COEN 241 – Cloud Computing – Winter 2023
Professor: Sean Choi
Student: Yangzhang Zhou – 1630152

1. Host Environment and Configuration

Laptop Brand	Legion Y7000-1060
CPU	Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz 2.30 GHz
RAM	8.00 GB (7.85 GB Available)
System Type	64-bit operating system based on x64 processor(Windows 10)
Disk Space	1 TB

Because the operating system I used is windows 10 and I already installed wsl before, I just need to download and install VcXsrv, from https://sourceforge.net/projects/vcxsrv/, to enable using wsl with GUI. In order to get the GUI, I also installed Desktop Environment for your WSL. Here I used XFCE4. And using the following command to access the GUI

```
startfce4
```

2. Qemu and Docker Setup

a. Qemu Setup and VM configuration

First, I downloaded the .iso file from https://releases.ubuntu.com/16.04/ubuntu-16.04.7-server-amd64.iso for the later steps. And then I opened the wsl and installed the Qemu using the below command line.

```
$ sudo apt-get install qemu
```

Then creating the qemu Image by running the following command.

```
$ sudo qemu-img create ubuntu.img 10G -f qcow2
```

Here I met an issue that the terminal will show "qemu-img not found". In order to fix this issue, I installed qemu-system using the following command.

```
# sudo apt-get install qemu-system
```

After I installed qemu-system, I can create qemu image successfully. And then install the VM using the command below (which takes the iso file as a "cdrom" and the qemu image as a "hard disk"):

```
$ sudo qemu-system-x86_64 -hda ubuntu.img -boot d -cdrom
./[UBUNTU_SERVER_ISO_FILE_NAME] -m 2046 -boot strict=on
```

It will pop up a Qemu window and the Ubuntu Installation instruction is here. I just followed the instruction and installed the Ubuntu system.

```
1# sudo apt-get install qemu-system
```

After installed the Ubuntu system successfully, I exited from the Qemu to terminate the install processing. And then using the following code to start Ubuntu.

```
sudo qemu-system-x86_64 -hda ubuntu.img -m 2046 -boot strict=on
```

This command just removed the "-cdrom ./[iso file]" out. In this way, we can boot the Ubuntu system from the image file, instead of starting the Installation instruction again.

b. Docker Container Setup

For windows user, I chose to install Docker Desktop directly from

https://docs.docker.com/desktop/install/windows-install/ .

And then I opened the docker desktop, it will start the docker service as well. Otherwise, you can't use any operations of docker.

First step, we need to pull a base ubuntu image from Docker Hub.

```
D:\scu\Class\23 Winter\COEN 241\hw1\docker>docker pull ubuntu:16.04
16.04: Pulling from library/ubuntu
58690f9b18fc: Pull complete
b51569e7c507: Pull complete
da8ef40b9eca: Pull complete
fb15d46c38dc: Pull complete
Digest: sha256:1f1a2d56de1d604801a9671f301190704c25d604a416f59e03c04f5c6ffee0d6
Status: Downloaded newer image for ubuntu:16.04
docker.io/library/ubuntu:16.04
```

Using this image, I created a testUbuntu container that I will update the apt-get and install sysbench.

```
D:\scu\Class\23 Winter\COEN 241\hw1\docker>docker run -it --name=testUbuntu ubuntu:16.04
root@425aaf8c1d9b:/#
```

Like the screenshot shows, the terminal will jump to the container. This container id is 913136bc518e and name is testUbuntu defined by me.

I updated the apt-get before I installed the sysbench.

```
root@425aaf8c1d9b:/# apt-get update
Get:1 http://security.ubuntu.com/ubuntu xenial-security InRelease [99.8 kB]
Get:2 http://archive.ubuntu.com/ubuntu xenial InRelease [247 kB]
Get:3 http://security.ubuntu.com/ubuntu xenial-security/main amd64 Packages [2051 kB]
Get:4 http://archive.ubuntu.com/ubuntu xenial-updates InRelease [99.8 kB]
Get:5 http://security.ubuntu.com/ubuntu xenial-security/restricted amd64 Packages [15.9 kB]
Get:6 http://security.ubuntu.com/ubuntu xenial-security/universe amd64 Packages [984 kB]
Get:7 http://archive.ubuntu.com/ubuntu xenial-backports InRelease [97.4 kB]
Get:8 http://security.ubuntu.com/ubuntu xenial-security/multiverse amd64 Packages [8820 B]
Get:9 http://archive.ubuntu.com/ubuntu xenial/main amd64 Packages [1558 kB]
Get:10 http://archive.ubuntu.com/ubuntu xenial/restricted amd64 Packages [14.1 kB]
Get:11 http://archive.ubuntu.com/ubuntu xenial/universe amd64 Packages [9827 kB]
Get:12 http://archive.ubuntu.com/ubuntu xenial/multiverse amd64 Packages [176 kB]
Get:13 http://archive.ubuntu.com/ubuntu xenial-updates/main amd64 Packages [2560 kB]
Get:14 http://archive.ubuntu.com/ubuntu xenial-updates/restricted amd64 Packages [16.4 kB]
Get:15 http://archive.ubuntu.com/ubuntu xenial-updates/universe amd64 Packages [1544 kB]
Get:16 http://archive.ubuntu.com/ubuntu xenial-updates/multiverse amd64 Packages [26.2 kB]
Get:17 http://archive.ubuntu.com/ubuntu xenial-backports/main amd64 Packages [10.9 kB]
Get:18 http://archive.ubuntu.com/ubuntu xenial-backports/universe amd64 Packages [12.7 kB]
Fetched 19.3 MB in 4s (4120 kB/s)
Reading package lists... Done
```

```
root@425aaf8c1d9b:/# apt-get install sysbench
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libmysqlclient20 libssl1.0.0 mysql-common
The following NEW packages will be installed:
  libmysqlclient20 libssl1.0.0 mysql-common sysbench
0 upgraded, 4 newly installed, 0 to remove and 0 not upgraded.
Need to get 1839 kB of archives.
After this operation, 7757 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
```

The version is 0.4.12.

```
root@425aaf8c1d9b:/# sysbench --version
sysbench 0.4.12
```

After we setup the container, I built my own image based on this container using the following command:

```
D:\scu\Class\23 Winter\COEN 241\hw1\docker>docker commit -m "Yangzhang's Ubuntu image" -a "yangzhang" testUbuntu yangzhang/ubuntu:v1.0
sha256:8aee72a445c268fa43fb5816ccd71008ec3185bc67a1e0a071da298133353bfb
```

It returned the self-created image's ID:

8aee72a445c268fa43fb5816ccd71008ec3185bc67a1e0a071da298133353bfb

Or we can use the below command to check if we build the image successfully.

```
D:\scu\Class\23 Winter\COEN 241\hw1\docker>docker images
REPOSITORY          TAG       IMAGE ID       CREATED          SIZE
yangzhang/ubuntu    v1.0      8aee72a445c2   38 seconds ago   174MB
ubuntu              16.04     b6f507652425   17 months ago    135MB
```

And I used "docker history" to check the building history.

```
D:\scu\Class\23 Winter\COEN 241\hw1\docker>docker history yangzhang/ubuntu:v1.0
IMAGE          CREATED             CREATED BY                                      SIZE      COMMENT
8aee72a445c2   About a minute ago  /bin/bash                                       39.6MB    Yangzhang's Ubuntu image
b6f507652425   17 months ago       /bin/sh -c #(nop)  CMD ["/bin/bash"]            0B
<missing>      17 months ago       /bin/sh -c mkdir -p /run/systemd && echo 'do…   7B
<missing>      17 months ago       /bin/sh -c rm -rf /var/lib/apt/lists/*          0B
<missing>      17 months ago       /bin/sh -c set -xe   && echo '#!/bin/sh' > /…   745B
<missing>      17 months ago       /bin/sh -c #(nop) ADD file:11b425d4c08e81a3e…   135MB
```

If we use this image to create container, we can see the sysbench will be installed.

```
D:\scu\Class\23 Winter\COEN 241\hw1\docker>docker run -it --name=my_ubuntu yangzhang/ubuntu:v1.0
root@36ed73bf65e2:/# sysbench --version
sysbench 0.4.12
```

There are some useful operations:

(1) Check containers

```
D:\scu\Class\23 Winter\COEN 241\hw1\docker>docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES

D:\scu\Class\23 Winter\COEN 241\hw1\docker>docker ps -a
CONTAINER ID   IMAGE                COMMAND       CREATED              STATUS                       PORTS     NAMES
36ed73bf65e2   yangzhang/ubuntu:v1.0   "/bin/bash"   About a minute ago   Exited (127) 9 seconds ago             my_ubuntu
425aaf8c1d9b   ubuntu:16.04         "/bin/bash"   7 minutes ago        Exited (0) 4 minutes ago               testUbuntu
```

If we just use “docker ps”, it will just show the running container, so we add “-a” to show all containers.

(2) Check the log of the specific container

```
D:\scu\Class\23 Winter\COEN 241\hw1\docker>docker logs -f 36ed73bf65e2
root@36ed73bf65e2:/# sysbench --version
sysbench 0.4.12
root@36ed73bf65e2:/# eixt
bash: eixt: command not found
root@36ed73bf65e2:/# exit
exit
```

(3) Start, Restart, Stop, Rejoin, and Remove

```
docker start my_ubuntu
docker restart my_ubuntu
docker stop my_ubuntu
D:\scu\Class\23 Winter\COEN 241\hw1\docker>docker attach my_ubuntu
root@36ed73bf65e2:/#
docker rm my_ubuntu
```

3. Experiment

Proof of experiment:

QEMU:

```
root@LAPTOP-3NBFHMP9:/mnt/d/scu/Class/23 Winter/COEN 241/hw1# sudo qemu-system-x
86_64 -m 2048 -boot d -cpu EPYC -smp cores=1,threads=2 -nic user,hostfwd=tcp::22
22-:22 -boot strict=on -hda ubuntu.img
```

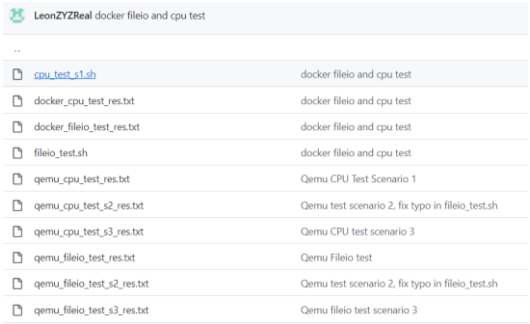
```
yangzhang@ubuntu:~/coen241/COEN_241_HW/hw1$ sysbench --test=cpu --cpu-max-prime=10000 --max-time=30
run
sysbench 0.4.12:  multi-threaded system evaluation benchmark

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

Doing CPU performance benchmark

Threads started!
Done.

Maximum prime number checked in CPU test: 10000


Test execution summary:
    total time:                          17.2935s
    total number of events:              10000
    total time taken by event execution: 17.2208
    per-request statistics:
         min:                                  1.56ms
         avg:                                  1.72ms
         max:                                  4.27ms
         approx.  95 percentile:               2.02ms

Threads fairness:
    events (avg/stddev):           10000.0000/0.00
    execution time (avg/stddev):   17.2208/0.00
```

Docker:

```
# sysbench --test=cpu --cpu-max-prime=20000 --max-time=30 ru
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: 20000


Test execution summary:
    total time:                          19.4316s
    total number of events:              10000
    total time taken by event execution: 19.4294
    per-request statistics:
         min:                                  1.86ms
         avg:                                  1.94ms
         max:                                  7.71ms
         approx.  95 percentile:               2.16ms

Threads fairness:
    events (avg/stddev):           10000.0000/0.00
    execution time (avg/stddev):   19.4294/0.00

# 
```

RAM 2.32 GB CPU 12.97% Connected to Hub

For other experiment results, I put all them on my git repository.



LeonZYZReal docker fileio and cpu test

..	
cpu_test_s1.sh	docker fileio and cpu test
docker_cpu_test_res.txt	docker fileio and cpu test
docker_fileio_test_res.txt	docker fileio and cpu test
fileio_test.sh	docker fileio and cpu test
qemu_cpu_test_res.txt	Qemu CPU Test Scenario 1
qemu_cpu_test_s2_res.txt	Qemu test scenario 2, fix typo in fileio_test.sh
qemu_cpu_test_s3_res.txt	Qemu CPU test scenario 3
qemu_fileio_test_res.txt	Qemu Fileio test
qemu_fileio_test_s2_res.txt	Qemu test scenario 2, fix typo in fileio_test.sh
qemu_fileio_test_s3_res.txt	Qemu fileio test scenario 3

Experiment Scenario

1. Environment

I set three scenarios for QEMU and normal docker container:

```
# sudo qemu-system-x86_64 -m 2048 -boot d -cpu EPYC -smp cores=1,threads=2 -boot strict=on -hda ubuntu.img
```

As the above command shows, it will create a virtual machine with 2G memory. And the virtual machine will have 1 core per die and 2 threads per core. The same as below command.

```
sudo qemu-system-x86_64 -m 2048 -boot d -cpu EPYC -smp cores=2,threads=4 -boot strict=on -hda ubuntu.img
```

```
sudo qemu-system-x86_64 -m 2048 -boot d -cpu EPYC -smp cores=4,threads=4 -boot strict=on -hda ubuntu.img
```

For the below command to create a container, the docker image I used is created by myself, it contains the sysbench and git installed. And the ubuntu version is the same as the virtual machine I used so that I will have the same sysbench version for virtual machine and container.

```
>docker run --rm -it --name=temp yangzhang/ubuntu:v2.0
```

1. CPU Testing

I used three numbers of threads to calculate of prime numbers up to 20000. And the numbers are 2, 4, and 6. For each case, I ran them five times to get the final result.

a. Shell Script

```
#!/bin/bash
threads='2 4 6'
dir=$(pwd)

target=$dir/qemu_cpu_test_res.txt
if [ ! -f $target ]
then
        touch $target
        echo "QEMU cput test result" >> $target
fi

for thread in $threads;
do
        echo "max prime $prime" >> $target
        for i in 1 2 3 4 5;
        do
        echo "iteration $i" >> $target
        sysbench --test=cpu --num-threads=$thread --cpu-max-prime=20000 --max-time=30 run >> $target
        done
done
```

b. Output

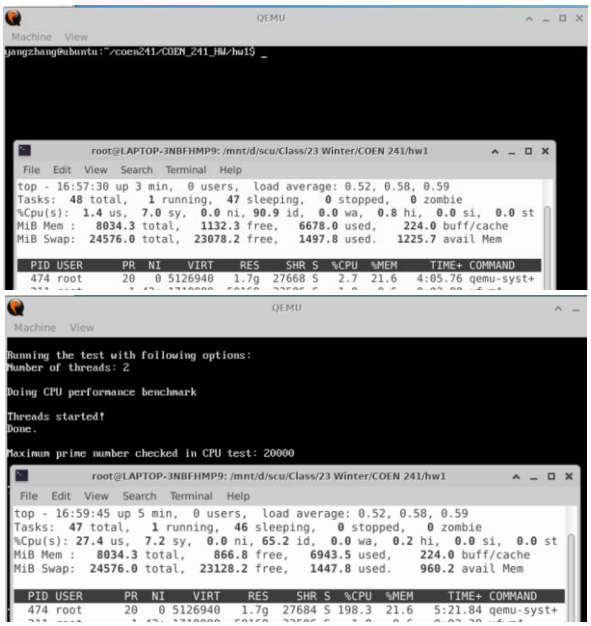
Experiment 1(num_of_threads: 2)				
	QEMU Scenario 1	QEMU Scenario 2	QEMU Scenario 3	Docker
Total time(avg of five tests)	21.33534 s	20.88482 s	20.83416 s	9.9381 s
min	3.822 ms	3.83 ms	3.83 ms	1.86 ms
max	13.648 ms	10.726 ms	11.052 ms	4.874 ms
avg	4.26 ms	4.16 ms	4.16 ms	1.986 ms
Experiment 2(num_of_threads: 4)				
	QEMU Scenario 1	QEMU Scenario 2	QEMU Scenario 3	Docker
Total time(avg of five tests)	20.83158 s	11.29022 s	11.16238 s	5.11654 s
min	3.82 ms	3.39 ms	3.93 ms	1.86 ms
max	23.2 ms	9.278 ms	7.376 ms	6.284 ms
avg	8.31 ms	4.506 ms	4.45 ms	2.45 ms

Experiment 3(num_of_threads: 6)				
	QEMU Scenario 1	QEMU Scenario 2	QEMU Scenario 3	Docker
Total time(avg of five tests)	20.9072 s	9.73472 s	9.56618 s	3.67994 s
min	3.82 ms	3.97 ms	3.922 ms	1.91 ms
max	40.816 ms	11.402 ms	10.66 ms	13.93 ms
avg	12.51 ms	5.828 ms	5.724 ms	2.20 ms

c. Data Analysis

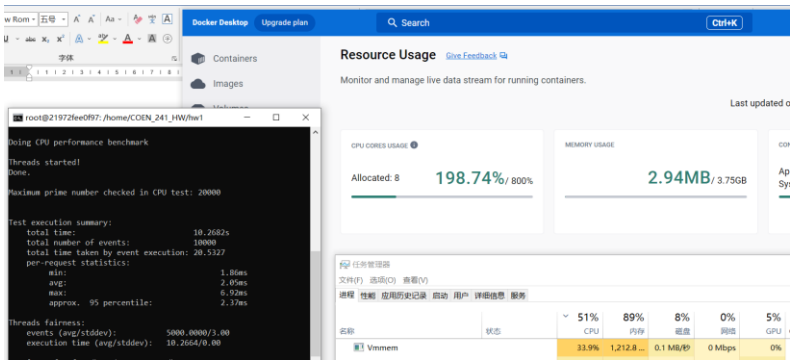
The experiments I set here, they are different with number of threads since I want to check if the number of threads will impact the execution time. The trend based on the data looks almost correct, if we have more and more worker threads, it will be faster. And also based on three different QEMU scenarios, I can find out that the more cores per die and more threads per core, it will run faster. And the performance of docker is obviously better than QEMU. I think it's because the host kernel is shared amongst docker containers.

In order to prove this point, I used some tools to monitor the CPU utilization respectively for kernel and user level. For QEMU, I used "top" command.



We can see the QEMU uses less system level CPU but more user level CPU.

For Docker I used Docker desktop resource usage extension and windows task manager. As we can see, the docker uses kernel level CPU than QEMU.



2. Fileio Test

For fileio testing, I changed the test mode for each experiment, here I used two modes that are seqwr(sequential write) and rndrw(randomly read and write). The same as CPU testing experiment, I ran each experiment five times.

a. Shell Script

```
#!/bin/bash
opers='seqwr rndrw'
dir=$(pwd)

target=$dir/qemu_fileio_test_res.txt
if [ ! -f $target ]
then
        touch $target
        echo "QEMU cput test result" >> $target
fi

for oper in $opers;
do
        echo "max prime $prime" >> $target
        for i in 1 2 3 4 5;
        do
        echo "iteration $i" >> $target
        sysbench --num-threads=16 --test=fileio --file-total-size=3G --file-test-mode=$oper prepare
        sysbench --num-threads=16 --test=fileio --file-total-size=3G --file-test-mode=$oper run >> $target
        sysbench --num-threads=16 --test=fileio --file-total-size=3G --file-test-mode=$oper cleanup
        done
done
```

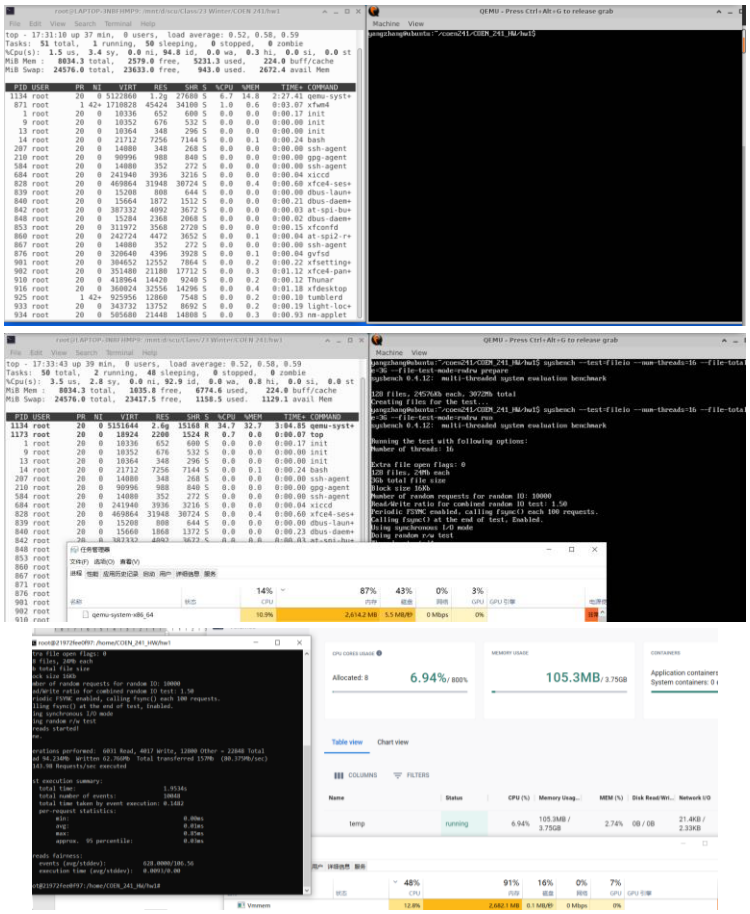
b. Output

Experiment 1(num_of_threads: 16 test_mode: seqwr)				
	QEMU Scenario 1	QEMU Scenario 2	QEMU Scenario 3	Docker
Total time(avg of five tests)	91.70786 s	46.167 s	20.83416 s	8.92602 s
min	0.06 ms	0.07 ms	3.83 ms	0.01 ms
max	4,964.104 ms	1,170.012 ms	11.052 ms	464.482 ms
avg	6.59 ms	3.216 ms	4.16 ms	0.7 ms

Experiment 2(num_of_threads: 16 test_mode: rndrw)				
	QEMU Scenario 1	QEMU Scenario 2	QEMU Scenario 3	Docker
Total time(avg of five tests)	138.7006 s	48.30476 s	44.49018 s	2.29444 s
min	0.02 ms	0.02 ms	0.02 ms	0.00 ms
max	14,899.842 ms	940.884 ms	786.24 ms	25.606 ms
avg	133.52 ms	45.474 ms	45.158 ms	0.06 ms

c. Data Analysis

For these two experiments, I want to see if there any difference when we use sequential writing and randomly reading&writing within different environment. Obviously, the second experiment is slower than previous one since we have one more operation --- reading. However, it's different within docker. And another finding is that docker is still faster than QEMU. The reason is the same as CPU, I think it's because docker is "closer" to physical hardware. I used the same tool to measure their performance.



Git Repository Information:

Username: LeonZYZReal

Repository name: COEN_241_HW

Git Link: https://github.com/LeonZYZReal/COEN_241_HW.git

Contents:

1. Two shell scripts
2. QEMU and Docker outputs
3. Report