

Justin Cole
 Professor Choi
 April 19, 2022

Assignment 1

1. Experimental Environment

A. Host Machine

All of the tests were performed on a ThinkPad X1 Carbon Gen 8 with an Ubuntu 20.04 OS. The device had 8Gb of ram, around 250 GB of memory provided by a M.2 2280 NVMe SSD, and a Core i7-10510U CPU that has 4 cores.

B. Guest OS

For this experiment 3 different setups were tested on both Docker and QEMU with an additional 4th specification utilizing hardware acceleration through KVM tested only on QEMU. The specifications for these setups are as follows:

1. Memory: 1024, CPUs: 1
2. Memory: 2048, CPUs: 2
3. Memory: 4096, CPUs: 4
4. (QEMU only) Acceleration: KVM, Memory: 4096, CPUs: 4

Explanation: These specifications were chosen for a number of reasons. First, the memory for these tests were chosen because the fileio test will be run with a total file size of 4GB. The memory was chosen to be one half, the same size, and double the size as the total file size to see if that made an impact. The CPUs were chosen because both the fileio and cpu tests will be run with 4 threads. These cpu configurations will ensure that on the third and fourth test case, each thread will get its own CPU maximizing their work and concurrency.

3. QEMU Configuration

A. General Configuration

In order to create the QEMU VM the following steps were performed:

1. `sudo apt install qemu qemu-kvm`
2. `sudo qemu-img create ubuntu.img 10G -f qcow2`
3. `sudo qemu-system-x86_64 -hda ubuntu.img -boot d -cdrom ./ubuntu-20.04.4-live-server-amd64.iso -m 2046 -boot strict=on`

B. Setup Configuration

1. Specification: Memory: 1024, CPUs: 1
 Command: `sudo qemu-system-x86_64 -drive file=ubuntu.img,index=0,media=disk,cache=none -boot d -m 1024 -smp 1 -boot strict=on`
2. Specification: Memory: 2048, CPUs: 2
 Command: `sudo qemu-system-x86_64 -drive file=ubuntu.img,index=0,media=disk,cache=none -boot d -m 2048 -smp 2 -boot strict=on`

3. Specification: Memory: 4096, CPUs: 4
 Command: `sudo qemu-system-x86_64 -drive file=ubuntu.img,index=0,media=disk,cache=none -boot d -m 4096 -smp 4 -boot strict=on`
4. Specification: Acceleration: KVM, Memory: 4096, CPUs: 4
 Command: `sudo qemu-system-x86_64 -machine type=ubuntu,accel=kvm -drive file=ubuntu.img,index=0,media=disk,cache=none -boot d -m 4096 -smp 4 -boot strict=on`

Explanation: The `-drive` option is specified in order to specify the `.img` file as well as remove caching so that the fileio tests work properly from run to run. The `m` and `smp` options were specified in order to set the memory and cpus respectively for each test. Finally the `-machine` with acceleration option was specified in order to enable hardware acceleration for the final test.

4. Docker Configuration

A. General Configuration

Docker was already installed on the device. In order to create the Docker containers utilized in the experiment the following Dockerfiles were created:

1. Dockerfile.fileio

```
FROM zyclonite/sysbench
COPY ./fileTest.sh /
ENTRYPOINT ["sh", "/fileTest.sh"]
```

2. Dockerfile.cpu

```
FROM zyclonite/sysbench
COPY ./cpuTest.sh /
ENTRYPOINT ["sh", "/cpuTest.sh"]
```

These docker files were then built using the following commands:

1. `docker build -t cpu -f Dockerfile.cpu .`
2. `docker build -t fileio -f Dockerfile.fileio .`

Explanation: The dockerfiles were used to import the testing script into the container environment so that the same testing script that will be used in the QEMU VM can also be used in the Docker container.

B. Setup Configuration

The following commands will be used in the `dockerTest.sh` bash file to create the required Docker containers for the experiment.

1. Specification: Memory: 1024, CPUs: 1
 Fileio command: `docker run -m 1024 --cpus 1 --memory-swap 2048m fileio`
 CPU command: `docker run -m 1024 --cpus 1 --memory-swap 2048m cpu`
2. Specification: Memory: 2048, CPUs: 2
 Fileio command: `docker run -m 2048 --cpus 2 --memory-swap 2048m fileio`
 CPU command: `docker run -m 2048 --cpus 2 --memory-swap 2048m cpu`

3. Specification: Memory: 4096, CPUs: 4

Fileio command: `docker run -m 4096 --cpus 4 --memory-swap 2048m fileio`

CPU command: `docker run -m 4096 --cpus 4 --memory-swap 2048m cpu`

Explanation: The `m` and `cpus` options were used to set the available memory and `cpus` respectively for each container. In addition the `memory-swap` option was used to ensure that the memory swap space was the same as what the default swap space was on the QEMU VMs.

5. Sysbench Test Configuration

A. sysbench fileio

This test will be conducted using 4 threads, 64 total files, a total file size of 2Gb, and 2 different test modes: `seqwr` and `rndwr`. Each of the tests will run for 30 seconds. These tests are run using the `fileTest.sh` bash file.

1. seqwr:

```
sysbench fileio --file-num=64 --file-total-size=2G prepare
sysbench fileio --file-num=64 --file-total-size=2G --time=30 --threads=4
--file-test-mode=seqwr run
sysbench fileio cleanup
```

2. rndwr:

```
sysbench fileio --file-num=64 --file-total-size=2G prepare
sysbench fileio --file-num=64 --file-total-size=2G --time=30 --threads=4
--file-test-mode=rndwr run
sysbench fileio cleanup
```

Explanation: These specifications were chosen as 2G over 64 files seemed to result in a good test of the hardware. In addition, using 4 threads allowed the test to take advantage of the increasing number of cores if at all possible. The two different test modes were used to try to see if having an increasing amount of memory would prove more beneficial to either random writes or sequential writes.

B. sysbench cpu

This test will be conducted using 4 threads and 2 different maximum primes: 100 and 100,000. Each of the tests will run for 30 seconds. These tests are run using the `cpuTest.sh` bash file.

1. 100: `sysbench cpu --cpu-max-prime=100 --threads=4 --time=30 run`

2. 100000: `sysbench cpu --cpu-max-prime=100000 --threads=4 --time=30 run`

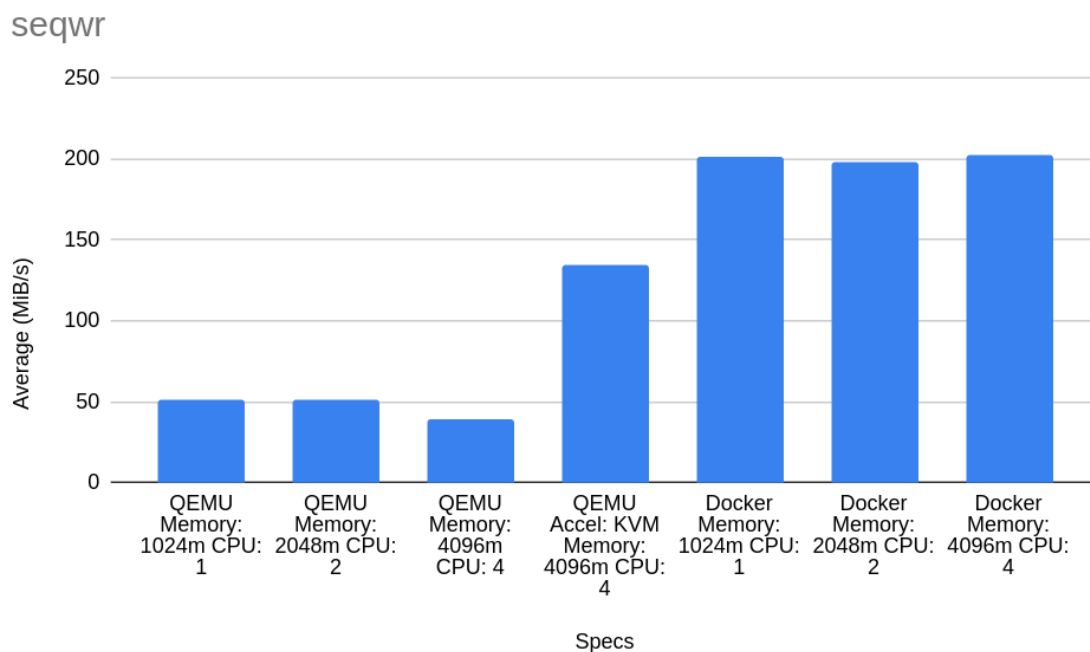
Explanation: These specifications were chosen as using 4 threads allowed the test to take advantage of the increasing number of cores if at all possible. In addition the different max prime values were chosen to specifically be very small and very large so that it could be seen if there was a difference in how the two technologies handled lots of small operations or relatively few, big operations.

6. Sysbench Test Results

The following results were gathered from the previously described test environments. The main testing scripts for all tests were the fileTest.sh and cpuTest.sh bash scripts. These scripts were utilized in both the Docker containers, which were created using the dockerTest.sh bash script, and the QEMU VMs. Most metrics were gathered from the sysbench output with two notable exceptions. The I/O utilization was gathered using the iotop program to monitor the average percentage of I/O utilization during each of the tests. Additionally, the user vs kernel cpu time for the cpu test was gathered using the time unix command.

A. Sysbench fileio for seqwr

| Specification for seqwr | Average (MiB/s) | Min | Max | Std | Latency | Utilization |
|--------------------------------------|-----------------|--------|--------|-------|---------|-------------|
| QEMU Memory: 1024m CPU: 1 | 51.75 | 50.4 | 53.47 | 1.23 | 0.73 | 0.61 |
| QEMU Memory: 2048m CPU: 2 | 51.58 | 48.8 | 54.54 | 2.05 | 0.75 | 0.63 |
| QEMU Memory: 4096m CPU: 4 | 39.522 | 33.15 | 44.8 | 5.54 | 0.86 | 0.62 |
| QEMU Accel: KVM Memory: 4096m CPU: 4 | 134.078 | 122.28 | 158.87 | 14.43 | 0.27 | 2.42 |
| Docker Memory: 1024m CPU: 1 | 201.062 | 184.68 | 231.22 | 17.89 | 0.20 | 75.12 |
| Docker Memory: 2048m CPU: 2 | 197.78 | 177.43 | 238.82 | 24.72 | 0.20 | 73.89 |
| Docker Memory: 4096m CPU: 4 | 202.394 | 164.94 | 248.43 | 31.39 | 0.21 | 75.54 |

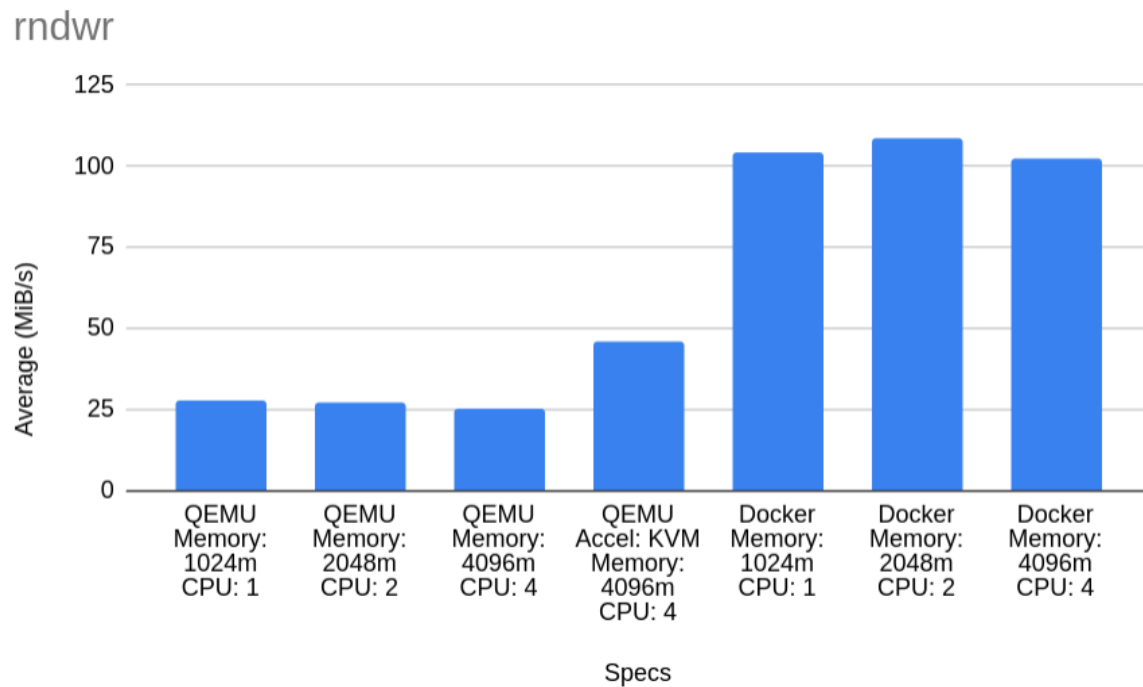


Analysis: The Docker containers appear to run much faster than all QEMU VMs. This is expected as there is less overhead with containers than full virtualization. The closest the QEMU VMs come is when they use hardware acceleration which, as expected, drastically improve the performance of the VMs. Lastly, it appears that adding more cores and more memory either causes no change or, in the case of the third QEMU configuration, a decrease in performance. This is likely because adding more processor cores and memory does nothing to help with writing to disk. In fact, it's possible that adding more memory increases access time for the relatively small amount of RAM the processes need.

Meaning that adding too much RAM decreases fileio performance.

B. Sysbench fileio for rndwr

| Configuration for rndwr | Average (MiB/s) | Min | Max | Std y | Latenc y | Utilizati on |
|--------------------------------------|--------------------|--------|--------|----------|-------------|-----------------|
| QEMU Memory: 1024m CPU: 1 | 27.608 | 27.24 | 28.03 | 0.28 | 1.35 | 2.1 |
| QEMU Memory: 2048m CPU: 2 | 27.34 | 25.6 | 30.17 | 2.28 | 1.38 | 2.2 |
| QEMU Memory: 4096m CPU: 4 | 25.536 | 25.32 | 25.91 | 0.23 | 1.47 | 2.0 |
| QEMU Accel: KVM Memory: 4096m CPU: 4 | 45.96 | 43.71 | 48.75 | 2.57 | 0.81 | 3.0 |
| Docker Memory: 1024m CPU: 1 | 104.48 | 100.02 | 107.99 | 2.86 | 0.36 | 88.7 |
| Docker Memory: 2048m CPU: 2 | 108.41 | 105.94 | 110.97 | 2.26 | 0.35 | 91.2 |
| Docker Memory: 4096m CPU: 4 | 102.35 | 99.82 | 106.26 | 2.72 | 0.37 | 90.4 |



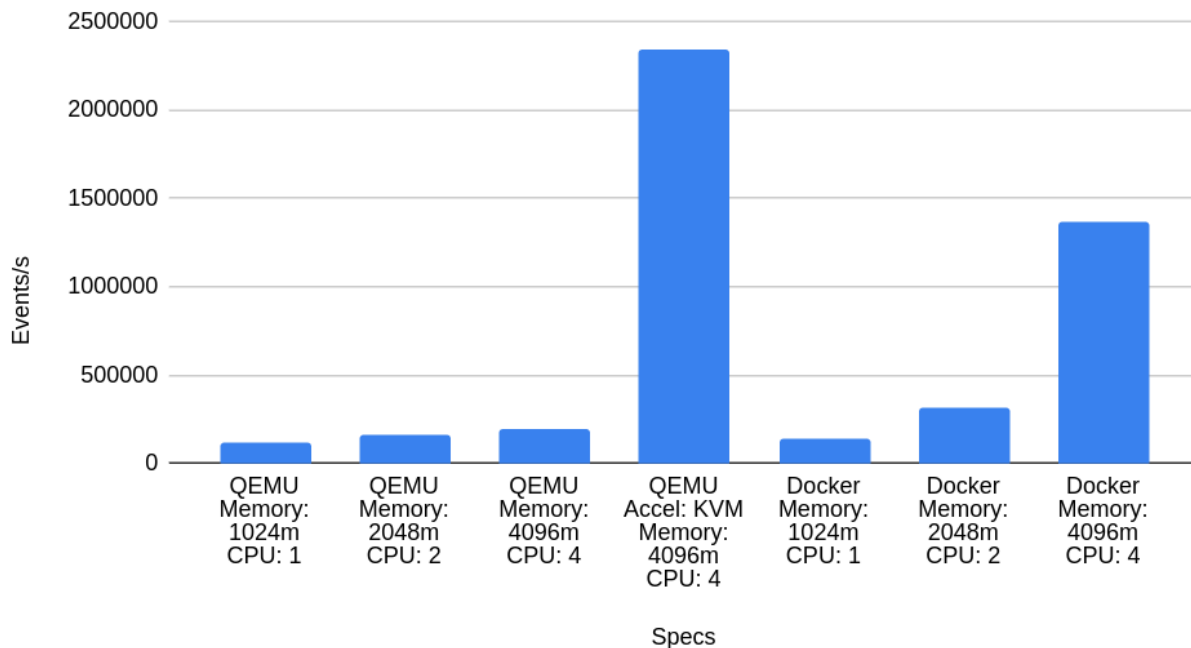
Analysis: Many of the same insights from the seqwr test mode results appear here as well. The Docker containers appear to run much faster than all QEMU VMs and it appears that adding more cores and more memory either causes the same or worse performance. There are, however, two minor differences between the rndwr and the seqwr test mode results. First, the rndwr has a lower average throughput than the seqwr across all environments. This is likely due to the random access nature of the writes requiring more work by the SSD. Additionally, it appears the KVM hardware acceleration does not have as much of an impact on the rndwr test mode as it did in the seqwr test mode. In the seqwr test mode the KVM acceleration made up over half of the performance difference between the QEMU and Docker environments. However, in the rndwr test mode it appears to cover less than half the difference.

C. Sysbench cpu for max prime 100

| Specification for max prime 100 | events/s | Min | Max | Std | User Time | Kernel Time |
|---------------------------------|-----------|-----------|-----------|----------|-----------|-------------|
| QEMU Memory: 1024m CPU: 1 | 119463.49 | 116900.2 | 122533.58 | 2077.93 | 29.96s | 0.15s |
| QEMU Memory: 2048m CPU: 2 | 160216.11 | 150527.31 | 178921.99 | 11025.32 | 25.1s | 34.4s |
| QEMU Memory: 4096m CPU: 4 | 199436.64 | 161652.79 | 261079.35 | 40379.16 | 50.13s | 1m9.8s |

| | | | | | | |
|--------------------------------------|------------|------------|------------|-----------|---------|-------|
| QEMU Accel: KVM Memory: 4096m CPU: 4 | 2346363.98 | 1956691.04 | 2893342.26 | 378936.16 | 2m0.5 | 0.56s |
| Docker Memory: 1024m CPU: 1 | 136206.14 | 127707.65 | 141985.5 | 5799.39 | 30.03s | 0.0s |
| Docker Memory: 2048m CPU: 2 | 313729.64 | 306198.37 | 322875.73 | 7423.63 | 1m0.08s | 0s |
| Docker Memory: 4096m CPU: 4 | 1371889.55 | 1225808.1 | 1735495.31 | 216788.37 | 2m | 0s |

Prime Limit: 100

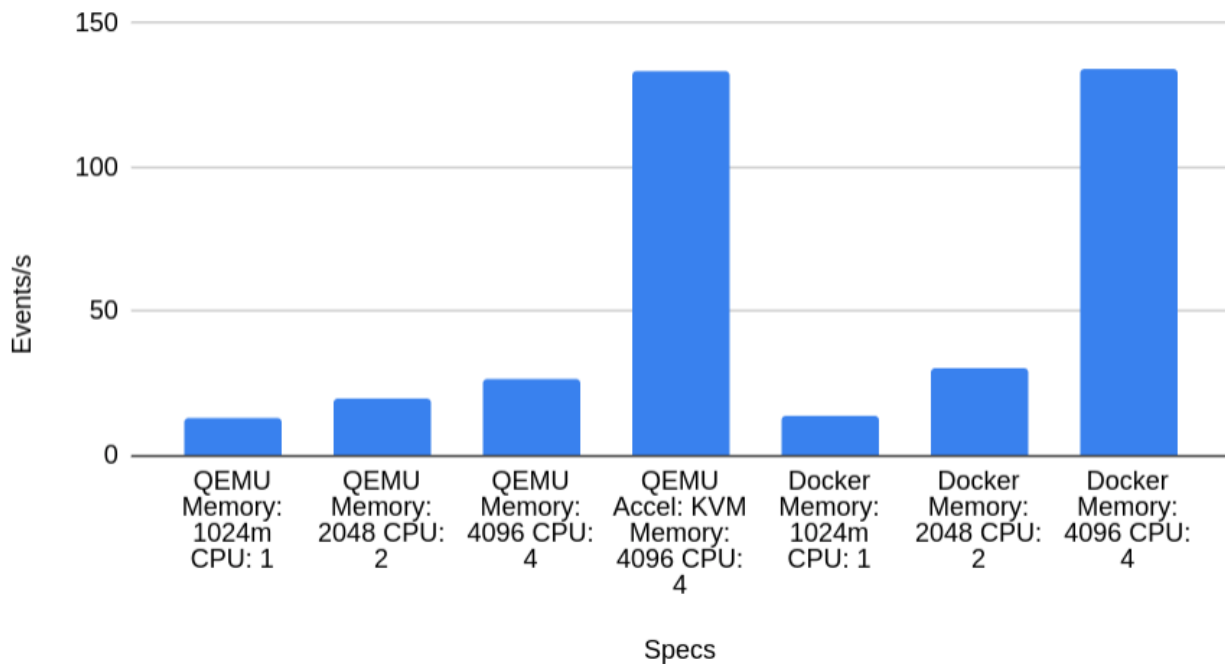


Analysis: On low powered environments, it appears that there is minimal difference in performance. However, as more resources are given to each environment there is a noticeable increase in work done. There is however something particularly noteworthy about the last test for each technology. The final configuration for the Docker environment has around four times as many events/s as the previous Docker environment. This remarkable boost in performance is most likely due to the removal of needing to swap between threads in the cpus as each thread has its own dedicated cpu in the final test. As for the last QEMU environment, astonishingly, once hardware acceleration is turned on it seems to outperform even the Docker environments. Perhaps there is some overhead that is present for lots of smaller operations that the KVM accelerated QEMU VM is able to deal with better than the Docker container can.

D. Sysbench cpu for max prime 100,000

| Specification for max prime 100,000 | events/s | Min | Max | Std | User Time | Kernel Time |
|-------------------------------------|----------|--------|--------|-------|-----------|-------------|
| QEMU Memory: 1024m CPU: 1 | 13.204 | 12.84 | 13.65 | 0.29 | 29.93s | 0.15s |
| QEMU Memory: 2048 CPU: 2 | 19.876 | 18.7 | 22.72 | 1.7 | 59.79s | 0.122s |
| QEMU Memory: 4096 CPU: 4 | 26.638 | 26.34 | 26.81 | 0.18 | 1m59s | 0.2s |
| QEMU Accel: KVM Memory: 4096 CPU: 4 | 133.374 | 132.7 | 133.88 | 0.50 | 2m0.45s | 0.041 |
| Docker Memory: 1024m CPU: 1 | 13.782 | 12.98 | 14.54 | 0.56 | 30.01s | 0s |
| Docker Memory: 2048 CPU: 2 | 30.288 | 29.46 | 30.77 | 0.56 | 1m0.05s | 0s |
| Docker Memory: 4096 CPU: 4 | 134.448 | 128.32 | 157.61 | 12.95 | 1m59.9s | 0s |

Prime Limit: 100000



Analysis: As with the lower prime limit a steady increase in power can be seen as more resources are granted to the VMs and containers with the final configuration for both technologies displaying by far the most performance. However, unlike the lower prime limit, the KVM accelerated QEMU VM and the final Docker container appear to be much more evenly matched.

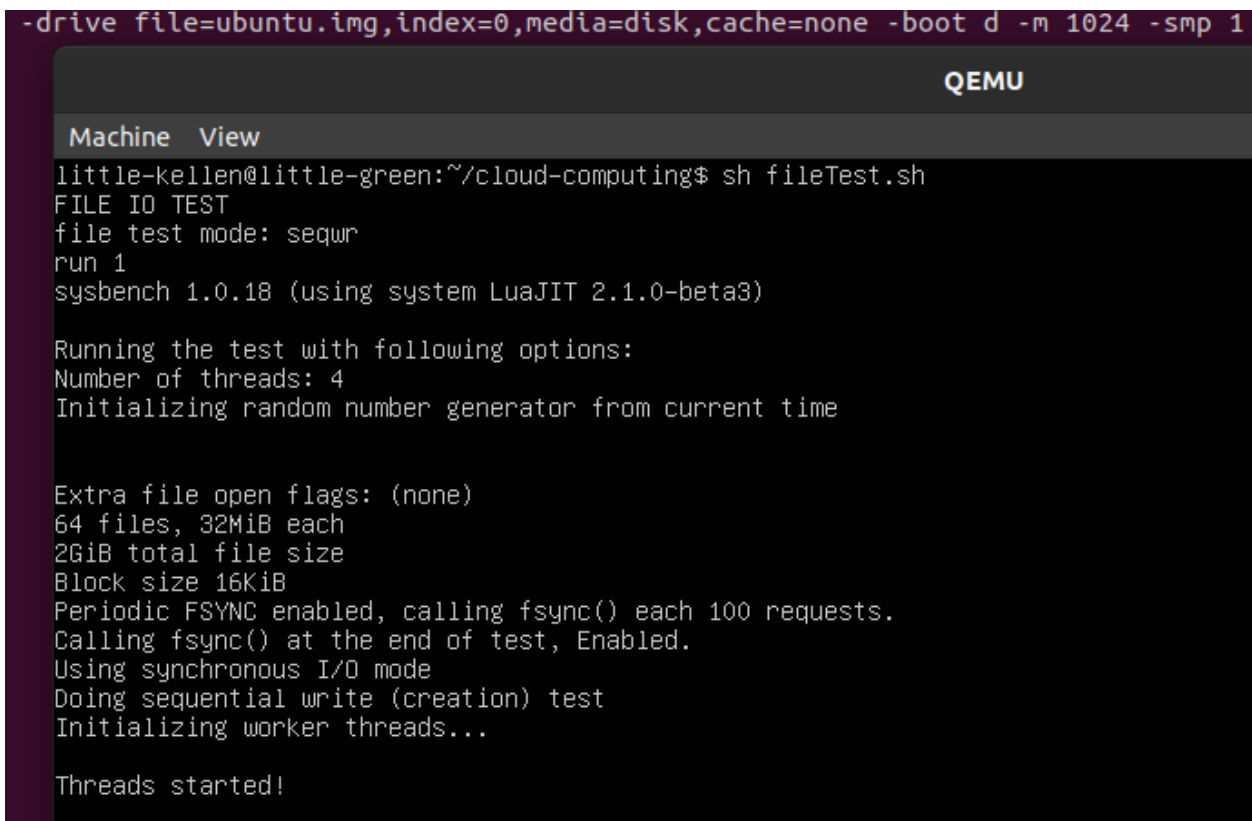
7. Conclusion

Both QEMU VMs and Docker containers are incredible technologies that both have their various use cases. Depending on what is required for a given project one may be better than the other on various merits, however, if raw performance is required then it appears that in most cases the Docker container is the superior technology. That being said, as the small prime limit showed, there may be niche cases where KVM accelerated QEMU VMs may indeed outperform Docker containers. Both technologies must be carefully considered when deciding which one to incorporate into a project.

8. Screenshots

QEMU 1024m and 1 CPU Fileio test

```
-drive file=ubuntu.img,index=0,media=disk,cache=none -boot d -m 1024 -smp 1
```



```
QEMU
```

```
Machine View
```

```
little-kellen@little-green:~/cloud-computing$ sh fileTest.sh
```

```
FILE IO TEST
```

```
file test mode: seqwr
```

```
run 1
```

```
sysbench 1.0.18 (using system LuaJIT 2.1.0-beta3)
```

```
Running the test with following options:
```

```
Number of threads: 4
```

```
Initializing random number generator from current time
```

```
Extra file open flags: (none)
```

```
64 files, 32MiB each
```

```
2GiB total file size
```

```
Block size 16KiB
```

```
Periodic FSYNC enabled, calling fsync() each 100 requests.
```

```
Calling fsync() at the end of test, Enabled.
```

```
Using synchronous I/O mode
```

```
Doing sequential write (creation) test
```

```
Initializing worker threads...
```

```
Threads started!
```

QEMU 1024m and 1 CPU CPU test

```
-drive file=ubuntu.img,index=0,media=disk,cache=none -boot d -m 1024 -smp 1
```

QEMU

Machine View

sysbench 1.0.18 (using system LuaJIT 2.1.0-beta3)

Running the test with following options:
Number of threads: 4
Initializing random number generator from current time

Prime numbers limit: 100

Initializing worker threads...

Threads started!

CPU speed:
events per second: 114611.42

QEMU 2048m and 2 CPU Fileio test

```
-drive file=ubuntu.img,index=0,media=disk,cache=none -boot d -m 2048 -smp 2
```

QEMU

Machine View

little-kellen@little-green:~\$ ls
cloud-computing key
little-kellen@little-green:~\$ cd cloud-computing/
little-kellen@little-green:~/cloud-computing\$ sh fileTest.sh
FILE IO TEST
file test mode: sequen
run 1
sysbench 1.0.18 (using system LuaJIT 2.1.0-beta3)

Running the test with following options:
Number of threads: 4
Initializing random number generator from current time

Extra file open flags: (none)
64 files, 32MiB each
2GiB total file size
Block size 16KiB
Periodic FSYNC enabled, calling fsync() each 100 requests.
Calling fsync() at the end of test, Enabled.
Using synchronous I/O mode
Doing sequential write (creation) test
Initializing worker threads...

Threads started!

QEMU 2048m and 2 CPU CPU test

```
-drive file=ubuntu.img,index=0,media=disk,cache=none -boot d -m 2048 -smp 2
```

QEMU

Machine View

sysbench 1.0.18 (using system LuaJIT 2.1.0-beta3)

Running the test with following options:

Number of threads: 4

Initializing random number generator from current time

Time numbers limit: 100

Initializing worker threads...

Threads started!

CPU speed:

events per second: 152526.32

General statistics:

| | |
|-------------------------|----------|
| total time: | 30.0016s |
| total number of events: | 4576887 |

Latency (ms):

| | |
|------------------|----------|
| min: | 0.00 |
| avg: | 0.02 |
| max: | 20.22 |
| 95th percentile: | 0.01 |
| sum: | 78462.82 |

QEMU 4096m and 4 CPU Fileio test

```
-drive file=ubuntu.img,index=0,media=disk,cache=none -boot d -m 4096 -smp 4
```

QEMU

Machine View

Support: <https://ubuntu.com/advantage>

System information as of Tue 19 Apr 2022 06:16:50 PM UTC

System load: 3.42

Usage of /: 69.6% of 8.90GB

Memory usage: 5%

Swap usage: 0%

Processes: 149

Users logged in: 0

IPv4 address for ens3: 10.0.2.15

IPv6 address for ens3: fec0::5054:ff:fe12:3456

Super-optimized for small spaces - read how we shrank the memory footprint of MicroK8s to make it the smallest full K8s around.

<https://ubuntu.com/blog/microk8s-memory-optimisation>

updates can be applied immediately, see these additional updates run: `apt list --upgradable`

Last login: Tue Apr 19 18:09:40 UTC 2022 on tty1

little-kellen@little-green:~\$ cd cloud-computing/

little-kellen@little-green:~/cloud-computing\$ sh fileTest.sh

FILE IO TEST

File test mode: sequen

h 1

sysbench 1.0.18 (using system LuaJIT 2.1.0-beta3)

QEMU 4096m and 4 CPU CPU test

```
-drive file=ubuntu.img,index=0,media=disk,cache=none -boot d -m 4096 -smp 4
```

| QEMU | |
|------------------|-----------|
| Machine View | |
| min: | 0.11 |
| avg: | 0.98 |
| max: | 73.30 |
| 95th percentile: | 1.93 |
| sum: | 117667.53 |

```

leads fairness:
  events (avg/stddev):      30024.2500/121.86
  execution time (avg/stddev): 29.4169/0.01

  2
bench 1.0.18 (using system LuaJIT 2.1.0-beta3)

Running the test with following options:
  Number of threads: 4
  Initializing random number generator from current time

  File open flags: (none)
  Files, 32MiB each
  Total file size
  Block size 16KiB
  Periodic FSYNC enabled, calling fsync() each 100 requests.
  Calling fsync() at the end of test, Enabled.
  Running synchronous I/O mode
  Running sequential write (creation) test
  Initializing worker threads...

Leads started!

kellen@little-green:~/cloud-computing$ sh cpuTest.sh
TEST
: 1

```

QEMU KVM accelerated 4096m and 4 CPU Fileio test

```
-machine type=ubuntu,accel=kvm -drive file=ubuntu.img,index=0,media=disk,cache=none
```

| QEMU | |
|------|--|
| View | |

```

kellen@little-green:~$ cd cloud-computing/
kellen@little-green:~/cloud-computing$ sh fileTest.sh
TEST
Mode: sequen

  1.0.18 (using system LuaJIT 2.1.0-beta3)

Running the test with following options:
  Number of threads: 4
  Initializing random number generator from current time

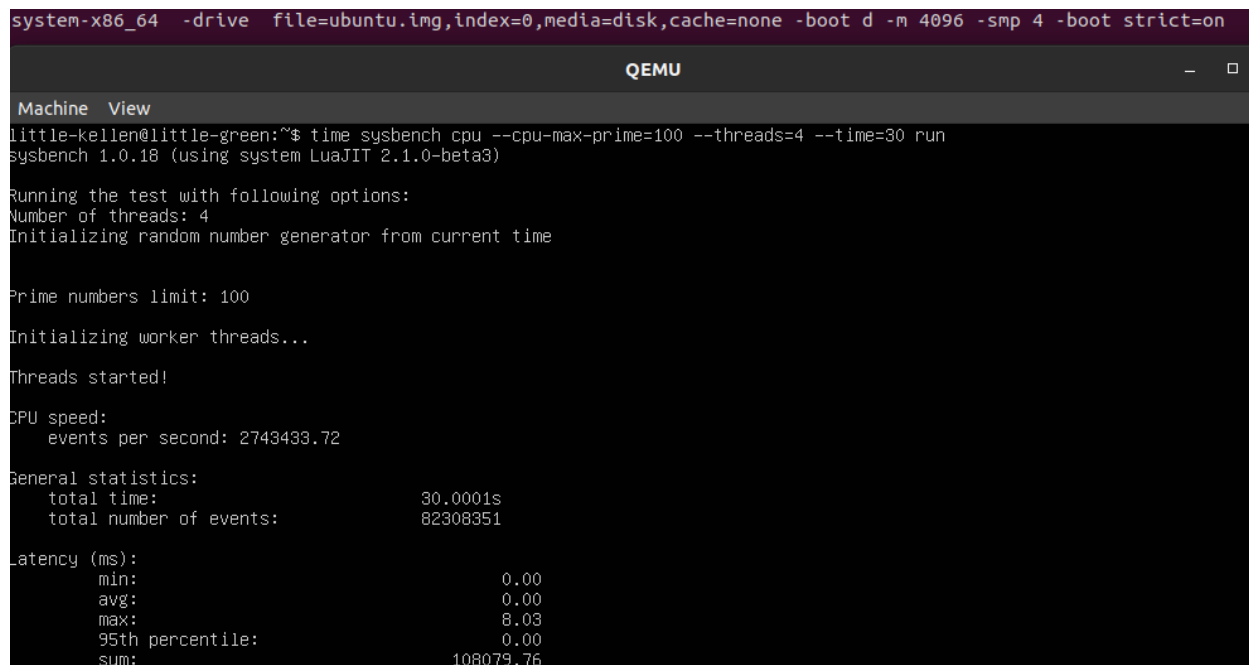
  File open flags: (none)
  Files, 32MiB each
  Total file size
  Block size 16KiB
  Periodic FSYNC enabled, calling fsync() each 100 requests.
  Calling fsync() at the end of test, Enabled.
  Running synchronous I/O mode
  Running sequential write (creation) test
  Initializing worker threads...

Leads started!

```

QEMU KVM accelerated 4096m and 4 CPU CPU test

```
system-x86_64 -drive file=ubuntu.img,index=0,media=disk,cache=none -boot d -m 4096 -smp 4 -boot strict=on
```



```
Machine View
little-kellen@little-green:~$ time sysbench cpu --cpu-max-prime=100 --threads=4 --time=30 run
sysbench 1.0.18 (using system LuaJIT 2.1.0-beta3)

Running the test with following options:
Number of threads: 4
Initializing random number generator from current time

Prime numbers limit: 100
Initializing worker threads...
Threads started!

CPU speed:
  events per second: 2743433.72

General statistics:
  total time:          30.0001s
  total number of events: 82308351

Latency (ms):
  min:                 0.00
  avg:                 0.00
  max:                 8.03
  95th percentile:    0.00
  sum:                108079.76
```

Docker 1024 and 1 CPU Fileio test

```
FILE IO TEST
file test mode: seqwr
run 1
sysbench 1.0.20-6ef8a4d4d7 (using bundled LuaJIT 2.1.0-beta2)

Running the test with following options:
Number of threads: 4
Initializing random number generator from current time

Extra file open flags: (none)
64 files, 32MiB each
2GiB total file size
Block size 16KiB
Periodic FSYNC enabled, calling fsync() each 100 requests.
Calling fsync() at the end of test, Enabled.
Using synchronous I/O mode
Doing sequential write (creation) test
Initializing worker threads...

Threads started!
```

Docker 1024 and 1 CPU CPU test

```
CPU TEST
run: 1
sysbench 1.0.20-6ef8a4d4d7 (using bundled LuaJIT 2.1.0-beta2)

Running the test with following options:
Number of threads: 4
Initializing random number generator from current time


Prime numbers limit: 100

Initializing worker threads...

Threads started!
```

Docker 2048 and 2 CPU Fileio test

```
kellen@kellen-ThinkPad-X1-Carbon-Gen-8:~/Documents/class/cloud/tests$ sudo docker run -m 2048m --cpus 2 --memory-swap 2048m fileio
FILE IO TEST
file test mode: seqwr
run 1
sysbench 1.0.20-6ef8a4d4d7 (using bundled LuaJIT 2.1.0-beta2)

Running the test with following options:
Number of threads: 4
Initializing random number generator from current time


Extra file open flags: (none)
64 files, 32MiB each
2GiB total file size
Block size 16KiB
Periodic FSYNC enabled, calling fsync() each 100 requests.
Calling fsync() at the end of test, Enabled.
Using synchronous I/O mode
Doing sequential write (creation) test
Initializing worker threads...

Threads started!
```

Docker 2048 and 2 CPU CPU test

```
kellen@kellen-ThinkPad-X1-Carbon-Gen-8:~/Documents/class/cloud/tests$ sudo docker run -m 2048m --cpus 2 --memory-swap 2048m cpu
CPU TEST
run: 1
sysbench 1.0.20-6ef8a4d4d7 (using bundled LuaJIT 2.1.0-beta2)

Running the test with following options:
Number of threads: 4
Initializing random number generator from current time


Prime numbers limit: 100

Initializing worker threads...

Threads started!
```

Docker 4096 and 4 CPU Fileio test

```
kellen@kellen-ThinkPad-X1-Carbon-Gen-8:~/Documents/class/cloud/tests$ sudo docker run -m 4096m --cpus 4 --memory-swap 2048m fileio
FILE IO TEST
file test mode: seqwr
run: 1
sysbench 1.0.20-6ef8a4d4d7 (using bundled LuaJIT 2.1.0-beta2)

Running the test with following options:
Number of threads: 4
Initializing random number generator from current time


Extra file open flags: (none)
64 files, 32MiB each
2GiB total file size
Block size 16KiB
Periodic FSYNC enabled, calling fsync() each 100 requests.
Calling fsync() at the end of test, Enabled.
Using synchronous I/O mode
Doing sequential write (creation) test
Initializing worker threads...

Threads started!
```

Docker 4096 and 4 CPU CPU test

```
kellen@kellen-ThinkPad-X1-Carbon-Gen-8:~/Documents/class/cloud/tests$ sudo docker run -m 4096m --cpus 4 --memory-swap 2048m cpu
CPU TEST
run: 1
sysbench 1.0.20-6ef8a4d4d7 (using bundled LuaJIT 2.1.0-beta2)

Running the test with following options:
Number of threads: 4
Initializing random number generator from current time


Prime numbers limit: 100

Initializing worker threads...

Threads started!
```