

System vs OS Virtualization

Configurations of the Experimental Setups:

I planned to run the VM and the Container in 3 different setups:

- 2GB of memory, 1 CPU cores, 1 thread per core
- 2GB of memory, 2 CPU cores, 1 thread per core
- 4GB of memory, 4 CPU cores, 1 thread per core

Configs 1 and 2 were meant to test the effect of an additional CPU core while 3 was meant to test the effect of double the resources in comparison to config 2.

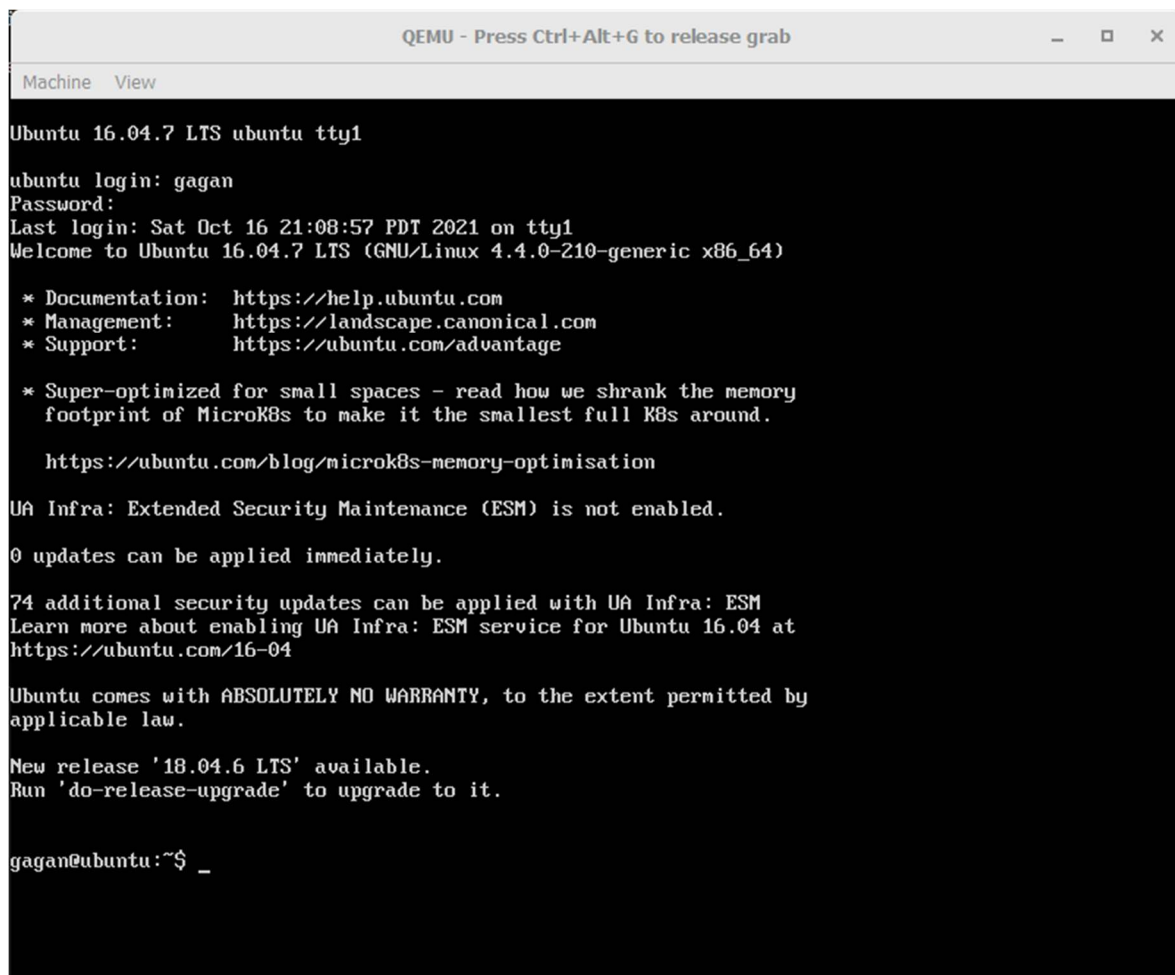
How to run QEMU VM:

QEMU requires the following steps to run a VM:

1. Get QEMU and the desired VM image
 2. Make sure you have the correct admin privileges
 3. Set up your environmental variables if applicable
 4. Create the QEMU image
 - a. e.x. `qemu-img.exe create ubuntu.img 10G -f qcow2`
 5. Run the install of your VM image
 - a. e.x. `qemu-system-x86_64.exe -hda .\ubuntu.img -boot d -cdrom .\ubuntu-16.04.7-server-amd64.iso -m 2046 -boot strict=on`
 6. Boot into the VM using a modified version of the previous command
 - a. e.x. `qemu-system-x86_64.exe -hda .\ubuntu.img -boot d -m 2046 -boot strict=on`
 7. Go through all the setup steps and finally boot into your desired VM
- Notes on possible resource flags in command
 - a. `-m`
 - Changes the amount of memory your VM has allocated
 - a. `-smp`
 - Changes the cpu configuration

- -accel
 - Changes the form of acceleration your VM will use
- My commands for the 3 different configurations
 - qemu-system-x86_64.exe -hda .\ubuntu.img -boot d -m 2G -boot strict=on
 - qemu-system-x86_64.exe -hda .\ubuntu.img -boot d -m 2G -smp 2,sockets=2,maxcpus=2 --accel tcg,thread=single -boot strict=on
 - qemu-system-x86_64.exe -hda .\ubuntu.img -boot d -m 4G -smp 4,sockets=4,maxcpus=4 --accel tcg,thread=single -boot strict=on

Screenshots of QEMU VM:



The screenshot shows a QEMU window titled "QEMU - Press Ctrl+Alt+G to release grab". The window contains a terminal window titled "Machine View" showing the Ubuntu 16.04.7 LTS login screen. The terminal output is as follows:

```
Ubuntu 16.04.7 LTS ubuntu tty1
ubuntu login: gagan
Password:
Last login: Sat Oct 16 21:08:57 PDT 2021 on tty1
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

* 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

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

0 updates can be applied immediately.

74 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

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

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

gagan@ubuntu:~$ _
```

```
QEMU - Press Ctrl+Alt+G to release grab
Machine View
Learn more about enabling UA Infra: ESM service for Ubuntu 16.04 at
https://ubuntu.com/16-04

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

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

gagan@ubuntu:~$ ls
coen241
gagan@ubuntu:~$ cd coen241/
gagan@ubuntu:~/coen241$ ls
hw1 hw1-backup
gagan@ubuntu:~/coen241$ cd hw1
gagan@ubuntu:~/coen241/hw1$ ls
QEMU_sysbench_tests.sh QEMU_test_log_config1.txt QEMU_test_log_config3.txt
QEMU_test_log1.txt      QEMU_test_log_config2.txt QEMU_test_log.txt
gagan@ubuntu:~/coen241/hw1$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        deleted:    sysbench_cpu_fileio_test.sh
        deleted:    test_log.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        .git-old/
        QEMU_test_log1.txt

no changes added to commit (use "git add" and/or "git commit -a")
gagan@ubuntu:~/coen241/hw1$ _
```

How to run Docker Container:

Docker requires the following steps to run a VM:

1. Download/Install Docker
 2. Pick the image you would like to use from the online repo or elsewhere and download it
 3. Use the run command with the required arguments and flags to boot your container
 - a. e.x. `docker run -itd [IMAGE] bash`
 - i. `-d` means the container will stay running in the background
 - ii. `"bash"` at the end makes the CLI open after you run the command assuming the container didn't instantly quit
- Notes on possible resource flags in command
 - a. `--memory`
 - Changes the amount of memory your VM has allocated

- --cpus
 - Changes the cpu configuration
- My commands for the 3 different configurations
 - docker run --name sysbench --memory="2G" --cpus="1.0" --privileged -itd csminpp/ubuntu-sysbench bash
 - docker run --name sysbench --memory="2G" --cpus="2.0" --privileged -itd csminpp/ubuntu-sysbench bash
 - docker run --name sysbench --memory="4G" --cpus="4.0" --privileged -itd csminpp/ubuntu-sysbench bash

Screenshots of Docker Container:

```

root@2e32de4754e: /home
Microsoft Windows [Version 10.0.19043.1288]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Gagan Gupta\Documents\SCU\4 - SCU Senior Year\1 - Fall 2021\COEN 241\COEN241_HM1_Windows>echo "export DOCKER_HOST=tcp://localhost:2375" >> ~/.bashrc && source ~/.bashrc
The system cannot find the path specified.

C:\Users\Gagan Gupta\Documents\SCU\4 - SCU Senior Year\1 - Fall 2021\COEN 241\COEN241_HM1_Windows>docker run --name sysbench --memory="2g" --cpus 1.0 --privileged -itd csminpp/ubuntu-sysbench ba
sh
48358a616475ba260839a36ce9e34a1dc8f0ba3e2ae22eda2e76a44898bc67b5

C:\Users\Gagan Gupta\Documents\SCU\4 - SCU Senior Year\1 - Fall 2021\COEN 241\COEN241_HM1_Windows>docker run --name sysbench --memory="2g" --cpus 1.0 --privileged -it csminpp/ubuntu-sysbench bas
h
root@11ff90b4db50:/# sysbench --test=cpu --cpu-max-prime=40000 run
sysbench 0.4.12: multi-threaded system evaluation benchmark

Running the test with following options:
Number of threads: 1

Doing CPU performance benchmark

Threads started!
Done.

Maximum prime number checked in CPU test: 40000

Test execution summary:
total time:                40.2394s
total number of events:    10000
total time taken by event execution: 40.2379
per-request statistics:
  min:                    3.95ms
  avg:                    4.02ms
  max:                    5.14ms
  approx. 95 percentile:  4.15ms

Threads fairness:
  events (avg/stddev):    10000.0000/0.00
  execution time (avg/stddev): 40.2379/0.00

root@11ff90b4db50:/# sysbench --test=cpu --cpu-max-prime=10000 run
sysbench 0.4.12: multi-threaded system evaluation benchmark

```

```

root@2e32de4754ea: /home
Setting up usbutils (1:007-2ubuntu1.1) ...
Processing triggers for libc-bin (2.19-0ubuntu6.6) ...
root@2e32de4754ea:/home# sudo apt install nano
Reading package lists... Done
Building dependency tree
Reading state information... Done
Suggested packages:
  spell
The following NEW packages will be installed:
  nano
0 upgraded, 1 newly installed, 0 to remove and 131 not upgraded.
Need to get 194 kB of archives.
After this operation, 614 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu/ trusty/main nano amd64 2.2.6-1ubuntu1 [194 kB]
Fetched 194 kB in 0s (211 kB/s)
Selecting previously unselected package nano.
(Reading database ... 17434 files and directories currently installed.)
Preparing to unpack .../nano.2.2.6-1ubuntu1_amd64.deb ...
Unpacking nano (2.2.6-1ubuntu1) ...
Setting up nano (2.2.6-1ubuntu1) ...
update-alternatives: using /bin/nano to provide /usr/bin/editor (editor) in auto mode
update-alternatives: using /bin/nano to provide /usr/bin/pico (pico) in auto mode
root@2e32de4754ea:/home# touch script_Docker.sh
root@2e32de4754ea:/home# nano script_Docker.sh
root@2e32de4754ea:/home# bash script_Docker.sh test_log_docker.txt
Architecture:      x86_64
CPU op-mode(s):    32-bit, 64-bit
Byte Order:        Little Endian
CPU(s):            20
On-line CPU(s) list: 0-19
Thread(s) per core: 2
Core(s) per socket: 10
Socket(s):         1
Vendor ID:         GenuineIntel
CPU family:        6
Model:             165
Stepping:          5
CPU MHz:           3600.010
BogoMIPS:          7200.02
Hypervisor vendor: Microsoft
Virtualization type: full
L1d cache:         32K
L1i cache:         32K

```

```

root@2e32de4754ea: /home
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
root@2e32de4754ea:/home# git pull origin master
warning: no common commits
remote: Enumerating objects: 30, done.
remote: Counting objects: 100% (30/30), done.
remote: Compressing objects: 100% (22/22), done.
remote: Total 30 (delta 15), reused 22 (delta 7), pack-reused 0
Unpacking objects: 100% (30/30), done.
From https://github.com/gaganSCU/COEN241_HW1
 * branch      master      -> FETCH_HEAD
 * [new branch] master      -> origin/master
Merge made by the 'recursive' strategy.
 QEMU_sysbench_tests.sh | 45 +++++
 QEMU_test_log.txt       | 383 +++++
 QEMU_test_log_config1.txt | 390 +++++
 QEMU_test_log_config2.txt | 390 +++++
 QEMU_test_log_config3.txt | 390 +++++
 sysbench_cpu_fileio_test.sh | 37 +++++
 test_log.txt           | 383 +++++
 7 files changed, 2018 insertions(+)
 create mode 100644 QEMU_sysbench_tests.sh
 create mode 100644 QEMU_test_log.txt
 create mode 100644 QEMU_test_log_config1.txt
 create mode 100644 QEMU_test_log_config2.txt
 create mode 100644 QEMU_test_log_config3.txt
 create mode 100644 sysbench_cpu_fileio_test.sh
 create mode 100644 test_log.txt
root@2e32de4754ea:/home# git push origin master
Username for 'https://github.com': gaganSCU
Password for 'https://gaganSCU@github.com':
Counting objects: 7, done.
Delta compression using up to 20 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 2.42 KiB | 0 bytes/s, done.
Total 6 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
To https://github.com/gaganSCU/COEN241_HW1
 c44de64..37580e9 master -> master
root@2e32de4754ea:/home#

```

Data/Measurements:

The script (pasted later in this document) filtered and fed the data of each config into a .txt file from which I moved the data to excel. Here are the tables:

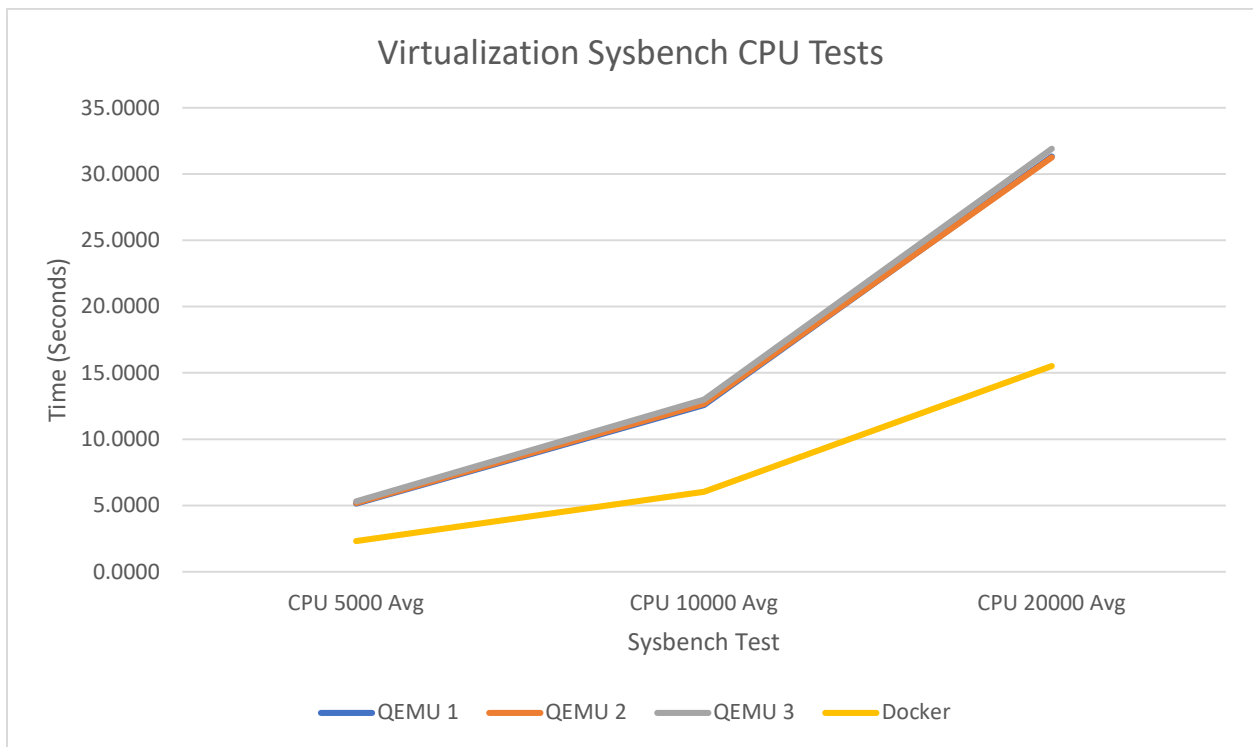
	QEMU 1	QEMU 2	QEMU 3	Docker
CPU 5000 Run 1	5.1210	5.2257	5.2136	2.3276
CPU 5000 Run 2	5.1339	5.2020	5.3371	2.3304
CPU 5000 Run 3	5.1939	5.2034	5.4107	2.3239
CPU 5000 Run 4	5.1406	5.1651	5.4265	2.3222
CPU 5000 Run 5	5.1565	5.1703	5.2203	2.3265
CPU 5000 Avg	5.1492	5.1933	5.3216	2.3261
CPU 10000 Run 1	12.5998	12.7124	12.7133	6.0213
CPU 10000 Run 2	12.6058	12.6202	13.1793	6.0066
CPU 10000 Run 3	12.5690	12.6644	12.7381	6.0075
CPU 10000 Run 4	12.5482	12.6476	13.2559	6.0225
CPU 10000 Run 5	12.5829	12.7417	13.0310	6.0527
CPU 10000 Avg	12.5811	12.6773	12.9835	6.0221
CPU 20000 Run 1	31.6840	31.2162	32.1740	15.5328
CPU 20000 Run 2	31.3436	31.2764	31.2879	15.5063
CPU 20000 Run 3	31.3811	31.3725	32.0950	15.5198
CPU 20000 Run 4	31.2209	31.1514	31.4477	15.5339
CPU 20000 Run 5	31.0576	31.2730	32.5930	15.5279
CPU 20000 Avg	31.3374	31.2579	31.9195	15.5241
FileIO 2G Run 1	6.7332	11.4627	0.6213	0.0669
FileIO 2G Run 2	8.8622	12.7997	0.4923	0.0798
FileIO 2G Run 3	9.9212	16.7610	1.0974	0.0816
FileIO 2G Run 4	12.3063	17.6150	0.7046	0.0775
FileIO 2G Run 5	12.1168	18.4183	0.8099	0.0838
FileIO 2G Avg	9.9879	15.4113	0.7451	0.0779
FileIO 4G Run 1	27.1652	40.4482	0.9527	0.4138
FileIO 4G Run 2	28.0483	40.7813	0.5343	0.4103
FileIO 4G Run 3	31.8298	40.1434	0.7297	0.4175
FileIO 4G Run 4	31.8566	43.0382	0.5430	0.4151
FileIO 4G Run 5	32.6931	44.1430	0.6336	0.4267
FileIO 4G Avg	30.3186	41.7108	0.6787	0.4167

Analysis and Discussion of the Results:

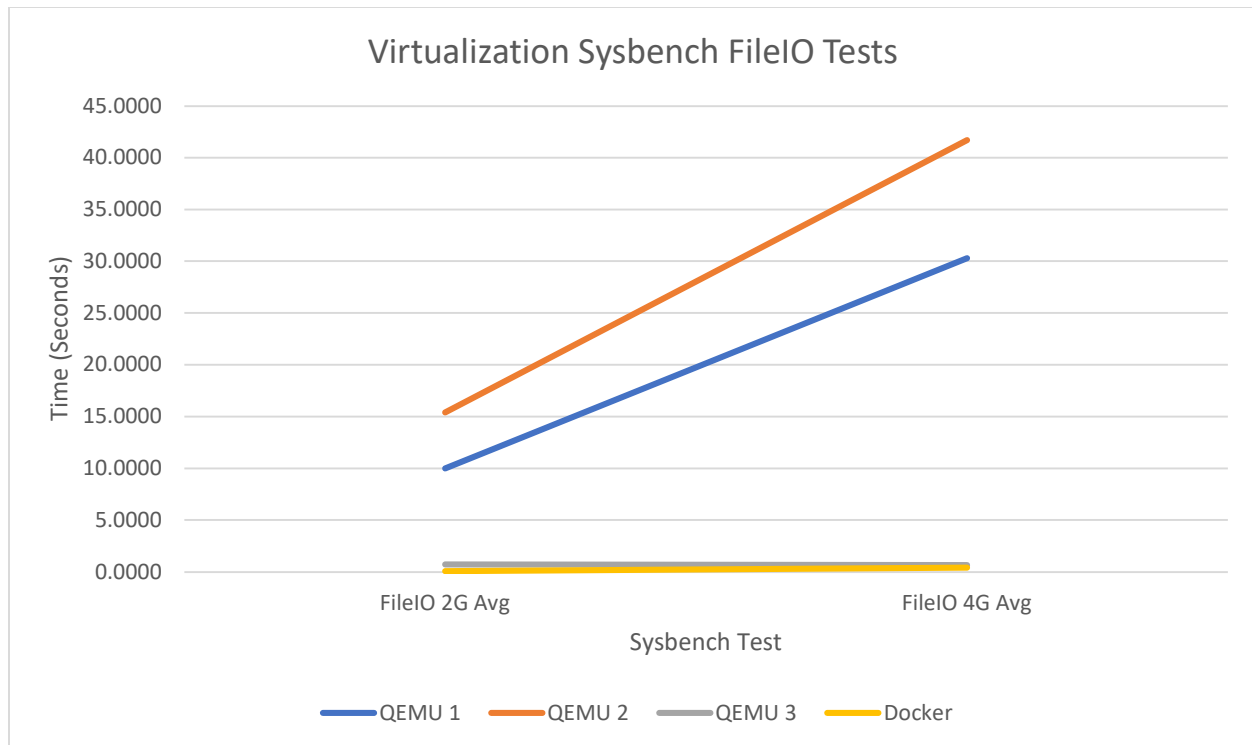
Averages of all the runs:

	QEMU 1	QEMU 2	QEMU 3	Docker
CPU 5000 Avg	5.1492	5.1933	5.3216	2.3261
CPU 10000 Avg	12.5811	12.6773	12.9835	6.0221
CPU 20000 Avg	31.3374	31.2579	31.9195	15.5241
FileIO 2G Avg	9.9879	15.4113	0.7451	0.0779
FileIO 4G Avg	30.3186	41.7108	0.6787	0.4167

If we plot all the averages for the CPU, we get the following:



This trend doesn't make sense for the QEMU lines as QEMU 1 had 1 core, QEMU 2 had 2 cores, and QEMU 3 had 4 cores. The lines should space out relatively proportional to this however this isn't seen in the slightest.



The weird trends with QEMU continued into the FileIO tests as well as 1 was faster than 2 and 3 was unreasonably fast compared to the other two QEMU FileIO tests.

Even after running `lscpu` and `lshw` to check the CPU and memory stats in the VM and confirming that they were correct for the QEMU configs, the performance did not show this. I am unsure why QEMU behaved like this even after all the sysbench commands and `dropcache`'s worked correctly. Docker had many issues with getting the configuration correct as the core count and the memory amount were always identical to the base system and never were affected by the altering of flags like `-memory` and `-cpu` or even adjusting the `.wslconf` file for WSL 2 windows 10. This is why docker was so much faster than the other QEMU tests as it was running with 20 cpus and 24GiB of memory. All these mishaps together mean that there are no solid findings that can be represented however with a bit more time for debugging the resource allocation issues this would work.

Script for running experiments:

This script worked in both setups of Linux (the VM and the container) but it requires the installation of packages/commands like `lshw`, `lscpu`, and `sysbench`. This script takes in one command line argument of output file name (e.g. `bash script.sh output.txt`).

```
outputFile=$1
```

```
sync; sudo sh -c "echo 3 > /proc/sys/vm/drop_caches"
```



```
touch $outputFile
> $outputFile
runCPU(){
    number=$1
    shift
    for i in {1..5}; do
        sysbench --test=cpu --cpu-max-prime=$number run | tail
-n 2 | head -n 1 >> $outputFile
    done
}

runFileIO(){
    size=$1
    mode=$2
    shift
    for i in {1..5}; do
        sysbench --num-threads=16 --test=fileio --file-total-
size=$size --file-test-mode=rndrw prepare
        sysbench --num-threads=16 --test=fileio --file-total-
size=$size --file-test-mode=rndrw run >> $outputFile
        sysbench --num-threads=16 --test=fileio --file-total-
size=$size --file-test-mode=rndrw cleanup
        sync; sudo sh -c "echo 3 > /proc/sys/vm/drop_caches"
    done
}

echo "CPU Config" >> $outputFile
sudo lscpu | tee >(grep 'CPU(s):' >> $outputFile) >(grep
'Socket' >> $outputFile) >(grep 'Thread' >> $outputFile)
sleep 3
```

```

echo "RAM Amount" >> $outputFile
sudo lshw -c memory | grep capacity >> $outputFile
echo "---CPU Tests---" >> $outputFile
echo "-----5000" >> $outputFile
runCPU 5000
echo "-----10000" >> $outputFile
runCPU 10000
echo "-----20000" >> $outputFile
runCPU 20000

echo "-----" >>
$outputFile

echo "---File IO Tests---" >> $outputFile
echo "-----2G 128Files 16Threads" >> $outputFile
runFileIO 2G
echo "-----4G 128Files 16Threads" >> $outputFile
runFileIO 4G

```

Usage of performance tools:

The tool we used to collect and show performance data of a virtualized space is the sysbench tool for the Linux environment.

- The sysbench commands used
 - sysbench --test=cpu [FLAGS] run
 - --cpu-max-prime=N
 - Prime generator test for cpu
 - sysbench --test=fileio [FLAGS] [prepare|run|cleanup]
 - --file-num=N
 - Number of files, default amount is 128
 - --file-total-size=SIZE
 - Size of all files combined
 - --file-test-mode=STRING

- Type of FileIO test: seqwr, seqrewr, seqrd, rndrd, rndwr, or rndrw

If the resource allocation of both environments were the same then much could be said about how CPU utilization differs in the user-space vs the kernel space however due to the erratic nature of my configurations we cannot make these connections. The fileIO throughput can be seen in the kb/sec measurement, the latency would be seen if you compare the “total time taken by event execution” of 2 different systems, and the disk utilization can be calculated by taking the total operations performed and dividing that by the “total time taken by event execution” resulting in a disk operations per second measurement.

Final Thoughts/Feedback:

Setting up virtualization for an assignment should be done in a controlled environment set up by the school where students can follow clean and organized instructions so that they can learn how to setup VMs and containers without having to waste time heavily debugging. Students’ personal machines varied in architecture and OS therefore making the setup process vastly different from everyone else. This stifled peer-to-peer review and analysis of bugs and resulted in very confusing rabbit holes of online solutions. Not many of my configurations worked well and I spent 2-3 days debugging trying to figure out why but most online sources conflict and its hard to relay these kinds of bugs on piazza. All-in-all if this were to be done again, I would highly prefer a controlled environment where I can spend more time learning about different types of virtualization and less time on debugging to no avail.

Github Repo:

https://github.com/gaganSCU/COEN241_HW1