Ryan Lam

COEN 241: Cloud Computing

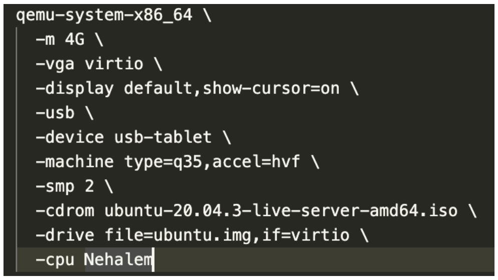
Professor Choi

10/15/21

Assignment 1: QEMU and Docker Benchmark Comparison

<u>Installation Instructions:</u>

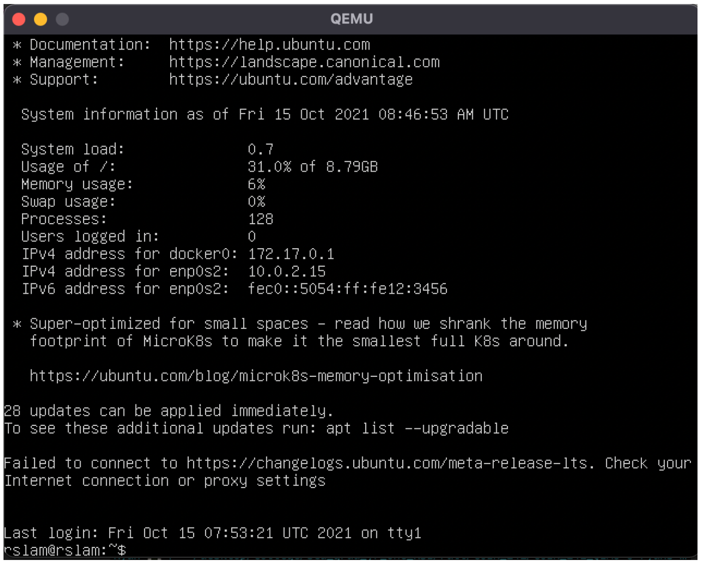
Steps to install QEMU:

1. "brew install qemu"
    - This installs QEMU to the local machine.
2. "qemu-img create ubuntu.img 10G"
    - This step creates an image with 10G of memory that we will use as the drive.
3. 

```
qemu-system-x86_64 \
  -m 4G \
  -vga virtio \
  -display default,show-cursor=on \
  -usb \
  -device usb-tablet \
  -machine type=q35,accel=hvf \
  -smp 2 \
  -cdrom ubuntu-20.04.3-live-server-amd64.iso \
  -drive file=ubuntu.img,if=virtio \
  -cpu Nehalem
```

    - This step boots up the machine with the iso as the cdrom, and the specifications we want the machine to have. The specifications are described in depth below.
4. Follow the instructions to install the OS.

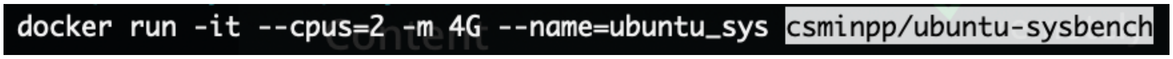
Running QEMU:



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

  System information as of Fri 15 Oct 2021 08:46:53 AM UTC

  System load:              0.7
  Usage of /:               31.0% of 8.79GB
  Memory usage:             6%
  Swap usage:               0%
  Processes:                128
  Users logged in:          0
  IPv4 address for docker0: 172.17.0.1
  IPv4 address for enp0s2:  10.0.2.15
  IPv6 address for enp0s2:  fec0::5054:ff:fe12:3456

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

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

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

Failed to connect to https://changelogs.ubuntu.com/meta-release-lts. Check your
Internet connection or proxy settings


Last login: Fri Oct 15 07:53:21 UTC 2021 on tty1
rslam@rslam:~$
```

Steps to install Docker:

1. Install the docker desktop app from the website.
2. Pull the docker image using “docker pull <image>”
3. Execute docker run to create the container.



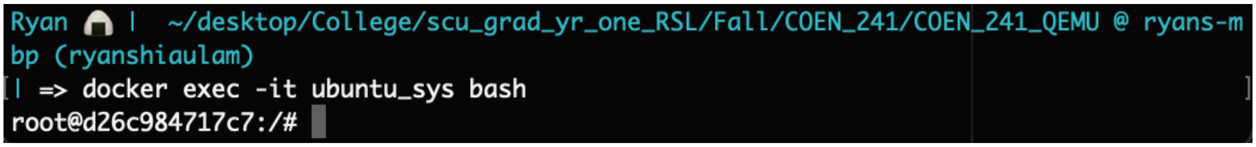
```
docker run -it --cpus=2 -m 4G --name=ubuntu_sys csminpp/ubuntu-sysbench
```

   - In this example we create a docker container with 2 cores and 4G of memory using the csminpp/ubuntu-sysbench image.

4. Run the command: “docker exec -it <name> bash”

   - This command allows the user to interact with the bash interface from the terminal.

Running docker with the bash interface:



```
Ryan 🍙 |  ~/desktop/College/scu_grad_yr_one_RSL/Fall/COEN_241/COEN_241_QEMU @ ryans-m
bp (ryanshiaulam)
[| => docker exec -it ubuntu_sys bash                                               ]
root@d26c984717c7:/#
```

Specifications:

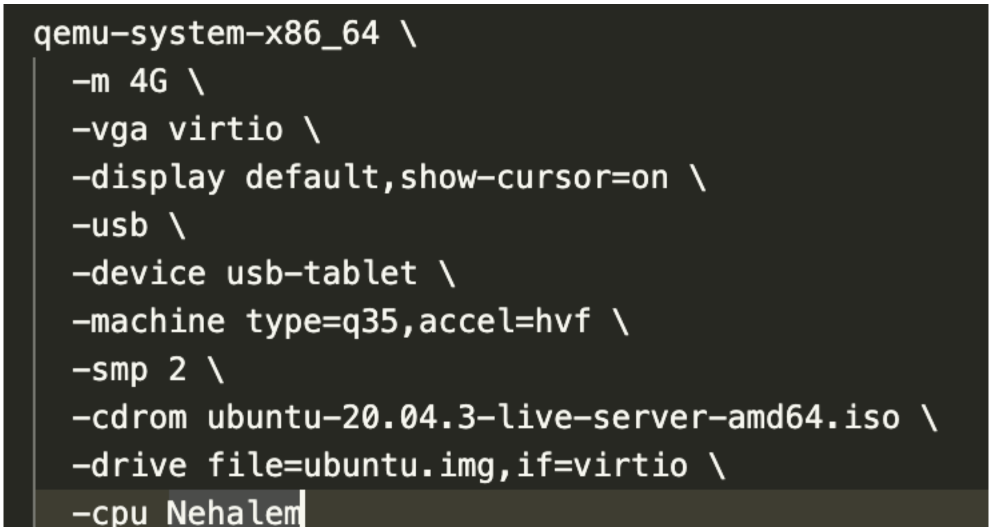
Host Machine Specification:

The host machine I utilized to run the following experiments in the report is an i5 intel macbook pro with 4 cores.

QEMU Specifications:

To begin with, I installed QEMU using "brew install qemu", and created a file named "ubuntu.img", which I allocated 10G worth of memory, to act as the drive. To do this, I entered the command "qemu-img create ubuntu.img 10G".

To boot QEMU, I placed the command in a "start.sh" shell script located in the directory. The exact system specifications are as follows:

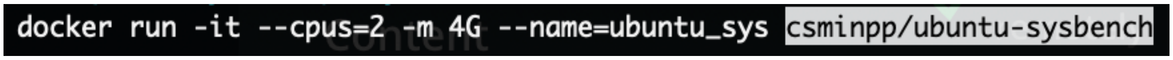


```
qemu-system-x86_64 \
  -m 4G \
  -vga virtio \
  -display default,show-cursor=on \
  -usb \
  -device usb-tablet \
  -machine type=q35,accel=hvf \
  -smp 2 \
  -cdrom ubuntu-20.04.3-live-server-amd64.iso \
  -drive file=ubuntu.img,if=virtio \
  -cpu Nehalem
```

For the machine, I allocated 4G of memory, specified virtio as the display device I used for the virtual graphics array or VGA. Additionally, to assist in simplicity between switching from my host machine to the emulator, I enabled show-cursor to allow me to easily traverse between the two. As my host machine is a mac, I utilized hvf as my accelerator, because from my understanding, as it is the native framework, it allows for passthrough of various hardware and registers, which would be beneficial to the overall performance. Additionally, I allocated the guest the permission of utilizing 2 cores by specifying the number following the "smp" flag. As my drive, I created an ubuntu.img with 10G of memory and utilized the iso provided in the assignment for my cdrom. For the last flag, I specified a Nehalem cpu, because it seemed to most closely mimic my current cpu, from the list provided.

Docker Specifications:

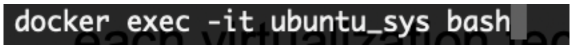
I found it a tad more difficult to find information on docker container specifications, however, I allocated the same number of cores and memory that I provided to QEMU. I figured these were the most necessary attributes to specify, in order to mimic a similar testing environment. To do this, I ran the following command:



```
docker run -it --cpus=2 -m 4G --name=ubuntu_sys csminpp/ubuntu-sysbench
```

This command creates a container that limits the cpu to 2, allocates 4G of memory, names the container to “ubuntu_sys” and specifies the image provided in the assignment, which is a version of ubuntu with sysbench pre-installed.

To access the container with the provided image, I executed the following command:



```
docker exec -it ubuntu_sys bash
```

The command allows me to access the container from a bash terminal.

Sysbench Experiments:

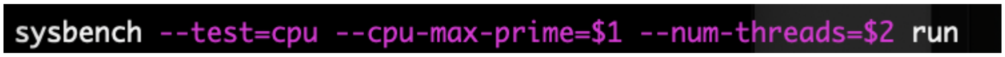
The scripts in the QEMU machine are in the directory: “~/assignment_1/scripts”.

The scripts in the docker container are in the directory: “~/sys_scripts”

For both experiments, the screenshots included below are formatted such that QEMU is located on the left and docker is located on the right.

CPU-Max-Prime:

- For this experiment, I created a shell script named “run_cpu.sh” on both QEMU and docker.
- The command contained in the shell scripts are as follows:



```
sysbench --test=cpu --cpu-max-prime=$1 --num-threads=$2 run
```

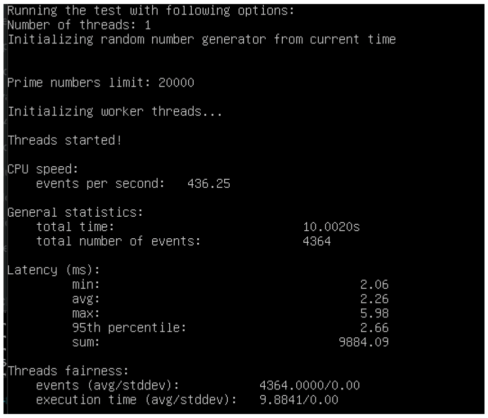
  - This command runs the cpu test and takes in the number of cpu-max-primes, as well as the number of threads.
- The command is executed using “sh run_cpu.sh [# primes] [# threads]”.

As the experiments progress, I increase the prime numbers to increase the load. Additionally, I ran all the experiments, except for the last one, with both 1 and 2 threads. Considering the last experiment has a load of 1000000, I only ran the experiment with 2 threads because 1 thread would take too long. The reason behind my decision to conduct the experiment as so is since it allows me to gauge the performance for single threaded and multithreaded processes using different number of workloads.

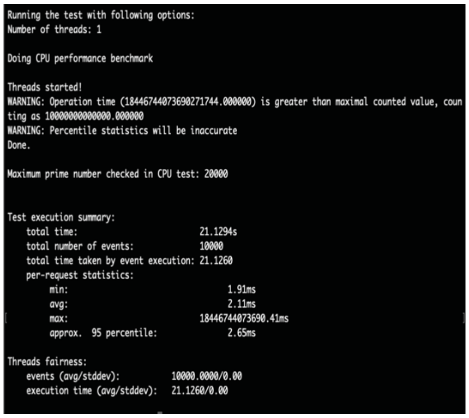
Results:

(cpu-max-prime: 20000, 1 thread)

	QEMU	Docker
Avg:	2.26 ms	1.91 ms
Min:	2.06 ms	2.11 ms
Max:	5.98 ms	18336744073690.4 ms
STD:	9.8841 ms	21.1260 ms



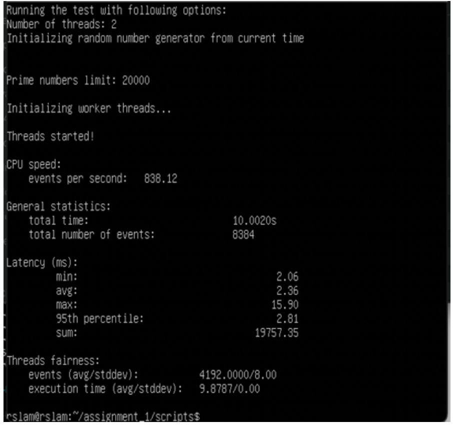
```
Running the test with following options:
Number of threads: 1
Initializing random number generator from current time


Prime numbers limit: 20000

Initializing worker threads...

Threads started!

CPU speed:
    events per second:   436.25

General statistics:
    total time:                          10.0020s
    total number of events:              4364

Latency (ms):
         min:                                    2.06
         avg:                                    2.26
         max:                                    5.98
         95th percentile:                        2.66
         sum:                                 9884.09

Threads fairness:
    events (avg/stddev):           4364.0000/0.00
    execution time (avg/stddev):   9.8841/0.00
```



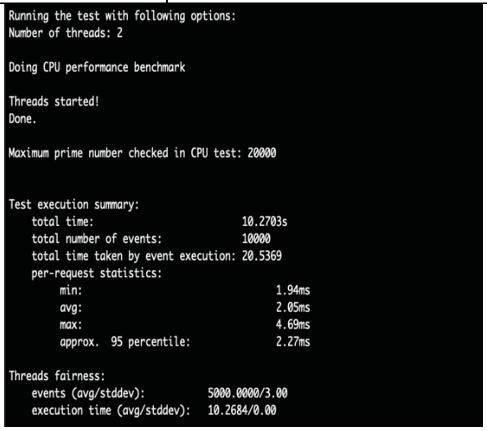
```
Running the test with following options:
Number of threads: 1

Doing CPU performance benchmark

Threads started!
WARNING: Operation time (18446744073690271744.000000) is greater than maximal counted value, coun
ting as 10000000000000.000000
WARNING: Percentile statistics will be inaccurate
Done.

Maximum prime number checked in CPU test: 20000


Test execution summary:
    total time:                          21.1294s
    total number of events:              10000
    total time taken by event execution: 21.1260
    per-request statistics:
         min:                                  1.91ms
         avg:                                  2.11ms
         max:                            18446744073690.41ms
         approx.  95 percentile:               2.65ms

Threads fairness:
    events (avg/stddev):           10000.0000/0.00
    execution time (avg/stddev):   21.1260/0.00
```

(cpu-max-prime: 20000, 2 thread)

	QEMU	**Docker**
Avg:	2.36 ms	2.05 ms
Min:	2.06 ms	1.94 ms
Max:	15.90 ms	4.69 ms
STD:	9.8787 ms	10.2684 ms



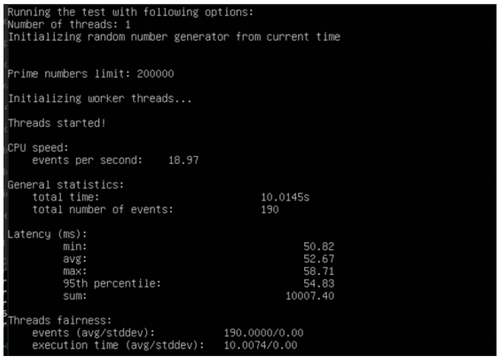
```
Running the test with following options:
Number of threads: 2
Initializing random number generator from current time


Prime numbers limit: 20000

Initializing worker threads...

Threads started!

CPU speed:
    events per second:   838.12

General statistics:
    total time:                          10.0020s
    total number of events:              8384

Latency (ms):
         min:                                    2.06
         avg:                                    2.36
         max:                                   15.90
         95th percentile:                        2.81
         sum:                                19757.35

Threads fairness:
    events (avg/stddev):           4192.0000/8.00
    execution time (avg/stddev):   9.8787/0.00

rslam@rslam:~/assignment_1/scripts$
```



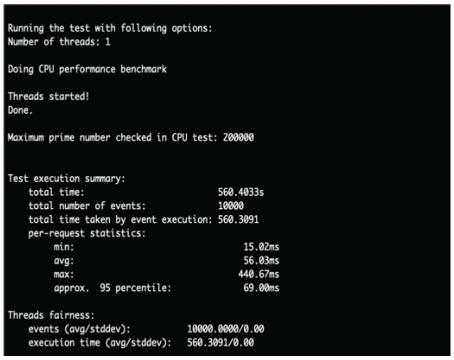
```
Running the test with following options:
Number of threads: 2

Doing CPU performance benchmark

Threads started!
Done.

Maximum prime number checked in CPU test: 20000


Test execution summary:
    total time:                          10.2703s
    total number of events:              10000
    total time taken by event execution: 20.5369
    per-request statistics:
         min:                                  1.94ms
         avg:                                  2.05ms
         max:                                  4.69ms
         approx.  95 percentile:               2.27ms

Threads fairness:
    events (avg/stddev):           5000.0000/3.00
    execution time (avg/stddev):   10.2684/0.00
```

(cpu-max-prime: 20000, 1 thread)

	QEMU	**Docker**
Avg:	52.67 ms	56.03 ms
Min:	50.82 ms	15.02 ms
Max:	58.71 ms	440.67 ms
STD:	54.83 ms	560.3091 ms



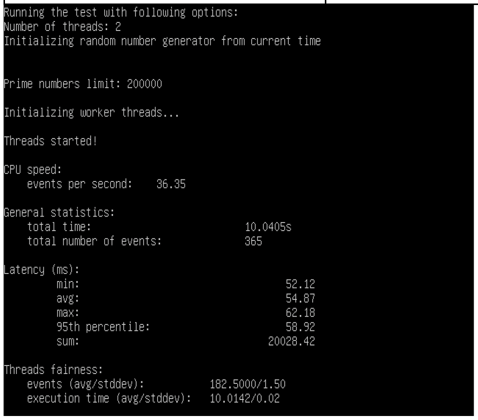
```
Running the test with following options:
Number of threads: 1
Initializing random number generator from current time


Prime numbers limit: 200000

Initializing worker threads...

Threads started!

CPU speed:
    events per second:    18.97

General statistics:
    total time:                          10.0145s
    total number of events:              190

Latency (ms):
         min:                                  50.82
         avg:                                  52.67
         max:                                  58.71
         95th percentile:                      54.83
         sum:                               10007.40

Threads fairness:
    events (avg/stddev):           190.0000/0.00
    execution time (avg/stddev):   10.0074/0.00
```



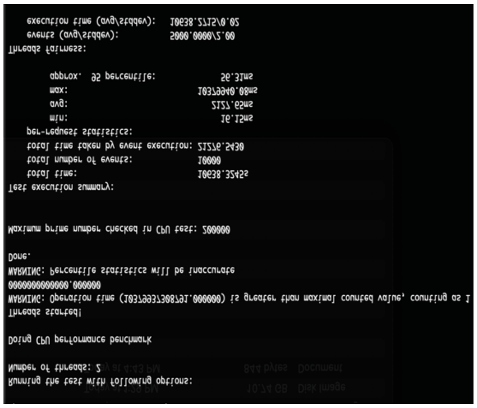
```
Running the test with following options:
Number of threads: 1

Doing CPU performance benchmark

Threads started!
Done.

Maximum prime number checked in CPU test: 200000


Test execution summary:
    total time:                          560.4033s
    total number of events:              10000
    total time taken by event execution: 560.3091
    per-request statistics:
         min:                                  15.02ms
         avg:                                  56.03ms
         max:                                 440.67ms
         approx.  95 percentile:               69.00ms

Threads fairness:
    events (avg/stddev):           10000.0000/0.00
    execution time (avg/stddev):   560.3091/0.00
```

(cpu-max-prime: 200000, 2 thread)

	<u>QEMU</u>	**<u>Docker</u>**
Avg:	54.87 ms	2127.65 ms
Min:	52.12 ms	16.15 ms
Max:	62.18 ms	10379940.08 ms
STD:	10.0142 ms	10638.2715 ms



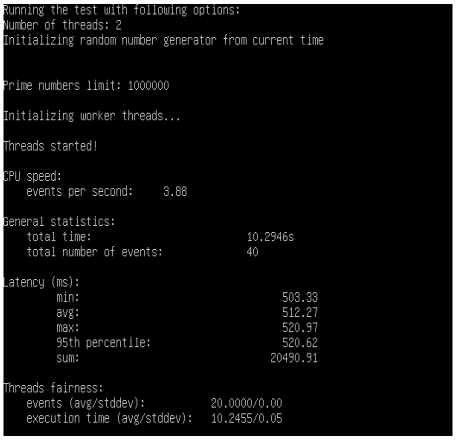
```
Running the test with following options:
Number of threads: 2
Initializing random number generator from current time


Prime numbers limit: 200000

Initializing worker threads...

Threads started!

CPU speed:
    events per second:    36.35

General statistics:
    total time:                          10.0405s
    total number of events:              365

Latency (ms):
         min:                                  52.12
         avg:                                  54.87
         max:                                  62.18
         95th percentile:                      58.92
         sum:                               20028.42

Threads fairness:
    events (avg/stddev):           182.5000/1.50
    execution time (avg/stddev):   10.0142/0.02
```



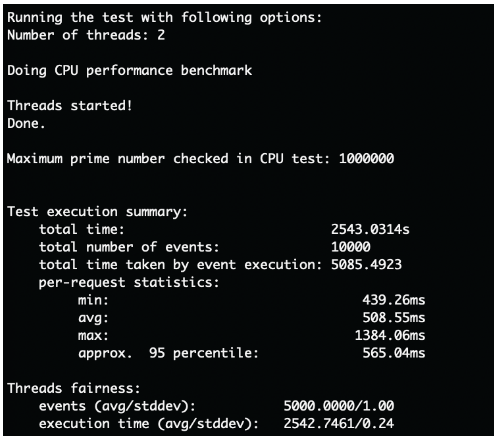
```
Running the test with following options:
Number of threads: 2

Doing CPU performance benchmark

Threads started!
WARNING: Operation time (103799337308791.000000) is greater than maximal counted value, counting as 1
0000000000000.000000
WARNING: Percentile statistics will be inaccurate
Done.

Maximum prime number checked in CPU test: 200000


Test execution summary:
    total time:                          10638.3245s
    total number of events:              10000
    total time taken by event execution: 21276.5430
    per-request statistics:
         min:                                  16.15ms
         avg:                                2127.65ms
         max:                            10379940.08ms
         approx.  95 percentile:               56.31ms

Threads fairness:
    events (avg/stddev):           5000.0000/2.00
    execution time (avg/stddev):   10638.2715/0.02
```

(cpu-max-prime: 1000000, 2 thread)

	<u>QEMU</u>	**<u>Docker</u>**
Avg:	512.27 ms	508.55 ms
Min:	503.33 ms	439.26 ms
Max:	520.97 ms	1384.06 ms
STD:	10.2455 ms	2542.7461 ms



```
Running the test with following options:
Number of threads: 2
Initializing random number generator from current time


Prime numbers limit: 1000000

Initializing worker threads...

Threads started!

CPU speed:
    events per second:     3.88

General statistics:
    total time:                          10.2946s
    total number of events:              40

Latency (ms):
         min:                                  503.33
         avg:                                  512.27
         max:                                  520.97
         95th percentile:                      520.62
         sum:                                20490.91

Threads fairness:
    events (avg/stddev):           20.0000/0.00
    execution time (avg/stddev):   10.2455/0.05
```



```
Running the test with following options:
Number of threads: 2

Doing CPU performance benchmark

Threads started!
Done.

Maximum prime number checked in CPU test: 1000000


Test execution summary:
    total time:                          2543.0314s
    total number of events:              10000
    total time taken by event execution: 5085.4923
    per-request statistics:
         min:                                439.26ms
         avg:                                508.55ms
         max:                               1384.06ms
         approx.  95 percentile:             565.04ms

Threads fairness:
    events (avg/stddev):           5000.0000/1.00
    execution time (avg/stddev):   2542.7461/0.24
```

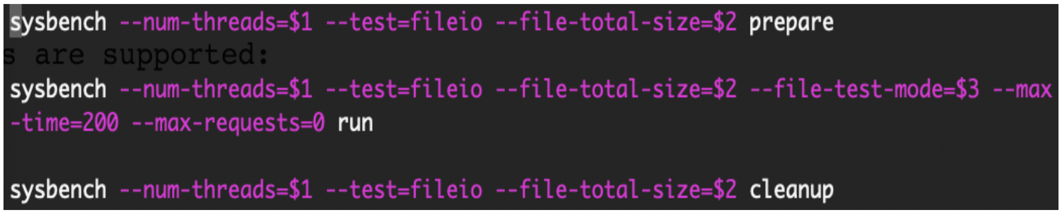
Overall, the results derived from QEMU tended to be more stable, however, the values and efficiency of docker was typically more efficient, as it completed all the tasks. I believe that there were a few outliers in each of the docker runs that ultimately increased the overall average, which can be seen in the std of the execution time. Although I’m not entirely sure why this had happened, as the environments are quite similar, I can only assume it is due to differences in sysbench versions, as I received some warnings on accuracy, which can be seen in the docker screenshots.

Fileio:

For the fileio benchmarks, I ran each test with 5 threads in order to view fileio operations when performed with multithreading, set the total file size to 2G, and gauged the performances on various tests, provided by sysbench. These tests include sequential write, sequential rewrite, random read, random write, and combined random read and write.

I determined and compared the performances by the metrics: Operations performed of reads, writes, and the average latencies. I was unable to find the metric for disk utilization, so, for the sake of this experiment, we will disregard that metrics. Furthermore, I was unable to find throughput for the docker metrics so we also have to disregard that metric, as there isn't sufficient data to compare.

The script I used for both QEMU and docker is named "run_fileio.sh"

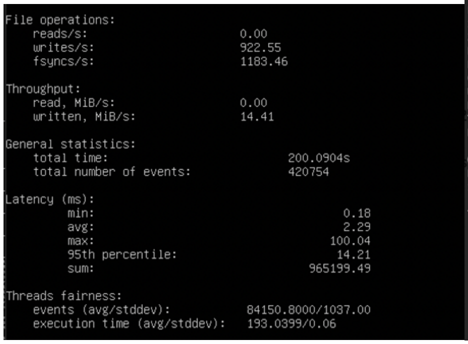


```
sysbench --num-threads=$1 --test=fileio --file-total-size=$2 prepare
s are supported:
sysbench --num-threads=$1 --test=fileio --file-total-size=$2 --file-test-mode=$3 --max
-time=200 --max-requests=0 run

sysbench --num-threads=$1 --test=fileio --file-total-size=$2 cleanup
```

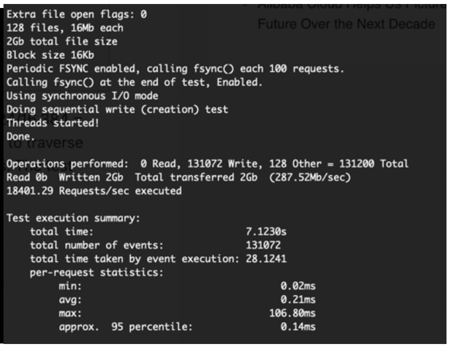
Results:

(Sequential write)

	QEMU	**Docker**
Operations (read)	0b	0b
Operations (write)	922.55	131072
Avg Latency	2.29 ms	.21 ms



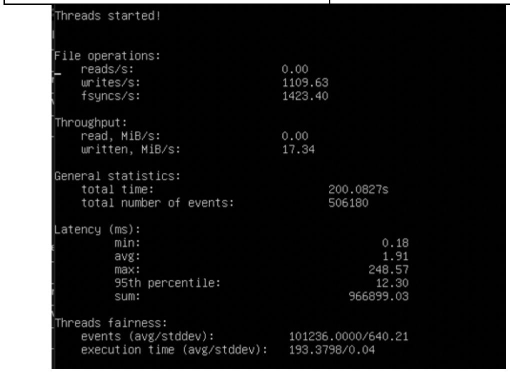
```
File operations:
    reads/s:                      0.00
    writes/s:                     922.55
    fsyncs/s:                     1183.46

Throughput:
    read, MiB/s:                  0.00
    written, MiB/s:               14.41

General statistics:
    total time:                          200.0904s
    total number of events:              420754

Latency (ms):
         min:                                  0.18
         avg:                                  2.29
         max:                                100.04
         95th percentile:                     14.21
         sum:                             965199.49

Threads fairness:
    events (avg/stddev):           84150.8000/1037.00
    execution time (avg/stddev):   193.0399/0.06
```



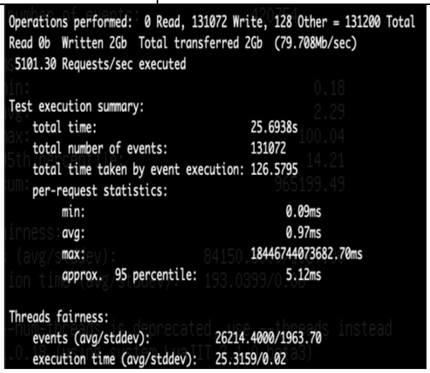
```
Extra file open flags: 0
128 files, 16Mb each
2Gb total file size
Block size 16Kb
Periodic FSYNC enabled, calling fsync() each 100 requests.
Calling fsync() at the end of test, Enabled.
Using synchronous I/O mode
Doing sequential write (creation) test
Threads started!
Done.

Operations performed:  0 Read, 131072 Write, 128 Other = 131200 Total
Read 0b  Written 2Gb  Total transferred 2Gb  (287.52Mb/sec)
18401.29 Requests/sec executed

Test execution summary:
    total time:                          7.1230s
    total number of events:              131072
    total time taken by event execution: 28.1241
    per-request statistics:
         min:                                  0.02ms
         avg:                                  0.21ms
         max:                                106.80ms
         approx.  95 percentile:               0.14ms
```

(Sequential rewrite)

	QEMU	**Docker**
Operations (read)	0b	0b
Operations (write)	1109.63	131072
Avg Latency	1.91 ms	.97 ms



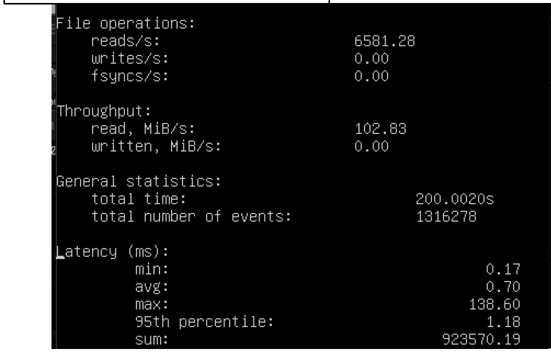
```
Threads started!

File operations:
    reads/s:                      0.00
    writes/s:                     1109.63
    fsyncs/s:                     1423.40

Throughput:
    read, MiB/s:                  0.00
    written, MiB/s:               17.34

General statistics:
    total time:                          200.0827s
    total number of events:              506180

Latency (ms):
         min:                                  0.18
         avg:                                  1.91
         max:                                248.57
         95th percentile:                     12.30
         sum:                             966899.03

Threads fairness:
    events (avg/stddev):           101236.0000/640.21
    execution time (avg/stddev):   193.3798/0.04
```



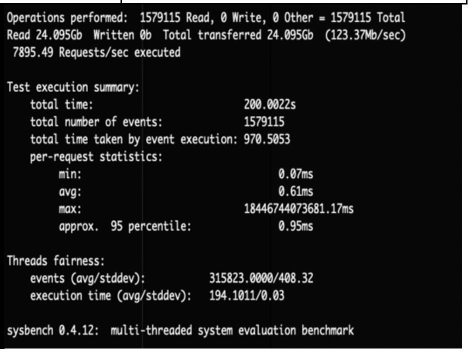
```
Operations performed:  0 Read, 131072 Write, 128 Other = 131200 Total
Read 0b  Written 2Gb  Total transferred 2Gb  (79.708Mb/sec)
 5101.30 Requests/sec executed

Test execution summary:
    total time:                          25.6938s
    total number of events:              131072
    total time taken by event execution: 126.5795
    per-request statistics:
         min:                                  0.09ms
         avg:                                  0.97ms
         max:                                18446744073682.70ms
         approx.  95 percentile:               5.12ms

Threads fairness:
    events (avg/stddev):           26214.4000/1963.70
    execution time (avg/stddev):   25.3159/0.02
```

(Random read)

	QEMU	**Docker**
Operations (read)	6581.28	1579115
Operations (write)	0 b	0 b
Avg Latency	.7 ms	.61 ms



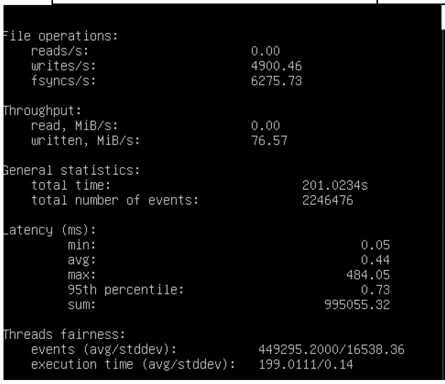
```
File operations:
    reads/s:                      6581.28
    writes/s:                     0.00
    fsyncs/s:                     0.00

Throughput:
    read, MiB/s:                  102.83
    written, MiB/s:               0.00

General statistics:
    total time:                          200.0020s
    total number of events:              1316278

Latency (ms):
         min:                                  0.17
         avg:                                  0.70
         max:                                138.60
         95th percentile:                      1.18
         sum:                             923570.19
```



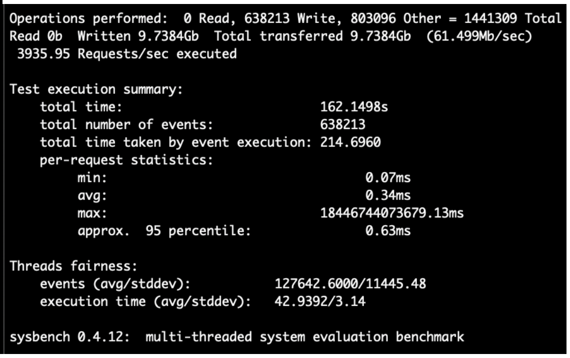
```
Operations performed:  1579115 Read, 0 Write, 0 Other = 1579115 Total
Read 24.095Gb  Written 0b  Total transferred 24.095Gb  (123.37Mb/sec)
 7895.49 Requests/sec executed

Test execution summary:
    total time:                          200.0022s
    total number of events:              1579115
    total time taken by event execution: 970.5053
    per-request statistics:
         min:                                  0.07ms
         avg:                                  0.61ms
         max:                                18446744073681.17ms
         approx.  95 percentile:               0.95ms

Threads fairness:
    events (avg/stddev):           315823.0000/408.32
    execution time (avg/stddev):   194.1011/0.03

sysbench 0.4.12:  multi-threaded system evaluation benchmark
```

(Random write)

	QEMU	**Docker**
Operations (read)	0b	0b
Operations (write)	4900.46	638213
Avg Latency	.44 ms	.34 ms



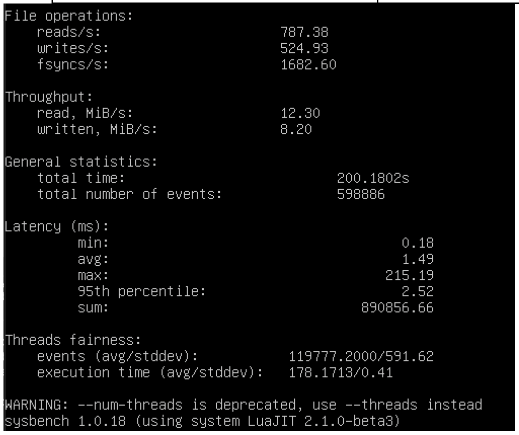
```
File operations:
    reads/s:                      0.00
    writes/s:                     4900.46
    fsyncs/s:                     6275.73

Throughput:
    read, MiB/s:                  0.00
    written, MiB/s:               76.57

General statistics:
    total time:                          201.0234s
    total number of events:              2246476

Latency (ms):
         min:                                    0.05
         avg:                                    0.44
         max:                                  484.05
         95th percentile:                        0.73
         sum:                               995055.32

Threads fairness:
    events (avg/stddev):           449295.2000/16538.36
    execution time (avg/stddev):   199.0111/0.14
```



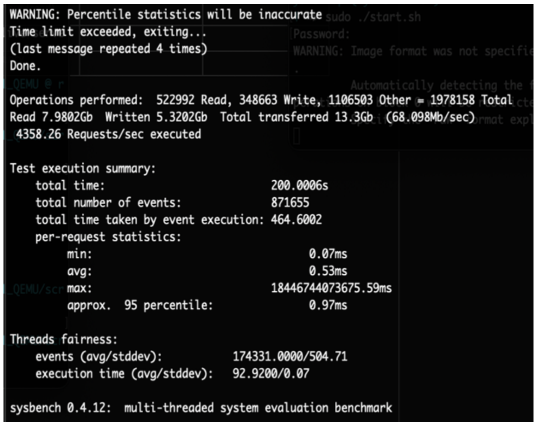
```
Operations performed:  0 Read, 638213 Write, 803096 Other = 1441309 Total
Read 0b  Written 9.7384Gb  Total transferred 9.7384Gb  (61.499Mb/sec)
 3935.95 Requests/sec executed

Test execution summary:
    total time:                          162.1498s
    total number of events:              638213
    total time taken by event execution: 214.6960
    per-request statistics:
         min:                                  0.07ms
         avg:                                  0.34ms
         max:                          18446744073679.13ms
         approx.  95 percentile:               0.63ms

Threads fairness:
    events (avg/stddev):           127642.6000/11445.48
    execution time (avg/stddev):   42.9392/3.14

sysbench 0.4.12:  multi-threaded system evaluation benchmark
```

(Combined random read and write)

	QEMU	**Docker**
Operations (read)	787.38	522992
Operations (write)	524.93	34663
Avg Latency	1.49 ms	.53 ms



```
File operations:
    reads/s:                      787.38
    writes/s:                     524.93
    fsyncs/s:                     1682.60

Throughput:
    read, MiB/s:                  12.30
    written, MiB/s:               8.20

General statistics:
    total time:                          200.1802s
    total number of events:              598886

Latency (ms):
         min:                                    0.18
         avg:                                    1.49
         max:                                  215.19
         95th percentile:                        2.52
         sum:                               890856.66

Threads fairness:
    events (avg/stddev):           119777.2000/591.62
    execution time (avg/stddev):   178.1713/0.41

WARNING: --num-threads is deprecated, use --threads instead
sysbench 1.0.18 (using system LuaJIT 2.1.0-beta3)
```



```
WARNING: Percentile statistics will be inaccurate
Time limit exceeded, exiting...
(last message repeated 4 times)
Done.

Operations performed:  522992 Read, 348663 Write, 1106503 Other = 1978158 Total
Read 7.9802Gb  Written 5.3202Gb  Total transferred 13.3Gb  (68.098Mb/sec)
 4358.26 Requests/sec executed

Test execution summary:
    total time:                          200.0006s
    total number of events:              871655
    total time taken by event execution: 464.6002
    per-request statistics:
         min:                                  0.07ms
         avg:                                  0.53ms
         max:                          18446744073675.59ms
         approx.  95 percentile:               0.97ms

Threads fairness:
    events (avg/stddev):           174331.0000/504.71
    execution time (avg/stddev):   92.9200/0.07

sysbench 0.4.12:  multi-threaded system evaluation benchmark
```

Although we were unable to obtain a comparison for the metrics, throughput and disk utilization, given the findings that the number of operations performed by docker greatly outweigh that of the QEMU operations performed, coupled with the fact that the docker simulated benchmarks typically have an overall lower average latency when performing fileio, we can conclude that the docker container of ubuntu is more efficient than QEMU.

Conclusion:

In this report, we've discussed the installation and utilization processes of both QEMU and docker, the system specifications required to replicate similar experimental environments, and compared the efficiency, measured by the cpu-max-prime tests and the fileio tests, between QEMU and docker. As a result of the testing, conducted by sysbench, the established conclusion is that the utilization of docker containers typically proves to be more efficient than that of QEMU machines.

Extra Credit:

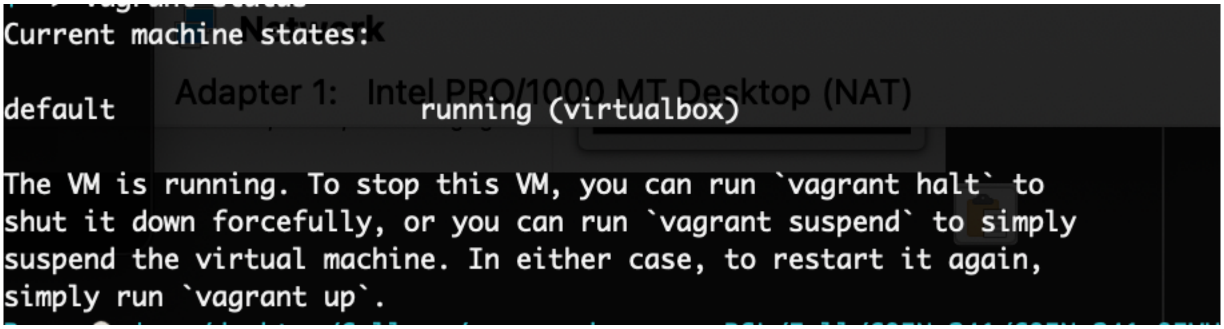
Vagrantfile (located in the main directory of the repository):

- Creates a virtualbox VM that is preconfigured with ubuntu and sysbench, as well as the scripts necessary for running the benchmarks.

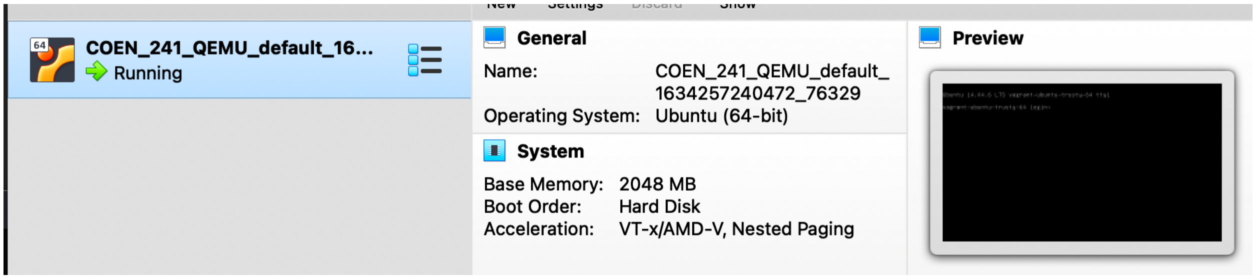
```
# -*- mode: ruby -*-
# vi: set ft=ruby :

VAGRANT_BOX = 'ubuntu/trusty64'

Vagrant.configure("2") do |config|

  # Select the vm box we want to use.
  config.vm.box = VAGRANT_BOX

  # Initialize the provider as virtualbox.
  config.vm.provider "virtualbox" do |vb|

    # Customize the amount of memory on the VM:
    vb.memory = "2048"
  end

  config.vm.provision "shell", inline: <<-SHELL
    # Update apt get to prime installation of sysbench.
    apt-get update -y
    # Install sysbench using apt-get install, when machine is booted up.
    apt-get install sysbench -y

    # Create directory and scripts for sysbench to run as benchmarks later.
    mkdir sysbench_scripts
    cd sysbench_scripts
    echo 'sysbench --test=cpu --cpu-max-prime=$1 --num-threads=$2 run' >> ./run_cpu.sh
    echo "sysbench --num-threads=$1 --test=fileio --file-total-size=3G prepare
    \nsysbench --num-threads=$1 --test=fileio --file-total-size=3G --file-test-mode=rndrw --max-time=300 --max-requests=0 run
    \nsysbench --num-threads=$1 --test=fileio --file-total-size=3G cleanup" >> ./run_fileio.sh

  SHELL
end
```

Steps to creating and using the vagrant file:

1. Execute “vagrant box add ubuntu/trusty64”
   - This command adds the vagrant box to the local machine.
2. Execute “vagrant init ubuntu/trusty64”
   - This command creates the vagrant file.
3. Write the vagrantfile as displayed above.
4. Execute “vagrant up --provisions”
   - Starts the virtual machine in virtual box.
5. Execute “vagrant ssh”
   - Allows the user to interface with the virtual machine through the terminal.



```
Current machine states:

default                   running (virtualbox)

The VM is running. To stop this VM, you can run `vagrant halt` to
shut it down forcefully, or you can run `vagrant suspend` to simply
suspend the virtual machine. In either case, to restart it again,
simply run `vagrant up`.
```



COEN_241_QEMU_default_16...
Running

General

Name: COEN_241_QEMU_default_1634257240472_76329
Operating System: Ubuntu (64-bit)

System

Base Memory: 2048 MB
Boot Order: Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging

Preview

Dockerfile (located in ./docker):

- A preconfigured dockerfile to create containers that allow the replication of the experiments conducted in the report.

Dockerfile:

```
FROM csminpp/ubuntu-sysbench

WORKDIR /app

COPY scripts /app

RUN apt-get update -y

RUN echo "Configuration successful!"

RUN sysbench --version
```

File explanation:

- FROM csminpp/ubuntu-sysbench
  - Creates a dockerfile using the image provided in the assignment.
- WORKDIR /app
  - Creates a directory called “app”.
- COPY scripts /app
  - Copies the run_cpu.sh and run_fileio.sh scripts in the scripts folder to the workdir app.
- RUN apt-get update -y
  - Executes apt-get update -y.
- RUN echo “Configuration successful!”
  - Prints Configuration successful! in the terminal.
- RUN sysbench --version
  - Shows the current version of sysbench installed in the container.

startdocker.sh

```
docker build –t ubuntu_docker –f Dockerfile .

docker image inspect ubuntu_docker

docker run –it ––rm ––name ubuntu_test_dockerfile ubuntu_docker

docker exec –it ubuntu_test_dockerfile bash
```

startdocker.sh is a shell script that invokes all the necessary commands to create a container from the dockerfile image we created above.