kvm_stat and Beyond
Past, Present and Future of Performance Monitoring with KVM

Christian Bornträger
Maintainer KVM on IBM Z
borntraeger@de.ibm.com
IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries.

The following are trademarks or registered trademarks of other companies.
- Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
- Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.
Motivation

If something in KVM does not work, or is too slow it often lands on my desk
Very unspecific
Reporter has no knowledge if the problem is in the hypervisor or in the guest

Problems like
“Is slower than z/VM”
“high cpu usage”
“not fast enough”

No quick logon possible
Hard to re-recreate setups
Private data on the system
Performance data needs to be collected at the system and transferred somewhere else
Getting Data Local and Remote

**Sysstat**
- Collects cpu (e.g. user, system, nice, interrupt), disk, memory data
- Available on almost all distributions
- Provides history data
- Pretty easy to setup
- Data can be transferred and analyzed somewhere else

**Other tools**
- Nagios, ELK, prometheus/grafana......

*Sometimes the standard Linux data gives an insight, sometime it does not*

*So what else do we have?*
Guest Time Accounting
Guests have the notion of steal time
I was runnable but the hypervisor scheduled me away

Sysstat understands both
Top/procps

Top does not understand guest time
Neither in summary nor in per process

Same for vmstat

In 2007 Laurent Vivier provided a patch for top to understand guest time
Procps was badly maintained at that point in time
Maintainer was reluctant and did not followup in discussions
Is it time to retry today with procps-ng?

```
$ vmstat -w 1
procs -----------------------memory---------------------- ---swap-- -----io---- -system-- --------cpu--------
     r  b         swpd         free         buff        cache   si   so    bi    bo   in   cs  us  sy  id  wa  st
56  0            0      5374312       198724      1611768    0    0     0     0 58127  363  61  17  22   0   0
56  0            0      5374060       198724      1611768    0    0     0     0 58166  500  60  17  22   0   0
56  0            0      5373556       198724      1611768    0    0     0     0 58224  428  61  17  22   0   0
56  0            0      5373304       198732      1611768    0    0     0  7864 64461 8044  62  16  22   0   0
56  0            0      5372548       198732      1611768    0    0     0     0 58869 1351  60  18  22   0   0
56  0            0      5371524       198732      1611768    0    0     0    16 58082  441  60  17  22   0   0
56  0            0      5371020       198732      1611768    0    0     0     0 58120  393  60  18  22   0   0
56  1            0      5370516       198740      1611760    0    0     0  1876 59387 2016  61  17  22   0   0
56  0            0      5370264       198740      1611768    0    0     0  4584 61922 5148  61  17  22   0   0
%Cpu(s):  4,0 us,  1,9 sy,  0,0 ni, 94,0 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
KiB Mem : 32451168 total, 4341112 free, 4257496 used, 23852560 buff/cache
KiB Swap: 32448508 total, 32318732 free, 129776 used, 27298876 avail Mem
```

No guest time
KVM Counters (kvm_stat)
Kvm_stat Counters

2 kinds of counters: per cpu and per guest
Very cheap
Debugfs shows accumulated values across all VMs+all CPUs
No measurable overhead with kvm-unit-tests

```
[root@kvm]# pwd
/sys/kernel/debug/kvm
[root@kvm# ls -R
.
7848-11  efer_reload
        exits
        fpu_reload
        halt_attempted_poll
        halt_exits
        halt_successful_poll

./7848-11:
        efer_reload
        exits
        fpu_reload
        halt_attempted_poll
        halt_exits
        halt_successful_poll
        halt_wakeup

./7848-11/vcpu0:
        tsc_offset
```
What Counters

Common code use
Only 4 common code counters
halt_attempted_poll
halt_successful_poll
halt_poll_invalid
halt_wakeup

Architectures differ a lot
Almost no use by ARM
S390 has highest use and growth

No presentation without a patch
ARM defined halt_* counters but did not expose those in debugfs → will increase the number of ARM counters to 10
kvm_stat Tool
## kvm_stat Tool

**Tool first shipped with qemu-kvm**  
Later moved into kernel

**Initial tool kvm_stat**  
56 lines of code

**Lots of additional features over time**

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Count 1</th>
<th>Count 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>efer_reload</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>exits</td>
<td>2435313</td>
<td>62176</td>
</tr>
<tr>
<td>fpu_reload</td>
<td>563348</td>
<td>24934</td>
</tr>
<tr>
<td>halt_attemp</td>
<td>24490</td>
<td>6</td>
</tr>
<tr>
<td>halt_exits</td>
<td>31591</td>
<td>10</td>
</tr>
<tr>
<td>halt_success</td>
<td>15606</td>
<td></td>
</tr>
<tr>
<td>halt_wakeup</td>
<td>16732</td>
<td>10</td>
</tr>
<tr>
<td>host_state</td>
<td>568124</td>
<td>24721</td>
</tr>
<tr>
<td>hypercalls</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>insn_emulat</td>
<td>298971</td>
<td>11699</td>
</tr>
<tr>
<td>insn_emulat</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>invlpg</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>io_exits</td>
<td>548885</td>
<td>24465</td>
</tr>
<tr>
<td>irq_exits</td>
<td>174556</td>
<td>13328</td>
</tr>
<tr>
<td>irq_injecti</td>
<td>197133</td>
<td>10160</td>
</tr>
<tr>
<td>irq_window</td>
<td>12064</td>
<td>626</td>
</tr>
<tr>
<td>l1d_flush</td>
<td>2059091</td>
<td>50214</td>
</tr>
<tr>
<td>largepages</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>mmio_exits</td>
<td>6441</td>
<td>7</td>
</tr>
<tr>
<td>mmu_cache_m</td>
<td>3351</td>
<td>49</td>
</tr>
<tr>
<td>mmu_flooded</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>mmu_pde_zap</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>mmu_pte_upd</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>mmu_pte_wri</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>mmu_recycle</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>mmu_shadow_</td>
<td>2314</td>
<td></td>
</tr>
<tr>
<td>mmu_unsync</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>nmi_injecti</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>nmi_window</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>pf_fixed</td>
<td>1333525</td>
<td></td>
</tr>
</tbody>
</table>

[...]
Per-guest Counters

**Initially kvm stats were global**
Accumulated over all guests over all vcpus
Data was available in code

Just needed to be exposed via debugfs

**kvm_stat tool**
--guest or g key allows to filter
Logging

Can be redirected into file and imported into spreadsheet
No timestamps
White space as separator
Banner string every 20 lines
A lot of data series
Needs improvement for import

Adding CSV output is relatively straightforward
Remove all ‘‘‘ and use ‘‘’’
Avoid banner line every 20 lines

What about logging into a file? Logrotation?
Integrate into journal?
Mixed with other messages (but can be filtered out)
Separate logfile?
Where to store the header? How often?
Trace Events
Trace Events

**Used to be the new cool kid in the house**
Trace events everywhere

**Plus sides**
Trace events allow to bind values to guest /cpus
Almost no overhead when off
Allows latency measurements

**Trace events for kvm have all the nice filters as normal trace events**
Very fine grained tracing if needed
Great for live debugging

**kvm_stat defaults to trace events**
I suggest to always use -d -t to get trace events and counters
Analyze events for all VMs, all VCPUs:

<table>
<thead>
<tr>
<th>Event</th>
<th>Samples</th>
<th>Samples%</th>
<th>Time%</th>
<th>Min Time</th>
<th>Max Time</th>
<th>Avg Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXTERNAL_INTERRUPT</td>
<td>2845</td>
<td>41.89%</td>
<td>27.40%</td>
<td>0.56us</td>
<td>22085.57us</td>
<td>67.79us ( +-  21.82% )</td>
</tr>
<tr>
<td>MSR_WRITE</td>
<td>1246</td>
<td>18.35%</td>
<td>16.14%</td>
<td>0.71us</td>
<td>14989.71us</td>
<td>91.17us ( +-  28.48% )</td>
</tr>
<tr>
<td>IO_INSTRUCTION</td>
<td>1186</td>
<td>17.46%</td>
<td>37.96%</td>
<td>2.93us</td>
<td>19441.21us</td>
<td>123.58us ( +-  15.43% )</td>
</tr>
<tr>
<td>EPT_VIOLATION</td>
<td>616</td>
<td>9.07%</td>
<td>10.81%</td>
<td>1.05us</td>
<td>21252.50us</td>
<td>123.58us ( +-  37.88% )</td>
</tr>
<tr>
<td>TPR_BELOW_THRESHOLD</td>
<td>265</td>
<td>3.90%</td>
<td>4.08%</td>
<td>0.72us</td>
<td>10350.12us</td>
<td>108.47us ( +-  49.82% )</td>
</tr>
<tr>
<td>PAUSE_INSTRUCTION</td>
<td>203</td>
<td>2.99%</td>
<td>0.09%</td>
<td>0.43us</td>
<td>158.07us</td>
<td>3.08us ( +-  28.50% )</td>
</tr>
<tr>
<td>CPUID</td>
<td>182</td>
<td>2.68%</td>
<td>0.77%</td>
<td>0.46us</td>
<td>4290.89us</td>
<td>29.71us ( +-  81.08% )</td>
</tr>
<tr>
<td>PENDING_INTERRUPT</td>
<td>122</td>
<td>1.80%</td>
<td>1.04%</td>
<td>0.71us</td>
<td>7084.68us</td>
<td>59.78us ( +-  97.11% )</td>
</tr>
<tr>
<td>VMCALL</td>
<td>93</td>
<td>1.37%</td>
<td>0.43%</td>
<td>0.66us</td>
<td>2796.13us</td>
<td>32.85us ( +-  91.42% )</td>
</tr>
<tr>
<td>EPT_MISCONFIG</td>
<td>13</td>
<td>0.19%</td>
<td>1.03%</td>
<td>6.22us</td>
<td>5349.92us</td>
<td>555.56us ( +-  75.00% )</td>
</tr>
<tr>
<td>PREEMPTION_TIMER</td>
<td>10</td>
<td>0.15%</td>
<td>0.00%</td>
<td>0.91us</td>
<td>1.36us</td>
<td>1.09us ( +-  3.83% )</td>
</tr>
<tr>
<td>HLT</td>
<td>8</td>
<td>0.12%</td>
<td>0.24%</td>
<td>3.45us</td>
<td>1143.19us</td>
<td>214.47us ( +-  62.21% )</td>
</tr>
<tr>
<td>DR_ACCESS</td>
<td>3</td>
<td>0.04%</td>
<td>0.00%</td>
<td>2.16us</td>
<td>2.65us</td>
<td>2.44us ( +-  5.98% )</td>
</tr>
</tbody>
</table>

Total Samples:6792, Total events handled time:703886.16us.
Trace Events Everywhere?

Change everything to trace events?
Not quite

Downsides
Measurable overhead for microbenchmarks
Must be activated to “measure” → no history
Kind of ABI (or not?)
libvirt
Libvirt Interfaces

Several interfaces to query domain...

- cpu-stats: show domain cpu statistics
- domblkstat: get device block stats for a domain
- domifstat: get network interface stats for a domain
- dommemstat: get memory statistics for a domain
- domstats: get statistics about one or multiple domains

...Or node information

- nodecpustats: Prints cpu stats of the node.
- nodememstats: Prints memory stats of the node.
Libvirt Interfaces - cpu-stats

Provides per-cpu times
vcpu understands guest time
Queried from cgroup cpuacct

```bash
virsh cpu-stats myguest
CPU0:
  cpu_time             38.250348806 seconds
  vcpu_time            36.329019159 seconds
CPU1:
  cpu_time             34.048214224 seconds
  vcpu_time            32.213941613 seconds
CPU2:
  cpu_time             39.577205065 seconds
  vcpu_time            37.958059656 seconds
CPU3:
  cpu_time             36.528732041 seconds
  vcpu_time            34.997068067 seconds
CPU4:
  cpu_time             27.706520847 seconds
  vcpu_time            27.414863236 seconds
CPU5:
  cpu_time             22.345352968 seconds
  vcpu_time            21.986850492 seconds
CPU6:
  cpu_time             28.786509907 seconds
  vcpu_time            27.919375487 seconds
CPU7:
  cpu_time             25.532488444 seconds
  vcpu_time            25.103798154 seconds
Total:
  cpu_time             252.775372302 seconds
  user_time            0.890000000 seconds
  system_time          90.320000000 seconds
```
Libvirt Interfaces - nodecpustats

Not all interfaces understand guest time
The internal interface virNodeGetCPUStats and virs nodecpustats do not know about guest time

```
$ virsh nodecpustats
user: 32885220000000
system: 26527760000000
idle: 3035468670000000
iowait: 869040000000

$ virsh nodecpustats
user: 32885220000000
system: 26527760000000
idle: 3035468670000000
iowait: 869040000000
```
Large scale tools
ELK Stack

Elastic search logstash kibana
History data
Can be analyzed on a different system
Metricbeat understands system, user, nice, irq steal and other times
No understanding of guest and guest nice time
KVM beat understands dommemstat from libvirt
No understanding of kvm counter and trace points
No understanding of other libvirt interfaces
Prometheus + Grafana

**Prometheus/graphana**
History data
Can be analyzed on a different system

**Several dashboards for libvirt**
Via prometheus or via collectd from libvirt
System-wide data (cpu, memory, swap)
Again, no kvm stat
Some dashboards understand guest time (hooray)
Moving Forward and Summary

Use the interfaces

Extend the interfaces if you miss something
Most tools only support a tiny subset of the data
We need to add things where appropriate

Talk about what's there
KVM provides more data than most people know
We need to get this message out

Next steps

More common kvm_stat counters
todo
Revalidate if we have the right counters
todo
Enable guest time in vmstat
Submitted
Enable guest time in top
Patch written
Find a new place for kvm stat (no debugfs)
Todo
Integrate kvm_stat in ELK stack
todo
Integrate kvm_stat in Prometheus/Grafana
todo
Integrate kvm_stat in sysstat
todo
Thanks

• Christian Borntraeger borntraeger@de.ibm.com
32 bit counters

Initially all counters 32 bit
Power introduced a new counter that added time: halt_poll_success_ns et al
64 bit needed

64 bit counters on 32 bit CPUs
Per cpu-counters are easy
Per VM counters would require locking
Idea: 64bit per cpu, unsigned long per vm.

Power-specific?
Is the idea of halt_poll_success_ns good for everyone?
On 22/01/19 21:48, Greg Kroah-Hartman wrote:

>> This also brings the question: shall we move these counters out of debugfs into something else?
> If you have code that relies on debugfs, yes, you need to move that out of debugfs because more and more systems are trying to disable it due to the obvious problems with it (i.e. leaking tons of debugging information).

> debugfs is for DEBUG information, not for "statistics about how my VM is working". That sounds like something you need to rely on, so debugfs is not the place for it.

Yes, we know that and tracepoints are already one replacement. However, they are slower than just a lock-free "vcpu->stats.foo_happened++".

Another idea that Steven Rostedt and I discussed a while ago is some kind of "statfs" which would already provide some code, similar to the one that KVM uses to accumulate statistics from multiple VMs or multiple VCPUs into a single counter.

Paolo
Potential micro optimization

Almost no measurable overhead with unit test
For testing I removed all vcpu→stat.<counter>++ lines

There is one thing that can be optimized
Group frequent counter with other frequent counters
to properly use the cache lines containing the counter
Usually 8 counters per cacheline, for a random workload I marked the most common ones
<table>
<thead>
<tr>
<th>Event</th>
<th>Total</th>
<th>%Total</th>
<th>CurAvg/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>kvm_exit</td>
<td>3943</td>
<td>17.1</td>
<td>236</td>
</tr>
<tr>
<td>kvm_exit(MSR_WRITE)</td>
<td>2647</td>
<td>67.1</td>
<td>162</td>
</tr>
<tr>
<td>kvm_exit(HLT)</td>
<td>936</td>
<td>23.7</td>
<td>56</td>
</tr>
<tr>
<td>kvm_exit(EXTERNAL_INTERRUPT)</td>
<td>209</td>
<td>5.3</td>
<td>12</td>
</tr>
<tr>
<td>kvm_exit(PENDING_INTERRUPT)</td>
<td>23</td>
<td>0.6</td>
<td>2</td>
</tr>
<tr>
<td>kvm_exit(IO_INSTRUCTION)</td>
<td>7</td>
<td>0.2</td>
<td>0</td>
</tr>
<tr>
<td>kvm_entry</td>
<td>3943</td>
<td>17.1</td>
<td>236</td>
</tr>
<tr>
<td>kvm_hv_timer_state</td>
<td>3587</td>
<td>15.5</td>
<td>217</td>
</tr>
<tr>
<td>kvm_msr</td>
<td>2647</td>
<td>11.5</td>
<td>162</td>
</tr>
<tr>
<td>kvm_fpu</td>
<td>1872</td>
<td>8.1</td>
<td>112</td>
</tr>
<tr>
<td>kvm_inj_virq</td>
<td>1061</td>
<td>4.6</td>
<td>61</td>
</tr>
<tr>
<td>kvm_eoi</td>
<td>1061</td>
<td>4.6</td>
<td>61</td>
</tr>
<tr>
<td>kvm_apic_accept_irq</td>
<td>1061</td>
<td>4.6</td>
<td>61</td>
</tr>
<tr>
<td>kvm_pv_eoi</td>
<td>1048</td>
<td>4.5</td>
<td>61</td>
</tr>
<tr>
<td>kvm_ple_window</td>
<td>937</td>
<td>4.1</td>
<td>56</td>
</tr>
<tr>
<td>kvm_vcpu_wakeup</td>
<td>936</td>
<td>4.1</td>
<td>56</td>
</tr>
<tr>
<td>kvm_wait_lapic_expire</td>
<td>913</td>
<td>4.0</td>
<td>56</td>
</tr>
<tr>
<td>kvm_halt_poll_ns</td>
<td>39</td>
<td>0.2</td>
<td>0</td>
</tr>
<tr>
<td>kvm_msi_set_irq</td>
<td>21</td>
<td>0.1</td>
<td>0</td>
</tr>
<tr>
<td>kvm_apic</td>
<td>13</td>
<td>0.1</td>
<td>0</td>
</tr>
<tr>
<td>kvm_pio</td>
<td>7</td>
<td>0.0</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>23089</td>
<td>1373</td>
<td></td>
</tr>
</tbody>
</table>