Disk I/O Performance of Kata Containers

Bharat Kunwar

StackHPC

Who am I?

StackHPC

- Live in Bristol, UK
- Work at StackHPC, a Bristol based HPC/Cloud consultancy
- Core reviewer for OpenStack Magnum, a project for deploying and managing Kubernetes cluster lifecycle which integrates with other OpenStack resources (e.g. Identity, Block Storage, LBaaS). Ussuri release supports:
    - Kubernetes v1.18.x
    - Fedora CoreOS 31 via podman
    - Containerd (consequently Kata)
    - Nodegroups
    - Rolling upgrades

What are katacontainers?

- Like **containers** but really **lightweight VMs**
- Has roots in **Intel Clear Containers** and **Hyper runV** technology
- Integrates seamlessly with **Docker** 🐳 and **Kubernetes** ☸️
- Often mentioned alongside **gVisor**, which aims to solve a similar problem by filtering and redirecting system calls to a separate user space kernel (which as a result suffers from runtime performance penalties).

Why does this matter?

- We may want to run **untrusted workloads** with the isolation gained by not sharing the OS kernel with the host (although this assumption is challenged in a recent survey of virtual machines and containers [1].)
- However, if the work is **I/O bound**, as HPC workloads often are, we may want to take into consideration the **trade-off** between the **security isolation** gained versus bare metal/runC container I/O performance.

Considerations: hardware

- Kata will only run on a machine configured to support **nested virtualisation**.
  - `egrep --color 'vmx|svm' /proc/cpuinfo`
- Kata requires **at least a Westmere** processor architecture

Considerations: virtio-9p vs virtio-fs

- virtio-9p is based on existing network protocol that is not optimized for virtualization use cases
- virtio-fs (available since Kata v1.7.0) takes advantage of the virtual machine’s co-location with the hypervisor
  - Experimental support for DAX where file contents can be mapped into a memory window on the host, allowing the guest to directly access data from the host page cache
    - Reduced memory footprint as guest cache is bypassed
    - No communication necessary, (hopefully) improving I/O performance

Deploying Kata

- Kata containers are **OCI conformant** which means that a Container Runtime Interface (CRI) that supports external runtime, e.g. CRI-O and containerd which use **runC** by default can instead use **kata-qemu** (since Kata 1.6.0 which uses 9pfs[2]) or **kata-qemu-virtiofs** runtimes (since Kata 1.9.0 but previously packaged into **kata-nemu** since Kata 1.7.0).
- From Kubernetes 1.14+ onwards, the **RuntimeClass** feature flag has been promoted to beta, therefore enabled by default. Consequently the setup is relatively straightforward (for **kata-qemu** using 9pfs at least).

Deploying Kata

- Clone Kata packaging repo:

```
git clone https://github.com/kata-containers/packaging -b stable-1.9
cd packaging
```

- Register RBAC, runtime classes and deploy Kata binaries:

```
kubectl apply -f kata-deploy/kata-rbac.yaml
kubectl apply -f kata-deploy/k8s-1.14/kata-qemu-runtimeClass.yaml
kubectl apply -f kata-deploy/k8s-1.14/kata-qemu-virtiofs-runtimeClass.yaml
kubectl apply -f kata-deploy/kata-deploy.yaml
```

- Add one of the following to your Pod spec:

```
runtimeClassName: kata-qemu
runtimeClassName: kata-qemu-virtiofs
```

Omit `runtimeClassName` for runC

Our test apparatus

- 1 master, 2 workers, all with 32 processing units and 125G RAM each
- BeeGFS 🐝 based NVME storage backend over 100Gbps Infiniband
    - Configured using our Ansible role available on Galaxy: stackhpc.beegfs
- Kubernetes v1.16.0 with containerd v1.2.6
    - Configured using Kubespray: https://github.com/kubernetes-sigs/kubespray since containerd support in Magnum is work in progress
- Kata v1.9.1
    - Deployed from Kubernetes templates: https://github.com/kata-containers/packaging

Challenges: BeeGFS 💔 virtio-fs v0.3

- There was a mismatch in syscalls instantiated by `virtiofsd` (v0.2 shipped with Kata v1.7.0 -> v0.3 shipped with Kata v1.9.1) to the underlying BeeGFS filesystem leading to `-EINVAL` error, symptom: FIO jobs never manage to run to completion.
- Additionally, we get an `-EIO` failure because of this check inside `fs/dax.c` where `inode->i_blkbits` resolves to 19 and `PAGE_SHIFT` resolves to 12:

```
if (WARN_ON_ONCE(inode->i_blkbits != PAGE_SHIFT))
    return -EIO;
```

- Additionally, `virtiofsd` (v0.3 shipped with Kata v1.9.1) was incompatible with the host OS kernel version (3.10.0-1062).

Solution: thanks stefanha & vgoyal 🙌

StackHPC

- Patch `virtio-fs-dev` branch of https://gitlab.com/virtio-fs/linux.git with this patch: https://gist.github.com/brtknr/5fe95642a67b8f28139db953413b91b0 and build the kernel
- Build `qemu-system-x86_64` and `virtiofsd` binaries from `virtio-fs-dev` branch of https://gitlab.com/virtio-fs/qemu.git for ensuring compatibility with host kernel (3.10.0-1062)
- Point `configuration-virtiofs.toml` file inside `/usr/local/bin/containerd-shim-kata-qemu-virtiofs-v2` to a config file targeting these custom kata binaries.

configuration-virtiofs.toml:

```
14,15d13
< path = "/opt/kata/bin/qemu-virtiofs-system-x86_64"
< kernel = "/opt/kata/share/kata-containers/vmlinuz-virtiofs.container"
16a15,16
> path = "/mnt/storage-nvme/kata/qemu/x86_64-softmmu/qemu-system-x86_64"
> kernel = "/mnt/storage-nvme/kata/linux/arch/x86/boot/bzImage"
105c105
< virtio_fs_daemon = "/opt/kata/bin/virtiofsd"
---
> virtio_fs_daemon = "/mnt/storage-nvme/kata/qemu/virtiofsd"
108c108
< virtio_fs_cache_size = 1024
---
> virtio_fs_cache_size = 0
131c131
< virtio_fs_cache = "always"
---
> virtio_fs_cache = "auto"
```

https://gist.github.com/brtknr/84aa4370c2e7c4ff2b00e30e677aefad

fio_jobfile.fio

```
[global]
; Do not use fallocate. Not all the filesystem types we can test (such as 9p) support
; this - which can then generate errors in the JSON datastream.
fallocate=none
; Limit runtime
runtime=30
; Ensure that jobs run for a specified time limit, not I/O quantity
time_based=1
; To model application load at greater scale, each test client will maintain
; a number of concurrent I/Os.
ioengine=libaio
iodepth=8
; Note: these two settings are mutually exclusive
; (and may not apply for Windows test clients)
direct=1
buffered=0
; Settings from Kata container repo
invalidate=1
ramp_time=0
; Set a number of workers on this client
thread=0
numjobs=4
group_reporting=1
; Each file for each job thread is this size
filesize=32g
size=32g
filename_format=$jobnum.dat
```

```
[fio-job]
rw=${FIO_RW}
```

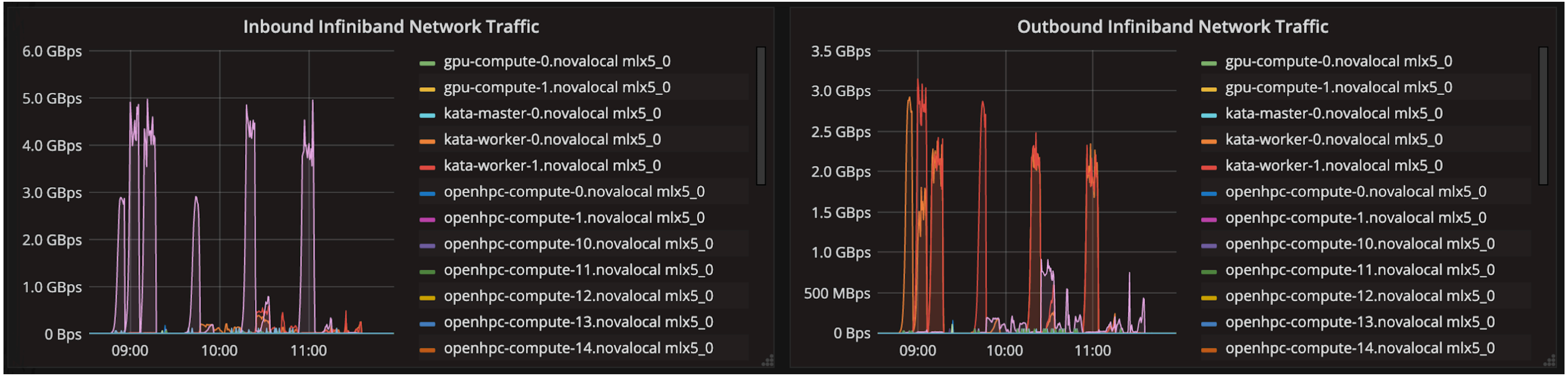
```
fio fio_jobfile.fio --directory=/beegfs/ --output-format=json+ --blocksize=65536 --output=65536.json
```

60 I/O scenarios (5x3x4)

Scenario	Number of clients	Disk I/O pattern (FIO_RW)
bare metal (3.10.0-1062)	1	(sequential) read
runC containers (3.10.0-1062)	8	randread
kata-qemu (4.19.75)	64	(sequential) write
kata-virtiofs (5.3.0-rc3+ with custom modifications, virtio_fs_cache_size = 0)		randwrite
kata-virtiofs (5.3.0-rc3+ with custom modifications, virtio_fs_cache_size = 1024) with DAX		

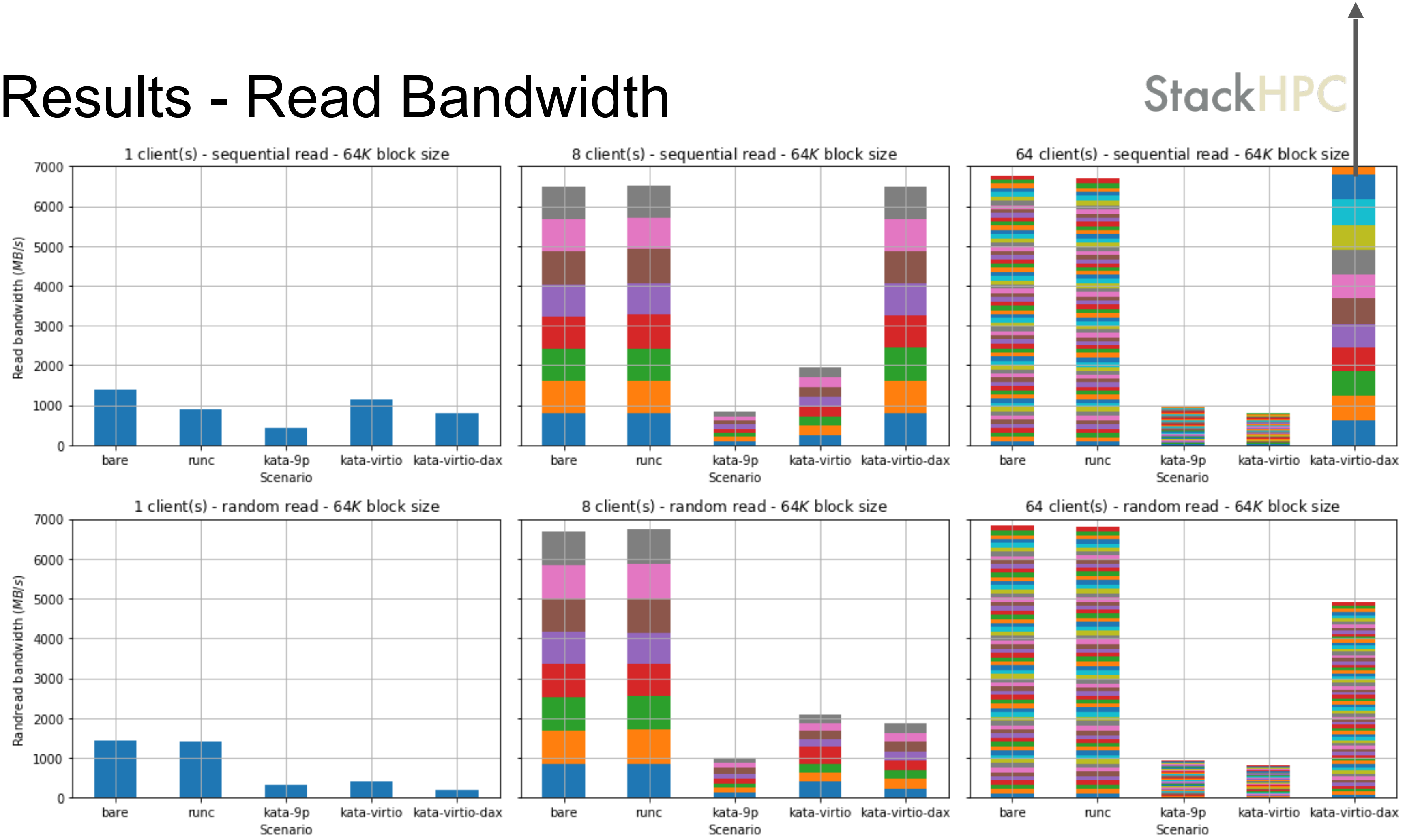
Results - Visualising an FIO run

StackHPC



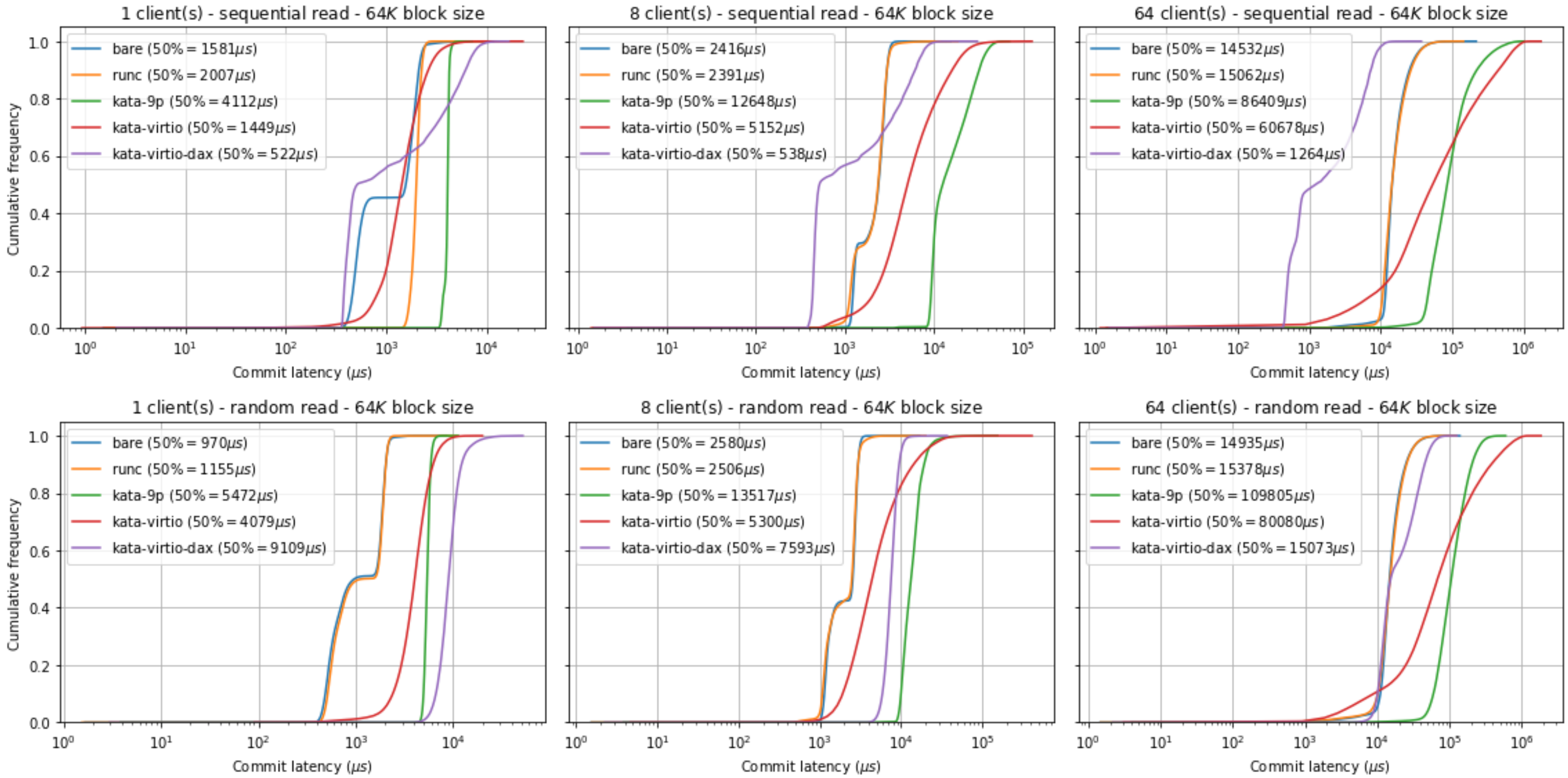
Results - Read Bandwidth

StackHPC



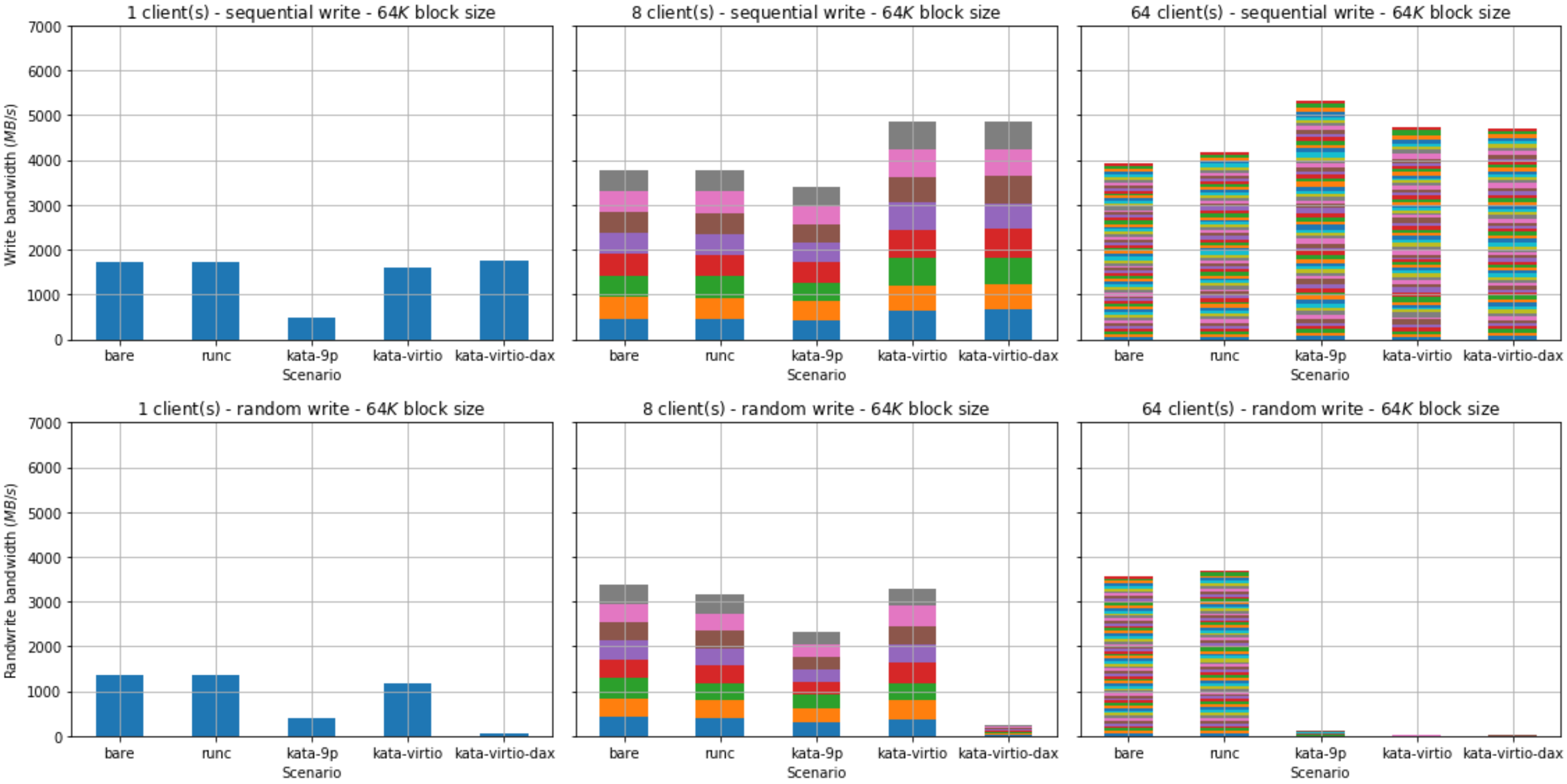
Results - Read Commit Latency

StackHPC



Results - Write Bandwidth

StackHPC



1 client(s) - sequential write - 64K block size

8 client(s) - sequential write - 64K block size

64 client(s) - sequential write - 64K block size

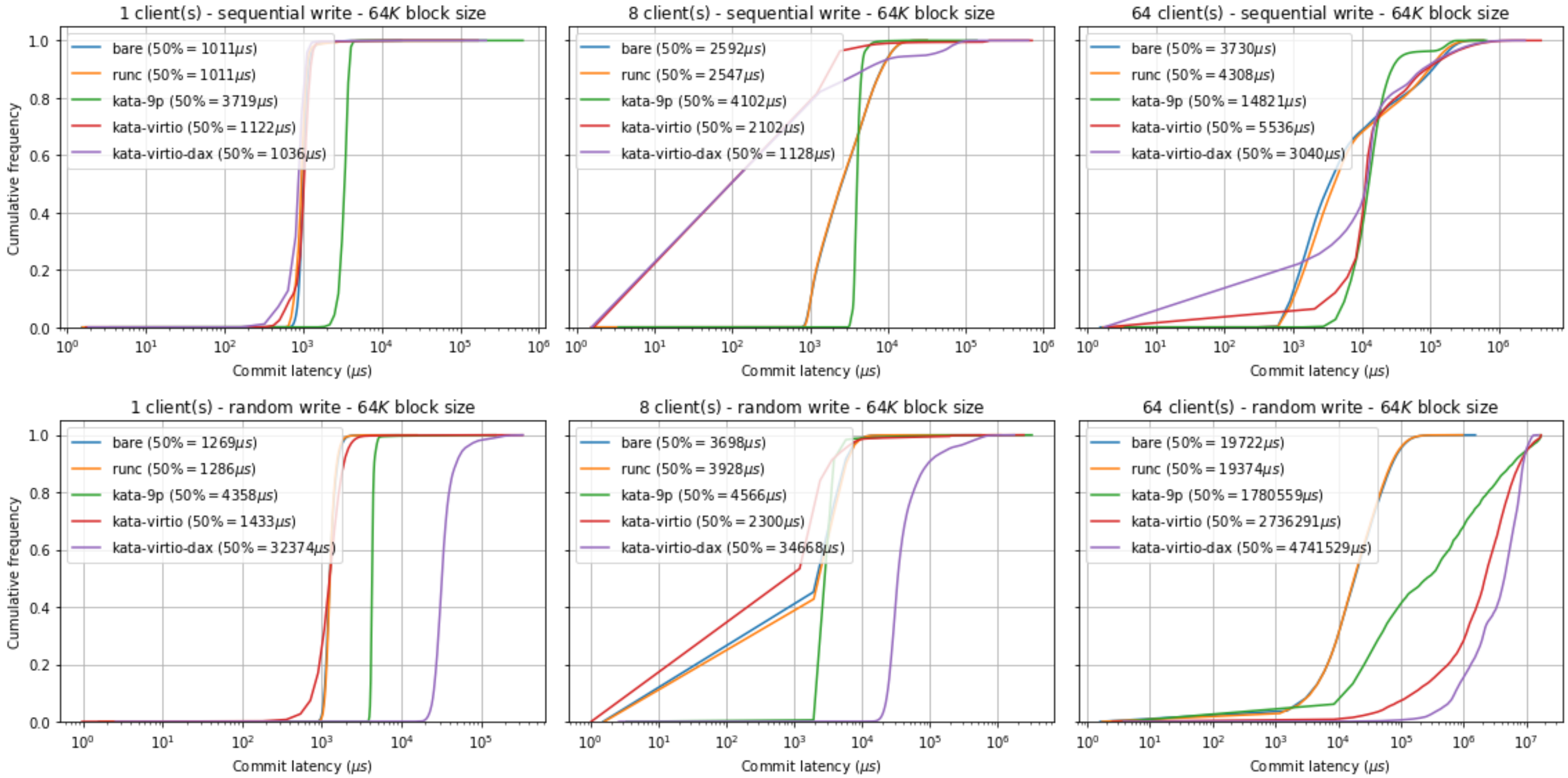
1 client(s) - random write - 64K block size

8 client(s) - random write - 64K block size

64 client(s) - random write - 64K block size

Results - Write Commit Latency

StackHPC



1 client(s) - sequential write - 64K block size

8 client(s) - sequential write - 64K block size

64 client(s) - sequential write - 64K block size

1 client(s) - random write - 64K block size

8 client(s) - random write - 64K block size

64 client(s) - random write - 64K block size

Observations

- Generally:
  - Not much discrepancy between baremetal and runC cases
- Sequential Write:
  - virtio-fs-dax appears to outperform baremetal?
- Random write:
  - virtio-fs-dax only slightly worse than baremetal
- Sequential Read:
  - virtio-fs-dax close to bare metal with fewer clients, outperforms 9p and virtio-fs without DAX
- Random Write:
  - 9p > virtio-fs and virtio-fs-dax

Conclusions

- virtio-9p works but considerable performance sacrifice and doesn’t appear to scale particularly well
- virtio-fs with DAX brings Kata containers much closer to bare metal/runC for read, randread and write scenarios, reservations for randwrite
- … although we may need to wait a little longer for the customisations to the kernel to be readily available if you are planning to use this with parallel file systems backends like BeeGFS/Ceph.

Special thanks

- Graham Whaley (gwhaley)
- Stefan Hajnoczi (stefanha)
- Vivek Goyal (vgoyal)

Thank you for your attention!

References

1. The Ideal Versus the Real: Revisiting the History of Virtual Machines and Containers - https://arxiv.org/abs/1904.12226
2. Grave Robbers from Outer Space: Using 9P2000 Under Linux - https://www.usenix.org/legacy/events/usenix05/tech/freenix/hensbergen.html
3. virtio-fs - https://virtio-fs.gitlab.io/
4. Our blog article - https://www.stackhpc.com/kata-io-1.html