Scalability and stability of libvirt: Experiences with very large hosts

—

Marc Hartmayer

Trademarks & Disclaimer

—

The following are trademarks of the International Business Machines Corporation in the United States and/or other countries. For a complete list of IBM Trademarks, see www.ibm.com/legal/copytrade.shtml:

IBM, the IBM logo, IBM Z, IBM z Systems, IBM z14, WebSphere, DB2 and Tivoli are trademarks of IBM Corporation in the United States and/or other countries. For a list of additional IBM trademarks, please see https://ibm.com/legal/copytrade.shtml.

The following are trademarks or registered trademarks of other companies: Java and all Java based trademarks and logos are trademarks of Sun Microsystems, Inc., in the United States and other countries or both Microsoft, Windows,Windows NT and the Windows logo are registered trademarks of Microsoft Corporation 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.
UNIX is a registered trademark of The Open Group in the United States and other countries or both. Linux is a trademark of Linus Torvalds in the United States, other countries, or both. Cell Broadband Engine is a trademark of Sony Computer Entertainment Inc. InfiniBand is a trademark of the InfiniBand Trade Association.
Other company, product, or service names may be trademarks or service marks of others.

NOTES: Linux penguin image courtesy of Larry Ewing (lewing@isc.tamu.edu) and The GIMP

Any performance data contained in this document was determined in a controlled environment. Actual results may vary significantly and are dependent on many factors including system hardware configuration and software design and configuration. Some measurements quoted in this document may have been made on development-level systems. There is no guarantee these measurements will be the same on generally-available systems. Users of this document should verify the applicable data for their specific environment. IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.
Information is provided “AS IS” without warranty of any kind. All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.

Trademarks & Disclaimer #2

—

This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.
All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.
Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Prices are suggested US list prices and are subject to change without notice. Starting price may not include a hard drive, operating system or other features. Contact your IBM representative or Business Partner for the most current pricing in your geography. Any proposed use of claims in this presentation outside of the United States must be reviewed by local IBM country counsel prior to such use. The information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any

Notice Regarding Specialty Engines

Any information contained in this document regarding Specialty Engines (“SEs”) and SE eligible workloads provides only general descriptions of the types and portions of workloads that are eligible for execution on Specialty Engines (e.g., zIIPs, zAAPs, and IFLs). IBM authorizes customers to use IBM SE only to execute the processing of Eligible Workloads of specific Programs expressly authorized by IBM as specified in the “Authorized Use Table for IBM Machines” provided at www.ibm.com/systems/support/machine_warranties/machine_code/aut.html (“AUT”).
No other workload processing is authorized for execution on an SE.
IBM offers SEs at a lower price than General Processors/Central Processors because customers are authorized to use SEs only to process certain types and/or amounts of workloads as specified by IBM in the AUT.

It all started with a performance bug

For multiple domains:

```
# while virsh start $vm && virsh destroy $vm; do : ; done
```

→ ~30s hang ups of the libvirtd main loop

Agenda

1. Test Setup and Scenarios
2. Stability
3. Performance
4. Summary and Outlook

Test Environment

All tests were conducted on the following system:

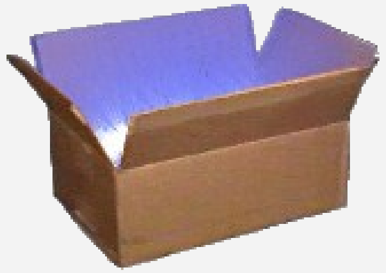
- 64 shared cores (z14)
- 4TB RAM
- Distro: Fedora 28, SELinux enforced
- Libvirt: commit 0a7101c89b78
- Kernel: 4.19+
- QEMU: 3.0.0

Source: https://mp.s81c.com/8034F2C/dal05/v1/AUTH_db1cfc7b-a055-460b-9274-1fd3f11fe689/266ef7f57b168d4e5dd7994d6a65327b/additionalOfferingImg__0_318d9711-7e49-4a46-ba5b-a262328c8204.png

Test Setup

Guest definition

- host kernel + minimal initrd (with Busybox)
- 1 vCPU
- 100mb RAM
- direct kernel boot
- SCLP console
- SCSI disks



```
<domain type='kvm'>
  <name>{{ name }}</name>
  <memory unit='MiB'>100</memory>
  <os>
    <type arch='s390x'>hvm</type>
    <kernel>/var/lib/libvirt/images/vmlinux-s390x</kernel>
    <initrd>/var/lib/libvirt/images/rfs-s390x.gz</initrd>
    <cmdline>root=/dev/ram</cmdline>
  </os>
  <devices>
    <console type='pty'>
      <target type='sclp'/>
    </console>
    {%- for disk in disks %}
    <disk type='block' device='disk'>
      <source dev="{{ disk.path }}"/>
      <target dev='{{ disk.dev }}' bus='scsi'/>
    </disk>
    {%- endfor %}
  </devices>
</domain>
```

Source: https://busybox.net/images/busybox1.png

Used system configuration

Adjusted the values suggested by the presentation from Jens Freimann *(“Pushing the limits: 1000 guests per host and beyond” - KVM forum 2015)*

- `sysctl -w kernel.pid_max=348160`
- `sysctl -w kernel.threads-max=33029620`
- `sysctl -w kernel.pty.max=20000`
- `sysctl -w fs.file-max=42653636`
- `sysctl -w fs.inotify.max_user_watches=524288`
- Increased `ulimit -n` for libvirtd accordingly

Used libvirt configuration

Default `libvirtd.conf` except

- `max_anonymous_clients = 100`
- `max_client_requests = 10`
- `max_workers = 64`
- `prio_workers = 10`

Default `qemu.conf` except

- `max_process = 16384`
- `max_files = 262144`

Test Setup

SCSI disks used for the guests

scsi_debug module used for the SCSI disks

- avoids the usage of real disks
- could be used for passthrough

```
# modprobe scsi_debug add_host=8 num_tgts=8 max_luns=256 \
    dev_size_mb=1
```

Test Scenarios

Trying to reproduce the bug

- Start/Destroy guests concurrently
- Define/Undefine guests concurrently
- Start/Managedsave concurrently

WHAT ELSE COULD POSSIBLY GO WRONG?

Encountered problems: deadlocks

Deadlock across fork() in virCommandExec()

- start/destroy in a loop for multiple domains
- fixed by commit 5fec1c3a5c0f

Race condition when counting unauthenticated clients

- results in a libvirtd that does not accept new connections
- connect/disconnect concurrently with multiple clients
- fixed by commit 94bbbcee1f23

Encountered problems: other race conditions

NULL pointer dereferencing when libvirtd reconnects to QEMU processes

- events were “handled” before the QEMU driver was initialized
- fixed by commit fef4d132c4e3

Double free’ing

- caused a segmentation fault
- define/undefine the same domain concurrently
- fixed by commit 7e760f61577e

after two days running...

after two days running… no segmentation faults

Back to the original bug

Main thread*

```
while True:
   poll(monitors, client sockets, …)
   virEventPollDispatchHandles
      qemuMonitorIO
         qemuProcessHandleMonitorEOF
            virObjectLock(vm)
```

Worker thread*

```
virNetServerHandleJob
   qemuDomainDestroyFlags
      qemuDomObjFromDomain
         virObjectLock(vm)
      qemuProcessStop
         qemuRemoveCgroup
            virDBusCall(..., timeout=30s)
```

* Very simplified

Back to the original bug

Main thread*

```
while True:
    poll(monitors, client sockets, …)
    virEventPollDispatchHandles
        qemuMonitorIO
            qemuProcessHandleMonitorEOF
                virObjectLock(vm)
```

Worker thread*

```
virNetServerHandleJob
    qemuDomainDestroyFlags
        qemuDomObjFromDomain
            virObjectLock(vm)
        qemuProcessStop
            qemuRemoveCgroup
                virDBusCall(..., timeout=30s)
```

* Very simplified

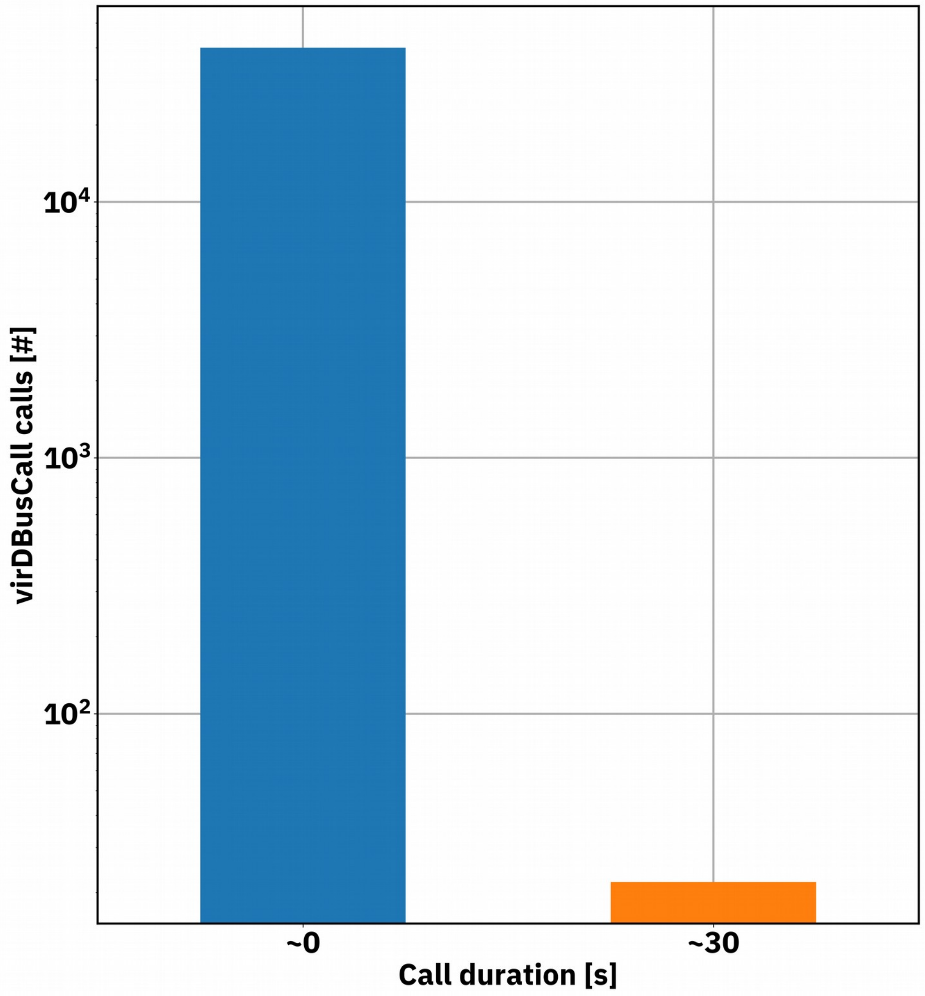
Problem observation

The D-Bus calls are

- either fast
- or they need the total timeout

but nothing in between.

No real timeout occurred!



* SystemTap used for instrumentation

Possible solutions

“If you use this low-level API directly, you're signing up for some pain.”*

Yep, we do so.

So we could either

- fix it within libvirt
- use another D-Bus library

* https://dbus.freedesktop.org/doc/api/html/ (visited on 2018.10.01)

NEVER EVER BLOCK YOUR MAIN LOOP!

Possible solutions

- no worker thread should block the main loop
- only dispatch the events in the main loop
- handle events in a thread pool*

* See presentation “Lessons in running libvirt at scale” from Prerna Saxena from last years KVM forum.

more on performance

How fast can we go?

Command line start of QEMU versus start via libvirtd

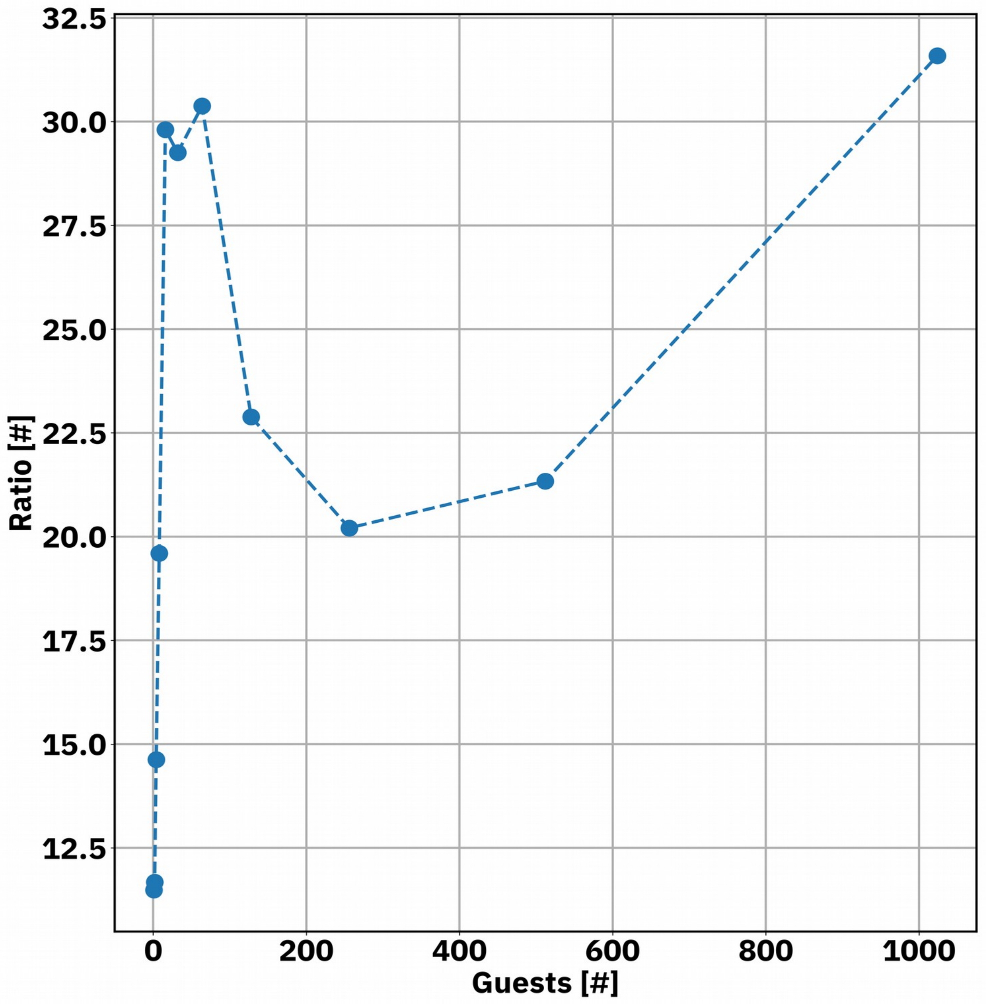
- it’s a real unfair comparison… since libvirt does so much more, but let’s approximate the “optimum”
- no disk per guest
- self-written Python3 test script:
  - using 64 threads
  - methods: direct command line and libvirt
- `# qemu-system-s390x -S $*`

Starting guests

Direct QEMU command line vs. libvirt

$$ratio(i) = \frac{t_{libvirt}(i)}{t_{cmdline}(i)}$$

Where does the time go?



Where does the time go?

Additionally, libvirt:

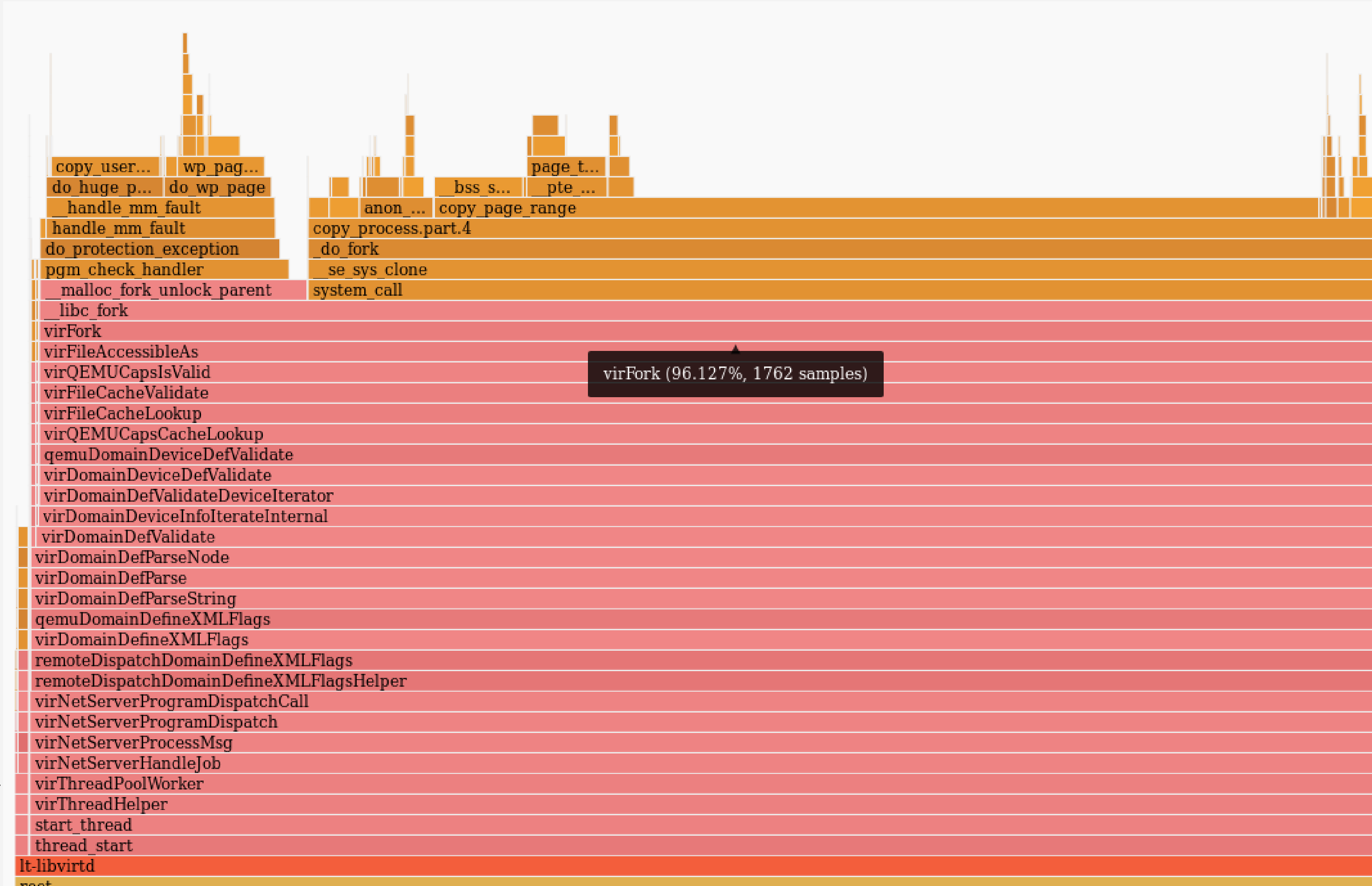
- prepares the host
  - vsock
  - hostdevs
  - …
- prepares the QEMU process
  - cgroups
  - namespaces
  - SELinux labels
  - …
- handles QEMU capabilities
- auditing
- logging
- …

Performance

Where does the time go for the **define** operation?

On-CPU flame graph* when defining guests for 60 seconds each with 20 SCSI disks

- 96.127% of the time is used for `virFork`
- which is called within `virDomainDefValidateDeviceIterator`



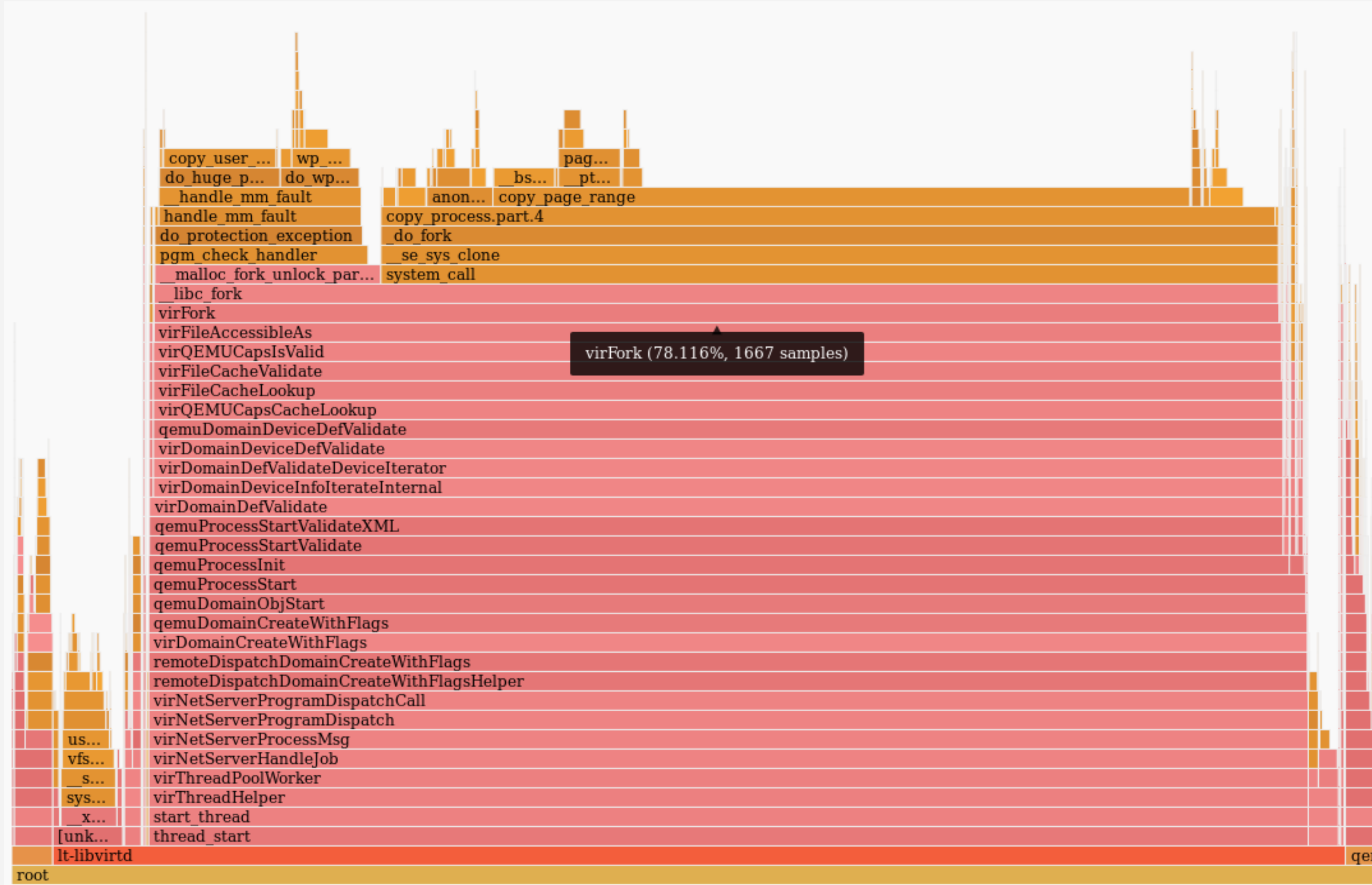
*See http://www.brendangregg.com/flamegraphs.html for more information

Performance

Where does the time go for the **start** operation?

On-CPU flame graph when starting guests for 60 seconds each with 20 SCSI disks

- 78.116% of the time is used for `virFork`
- which is called within `virDomainDefValidateDeviceIterator`



What does virQEMUCapsCacheLookup do?

- Probing the QEMU capabilities is expensive

  → Caching was introduced
- Looks up the QEMU capabilities for the domain in the cache + validates that these capabilities are still valid
  - Fork for verifying `/dev/kvm` is accessible as `qemu:qemu`

Do we really need this validation for **each** device of a domain?
Because the more devices a domain has the more expensive it is

Possible improvements

- Query the QEMU capabilities once for each task (e.g. define, start, ...)

  See my patch series “Avoid numerous calls of virQEMUCapsCacheLookup”*

  - this also avoids using different QEMU capabilities for the same task
- Use `vfork` + `execve` a dedicated program instead of a expensive fork

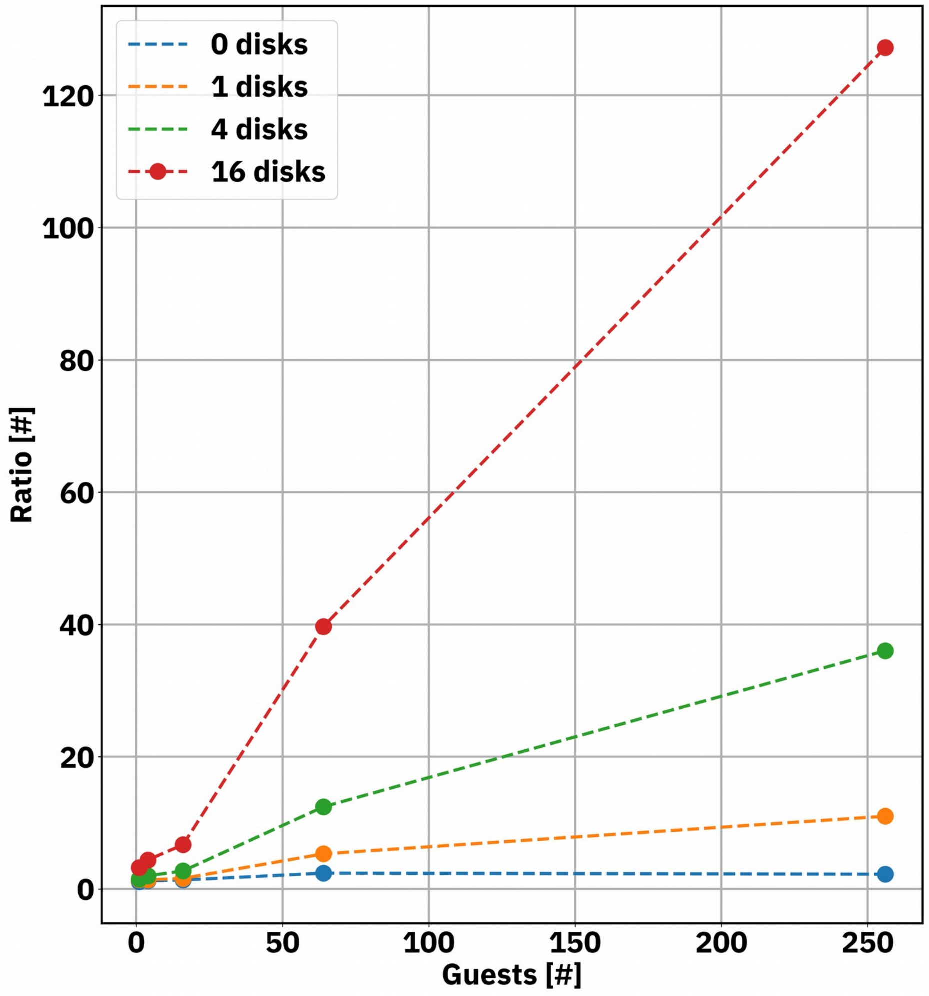
* https://www.redhat.com/archives/libvir-list/2018-September/msg01092.html

Performance results

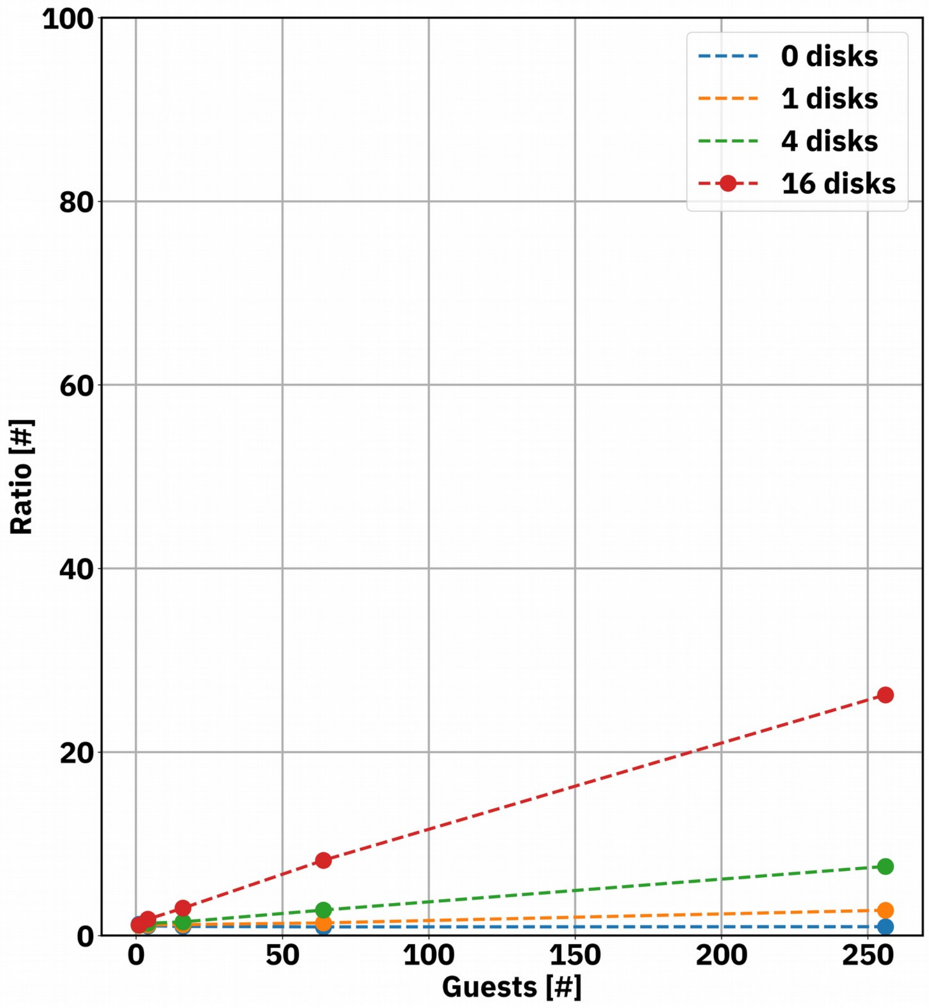
- baseline: libvirt (commit 0a7101c89b78)
- improved: libvirt (commit 0a7101c89b78) + my patch series “Avoid numerous calls of virQEMUCapsCacheLookup”

$$ratio(i) = \frac{t_{baseline}(i)}{t_{improved}(i)}$$

definition
256 guests
16 disks
127x



start
256 guests
16 disks
26x



What can be optimized?

- don’t block the main loop

  see “Lessons in running libvirt at scale”

- optimize QEMU capabilities usage

  see my patch series

- fix the 30 seconds D-Bus problem

Further analysis

- locking strategies
  - Optimize locking of `virDomainObjList` and `virDomainObj`
  - …

  → **Analyze Off-CPU times!**
- what happens for more sophisticated operations?

  e.g. live migration
- what happens if we kill QEMU processes randomly?

  e.g. during migration

Questions?