Observability in KVM
How to troubleshoot virtual machines

Stefan Hajnoczi <stefanha@redhat.com>
FOSDEM 2015
In this talk we can only scratch the surface (sorry)
About me

QEMU contributor since 2010

- Block layer co-maintainer
- Tracing and net subsystem maintainer
- Google Summer of Code & Outreach Program for Women mentor and administrator

I work in Red Hat's KVM virtualization team
Common questions on #qemu IRC

“My VM cannot connect to the internet. What's wrong?”

“Copying files is slow in the VM. How can I make it fast?”

These problems can be solved through troubleshooting, but QEMU is a black box to many users.

This talk is about how to get to the bottom of these types of issues.
What's required for troubleshooting?

Systematic approaches require a **mental model**

Knowing components and their relationships allows you to ask the right questions.
How to troubleshoot KVM issues

Get familiar with the components and key characteristics of KVM

Make use of observability tools:

- Performance statistics
- Network packet capture
- Log files
- Tracing

Use scientific process to determine root cause
Components in the KVM virtualization stack

Management for datacenters and clouds
- OpenStack
- oVirt

Management for one host
- libvirt

Emulation for one guest
- QEMU
- Guest

Host hardware access and resource mgmt
- Host kernel
- kvm.ko
General troubleshooting with libvirt and KVM

Use **virsh(1)** to inspect virtual machines

- Far too many commands to list, see “virsh help”

Libvirt keeps logs for each virtual machine at
/var/log/libvirt/qemu/<domain>.log

Also check dmesg(1) for kernel messages such as Out-of-Memory killer, segmentation faults, or error messages from kvm.ko module
Tracing

Tracing is useful for performance analysis, requires low-level knowledge and/or familiarity with code

Using `strace -f` on QEMU is noisy but can be done

kvm.ko kernel trace events available via `perf(1)` and `trace-cmd(1)`

Some distros ship QEMU with a `SystemTap` tapset

- Advantage: combine host kernel and QEMU traces
The big secret to troubleshooting KVM

Plain old Linux commands like `ps(1)`, `vmstat(1)`, `tcpdump(8)`, etc work!

There is less virtualization magic than one might think.
Part 1 - CPU
Virtual machine CPU execution (overview)

1 QEMU process per guest

1 “vcpu thread” per guest CPU

Host kernel schedules vcpu threads like normal threads
CPU utilization breakdown on KVM hosts

Useful CPU utilization categories:

1) Guest code (%guest)
   - Kernel and userspace

2) QEMU (%usr)
   - Device emulation, live migration, etc

3) Other host userspace (%usr)
   - Are you running bitcoind on the host?!

4) Host kernel (%sys, %irq, %soft)
   - Caused by I/O or userspace activity
Host shows high CPU utilization, what's wrong?

top(1) on host shows 25% user process CPU time

Tool: mpstat(1) from the “sysstat” package offers detailed processor statistics

<table>
<thead>
<tr>
<th></th>
<th>%usr</th>
<th>%nice</th>
<th>%sys</th>
<th>%iowait</th>
<th>%irq</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.40</td>
<td>0.00</td>
<td>0.40</td>
<td>0.30</td>
<td>0.00</td>
</tr>
<tr>
<td>%soft</td>
<td>0.00</td>
<td>0.00</td>
<td>25.01</td>
<td>0.00</td>
<td>73.89</td>
</tr>
</tbody>
</table>

25.01% guest means 1 out of 4 host CPUs is maxed out running guest code.

Result: Check if guest is stuck in an infinite loop or use <cputune> libvirt XML for cgroups resource control
Is my cloud guest getting enough CPU?

Host may report how long runnable vcpus wait to run on a physical CPU

Reported as %steal in mpstat(1)

Requires host to cooperate – may be disabled

Good for identifying overloaded hosts
Virtual machine CPU execution (low-level)

vcpu thread calls ioctl(KVM_RUN) repeatedly to run guest code

Kicked out of guest code by hardware register accesses, interrupts, model specific registers, etc

vcpu thread state machine
Observing low-level events with `kvm_stat`

`kvm_stat` is a `top(1)`-like tool for KVM event counters:

```
kvm_exit   809319  432
kvm_entry  809319  432
kvm_msr    593133  318
kvm_inj_virq 196268  112
kvm_eoi    196165  112
...```

These KVM trace events can also be observed with `perf record -a -e kvm:*`
100% CPU while sitting at the GRUB menu?

Suspicious events are typically >10,000 events/sec:

kvm_exit  ...  880112
kvm_cr    ...  805440

“cr” ← x86 control registers (e.g. changing into protected mode)

This could be a guest is spinning in a loop that transitions back and forth between real mode and protected mode.
Part 2 - Networking
Virtual machine networking

vhost_net with bridged networking is a popular configuration

Guest interface: eth0 emulated virtio-net NIC
Host interface: vnet0 tun software interface

External network connectivity through software bridge (virbr0)

Other guests can be connected to same bridge for guest<->guest connectivity
Troubleshooting bridged networking

tcpdump eth0 inside guest
  • Does guest receive traffic and get ARP responses?
tcpdump vnet0 on host
  • Does host see guest outgoing traffic?
  • Does the bridge forward guest incoming traffic?
tcpdump virbr0 on host
  • Does the bridge see traffic?
tcpdump eth0 on host
  • Does physical traffic look as expected?
Host-wide interface statistics

    # netstat -i
    Iface    MTU  RX-OK ...  TX-OK ...
    virbr0   1500  2669 ...  4611 ...
    virbr0-n 1500   0 ...    0 ...
    vnet0    1500  41 ...   502 ...
    wlp3s0   1500 1500554 ... 387876 ...

Guest network interface names can be queried:

    # virsh domiflist rhel7
    Interface Type    Source  Model  MAC
    vnet0    network default virtio 52:...
Popular NAT networking configuration

Guests on private bridge with iptables NAT rules for external connectivity
- Private guest IP range
- Only one public IP for host and guests
- Requires port-forwarding for incoming connections

DNS and DHCP services typically provided by host using dnsmasq
Now you can troubleshoot DHCP and DNS too

```
(host)# journalctl -r | head  # or syslog
dnsmasq-dhcp[1173]: DHCPDISCOVER(virbr0)
  192.168.122.252 52:54:00:52:fe:24

dnsmasq-dhcp[1173]: DHCPOFFER(virbr0)
  192.168.122.252 52:54:00:52:fe:24

dnsmasq-dhcp[1173]: DHCPREQUEST(virbr0)
  192.168.122.252 52:54:00:52:fe:24

dnsmasq-dhcp[1173]: DHCPACK(virbr0)
  192.168.122.252 52:54:00:52:fe:24
```
Part 3 – Disk I/O
Popular LVM local disk configuration

Storage provided to guest as virtio-blk PCI adapter

QEMU typically configured with cache=none to bypass host page cache

LVM offers good performance and storage management features

Guest kernel

virtio_blk

QEMU

Linux AIO

Host kernel

lv_guest01
Why can't QEMU open the disk image file?

Libvirt can launch QEMU as an unprivileged user with SELinux isolation

Check that QEMU process uid/gid can access disk image file

Check SELinux audit logs in /var/log/audit/audit.log for denials

Libvirt SELinux configuration in /etc/libvirt/qemu.conf
Benchmarking disk performance

Apples-to-oranges comparisons are very common!

Use `fio -direct=1` for benchmarking to bypass page cache

Use `fio -rw=randwrite` for a random pattern that avoids QEMU virtio-blk write merging
I/O statistics with `iostat(1)`

```bash
$ iostat -k -x 1
Device: ... r/s  w/s  rkB/s  wkB/s
sda    0.00 13.00  0.00  51.20
        avgreq-sz   avgqus-sz ... 
        7.88  0.01
```

Compare guest and host to identify unexpected changes including:

- Page cache usage (request not sent to device)
- Request merging
- Request parallelism (queue depth)
I/O patterns with blktrace(8)

To study the exact pattern of I/O requests:

8,0 3 1 0.0000000000 21846 A W ...
8,0 3 2 0.0000000770 21846 Q W ...
8,0 3 3 0.0000004564 21846 G W ...
8,0 3 4 0.0000006611 21846 I W ...
8,0 3 5 0.0000017716 21846 D W ...
8,0 0 1 0.001158278 0 C W ...

This truncated example shows a write request on device 8,0 taking 1.16 milliseconds.
Questions?

Email: stefanha@redhat.com
IRC: stefanha on #qemu irc.oftc.net
Blog: http://blog.vmssplice.net/

QEMU: http://qemu-project.org/

Slides available on my website: http://vmssplice.net/