Almas Khan

HW1

System vs OS virtualization

Performed Experiments using QEMU and Docker to understand system virtualization and OS virtualization. This report gives an overview of experiments performed.

# Environment

## Host environment

This experiment is carried on personal laptop with configuration as below:

Operating System: MacOS Big Sur version 11.2.3

Processor: 2.3 GHz Dual-Core Intel Core i5

Memory: 8 GB 2133 MHz LPDDR3

Graphics: Intel Iris Plus Graphics 640 1536 MB

## QEMU environment

There were issues in running QEMU with Ubuntu 20.04. But Combination of QEMU and Ubuntu 16.04 worked successfully

Commands to setup QEMU

1. To install QEMU command

### brew install qemu

1. Downloading ubuntu server image

curl -o ubuntu-20.04.4-live-server-amd64.iso

[https://releases.ubuntu.com/16.04/ubuntu-16.04.7-server-amd64](https://releases.ubuntu.com/16.04/ubuntu-16.04.7-server-amd64.iso)

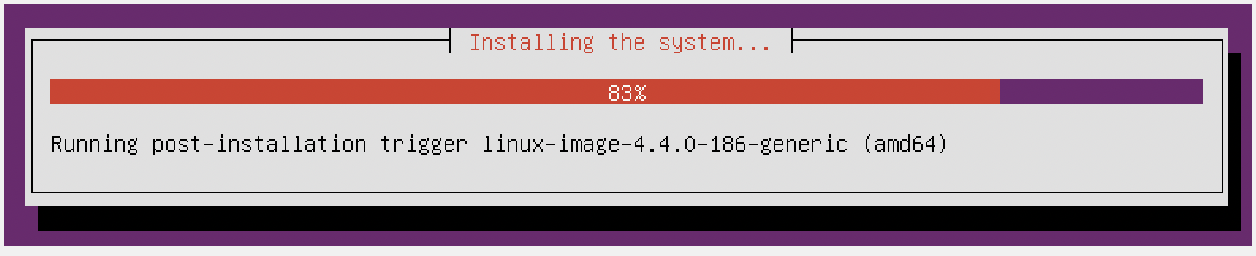
[.iso](https://releases.ubuntu.com/16.04/ubuntu-16.04.7-server-amd64.iso)

1. Create QEMU image

### qemu-img create -f qcow2 ubuntu.qcow 10G

1. Install OA in the QEMU image

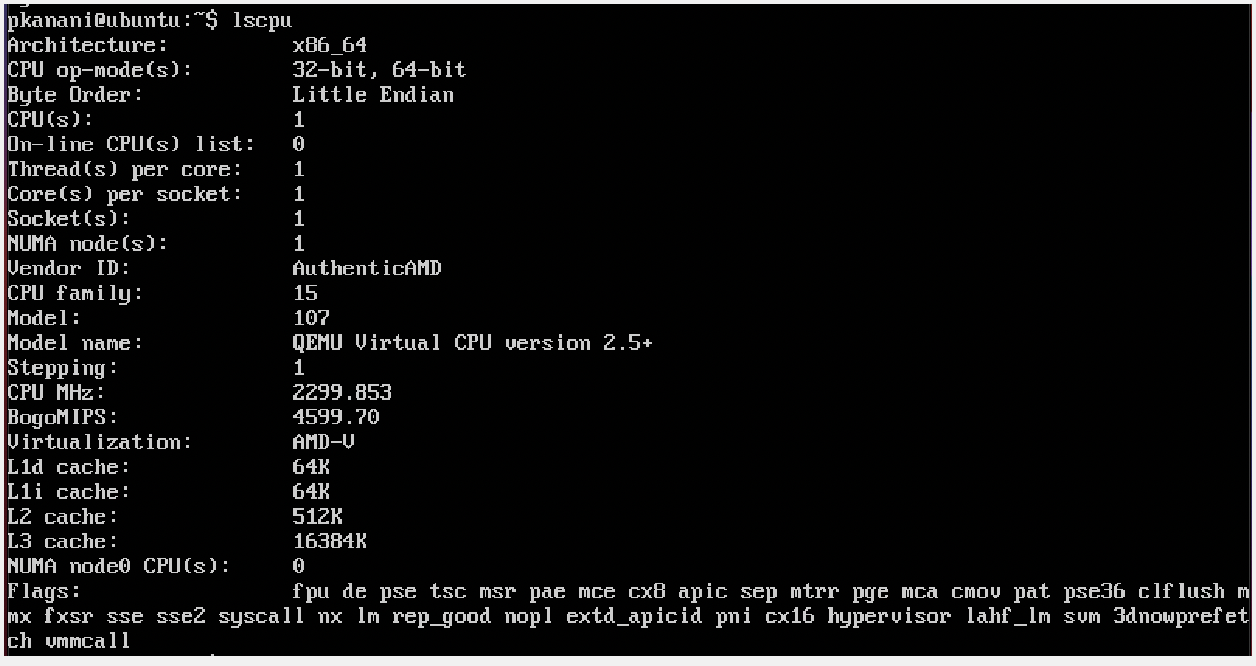
### qemu-system-x86\_64 -hda ubuntu.img -boot d -cdrom ubuntu-16.04.7-server-amd64.iso -m 2046 -boot strict=on



1. Running QEMU after installation

### qemu-system-x86\_64 -hda ubuntu.img -m 2046

CPU configurations



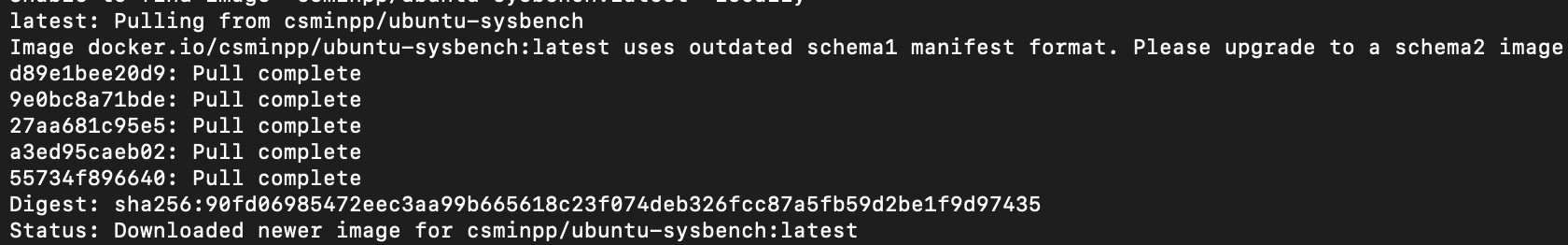
## Docker environment

1. Installing docker

Followed instructions according to <https://docs.docker.com/desktop/mac/install/>

1. Download required docker image

Docker pull csminpp/ubuntu-sysbench



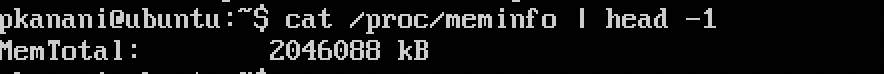
# Useful arguments for QEMU

Below are some useful arguments experimented in QEMU

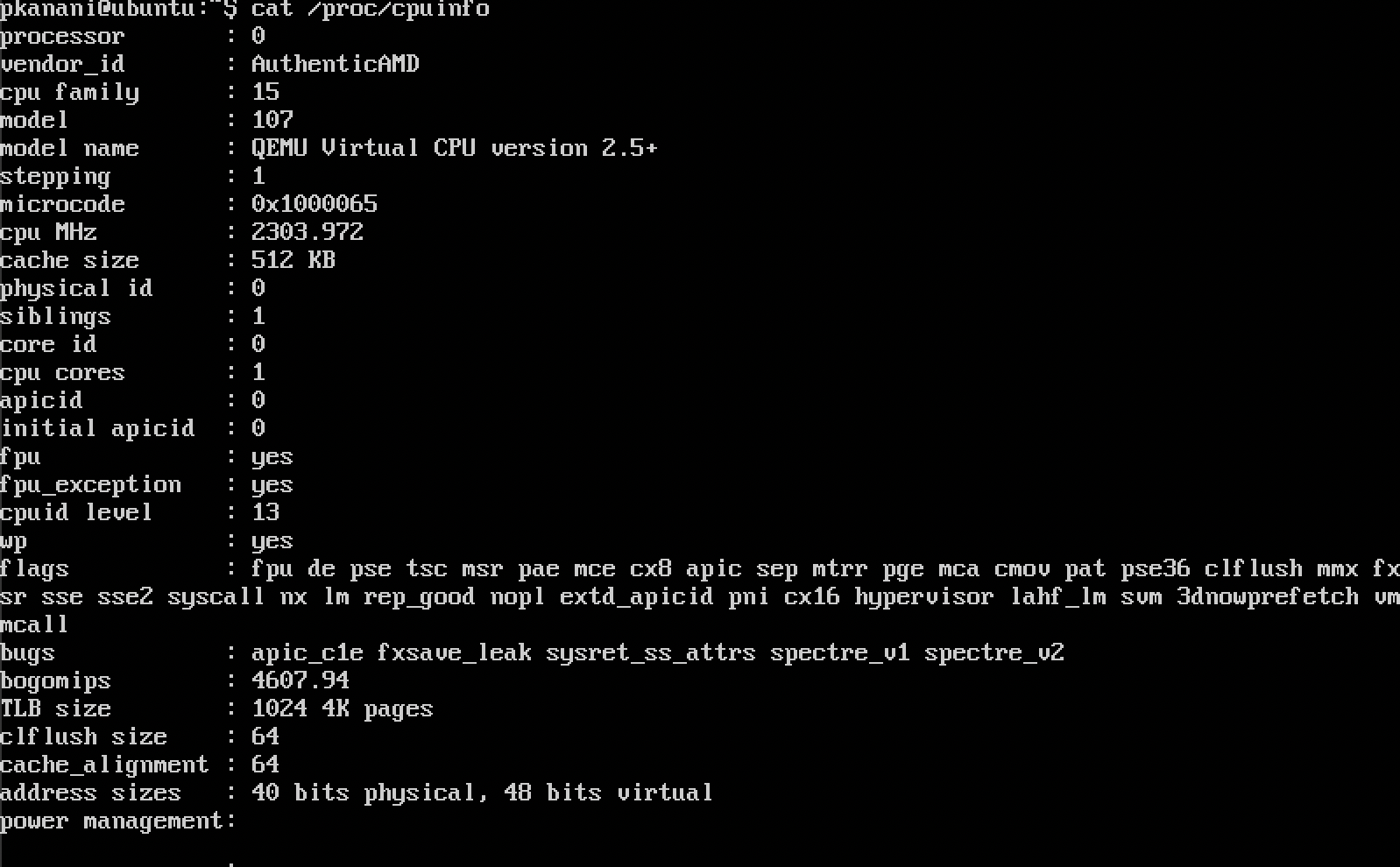
-m :

-m is used to define how much memory do we want to assign to QEMU

The initial value is set to 2046MB.



-smp :

-smp is used to define how many CPU cores we want in our VM. Initial value of cpu cores can be seen as 1 in below screenshot.

After running QEMU with value of -smp 2, the value of CPU cores has now changed from 1 to 2.



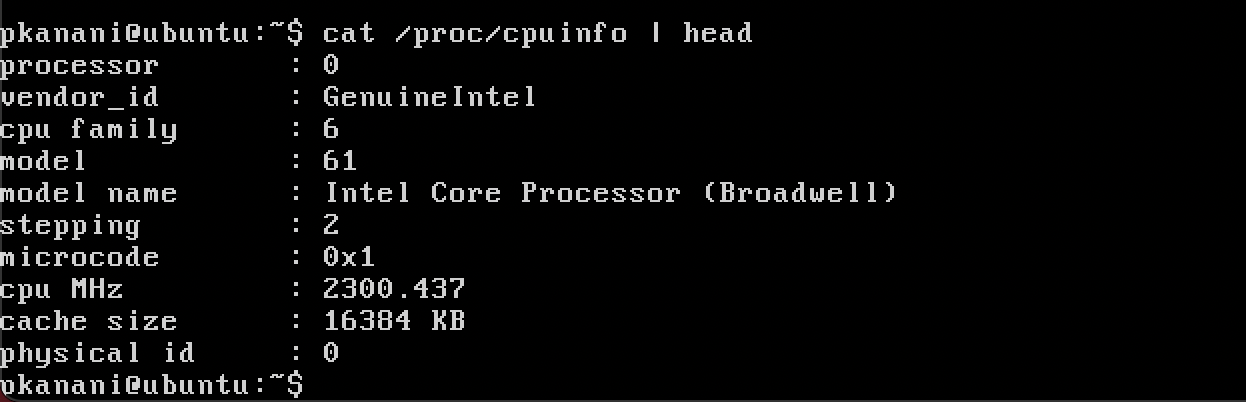
-accel :

Using the accel option we can enable the accelerator. It helps with the speed of VM.

-cpu :

Using cpu option we can emulate different CPU models. In initial screen shot we can see that default CPU model is “QEMU Virtual CPU version 2.5+”

After running it with option cpu and value “Broadwell-v1” cpu model have now changed to “Intel Core Processor (Broadwell)”



-boot

Using boot option we can pass values which will be used during booting process

-hda

Supplies hard disk to VM

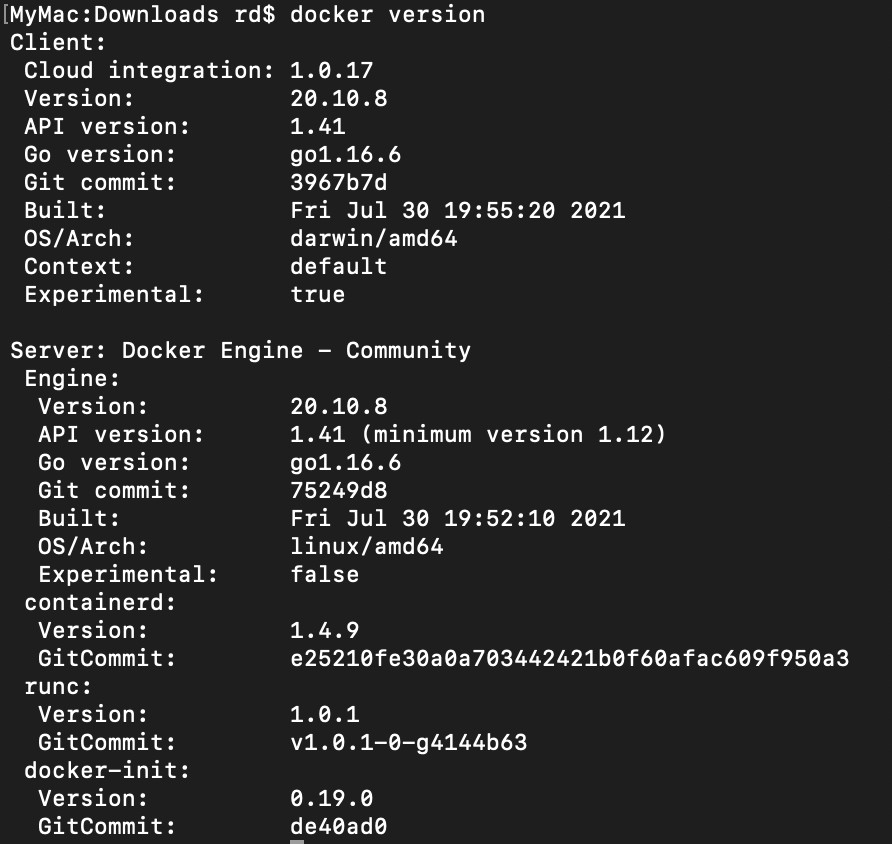
-cdrom

Supplies cdrom to VM

# Useful Docker commands

### docker version

This command gives information about version of the docker running on the machine. It’s output looks similar to the screenshot below:



### docker images

Lists all available images in a docker.

Output should be similar to screenshot below:



### docker images –help

Putting –help at the end of any command provides information related to that command, like syntax and available options.

Example output:



### docker ps

Gives list of containers that are running currently. It also provides option to get list of containers in other status like stopped.

### docker pull image

To fetch docker image

### docker container run imageName

Creates new container and starts it

### docker container stop containerName

Stops container with name containerName

### docker container rm containerName

Deletes docker container with name containerName

### docker image rm imageName

Deletes docker image with name imageName

# CPU test

3 CPU tests with different max-prime values of 10000, 20000 and 50000 were run. Steps and test results for each platform are described in this section.

## QEMU

qemu\_cpu\_test.sh contains the actual command and runs it 5 times. qemu\_cpu\_test\_main.sh is the main script to perform cpu tests. It executes qemu\_cpu\_test.sh with parameters 10000, 20000, and 50000. These parameters will be passed to actual command which as below to check different test cases:

### sysbench --test=cpu --cpu-max-prime=$prime run

Test was run using following command

### sh qemu\_cpu\_test\_main.sh > qemu\_cpu\_test.txt

Hence test results for QEMU test are in qemu\_cpu\_test.txt file

## Docker

docker\_cpu\_test.sh contains the command to run the CPU test on docker and runs it five times for given parameters. docker\_cpu\_test\_main.sh is main scripts which executes docker\_cpu\_test.sh with parameters same as QEMU, i.e., 10000, 20000, and 50000. Command for docker is as below:

### docker run csminpp/ubuntu-sysbench sysbench --test=cpu

--cpu-max-prime=$prime run

Docker test can be run using following command:

sh docker\_cpu\_test\_main.sh > docker\_cpu\_test.txt

## Results

Below we can see test results for both QEMU and docker using different parameters.

For each case performance of docker is better than QEMU. This could be partially because of better CPU but even with same CPU, OS virtualization is usually faster than system virtualization.

Also we can see that with increase in value of max-prime, timing has increased for both systems. This is simply because of increase in amount of work.

Test 1

|  |  |  |
| --- | --- | --- |
| max-prime = 10000 | | |
|  | QEMU | Docker |
| Test-num | Total time (in s) | Total time (in s) |
| 1 | 18.0312 | 7.2224 |
| 2 | 17.2081 | 7.465 |
| 3 | 29.413 | 8.0328 |
| 4 | 38.0274 | 7.0191 |

|  |  |  |
| --- | --- | --- |
| 5 | 42.5024 | 7.0951 |
| Min | 18.0312 | 7.0191 |
| Max | 42.5024 | 8.0548 |
| Avg | 31.24662 | 8.39368 |
| StdDev | 10.11875 | 0.35536 |

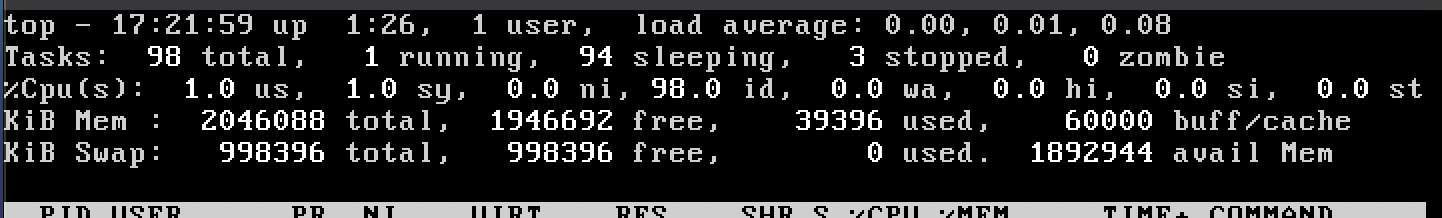
Test 2

|  |  |  |
| --- | --- | --- |
| max-prime = 20000 | | |
|  | QEMU | Docker |
| Test-num | Total time (in s) | Total time (in s) |
| 1 | 65.0395 | 20.45764 |
| 2 | 101.1749 | 20.5626 |
| 3 | 97.0249 | 20.45 |
| 4 | 55.4144 | 20.0697 |
| 5 | 73.3548 | 20.0789 |
| Min | 55.4144 | 20.45 |
| Max | 102.1749 | 21.0789 |
| Avg | 79.6017 | 21.76776 |
| StdDev | 18.32759 | 0.28954 |

Test 3

|  |  |  |
| --- | --- | --- |
| max-prime = 50000 | | |
| max-prime = 50000 | QEMU | Docker |
| Test-num | Total time (in s) | Total time (in s) |
| 1 | 161.1985 | 76.7185 |
| 2 | 160.8651 | 78.192 |
| 3 | 161.7101 | 78.5719 |
| 4 | 162.3358 | 79.4365 |
| 5 | 161.7146 | 77.4922 |
| Min | 160.8651 | 76.7185 |
| Max | 111.7146 | 79.4365 |
| Avg | 161.16482 | 78.08222 |

|  |  |  |
| --- | --- | --- |
| StdDev | 1.52974 | 1.03607 |

In the below screenshot we can see that OS vs kernel space usage of CPU is almost similar for the QEMU.

It is observed that during usage of QEMU, user CPU usage spiked up. However during docker there was just small change in system CPU utilization.

# File I/O test

I have run 4 file I/O tests. For each test a different value of num-threads and file-test-mode was used. Parameters that I have used for file I/O tests are as below:

* 8 threads seqwr mode (sequential write)
* 16 threads seqwr mode (sequential write)
* 8 threads rndwr mode (Random write)
* 16 threads rndwr mode (Random write)

Steps and test results for each platform are described in this section.

## QEMU

qemu\_fileio\_test.sh contains the actual command and runs it 5 times for given set of parameters. qemu\_fileio\_test\_main.sh is the main script to perform fileio tests. It executes qemu\_fileio\_test.sh with parameters described earlier. These parameters will be passed to set of commands which as below to check different test cases:

To prepare files for test

sysbench --num-threads=$threads --test=fileio

### --file-total-size=1G --file-test-mode=$mode prepare

To run the actual test

sysbench --num-threads=$threads --test=fileio

### --file-total-size=1G --file-test-mode=$mode run

To cleanup afterwards

sysbench --num-threads=$threads --test=fileio

### --file-total-size=1G --file-test-mode=$mode cleanup

Test was run using following command

### sh qemu\_fileio\_test\_main.sh > qemu\_fileio\_test.txt

Hence test results for QEMU test are in qemu\_fileio\_test.txt file

## Docker

docker\_fileio\_test.sh contains the command to run the File I/O test on docker and runs it five times for given parameters. docker\_fileio\_test\_main.sh is main scripts which executes docker\_file\_test.sh with parameters described earlier. Command for docker is as below:

To prepare files for test

docker run --rm csminpp/ubuntu-sysbench sysbench

--num-threads=$threads --test=fileio --file-total-size=1G

--file-test-mode=$mode prepare

To run actual test

docker run --rm csminpp/ubuntu-sysbench sysbench

--num-threads=$threads --test=fileio --file-total-size=1G

--file-test-mode=$mode --max-time=300 run

To do cleanup after test

docker run --rm csminpp/ubuntu-sysbench sysbench

--num-threads=$threads --test=fileio --file-total-size=1G

--file-test-mode=$mode cleanup

Docker test can be run using following command:

sh docker\_fileio\_test\_main.sh > docker\_fileio\_test.txt

## Results

Below we can see test results for both QEMU and docker using different parameters.

For each case performance of docker is better than QEMU. so performance of OS virtualization is better in terms of I/O as well.

Also with the increase in number of threads performance improved for random write in both systems. In sequential write performance did improve with change but the difference isn’t significant.

Test 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| num-threads=8 and file-test-mode = seqwr | | | | |
|  | QEMU | | Docker | |
| Test-num | Transfer rate(in Mb/s) | Request/sec | Transfer rate(in Mb/s) | Request/sec |
| 1 | 32.523 | 2081.46 | 396.77 | 25393.38 |
| 2 | 33.235 | 2127.04 | 235.19 | 15051.89 |
| 3 | 32.652 | 2089.76 | 435.57 | 27876.6 |
| 4 | 34.2 | 2188.83 | 303.26 | 19408.66 |
| 5 | 34.38 | 2200.33 | 414.68 | 26539.35 |
| Min | 32.523 | 2081.46 | 235.19 | 15051.89 |
| Max | 34.38 | 2200.33 | 435.57 | 27876.6 |
| Avg | 33.398 | 2137.484 | 357.094 | 22853.976 |
| StdDev | 0.85969 | 55.02585 | 84.86386 | 5431.3915 |

Test 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| num-threads=16 and file-test-mode = seqwr | | | | |
|  | QEMU | | Docker | |
| Test-num | Transfer rate(in Mb/s) | Request/sec | Transfer rate(in Mb/s) | Request/sec |
| 1 | 33.889 | 2168.89 | 385.05 | 24643.2 |
| 2 | 32.803 | 2099.37 | 371.88 | 23800.21 |
| 3 | 33.053 | 2115.37 | 461.64 | 29545.13 |
| 4 | 34.081 | 2181.21 | 386.58 | 24741.3 |
| 5 | 33.063 | 2116 | 427.98 | 27390.62 |
| Min | 32.803 | 2099.37 | 371.88 | 23800.21 |
| Max | 34.081 | 2181.21 | 461.64 | 29545.13 |
| Avg | 33.3778 | 2136.168 | 406.626 | 26024.092 |
| StdDev | 0.56807 | 36.37619 | 37.27326 | 2385.53854 |

Test 3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| num-threads=8 and file-test-mode = rndwr | | | | |
|  | QEMU | | Docker | |
| Test-num | Transfer rate(in Mb/s) | Request/sec | Transfer rate(in Mb/s) | Request/sec |
| 1 | 14.018 | 897.13 | 65.39 | 4184.96 |
| 2 | 13.978 | 894.58 | 63.598 | 4070.26 |
| 3 | 13.968 | 893.98 | 59.213 | 3789.62 |
| 4 | 13.724 | 878.32 | 51.868 | 3319.55 |
| 5 | 13.023 | 833.48 | 61.956 | 3965.21 |
| Min | 13.023 | 833.48 | 51.868 | 3319.55 |
| Max | 14.018 | 897.13 | 65.39 | 4184.96 |
| Avg | 13.7422 | 879.498 | 60.405 | 3865.92 |
| StdDev | 0.4184 | 26.77278 | 5.28565 | 338.28306 |

Test 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| num-threads=16 and file-test-mode = rndwr | | | | |
|  | QEMU | | Docker | |
| Test-num | Transfer rate(in Mb/s) | Request/sec | Transfer rate(in Mb/s) | Request/sec |
| 1 | 12.045 | 770.87 | 87.443 | 5596.36 |
| 2 | 13.818 | 884.37 | 75.815 | 4852.15 |
| 3 | 14.07 | 900.48 | 81.401 | 5209.65 |
| 4 | 13.786 | 882.33 | 80.995 | 5183.7 |
| 5 | 14.617 | 935.49 | 64.646 | 4137.33 |
| Min | 12.045 | 770.87 | 64.646 | 4137.33 |
| Max | 14.617 | 935.49 | 87.443 | 5596.36 |
| Avg | 13.6672 | 874.708 | 78.06 | 4995.838 |
| StdDev | 0.96602 | 61.83128 | 8.55559 | 547.56657 |

# Conclusion

It is observed that OS virtualization is better than system virtualization.