System vs OS Virtualization Report

TABLE OF CONTENTS

1 System and Version Information

Type	Version
OS	Windows 10
CPU	Core i5 (4 cores)
Memory	8 GB
QEMU	6.2.0
Ubuntu	16.04
Sysbench	1.0.20
Docker desktop	20.10.14

Note: I had already installed docker desktop and its performance was good. Also, the docker website recommended using docker desktop instead of docker binaries [3]. So, I used docker desktop for the experiments. However, I only used CLI for executing commands.

2 System Virtualization using QEMU

After installing QEMU and adding its executables to the system's PATH, the below steps were followed to run Ubuntu on QEMU.

2.1 Creating an image

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

The above command creates an image with a size of 10G (like a hard disk) with qcow2 format.

```
C:\Folders\Study\COEN 241>qemu-img create ubuntu.img 10G -f qcow2
Formatting 'ubuntu.img', fmt=qcow2 cluster_size=65536 extended_l2=off compression_type=zlib size=10737418240
 lazy_refcounts=off refcount_bits=16
```

2.2 Boot and Install Ubuntu

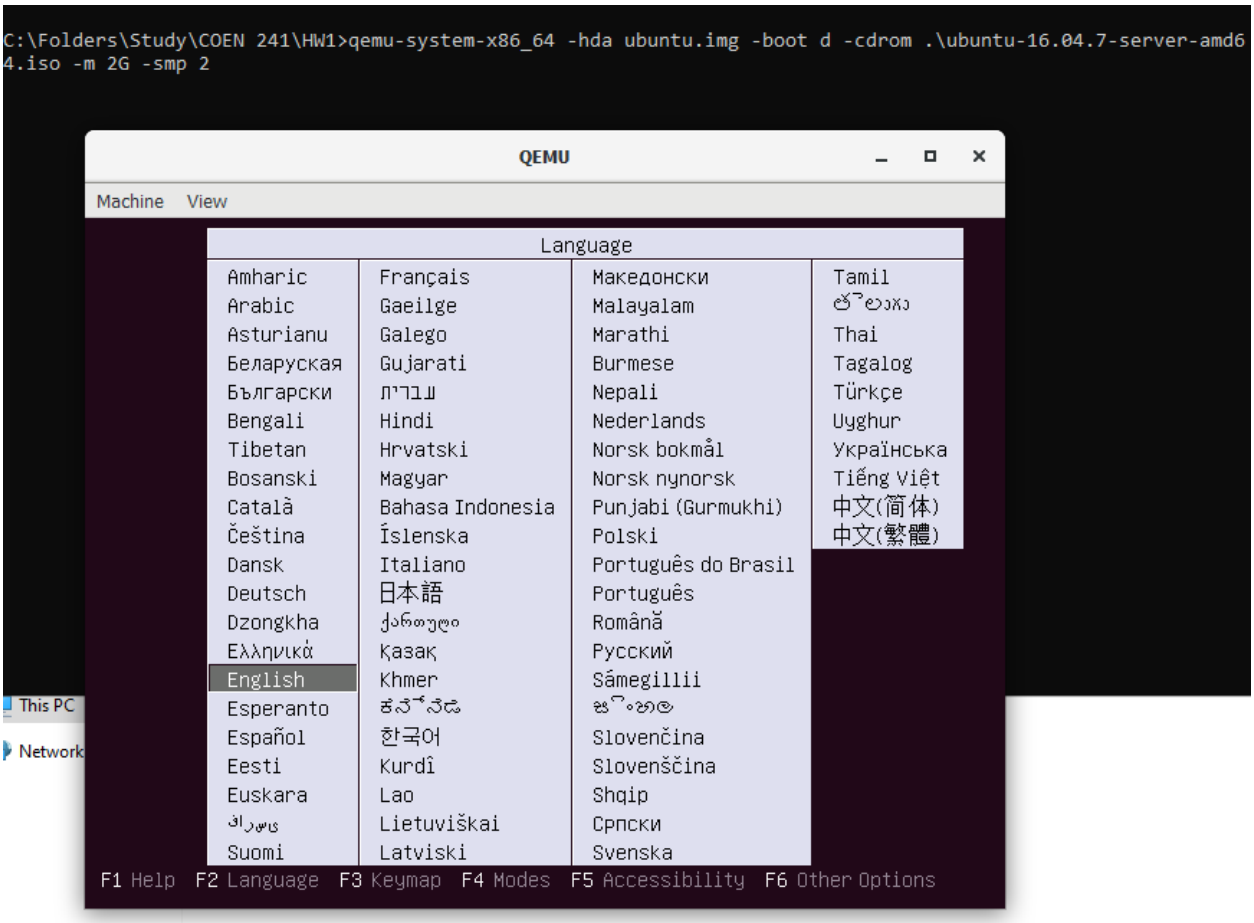
qemu-system-x86_64 -hda ubuntu.img -boot d -cdrom .\ubuntu-16.04.7-server-amd64.iso -m 2G -smp 2

- ISO image of ubuntu for windows is downloaded from here:
  - https://releases.ubuntu.com/16.04/ubuntu-16.04.7-server-amd64.iso
- To use this ISO as cdrom, I specified -cdrom option in the command above.
- To boot from this cdrom, -boot option is used in the command with drive "d" because that is the first cdrom drive in x86 architecture PCs.
- -hda is used to specify the image to use (that was created in the previous step).
- It is assigned 2G memory using -m option and 2 cores using -smp option.

Note: When using -smp 2, QEMU 6.2 (the version that I used) creates 2 cores and 1 socket. Prior to this version, it created 2 sockets and 1 core by default.

- After running this command, it opens up a configuration screen for Ubuntu server. I just followed the steps and completed the installation.

Screenshot: boot from cdrom:



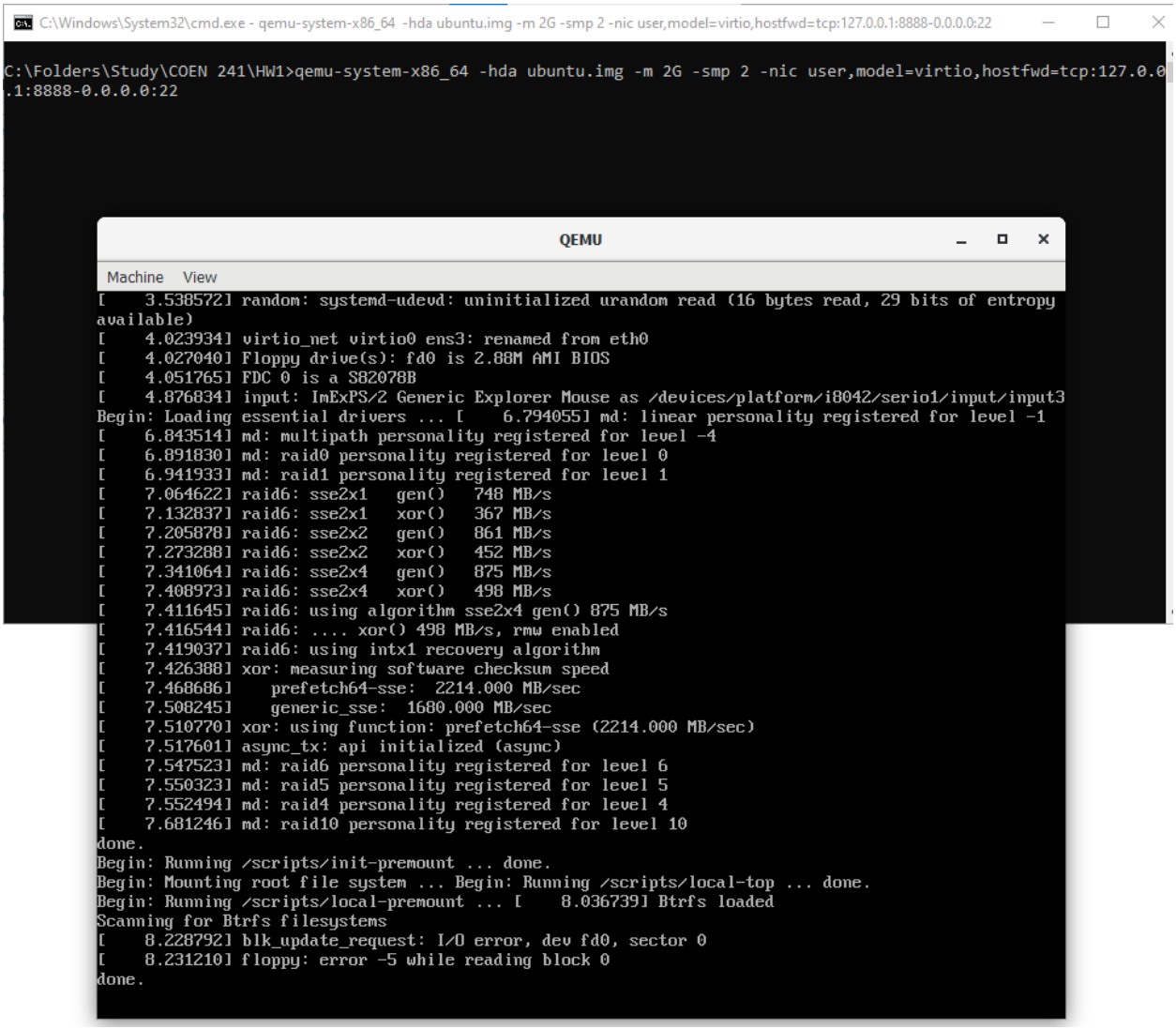
2.3 RUNNING THE VM FROM THE IMAGE

Once the image is all set-up, the boot option is no longer needed, and running the VM can be done using the below command.

```
qemu-system-x86_64 -hda ubuntu.img -m 2G -smp 2 -nic
user,model=virtio,hostfwd=tcp:127.0.0.1:8888-0.0.0.0:22
```

1. Again, -hda is used to specify the image that I created and configured above. Memory and cores have been kept the same.
2. To do SSH, I have used hostfwd configuration of the -nic option. This will forward port 22 of the VM to port 8888 of the localhost.

   Screenshot below shows how it boots Ubuntu from the image:



3. Once the VM is up, SSH to this VM using:
   ssh -p 8888 naveen@localhost

   -p option is used to specify a different port (mapped in the previous step) instead of the default port 22.

   Below is the screenshot for SSH to this VM:

```
C:\Users\navee>ssh -p 8888 naveen@localhost
naveen@localhost's password:
Welcome to Ubuntu 16.04.7 LTS (GNU/Linux 4.4.0-186-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage


113 packages can be updated.
80 updates are security updates.

New release '18.04.6 LTS' available.
Run 'do-release-upgrade' to upgrade to it.


Last login: Thu Apr 14 10:38:00 2022 from 10.0.2.2
naveen@ubuntu:~$
```

4. Install the latest version of sysbench using the below 2 commands (Ref [1])

   curl -s https://packagecloud.io/install/repositories/akopytov/sysbench/script.deb.sh | sudo bash

   sudo apt -y install sysbench

5. Copy the sysbench-script.sh into this VM
   SCP can be used for that:

   scp -P 8888 sysbench-script.sh naveen@localhost:/home/naveen

   The custom port is specified using -P option.

```
C:\Folders\Study\COEN 241\HW1>scp -P 8888 sysbench-script.sh naveen@localhost:/home/naveen
naveen@localhost's password:
sysbench-script.sh                                            100%  577   127.2KB/s   00:00
```

6. Run the test using this command:

   bash sysbench-script.sh [number of threads]

   Number of threads is optional. More details about the script in Shell Scripts section.

3 OS VIRTUALIZATION USING DOCKER

After installing Docker desktop, I created a Dockerfile to create an image.

3.1 DOCKERFILE

```
FROM ubuntu:16.04
RUN apt update && apt install -y curl && curl -s
https://packagecloud.io/install/repositories/akopytov/sysbench/script.deb.sh | bash && apt install
-y sysbench
COPY sysbench-script.sh /var/scripts/sysbench-script.sh
WORKDIR /var/scripts
```

The first line (FROM) takes the base image of ubuntu. I preferred 16.04 version because that is the version of Ubuntu which I used in QEMU and I wanted to keep the environments as similar as possible.

The second line (RUN) installs sysbench latest version.

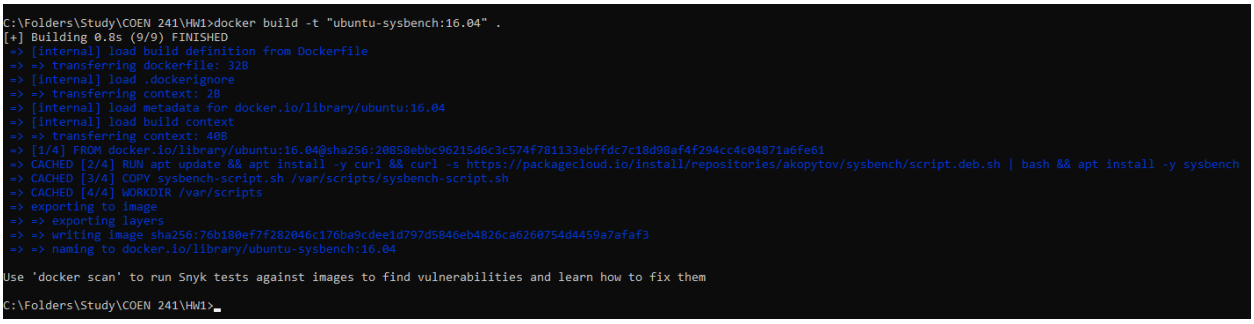
The third line (COPY) copies the sysbench script to directory /var/scripts in ubuntu.

The fourth line (WORKDIR) switches to the work directory where the sysbench script was copied.

3.2 Build Docker Image

To build the image, the below command is used. -t is used to tag the image and "." is the current path for the build context (Dockerfile and other required files).

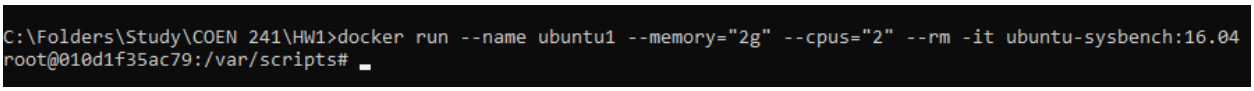
```
docker build -t "ubuntu-sysbench:16.04" .
```



3.3 Run Container from this Image

To run the container, below command is used. --name gives the container a friendly name. --memory is used to provide memory to this container. --cpus gives it number of cores. --rm performs clean-up after the container exits. Shell is an interactive process, so -i option is specified for that (with -t for tty).

```
docker run --name ubuntu1 --memory="2g" --cpus="2" --rm -it ubuntu-sysbench:16.04
```



After the container is up, just need to run the sysbench-script.sh.

3.4 Other Docker Commands Used

docker images	To list the docker images present locally
docker ps	To see currently running containers
docker image rm <imageid>	To remove an image
docker exec -it ubuntu1 /bin/bash	bash into the container that is already running. I used it when monitoring performance while the tests were running in a separate window.

4 RUNNING SYSBENCH TESTS

I ran all the sysbench tests for 30 seconds, so that it gets enough events to process.

4.1 CPU TEST

sysbench --test=cpu --time=30 --cpu-max-prime=20000 run

The default cpu-max-prime=20000 ran sufficient number of events for comparison.

I noticed that when using 2 CPUs, only 1 CPU was being utilized in the sysbench tests because sysbench uses only one thread by default. The below screenshot shows only 1 CPU being utilized.

```
naveen@ubuntu:~$ mpstat -P 0,1 2
Linux 4.4.0-186-generic (ubuntu)        04/15/2022      _x86_64_        (2 CPU)

02:45:57 PM  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
02:45:59 PM    0   97.99    0.00    2.01    0.00    0.00    0.00    0.00    0.00    0.00    0.00
02:45:59 PM    1    0.00    0.00    0.50    0.00    0.00    0.00    0.00    0.00    0.00   99.50
02:46:01 PM    0   98.99    0.00    1.01    0.00    0.00    0.00    0.00    0.00    0.00    0.00
02:46:01 PM    1    0.50    0.00    1.51    0.00    0.00    0.00    0.00    0.00    0.00   97.99
02:46:03 PM    0   98.00    0.00    2.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
02:46:03 PM    1    0.50    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00   99.50
02:46:05 PM    0   97.50    0.00    2.50    0.00    0.00    0.00    0.00    0.00    0.00    0.00
02:46:05 PM    1    0.50    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00   99.50
02:46:07 PM    0   97.99    0.00    2.01    0.00    0.00    0.00    0.00    0.00    0.00    0.00
02:46:07 PM    1    0.00    0.00    0.50    0.00    0.00    0.00    0.00    0.00    0.00   99.50
```

So, whenever I have more than one CPU, I am running sysbench with --threads option. For example, the command used for 2 CPUs will be:

sysbench --test=cpu --time=30 --threads=2 --cpu-max-prime=20000 run

With 2 threads, both CPUs are utilized. This is corroborated by a greater number of events per second in sysbench results when using 2 threads.

```
naveen@ubuntu:~$ mpstat -P 0,1 2
Linux 4.4.0-186-generic (ubuntu)        04/15/2022      _x86_64_        (2 CPU)

02:47:42 PM  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
02:47:44 PM    0   97.99    0.00    2.01    0.00    0.00    0.00    0.00    0.00    0.00    0.00
02:47:44 PM    1   98.49    0.00    1.51    0.00    0.00    0.00    0.00    0.00    0.00    0.00
02:47:46 PM    0   96.98    0.00    3.02    0.00    0.00    0.00    0.00    0.00    0.00    0.00
02:47:46 PM    1   97.00    0.00    2.50    0.00    0.00    0.50    0.00    0.00    0.00    0.00
02:47:48 PM    0   98.00    0.00    2.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
02:47:48 PM    1   97.99    0.00    2.01    0.00    0.00    0.00    0.00    0.00    0.00    0.00
02:47:50 PM    0   97.49    0.00    2.51    0.00    0.00    0.00    0.00    0.00    0.00    0.00
02:47:50 PM    1   97.99    0.00    2.01    0.00    0.00    0.00    0.00    0.00    0.00    0.00
02:47:52 PM    0   97.45    0.00    2.55    0.00    0.00    0.00    0.00    0.00    0.00    0.00
02:47:52 PM    1   97.97    0.00    2.03    0.00    0.00    0.00    0.00    0.00    0.00    0.00
02:47:54 PM    0   96.89    0.00    3.11    0.00    0.00    0.00    0.00    0.00    0.00    0.00
```

Below is the screenshot from QEMU:

```
naveen@ubuntu:~$ sysbench --time=30 --cpu-max-prime=20000 cpu run
sysbench 1.0.20 (using bundled LuaJIT 2.1.0-beta2)

Running the test with following options:
Number of threads: 1
Initializing random number generator from current time


Prime numbers limit: 20000

Initializing worker threads...

Threads started!

CPU speed:
    events per second:   342.31

General statistics:
    total time:                          30.0020s
    total number of events:              10272

Latency (ms):
         min:                                    2.51
         avg:                                    2.91
         max:                                   12.92
         95th percentile:                        3.89
         sum:                                29921.02

Threads fairness:
    events (avg/stddev):           10272.0000/0.00
    execution time (avg/stddev):   29.9210/0.00
```

Below is the screenshot from Docker:

```
root@321e53228caf:/var/scripts# sysbench --time=30 --cpu-max-prime=20000 cpu run
sysbench 1.0.20 (using bundled LuaJIT 2.1.0-beta2)

Running the test with following options:
Number of threads: 1
Initializing random number generator from current time


Prime numbers limit: 20000

Initializing worker threads...

Threads started!

CPU speed:
    events per second:  1342.98

General statistics:
    total time:                          30.0007s
    total number of events:              40294

Latency (ms):
         min:                                    0.71
         avg:                                    0.74
         max:                                    2.56
         95th percentile:                        0.81
         sum:                                29977.90

Threads fairness:
    events (avg/stddev):           40294.0000/0.00
    execution time (avg/stddev):   29.9779/0.00
```

4.2 FileIO Test

This test has 3 steps:

4.2.1 Cleanup (remove any previous files)

sysbench fileio cleanup

```
naveen@ubuntu:~$ sysbench fileio cleanup
sysbench 1.0.20 (using bundled LuaJIT 2.1.0-beta2)

Removing test files...
```

4.2.2 Prepare (create files for the test)

sysbench --file-total-size=4G --file-test-mode=rndrw fileio prepare

```
naveen@ubuntu:~$ sysbench --file-total-size=4G --file-test-mode=rndrw fileio prepare
sysbench 1.0.20 (using bundled LuaJIT 2.1.0-beta2)

128 files, 32768Kb each, 4096Mb total
Creating files for the test...
Extra file open flags: (none)
Creating file test_file.0
Creating file test_file.1
Creating file test_file.2
Creating file test_file.3
Creating file test_file.4
Creating file test_file.5
Creating file test_file.6
Creating file test_file.7
Creating file test_file.8
Creating file test_file.9
Creating file test_file.10
Creating file test_file.11
Creating file test_file.12
Creating file test_file.13
Creating file test_file.14
Creating file test_file.15
Creating file test_file.16
Creating file test_file.17
Creating file test_file.18
```

```
4294967296 bytes written in 30.56 seconds (134.03 MiB/sec).
naveen@ubuntu:~$
```

A total file size of 4G is good enough for the test because the files cannot stay in the main memory.

4.2.3 Run

sysbench --time=30 --file-test-mode=rndrw fileio run

rndrw mode is used for combined and random read/write operations.

Below is the screenshot for QEMU:

```
naveen@ubuntu:~$ sysbench --time=30 --file-test-mode=rndrw fileio run
sysbench 1.0.20 (using bundled LuaJIT 2.1.0-beta2)

Running the test with following options:
Number of threads: 1
Initializing random number generator from current time


Extra file open flags: (none)
128 files, 16MiB each
2GiB total file size
Block size 16KiB
Number of IO requests: 0
Read/Write ratio for combined random IO test: 1.50
Periodic FSYNC enabled, calling fsync() each 100 requests.
Calling fsync() at the end of test, Enabled.
Using synchronous I/O mode
Doing random r/w test
Initializing worker threads...

Threads started!


File operations:
    reads/s:                      427.52
    writes/s:                     285.02
    fsyncs/s:                     916.11

Throughput:
    read, MiB/s:                  6.68
    written, MiB/s:               4.45

General statistics:
    total time:                          30.0279s
    total number of events:              48786

Latency (ms):
         min:                                    0.01
         avg:                                    0.61
         max:                                   11.72
         95th percentile:                        2.71
         sum:                                29639.43

Threads fairness:
    events (avg/stddev):           48786.0000/0.00
    execution time (avg/stddev):   29.6394/0.00
```

Below is the screenshot for Docker:

```
root@321e53228caf:/var/scripts# sysbench --time=30 --file-test-mode=rndrw fileio run
sysbench 1.0.20 (using bundled LuaJIT 2.1.0-beta2)

Running the test with following options:
Number of threads: 1
Initializing random number generator from current time


Extra file open flags: (none)
128 files, 16MiB each
2GiB total file size
Block size 16KiB
Number of IO requests: 0
Read/Write ratio for combined random IO test: 1.50
Periodic FSYNC enabled, calling fsync() each 100 requests.
Calling fsync() at the end of test, Enabled.
Using synchronous I/O mode
Doing random r/w test
Initializing worker threads...

Threads started!


File operations:
    reads/s:                      2388.44
    writes/s:                     1592.30
    fsyncs/s:                     5097.01

Throughput:
    read, MiB/s:                  37.32
    written, MiB/s:               24.88

General statistics:
    total time:                          30.0160s
    total number of events:              272382

Latency (ms):
         min:                                    0.00
         avg:                                    0.11
         max:                                   99.85
         95th percentile:                        0.17
         sum:                                29860.57

Threads fairness:
    events (avg/stddev):           272382.0000/0.00
    execution time (avg/stddev):   29.8606/0.00
```

Note: Running FileIO test multiple times without cleanup gives way better results for subsequent runs. I downloaded and used the utility "Sync" but that didn't solve this problem. Therefore, I am doing cleanup (and prepare) before any FileIO test is run because that solves this problem.

5 MONITORING PERFORMANCE

For monitoring CPU and I/O performance of VMs on QEMU, I have used 2 utilities:

1. Sysstat for CPU:

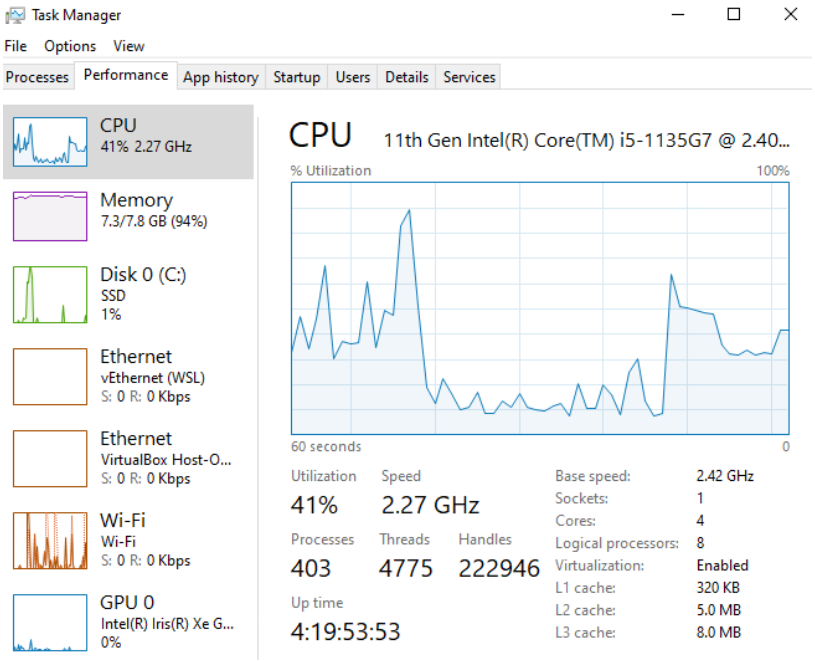
   Installation: sudo apt install sysstat

   Usage (for 2 CPUs, updates every 2 seconds): mpstat -P 0,1 2

   When running CPU test on QEMU, with 2 CPUs and 2G memory, the user-level CPU utilization (%usr) is almost 98% for both CPUs. The kernel-level CPU utilization (%sys) stays as low as 2%.

```
naveen@ubuntu:~$ mpstat -P 0,1 2
Linux 4.4.0-186-generic (ubuntu)        04/15/2022      _x86_64_        (2 CPU)

02:56:52 PM  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
02:56:54 PM    0   98.00    0.00    2.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
02:56:54 PM    1   97.99    0.00    2.01    0.00    0.00    0.00    0.00    0.00    0.00    0.00
02:56:56 PM    0   97.97    0.00    2.03    0.00    0.00    0.00    0.00    0.00    0.00    0.00
02:56:56 PM    1   98.48    0.00    1.52    0.00    0.00    0.00    0.00    0.00    0.00    0.00
02:56:58 PM    0   98.49    0.00    1.51    0.00    0.00    0.00    0.00    0.00    0.00    0.00
02:56:58 PM    1   96.98    0.00    3.02    0.00    0.00    0.00    0.00    0.00    0.00    0.00
```

Expectedly, the host (windows) CPU utilization spikes from 9% to 40% during the test.

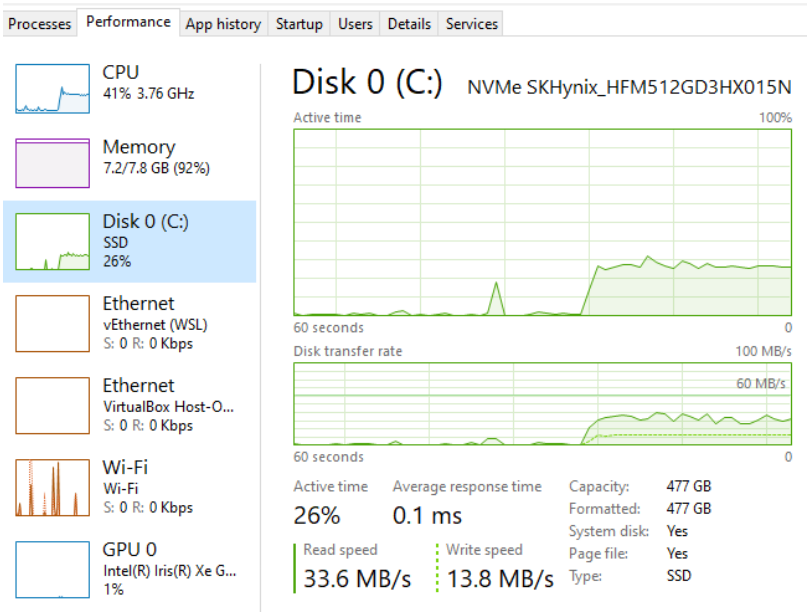


2. iotop for I/O:
   Installation: sudo apt install iotop
   Usage (monitor disk usage, updates every 2 seconds): iostat -d 2

   When running FileIO test on QEMU with 2 CPUs and 2G memory, the disk usage is around 3700 kB/s for reads, and for writes it is ranging between 2400 and 5700 kB/s.

```
Device:            tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
loop0             0.00         0.00         0.00          0          0
sda             780.50      3556.00      2436.00       7112       4872
fd0               0.00         0.00         0.00          0          0

Device:            tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
loop0             0.00         0.00         0.00          0          0
sda            1363.00      3680.00      4600.00       7360       9200
fd0               0.00         0.00         0.00          0          0

Device:            tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
loop0             0.00         0.00         0.00          0          0
sda            1714.00      4934.00      5702.00       9868      11404
fd0               0.00         0.00         0.00          0          0

Device:            tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
loop0             0.00         0.00         0.00          0          0
sda            1308.00      3704.00      4360.00       7408       8720
fd0               0.00         0.00         0.00          0          0

Device:            tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
loop0             0.00         0.00         0.00          0          0
sda            1312.50      3728.00      4408.00       7456       8816
fd0               0.00         0.00         0.00          0          0
```

As expected, the disk usage in the host OS also spikes as shown below:



The CPU utilization at the user-level is almost 0 for the FileIO test. However, the kernel-level CPU utilization increases up to 20%.

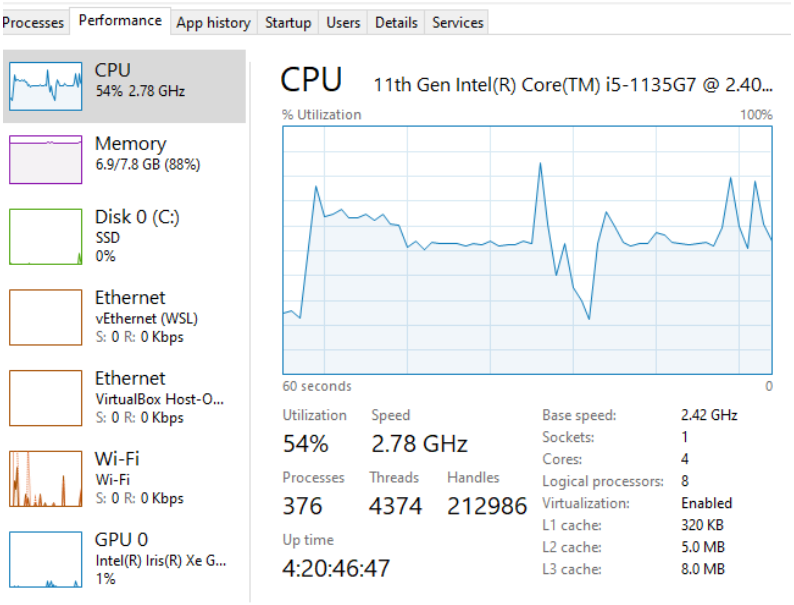
```
naveen@ubuntu:~$ mpstat -P 0,1 2
Linux 4.4.0-186-generic (ubuntu)        04/15/2022      _x86_64_        (2 CPU)

03:13:07 PM  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
03:13:09 PM    0    1.33    0.00   14.67   38.00    0.00    0.00    0.00    0.00    0.00   46.00
03:13:09 PM    1    0.00    0.00   18.45   20.39    0.00    1.94    0.00    0.00    0.00   59.22
03:13:11 PM    0    1.29    0.00   18.71   38.06    0.00    0.00    0.00    0.00    0.00   41.94
03:13:11 PM    1    1.83    0.00   20.18   19.27    0.00    2.75    0.00    0.00    0.00   55.96
03:13:13 PM    0    1.30    0.00   13.64   40.91    0.00    0.00    0.00    0.00    0.00   44.16
03:13:13 PM    1    0.00    0.00   17.31   23.08    0.00    1.92    0.00    0.00    0.00   57.69
03:13:15 PM    0    0.68    0.00   13.51   38.51    0.00    0.00    0.00    0.00    0.00   47.30
03:13:15 PM    1    0.00    0.00   20.56   18.69    0.00    3.74    0.00    0.00    0.00   57.01
03:13:17 PM    0    0.66    0.00   13.91   40.40    0.00    0.00    0.00    0.00    0.00   45.03
03:13:17 PM    1    0.00    0.00   22.02   18.35    0.00    1.83    0.00    0.00    0.00   57.80
```

For docker, I have used the command docker stats to monitor the CPU utilization.

When running CPU test on docker with 2 CPUs and 2G memory, the CPU is completely utilized as shown below, 200% indicating that both CPUs are utilized.

```
CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT   MEM %     NET I/O          BLOCK I/O   PIDS
a000edca7c3c   ubuntu1   198.51%   6.082MiB / 200MiB   3.04%     492kB / 18.3kB   0B / 0B     5
```

The CPU utilization displayed on the Host OS spikes as well.



It's worth reiterating that running sysbench with the default value of --threads (i.e., 1) does not utilize the CPU completely. This is visible in the below screenshot where the CPU utilization is 99% instead of 198% as shown in the above scenario.

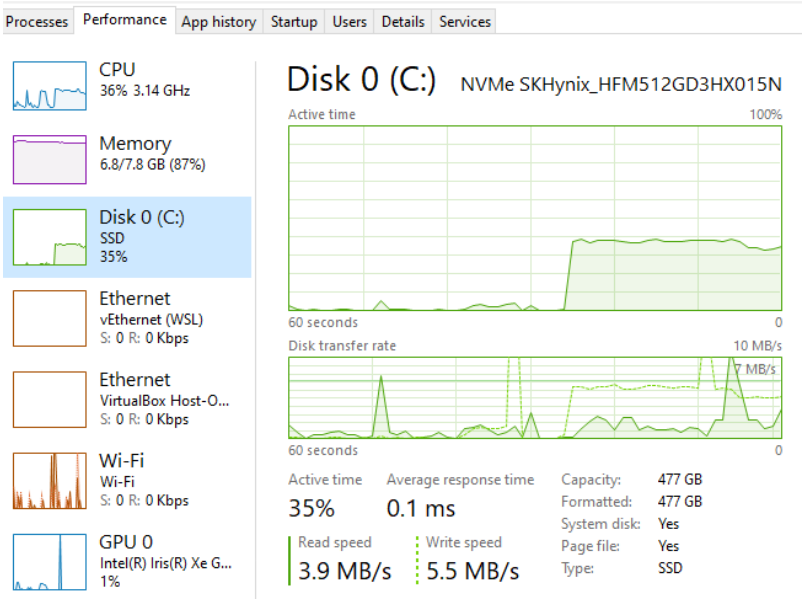
```
CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT   MEM %     NET I/O          BLOCK I/O   PIDS
a000edca7c3c   ubuntu1   99.75%    6.164MiB / 200MiB   3.08%     492kB / 18.3kB   0B / 0B     4
```

Running FileIO test doesn't display any stats using docker stats command. So, for this, I went back to iostat.

When running FileIO test on docker with 2 CPUs and 2G memory, the rate of reads and writes is almost 20 times the rate of QEMU VM.

```
Device:            tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
loop0             0.00         0.00         0.00          0          0
loop1             0.00         0.00         0.00          0          0
sda               0.00         0.00         0.00          0          0
sdb               0.00         0.00         0.00          0          0
sdc           17862.00     71768.00     56982.00     143536     113964

Device:            tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
loop0             0.00         0.00         0.00          0          0
loop1             0.00         0.00         0.00          0          0
sda               0.00         0.00         0.00          0          0
sdb               0.00         0.00         0.00          0          0
sdc           17583.50     71000.00     56164.00     142000     112328
```

The Host disk usage also seems to be higher than when QEMU was running.



6 SYSBENCH TEST RESULTS

All the screenshots can be found in Appendix.

6.1 TEST 1: MEMORY 2G, CPU 2

Results from QEMU:

Statistic → Test ↓	Min	Max	Avg	StdDev
CPU (events/sec)	336	611	470	87
FileIO Read throughput (MiB/s)	5.48	7.99	6.37	0.85

FileIO Write throughput (MiB/s)	3.65	5.33	4.25	0.56
FileIO Avg Latency	1.02	1.46	1.28	0.14

Results from Docker

Statistic → Test ↓	**Min**	**Max**	**Avg**	**StdDev**
CPU (events/sec)	1062.10	2155.73	1482.35	398.7
FileIO Read throughput (MiB/s)	37.18	52.62	44.7	5.5
FileIO Write throughput (MiB/s)	24.79	35.08	29.8	3.68
FileIO Avg Latency	0.15	0.22	0.18	0.02

6.2 TEST 2: MEMORY 1G, CPU 2

Results from QEMU:

Statistic → Test ↓	**Min**	**Max**	**Avg**	**StdDev**
CPU (events/sec)	284.76	475.8	371.58	61.07
FileIO Read throughput	4.12	6.42	4.99	0.76
FileIO Write throughput	2.74	4.28	3.32	0.5
FileIO Avg Latency	1.26	1.95	1.64	0.22

Results from Docker:

Statistic → Test ↓	**Min**	**Max**	**Avg**	**StdDev**
CPU (events/sec)	1348.54	2071.1	1809.01	275.1
FileIO Read throughput	69.86	85.23	75.11	6.22

FileIO Write throughput	46.57	56.82	50.07	4.15
FileIO Avg Latency	0.10	0.12	0.11	0.009

6.3 TEST 3: MEMORY 2G, CPU 1

Results from QEMU:

Statistic → Test ↓	**Min**	**Max**	**Avg**	**StdDev**
CPU (events/sec)	244	304.4	280.4	21.18
FileIO Read throughput	3.9	4.5	4.23	0.21
FileIO Write throughput	2.6	3	2.82	0.14
FileIO Avg Latency	0.9	1.04	0.96	0.04

Results from Docker:

Statistic → Test ↓	**Min**	**Max**	**Avg**	**StdDev**
CPU (events/sec)	989.4	1325.36	1200.5	119.12
FileIO Read throughput	23.6	36.8	29.92	5.03
FileIO Write throughput	15.74	24.52	19.94	3.35
FileIO Avg Latency	0.11	0.17	0.138	0.02

6.4 TEST 4: WHPX ACCELERATOR (QEMU SPECIFIC)

QEMU uses tcg as the default accelerator. This test runs the VM with whpx accelerator.

```
qemu-system-x86_64 -hda ubuntu.img -accel whpx -m 2G -smp 2 -nic
user,model=virtio,hostfwd=tcp:127.0.0.1:8888-0.0.0.0:22
```

Statistic → Test ↓	**Min**	**Max**	**Avg**	**StdDev**
CPU (events/sec)	1900.94	2259.98	2062.8	131.01
FileIO Read throughput	4.23	4.68	4.47	0.15

FileIO Write throughput	2.82	3.12	2.98	0.1
FileIO Avg Latency	1.75	1.93	1.83	0.06

7 SYSBENCH RESULTS ANALYSIS

7.1 COMPARING CPU

Premise: Test 1 and Test 2 have the same CPU (i.e., 2) but Test 2 has just 1G memory compared to 2G memory in Test1.

The average CPU benchmark for QEMU in Test 2 (371.58) is quite similar to Test 1 (470) which is expected. The average CPU benchmark of docker increased in Test 2 (1809) compared to Test 1 (1482), but the max values remained the same (around 2100). Also considering a significant standard deviation in docker tests, I think benchmarks for docker are similar due to CPU being the same.

Premise: Test 1 uses 2 CPUs while Test 3 uses just 1 CPU.

The average CPU benchmark numbers for both QEMU and Docker decline significantly in Test 3 due to the use of just 1 CPU, which is the expected behavior.

7.2 COMPARING FILEIO

Premise: Test 1 uses 2G main memory while Test 2 uses just 1G memory.

The average read throughput for QEMU in Test 2 is 6.37 compared to 4.99 in Test2. This is only a slight decrease, possibly because of reduced memory. The same can be said for write throughput and average latency.

The read throughput of docker for Test 2 (75.11) is greater than Test 1 (44.7), which is surprising. I ran the tests again, but the results were consistent with previous results. Strangely, the memory utilization during the test stays around 100MiB for both tests. So, I think the memory doesn't affect the FileIO test.

Premise: Test 1 uses 2 CPUs while Test 3 uses just 1 CPU.

Since FileIO uses kernel CPU, reducing the CPU to 1 in Test 3 reduces the read/write throughput slightly in both QEMU and Docker when compared to Test 1 where 2 CPUs are used, which is expected.

7.3 COMPARING QEMU WITH DOCKER

CPU performance of Docker gave far better results in all test scenarios, which is mainly because it is an OS virtualization, making it lightweight when compared to System virtualization which has an added abstraction layer.

Similarly, FileIO tests in Docker perform better than QEMU across all test scenarios.

7.4 The WHPX Accelerator

Test 4 was surprising to me. I decided to try whpx (windows hypervisor platform) accelerator on QEMU instead of the default tcg accelerator and the CPU performance was far greater than all other QEMU tests and only slightly better than docker, which indicates that the whpx accelerator has excellent performance, at least for windows systems. However, FileIO test results remained the same.

8 Vagrant

To try out vagrant, I have used “virtualbox” as the provider.

```
Vagrant.configure("2") do |config|
  config.vm.define "ubuntu" do |ubuntu|
    ubuntu.vm.box = "ubuntu/xenial64"
    ubuntu.vm.boot_timeout=500
    ubuntu.vm.provider "virtualbox" do |vb|
      vb.memory = 2048
      vb.cpus = 1
    end
    ubuntu.vm.provision "file", source: "sysbench-script.sh", destination: "/tmp/sysbench-script.sh"
    ubuntu.vm.provision "shell", inline: <<-SHELL
      apt update
      apt install -y curl
      curl -s https://packagecloud.io/install/repositories/akopytov/sysbench/script.deb.sh | bash
      apt install -y sysbench
      mkdir /var/scripts
      cp /tmp/sysbench-script.sh /var/scripts/sysbench-script.sh
    SHELL
  end
end
```

The above Vagrantfile performs these steps:

1. Give the box name ubuntu using config.vm.define.
2. Select ubuntu/xenial64 as the vm box (image).
3. Select virtualbox as the VMM provider.
4. Assign memory and cpu to this VM using memory and cpus.
5. Install the latest version of sysbench on this VM.
6. Copy the sysbench-script file (which contains cpu and fileio tests) to /var/scripts directory of this VM. There was a permissions issue when copying directly to the guest VM’s var directory (because it uses SCP and SCP does not have sudo), due to which I had to first copy the script to /tmp and then copy from /tmp to /var/scripts.
7. Booting ubuntu timed out the first time I tried to install this VM, so I had to increase the timeout to 500, and then it boot-up just fine.

With the vagrantfile, managing the VM becomes easy:

1. Bring up the VM (with the name of the VM given in Vagrantfile)
   vagrant up ubuntu

```
C:\Folders\Study\COEN 241\HW1>vagrant up ubuntu
Bringing machine 'ubuntu' up with 'virtualbox' provider...
==> ubuntu: Importing base box 'ubuntu/xenial64'...
==> ubuntu: Matching MAC address for NAT networking...
==> ubuntu: Checking if box 'ubuntu/xenial64' version '20211001.0.0' is up to date...
==> ubuntu: Setting the name of the VM: HW1_ubuntu_1650038752126_86441
==> ubuntu: Fixed port collision for 22 => 2222. Now on port 2200.
==> ubuntu: Clearing any previously set network interfaces...
==> ubuntu: Preparing network interfaces based on configuration...
    ubuntu: Adapter 1: nat
==> ubuntu: Forwarding ports...
    ubuntu: 22 (guest) => 2200 (host) (adapter 1)
==> ubuntu: Running 'pre-boot' VM customizations...
==> ubuntu: Booting VM...
==> ubuntu: Waiting for machine to boot. This may take a few minutes...
    ubuntu: SSH address: 127.0.0.1:2200
    ubuntu: SSH username: vagrant
    ubuntu: SSH auth method: private key
    ubuntu:
    ubuntu: Vagrant insecure key detected. Vagrant will automatically replace
    ubuntu: this with a newly generated keypair for better security.
```

2. SSH into the VM

   vagrant ssh ubuntu

   The screenshot below shows that the sysbench script is copied to the guest VM.

```
C:\Folders\Study\COEN 241\HW1>vagrant ssh ubuntu
Welcome to Ubuntu 16.04.7 LTS (GNU/Linux 4.4.0-210-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

UA Infra: Extended Security Maintenance (ESM) is not enabled.

1 update can be applied immediately.
To see these additional updates run: apt list --upgradable

95 additional security updates can be applied with UA Infra: ESM
Learn more about enabling UA Infra: ESM service for Ubuntu 16.04 at
https://ubuntu.com/16-04

New release '18.04.6 LTS' available.
Run 'do-release-upgrade' to upgrade to it.


vagrant@ubuntu-xenial:~$ cd /var/scripts/
vagrant@ubuntu-xenial:/var/scripts$ ls
sysbench-script.sh
vagrant@ubuntu-xenial:/var/scripts$
```

3. Destroy the VM (if not needed anymore)

   vagrant destroy ubuntu

```
C:\Folders\Study\COEN 241\HW1>vagrant destroy ubuntu
    ubuntu: Are you sure you want to destroy the 'ubuntu' VM? [y/N] y
==> ubuntu: Forcing shutdown of VM...
==> ubuntu: Destroying VM and associated drives...

C:\Folders\Study\COEN 241\HW1>
```

9 SHELL SCRIPTS

sysbench-script.sh is the main script that runs both CPU and FileIO test one time. It takes in one argument that specifies the number of threads to use. For example, to run with 2 threads,

bash sysbench-script.sh 2

If no argument is passed, it runs with default 1 thread (defined by sysbench).

Below is the script:

```
echo -e "\nRunning CPU Test\n"
if [ -z "$1" ]
   then
     sysbench --time=30 --cpu-max-prime=20000 cpu run
   else
     sysbench --time=30 --cpu-max-prime=20000 --threads="$1" cpu run
fi
echo -e "\nRunning FileIO Test\n"
echo -e "\nFileIO Test - Cleanup\n"
sysbench fileio cleanup
echo -e "\nFileIO Test - Prepare\n"
sysbench --file-total-size=4G --file-test-mode=rndrw fileio prepare
echo -e "\nFileIO Test - Run\n"
if [ -z "$1" ]
   then
     sysbench --time=30 --file-test-mode=rndrw fileio run
   else
     sysbench --time=30 --file-test-mode=rndrw --threads="$1" fileio run
fi
```

I have also included 2 more scripts which I used only to get the relevant lines from the sysbench output.

cpu-test.sh**:** Runs CPU test 5 times and prints only the line “events per second”. It has the same argument to specify the number of threads just like sysbench-script.

```
for i in {1..5}
do
  if [ -z "$1" ]
    then
      sysbench --time=30 --cpu-max-prime=20000 cpu run | grep "events per second"
    else
      sysbench --time=30 --cpu-max-prime=20000 --threads="$1" cpu run | grep "events per second"
  fi
done
```

A sample run:

```
naveen@ubuntu:~$ bash cpu-test.sh 2
    events per second:   611.98
    events per second:   484.97
    events per second:   453.95
    events per second:   467.32
    events per second:   336.47
```

fileio-test.sh**:** Runs FileIO test 5 times and prints only the lines with throughput and latency. The argument for the number of threads is the same as above.

```
for i in {1..5}
do
  sysbench fileio --verbosity=0 cleanup
  sysbench --file-total-size=4G --file-test-mode=rndrw --verbosity=0 fileio prepare
  if [ -z "$1" ]
    then
      sysbench --time=30 --file-test-mode=rndrw fileio run | grep -E "read, MiB/s|written, MiB/s|avg:"
    else
      sysbench --time=30 --file-test-mode=rndrw --threads="$1" fileio run | grep -E "read, MiB/s|written, MiB/s|avg:"
  fi
done
```

A sample run:

```
naveen@ubuntu:~$ bash fileio-test.sh 2
    read, MiB/s:                  6.02
    written, MiB/s:               4.01
         avg:                                 1.34
    read, MiB/s:                  6.11
    written, MiB/s:               4.07
         avg:                                 1.32
    read, MiB/s:                  5.48
    written, MiB/s:               3.65
         avg:                                 1.46
    read, MiB/s:                  6.29
    written, MiB/s:               4.19
         avg:                                 1.28
    read, MiB/s:                  7.99
    written, MiB/s:               5.33
         avg:                                 1.02
```

To not overcomplicate the script, I am redirecting this output to a file and then getting the required values:

For example, to get the read throughput values from 5 tests.

```
naveen@ubuntu:~$ cat fr1.txt | grep read | cut -d : -f 2 | awk 'NF { $1=$1; print }' | paste -sd "," -
6.02,6.11,5.48,6.29,7.99
naveen@ubuntu:~$
```

cat fr1.txt | grep read | cut -d : -f 2 | awk 'NF { $1=$1; print }' | paste -sd "," -

cat fr1.txt | grep written | cut -d : -f 2 | awk 'NF { $1=$1; print }' | paste -sd "," -

cat fr1.txt | grep avg| cut -d : -f 2 | awk 'NF { $1=$1; print }' | paste -sd "," -

(Ref for awk: [6]):

10 REFERENCES

[1] https://github.com/akopytov/sysbench#linux

[2] https://www.qemu.org/docs/master/system/invocation.html

[3] https://docs.docker.com/engine/install/binaries/

[4] https://app.vagrantup.com/ubuntu/boxes/xenial64

[5] https://www.vagrantup.com/docs/provisioning/file

[6] https://stackoverflow.com/questions/8562782/delete-empty-lines-and-trim-surrounding-spaces-in-bash

11 APPENDIX: SCREENSHOTS OF ALL EXPERIMENTS

A. Mem 2G, CPU 2

QEMU:

```
naveen@ubuntu:~$ bash cpu-test.sh 2
    events per second:   611.98
    events per second:   484.97
    events per second:   453.95
    events per second:   467.32
    events per second:   336.47
```

```
naveen@ubuntu:~$ bash fileio-test.sh 2
    read, MiB/s:                  6.02
    written, MiB/s:               4.01
         avg:                                 1.34
    read, MiB/s:                  6.11
    written, MiB/s:               4.07
         avg:                                 1.32
    read, MiB/s:                  5.48
    written, MiB/s:               3.65
         avg:                                 1.46
    read, MiB/s:                  6.29
    written, MiB/s:               4.19
         avg:                                 1.28
    read, MiB/s:                  7.99
    written, MiB/s:               5.33
         avg:                                 1.02
```

Docker:

```
C:\Folders\Study\COEN 241\HW1>docker run --name ubuntu1 --memory="2g" --cpus="2" --rm -it ubuntu-sysbench:16.04
root@4b1554f9951d:/var/scripts# bash cpu-test.sh 2
    events per second:  2155.73
    events per second:  1699.14
    events per second:  1062.10
    events per second:  1186.60
    events per second:  1308.18
root@4b1554f9951d:/var/scripts# bash fileio-test.sh 2 > fr1.txt
root@4b1554f9951d:/var/scripts# cat fr1.txt
    read, MiB/s:                 52.62
    written, MiB/s:              35.08
         avg:                                0.15
    read, MiB/s:                 37.18
    written, MiB/s:              24.79
         avg:                                0.22
    read, MiB/s:                 41.23
    written, MiB/s:              27.49
         avg:                                0.20
    read, MiB/s:                 49.15
    written, MiB/s:              32.76
         avg:                                0.17
    read, MiB/s:                 43.38
    written, MiB/s:              28.92
         avg:                                0.19
```

B. Mem 1G, CPU 2

QEMU:

```
naveen@ubuntu:~$ bash cpu-test.sh 2
    events per second:   475.84
    events per second:   373.58
    events per second:   356.78
    events per second:   366.98
    events per second:   284.76
naveen@ubuntu:~$ bash fileio-test.sh 2
    read, MiB/s:                 6.42
    written, MiB/s:              4.28
         avg:                                1.26
    read, MiB/s:                 4.77
    written, MiB/s:              3.18
         avg:                                1.67
    read, MiB/s:                 4.12
    written, MiB/s:              2.74
         avg:                                1.95
    read, MiB/s:                 4.82
    written, MiB/s:              3.22
         avg:                                1.66
    read, MiB/s:                 4.82
    written, MiB/s:              3.22
         avg:                                1.67
```

Docker:

```
root@76bfdbaf701b:/var/scripts# bash cpu-test.sh 2
    events per second:  2071.10
    events per second:  1348.54
    events per second:  1637.91
    events per second:  2002.70
    events per second:  1984.81
root@76bfdbaf701b:/var/scripts# bash fileio-test.sh 2
    read, MiB/s:                  85.23
    written, MiB/s:               56.82
         avg:                                  0.10
    read, MiB/s:                  79.60
    written, MiB/s:               53.07
         avg:                                  0.10
    read, MiB/s:                  70.49
    written, MiB/s:               46.99
         avg:                                  0.12
    read, MiB/s:                  70.38
    written, MiB/s:               46.92
         avg:                                  0.12
    read, MiB/s:                  69.86
    written, MiB/s:               46.57
         avg:                                  0.12
root@76bfdbaf701b:/var/scripts#
```

C. Mem 2G, CPU 1

QEMU:

```
naveen@ubuntu:~$ bash cpu-test.sh 1
    events per second:   304.41
    events per second:   283.18
    events per second:   244.03
    events per second:   273.18
    events per second:   297.29
naveen@ubuntu:~$ bash fileio-test.sh 1
    read, MiB/s:                  4.37
    written, MiB/s:               2.91
         avg:                                  0.93
    read, MiB/s:                  4.26
    written, MiB/s:               2.84
         avg:                                  0.95
    read, MiB/s:                  3.90
    written, MiB/s:               2.60
         avg:                                  1.04
    read, MiB/s:                  4.11
    written, MiB/s:               2.74
         avg:                                  0.99
    read, MiB/s:                  4.51
    written, MiB/s:               3.01
         avg:                                  0.90
naveen@ubuntu:~$
```

Docker:

```
root@24ff529794f3:/var/scripts# bash cpu-test.sh 1
    events per second:  1301.11
    events per second:  1325.36
    events per second:  1210.35
    events per second:  1176.46
    events per second:   989.40
root@24ff529794f3:/var/scripts# bash fileio-test.sh 1
    read, MiB/s:                  30.40
    written, MiB/s:               20.27
         avg:                                0.13
    read, MiB/s:                  33.82
    written, MiB/s:               22.55
         avg:                                0.12
    read, MiB/s:                  36.79
    written, MiB/s:               24.52
         avg:                                0.11
    read, MiB/s:                  23.62
    written, MiB/s:               15.74
         avg:                                0.17
    read, MiB/s:                  24.98
    written, MiB/s:               16.65
         avg:                                0.16
root@24ff529794f3:/var/scripts#
```

D. QEMU with 2 CPU and 2G Memory. But with whpx accelerator.

```
naveen@ubuntu:~$ bash cpu-test.sh 2
    events per second:  2259.98
    events per second:  2167.33
    events per second:  1997.82
    events per second:  1900.94
    events per second:  1987.97
naveen@ubuntu:~$ bash fileio-test.sh 2
    read, MiB/s:                  4.42
    written, MiB/s:               2.94
         avg:                                1.85
    read, MiB/s:                  4.57
    written, MiB/s:               3.05
         avg:                                1.79
    read, MiB/s:                  4.23
    written, MiB/s:               2.82
         avg:                                1.93
    read, MiB/s:                  4.45
    written, MiB/s:               2.97
         avg:                                1.84
    read, MiB/s:                  4.68
    written, MiB/s:               3.12
         avg:                                1.75
naveen@ubuntu:~$
```