(raw_syscalls.sys_enter)

9) Call the synthetic event with the data you want to store

- syscall - synthetic event to trigger on match
- id - the system call number
- $time - the latency measurement

```
# echo 'hist:keys=common_pid,id:
    time=common_timestamp.usecs-$ts:onmatch(raw_syscalls.sys_enter).trace(syscall,id,$time)'
    > events/raw_syscalls/sys_exit/trigger
```

©2019 VMware, Inc.

Histogram variables

Unique variables associated to a histogram bucket

```
# echo 'hist:keys=common_pid,id:ts=common_timestamp.usecs' > events/raw_syscalls/sys_enter/trigger
```

common_pid	id	$ts
1512	257	7975.394875
1512	72	7975.394892
1512	72	7975.394903
1512	72	7975.394912
1512	33	7975.394912

©2019 VMware, Inc.

Histogram variables

And can be passed via “onmatch()”

```
# echo 'hist:keys=common_pid,id:
    time=common_timestamp.usecs-$ts:onmatch(raw_syscalls.sys_enter).trace(syscall,id,$time)'
    > events/raw_syscalls/sys_exit/trigger
```

sys_enter

common_pid	id	$ts
1512	257	7975.394875
1512	72	7975.394892
1512	72	7975.394903
1512	72	7975.394912
1512	33	7975.394912

sys_exit

common_pid	id	$time
1512	257	69590
1512	72	5
1512	72	6
1512	72	3
1512	33	5

©2019 VMware, Inc.

Events are like tables!

Each event acts like a separate table

Each field of the event acts like a column of the table

Each instance of an event acts like a row in the table

©2019 VMware, Inc.

Events are like tables!

Each event acts like a separate table

Each field of the event acts like a column of the table

Each instance of an event acts like a row in the table

onmatch() joins two events (tables) to create a synthetic event

- Like creating a new table!

©2019 VMware, Inc.

histogram / synthetic events are hard!

The language is not like anything else

Not intuitive

- Need to constantly look up how to do it

If it is hard to use, it wont be used

- no matter how useful it is

```
# echo 'hist:keys=common_pid,id:ts=common_timestamp.usecs' > events/raw_syscalls/sys_enter/trigger
# echo 'syscall long id; u64 time' > synthetic_events
# echo 'hist:keys=common_pid,id:
    time=common_timestamp.usecs-$ts:onmatch(raw_syscalls.sys_enter).trace(syscall,id,$time)'
    > events/raw_syscalls/sys_exit/trigger
```

©2019 VMware, Inc.

histogram / synthetic events are hard!

But events can be treated as tables

A synthetic event is a combination of tables

What other language can we use?

©2019 VMware, Inc.

SQL!

©2019 VMware, Inc.

SQL!

SQL maps practically one to one with synthetic event logic

But we can't write a SQL interpreter in the kernel

How can we use this?

©2019 VMware, Inc.

SQL!

SQL maps practically one to one with synthetic event logic

But we can't write a SQL interpreter in the kernel

How can we use this?

trace-cmd!

- Can take in a string
- Can act like a compiler
- Translate a SQL language into the kernel language

©2019 VMware, Inc.

trace-cmd --select ?

Still in the design phase

- Looking for a SQL flex/bison templates
- Do not want to rewrite the wheel

Discussed this at Linux Plumbers in Lisbon (September 2019)

- Talked with Daniel Black (runner of the Database Microconference)
- Came up with a way to map 1 to 1

©2019 VMware, Inc.

trace-cmd --select ?

Still in the design phase

```
# echo 'hist:keys=common_pid,id:ts=common_timestamp.usecs' > events/raw_syscalls/sys_enter/trigger
# echo 'syscall long id; u64 time' > synthetic_events
# echo 'hist:keys=common_pid,id:
    time=common_timestamp.usecs-$ts:onmatch(raw_syscalls.sys_enter).trace(syscall,id,$time)'
    > events/raw_syscalls/sys_exit/trigger
```

```
# trace-cmd record --select_start e.id, x.$time - e.$time AS time \
  FROM raw_syscalls.sys_enter AS e \
  JOIN raw_syscalls.sys_exit AS x \
  ON e.common_pid == x.common_pid AND e.id == x.id \
  --select_end
```

©2019 VMware, Inc.

trace-cmd --select ?

Still in the design phase and with filters

```
# echo 'hist:keys=common_pid,id:ts=common_timestamp.usecs' > events/raw_syscalls/sys_enter/trigger
# echo 'syscall long id; u64 time' > synthetic_events
# echo 'hist:keys=common_pid,id:
    time=common_timestamp.usecs-$ts:onmatch(raw_syscalls.sys_enter).trace(syscall,id,$time)'
    > events/raw_syscalls/sys_exit/trigger if $time > 10
```

```
# trace-cmd record --select_start e.id, x.$time - e.$time AS time \
  FROM raw_syscalls.sys_enter AS e \
  JOIN raw_syscalls.sys_exit AS x \
  ON e.common_pid == x.common_pid AND e.id == x.id \
  WHERE time > 10
  --select_end
```

©2019 VMware, Inc.

Plenty to do

Currently all vapor wear

Need to learn SQL more

Need the parsing

Need user feedback

- So far a lot of admins have told me they like it

Need to make histograms in the kernel more robust

- Lots of bugs found
- No users, no testing :-(

©2019 VMware, Inc.

Conclusion

Histogram code can be useful for PREEMPT_RT

- No need for eBPF!
  - Not RT friendly

The functionality is mostly in the kernel

User space tools can facilitate the adaption by users

Think outside the box (SQL for tracing!)

©2019 VMware, Inc.

Thank You

vmware®