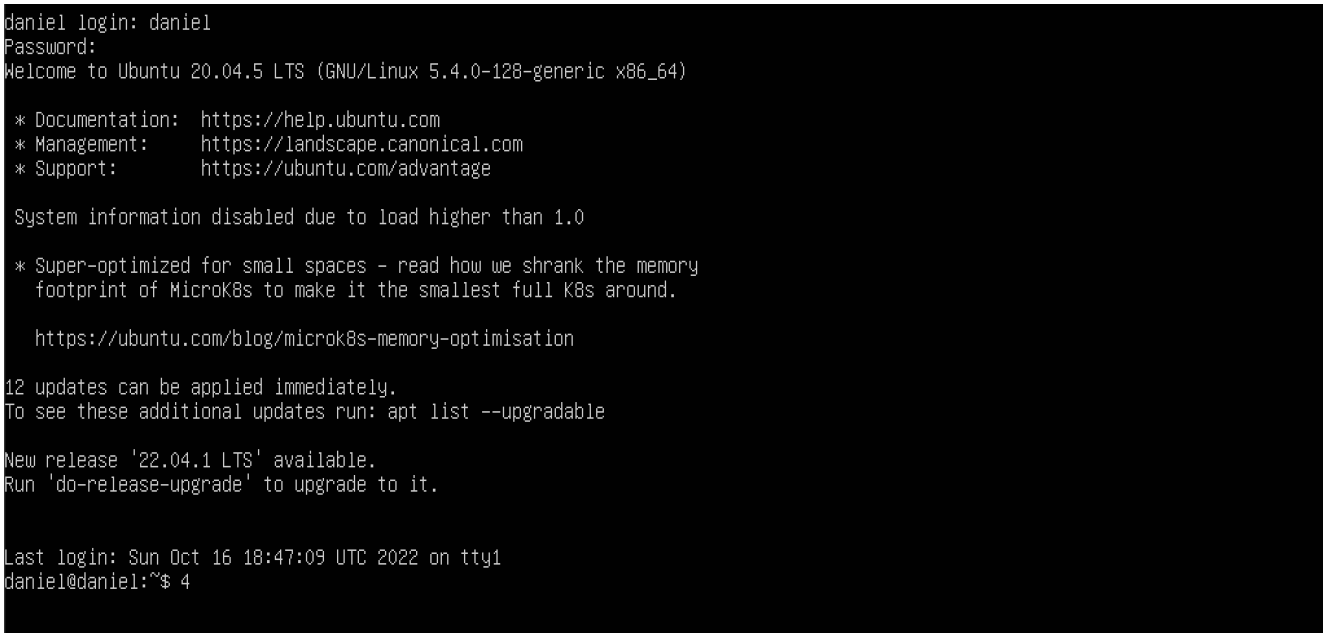
COEN 241 Assignment 1 Report

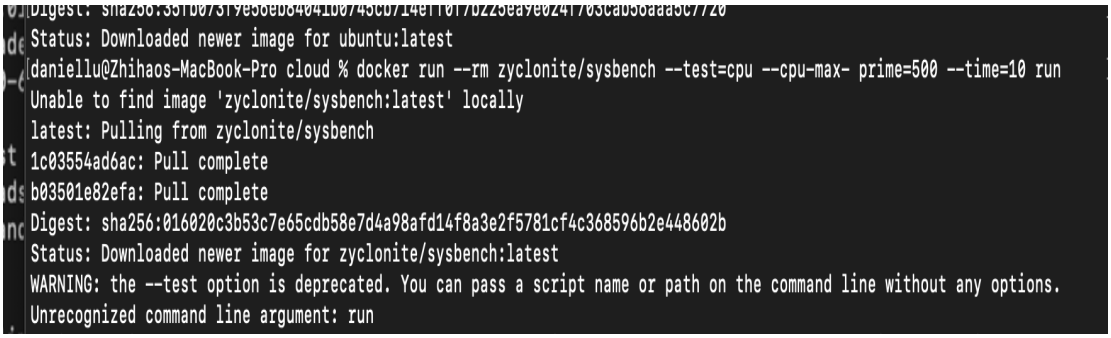
Set up QEMU VM and Sysbench

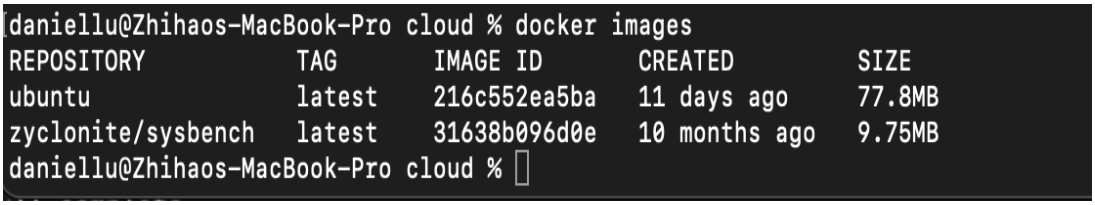
1. Install QEMU
    a. $sudo apt-get install qemu
2. Create QEMU image
    a. $sudo qemu-img create ubuntu.img 10G -f qcow2
3. Install VM
    a. $sudo qemu-system-x86_64 -hda ubuntu.img -boot d -cdrom ./server.iso -m 2046 -boot strict=on
4. Start VM and Run
    a. sudo qemu-system-x86_64 -hda ubuntu.img -boot d -cdrom ./server.iso -m 2046 -boot strict=on
5. Install sysbench
    a. $sudo apt update
    b. $sudo apt install sysbench
6. Screenshot of launching QEM



Set up Docker and Sysbench

1. Download docker desktop
2. Install ubuntu image
   $ docker pull ubuntu:latest
3. Install sysbench
   $ docker run --rm zyclonite/sysbench --test=cpu --cpu-max-prime=500 --time=10 run





4. Some useful docker command:
   a. $docker images: list all images
   b. $docker ps: list all running containers
   c. $docker rm:

Test Case and Scenarios

- I set up the following test cases for my experiment
- Test Case 1: prime number = 30000: sysbench --test=cpu--cpu-max-prime=30000 run
  a. Scenario 1: CPU = 1 & Memory = 1G
  b. Scenario 2: CPU = 2 & Memory = 2G
  c. Scenario 3: CPU = 2 & Memory = 4G
- Test Case 2: prime number = 25000: sysbench --test=cpu --cpu-max-prime=10000 run
  a. Scenario 1: CPU = 1 & Memory = 1G
  b. Scenario 2: CPU = 2 & Memory = 2G
  c. Scenario 3: CPU = 2 & Memory = 4G
- File IO : sysbench –num-threads –test=fileio –file-total-size=2G –file-test-mode=rndrw run
  a. Scenario 1: CPU = 1 & Memory = 1G
  b. Scenario 2: CPU = 2 & Memory = 2G
  c. Scenario 3: CPU = 2 & Memory = 4G

Experiment on QEMU — CPU Experiment:

- Set up different scenarios. Use the following command to set up QEMU configuration where -m represents the memory and -smp represents the total number of CPUs.

  ```
  $ qemu-system-x86_64 -hda ubuntu.img -boot d -m 2048 -smp 2 --accel tcg -boot strict=on
  ```

- I wrote two bash scripts to set up two different prime numbers as test cases and * running the scripts for each QEMU configurations (scenarios)

```
# !/bin/bash
echo "Test case: prime number = 30000"
for value in {1..5}
do
    echo "Try: " $value
    sysbench --test=cpu --cpu-max-prime=30000 --time=10 run
done
```

```
# !/bin/bash
echo "Test case: prime number = 10000"
for value in {1..5}
do
        echo "Try: " $value
        sysbench --test=cpu --cpu-max-prime=10000 --time=10 run
done
```

- **CPU Example Experiment** – Scenario 1 with test case 2:
  - Set up scenarios with parameter -m and -smp: 2G memory and 2 CPU:

    ```
    $ qemu-system-x86_64 -hda ubuntu.img -boot d -m 2048 -smp 2 --accel tcg -boot strict=on
    ```

  - Running test case 2 bash script and writing output to a text file to collect data the content of the output is shown below.

```
Try:  5
sysbench 1.0.18 (using system LuaJIT 2.1.0-beta3)

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


Prime numbers limit: 10000

Initializing worker threads...

Threads started!

CPU speed:
    events per second:   428.76

General statistics:
    total time:                          10.0030s
    total number of events:              4291

Latency (ms):
         min:                                    2.14
         avg:                                    2.32
         max:                                    6.39
         95th percentile:                        2.57
         sum:                                 9974.70

Threads fairness:
    events (avg/stddev):           4291.0000/0.00
    execution time (avg/stddev):   9.9747/0.00
```

Experiment on QEMU — File IO Experiment:

- Set up different scenarios. Use the following command to set up QEMU configuration where -m represents the memory and -smp represents the total number of CPUs.

```
$ qemu-system-x86_64 -hda ubuntu.img -boot d -m 2048 -smp 2 --accel tcg -boot strict=on
```

- Ran the IO test with the following commands as a script under different scenarios

```
$sysbench –num-threads=8 –test=fileio –file-total-size=2G –file-test-mode=rndrw prepare
$sysbench –num-threads=8 –test=fileio –file-total-size=2G –file-test-mode=rndrw run
$sysbench –num-threads=8 –test=fileio –file-total-size=2G –file-test-mode=rndrw cleanup
```

```
#! /bin/bash
echo "File IO Test"
for value in {1..5}
do
        echo "Try" $value
        sysbench --num-threads=8 --test=fileio --file-total-size=2G --file-test-mode=rndrw prepare
        sysbench --num-threads=8 --test=fileio --file-total-size=2G --file-test-mode=rndrw run
        sysbench --num-threads=8 --test=fileio --file-total-size=2G --file-test-mode=rndrw cleanup
done
~
~
~
~
```

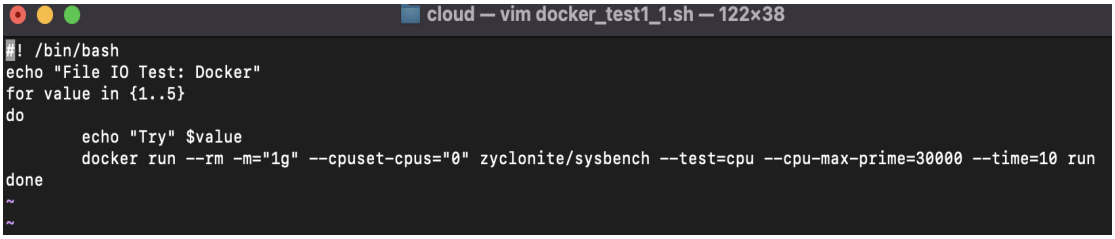
- Collect data: the output of the bash script is written into a text file as shown below for each scenario there is an output file.

```
Creating file test_file.124
Creating file test_file.125
Creating file test_file.126
Creating file test_file.127
2147483648 bytes written in 18.94 seconds (108.11 MiB/sec).
WARNING: --num-threads is deprecated, use --threads instead
sysbench 1.0.18 (using system LuaJIT 2.1.0-beta3)

Running the test with following options:
Number of threads: 8
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:                      1194.42
    writes/s:                     795.95
    fsyncs/s:                     2644.05

Throughput:
    read, MiB/s:                  18.66
    written, MiB/s:               12.44

General statistics:
    total time:                          10.1949s
    total number of events:              46243

Latency (ms):
         min:                                    0.02
         avg:                                    1.71
         max:                                   27.40
         95th percentile:                        5.28
         sum:                                79253.71
```

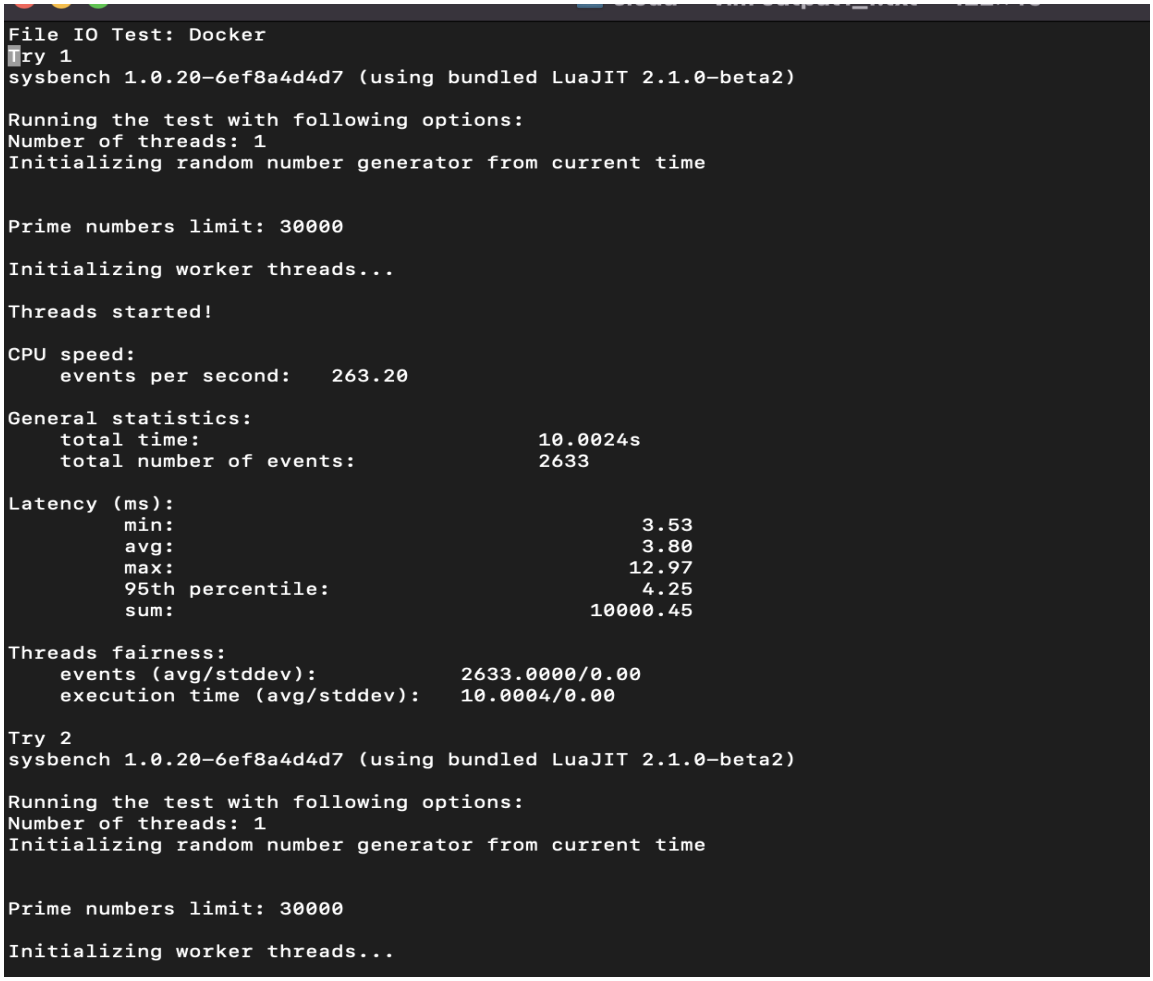
Experiment on Docker — CPU Experiment

- Use the following command to set up docker configurations (scenarios)
  - $docker run --rm -m="1g" --cpuset-cpus="0" zyclonite/sysbench --test=cpu --cpu-max-prime=10000 --time=10 run
    - -m represents the amount of memory used
    - -cpuset-cpus=”0” represents the number of CPUs used.
    - -cpu-max-prime set up the prime number that needs to be calculated

- Example : Write the command as a script Example:



- Collect data: the output of the script above is written into a text file as shown below



Experiment on Docker — File IO Experiment

- Use following commands to set up different scenarios in docker
  $ docker run --rm -it -m="1g" -- cpuset-cpus="0" --entrypoint /bin/sh zyclonite/sysbench
  $ docker run --rm -it -m="2g" -- cpuset-cpus="0-1" --entrypoint /bin/sh zyclonite/sysbench
  $ docker run --rm -it -m="4g" -- cpuset-cpus="0-1" --entrypoint /bin/sh zyclonite/sysbench

- The three containers created by above commands

```
daniellu@Zhihaos-MacBook-Pro clous % docker ps
CONTAINER ID   IMAGE                COMMAND     CREATED          STATUS          PORTS
  NAMES
be6f8095c2ed   zyclonite/sysbench   "/bin/sh"   6 minutes ago    Up 6 minutes
  elated_elgamal
40c90ca63870   zyclonite/sysbench   "/bin/sh"   15 minutes ago   Up 15 minutes
  distracted_austin
c5d86082ab55   zyclonite/sysbench   "/bin/sh"   48 minutes ago   Up 48 minutes
  admiring_swartz
daniellu@Zhihaos-MacBook-Pro clous %
```

- I wrote a shell script to test File IO, and each scenarios need to run this script.

```
#! /bin/bash
echo "Docker File IO Test"
for value in 1 2 3 4 5
do
        echo "Try" $value
        sysbench --num-threads=8 --test=fileio --file-total-size=2G --file-test-mode=rndrw prepare
        sysbench --num-threads=8 --test=fileio --file-total-size=2G --file-test-mode=rndrw run
        sysbench --num-threads=8 --test=fileio --file-total-size=2G --file-test-mode=rndrw cleanup
done
~
~
```

- Running script in container

```
/home # ls
docker_io_test1.sh
/home # sh docker_io_test1.sh > io_output_2.txt
WARNING: the --test option is deprecated. You can pass a script name or path on the command line without an
y options.
WARNING: the --test option is deprecated. You can pass a script name or path on the command line without an
y options.
WARNING: the --test option is deprecated. You can pass a script name or path on the command line without an
y options.
WARNING: the --test option is deprecated. You can pass a script name or path on the command line without an
y options.
WARNING: the --test option is deprecated. You can pass a script name or path on the command line without an
y options.
WARNING: the --test option is deprecated. You can pass a script name or path on the command line without an
y options.
WARNING: the --test option is deprecated. You can pass a script name or path on the command line without an
y options.
WARNING: the --test option is deprecated. You can pass a script name or path on the command line without an
y options.
WARNING: the --test option is deprecated. You can pass a script name or path on the command line without an
y options.
WARNING: the --test option is deprecated. You can pass a script name or path on the command line without an
y options.
WARNING: the --test option is deprecated. You can pass a script name or path on the command line without an
y options.
WARNING: the --test option is deprecated. You can pass a script name or path on the command line without an
y options.
WARNING: the --test option is deprecated. You can pass a script name or path on the command line without an
y options.
WARNING: the --test option is deprecated. You can pass a script name or path on the command line without an
y options.
WARNING: the --test option is deprecated. You can pass a script name or path on the command line without an
y options.
```

- Collect data: the output data is written into a text file

```
Creating file test_file.127
2147483648 bytes written in 3.20 seconds (639.35 MiB/sec).
WARNING: --num-threads is deprecated, use --threads instead
sysbench 1.0.20-6ef8a4d4d7 (using bundled LuaJIT 2.1.0-beta2)

Running the test with following options:
Number of threads: 8
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:                      8085.58
    writes/s:                     5390.38
    fsyncs/s:                     17339.30

Throughput:
    read, MiB/s:                  126.34
    written, MiB/s:               84.22

General statistics:
    total time:                          10.0236s
    total number of events:              307907

Latency (ms):
         min:                                    0.00
         avg:                                    0.26
         max:                                 1007.73
         95th percentile:                        0.62
         sum:                                79856.73

Threads fairness:
    events (avg/stddev):           38488.3750/567.74
    execution time (avg/stddev):   9.9821/0.00

WARNING: --num-threads is deprecated, use --threads instead
sysbench 1.0.20-6ef8a4d4d7 (using bundled LuaJIT 2.1.0-beta2)

Removing test files...
```

Experiment Result

- **CPU Test results —- QEMU**

CPU Test --- QEMU						
	Test Case 1: prime = 30000			Test Case 2: prime = 10000		
	Scenario_1: 1G Memory and 1 CPU	Scenario_2: 2G Memory and 2 CPU	Scenario_3: 4G Memory and 2 CPU	Scenario_1: 1G Memory and 1 CPU	Scenario_2: 2G Memory and 2 CPU	Scenario_3: 4G Memory and 2 CPU
Try 1	98.58	99.04	96.2	438.84	403.91	411.09
Try 2	99.77	98.58	92.82	438.86	406.96	401.12
Try 3	99.31	99.15	96.65	440.3	409.49	408.05
Try 4	100.24	98.75	98.63	433.49	412.53	420.65
Try 5	99.25	99.07	99.48	441.58	428.76	409.08
Max	100.24	99.15	99.48	441.58	428.76	420.65
Min	98.58	98.58	92.82	433.49	403.91	401.12
Avg	99.43	98.918	96.756	438.614	412.33	409.998
Std	0.6206851053	0.2420123964	2.585755209	3.082479521	9.718330618	7.650727199

- **CPU Test results — Docker**

CPU Test --- Docker						
	Test Case 1: prime = 30000			Test Case 2: prime = 10000		
	Scenario_1: 1G Memory and 1 CPU	Scenario_2: 2G Memory and 2 CPU	Scenario_3: 4G Memory and 2 CPU	Scenario_1: 1G Memory and 1 CPU	Scenario_2: 2G Memory and 2 CPU	Scenario_3: 4G Memory and 2 CPU
Try 1	263.2	263.62	264.32	1175.83	1186.31	1120.53
Try 2	261.7	256.94	258.3	1160.8	1178.09	1123.18
Try 3	271.49	265.9	262.36	1071.56	1176.92	1162.87
Try 4	274.97	260.51	269.22	1082.51	1182.99	1170.8
Try 5	274.87	254.65	266	1115.17	1182.96	1171.02
Max	274.97	265.9	269.22	1175.83	1186.31	1171.02
Min	261.7	254.65	258.3	1071.56	1176.92	1120.53
Avg	269.246	260.324	264.04	1121.174	1181.454	1149.68
Std	6.382110153	4.626978496	4.078063266	46.23310535	3.87561995	25.62907821

- **FIle IO Test results — QEMU**

File IO --- QEMU									
	Scenario 1: 1G Memory and 1 CPU			Scenario 2: 2G Memory and 2 CPU			Scenario 3: 4G Memory and 2 CPU		
	Throughput Write	Throught Read	Lantency	Throughput Write	Throught Read	Lantency	Throughput Write	Throught Read	Lantency
Try 1	12.65	18.98	1.7	13.74	20.62	1.56	17.17	25.75	1.25
Try 2	12.75	19.13	1.68	13.81	20.73	1.55	16.96	25.43	1.26
Try 3	12.59	18.89	1.7	14.07	21.11	1.52	16.96	25.43	1.26
Try 4	11.68	17.53	1.84	13.86	20.81	1.54	17.14	25.7	1.25
Try 5	12.72	19.09	1.69	12.44	18.66	1.71	17.07	25.61	1.25
Max	12.75	19.13	1.84	14.07	21.11	1.71	17.17	25.75	1.26
Min	11.68	17.53	1.68	12.44	18.66	1.52	16.96	25.43	1.25
Avg	12.478	18.724	1.722	13.584	20.386	1.576	17.06	25.584	1.254
Std	0.4504109235	0.6740771469	0.066483080	0.6512526392	0.9818502941	0.076354436	0.09823441352	0.1492648653	0.005477225

- **File IO Test results — Docker**

File IO --- Docker									
	Scenario 1: 1G Memory and 1 CPU			Scenario 2: 2G Memory and 2 CPU			Scenario 3: 4G Memory and 2 CPU		
	Throughput Write	Throught Read	Lantency	Throughput Write	Throught Read	Lantency	Throughput Write	Throught Read	Lantency
Try 1	54.77	82.17	0.4	84.22	126.34	0.26	95.03	142.55	0.23
Try 2	44.48	66.73	0.49	75.18	112.77	0.29	86.45	129.67	0.25
Try 3	51.26	76.89	0.43	82.12	123.18	0.27	96.86	145.29	0.23
Try 4	50.43	75.65	0.43	77.42	116.12	0.28	81.43	122.15	0.27
Try 5	29.71	44.58	0.74	82.74	124.11	0.26	83.89	125.84	0.26
Max	54.77	82.17	0.74	84.22	126.34	0.29	96.86	145.29	0.27
Min	29.71	44.58	0.4	75.18	112.77	0.26	81.43	122.15	0.23
Avg	46.13	69.204	0.498	80.336	120.504	0.272	88.732	133.1	0.248
Std	9.896911134	14.84254628	0.1391761474	3.844955136	5.77185672	0.013038404	6.850198537	10.27464841	0.017888543

Analysis and conclusion

From the results data, we can see that Docker's CPU performance is better than QEMU's performance. When calculating the prime number of 30000, the avg speed of the QEMU CPU test is about 100. When calculating the prime number of 10000 the avg speed of QEMU is 420. For docker the avg speed is about 270 and 1100 respectively. Dockers's Fille IO speed is also much better than QEMU. From the data we can see that for each scenario, Docker's write speed, read speed and latency are all much better than QEMU.

Another thing we can see from the data is as we increase the prime limit, the CPU speed will decrease for both docker and QEMU. In addition, we can see that as we increase the memory and the number of CPUs, the file IO performance including write speed, read speed and latency all have outstanding improvement for both docker and QEMU.