CS 552

Introduction to Cloud Computing

Mini Project – 1

Name – Radhika Dotihal

B-Number – B0087083

Detailed configurations of Google cloud instance is as below –

1. Login to GCP using ur personal Gmail account.

2. In the compute engine category, Click on VM instances and then click on create instance.

3. I have created an instance with name project-1 under project “My first Project”.

4. Below are the configurations chosen for the instance while creating - 2 VCPU, 4GB RAM and 30GB disk size and Operating System: Ubuntu (18.04 LTS). This instance is considered as native environment.

Steps to enable Docker container are as below -

1.Before installing the docker switched the system login from normal user to root user by using commands,

$sudo su

2.In the next step followed the process documents given to install the Docker CE on the Native OS.

3.Download the sysbench application inside the docker using command

$docker run csminpp/ubuntu-sysbench

The above command pulls the image from the docker registry and ensures it is saved inside docker repository not in native OS.

```
root@prj-1:/home/radhikardotihal# docker run -it csminpp/ubuntu-sysbench /bin/bash
root@3e25062a19cf:/# sysbench --test=cpu --cpu-max-prime=20000 run
```

3. After the installation of Docker, I performed some basic commands like below, and they are as following -

$docker pull – to pull an image from the docker registry on to the system.

```
radhikardotihal@prj-1:~$ sudo su
root@prj-1:/home/radhikardotihal# docker pull csminpp/ubuntu-sysbench
Using default tag: latest
latest: Pulling from csminpp/ubuntu-sysbench
d89e1bee20d9: Pull complete
9e0bc8a71bde: Pull complete
27aa681c95e5: Pull complete
a3ed95caeb02: Pull complete
55734f896640: Pull complete
Digest: sha256:90fd06985472eec3aa99b665618c23f074deb326fcc87a5fb59d2be1f9d97435
Status: Downloaded newer image for csminpp/ubuntu-sysbench:latest
root@prj-1:/home/radhikardotihal#
```

$docker run –creates and runs the container

$docker ps -a –to list all the containers that we ran along with their status

```
root@prj-1:/home/radhikardotihal# docker ps -a
CONTAINER ID   IMAGE                      COMMAND                  CREATED          STATUS                      PORTS     NAMES
7e651fd0ccf3   csminpp/ubuntu-sysbench    "--test=cpu --cpu-ma…"   2 minutes ago    Created                             angry_blackwell
ec30e1fc729a   csminpp/ubuntu-sysbench    "echo 'Hello From Do…"   3 minutes ago    Exited (0) 3 minutes ago            quirky_davinci
061a282a85e2   csminpp/ubuntu-sysbench    "ls -l"                  4 minutes ago    Exited (0) 4 minutes ago            happy_nash
0eb8b61af82b   csminpp/ubuntu-sysbench    "/bin/bash"              6 hours ago      Exited (0) 6 hours ago              jovial_golick
9e6e6e19dd9a   hello-world                "/hello"                 2 days ago       Exited (0) 2 days ago               gracious_pike
913ceaceeb72   alpine                     "/bin/sh"                4 days ago       Exited (255) 2 days ago             focused_gates
cd9ebbdc4fd4   alpine                     "/bin/sh"                4 days ago       Exited (0) 4 days ago               determined_newton
7d08c45db323   alpine                     "/bin/sh"                4 days ago       Exited (0) 4 days ago               bold_feistel
3db0be3fc92d   alpine                     "echo 'Hello from al…"   4 days ago       Exited (0) 4 days ago               bold_moore
92ca835bf0d3   alpine                     "echo 'Hello from al…"   4 days ago       Exited (0) 4 days ago               sharp_clarke
26d7957e7afc   alpine                     "ls -l"                  4 days ago       Exited (0) 4 days ago               intelligent_ishizaka
dfe2087fc249   hello-world                "/hello"                 4 days ago       Exited (0) 4 days ago               hardcore_easley
root@prj-1:/home/radhikardotihal# docker exec -it csminpp/ubuntu-sysbench /bin/bash
```

After the installation was complete, I run the basic “Hello World” program in the container to check if the Docker container is up and running. The output was a success and is as below –

```
root@prj-1:/home/radhikardotihal# sudo docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/

root@prj-1:/home/radhikardotihal#
```

Docker Measurement

Docker is installed and docker container performance is measured using Sysbench Benchmark tool. Docker container image csminpp/ubuntu-sysbench is used to check the system performance.

Below commands are used to use docker container image. Mainly two test modes are performed – cpu test mode and io test mode.

1) **CPU Utilization**

I performed cpu utilization operation. CPU utilization test enables multiple threads to execute prime number calculation until the command is satisfied (i.e till the maximum prime number mentioned in the command) and gives output the total time taken.

On the basis of my observation and executing the command for several iterations, I decided and selected maximum prime number to be checked as 27000 as the time taken to complete execution was between 30 sec to 50 sec. I performed the above test three times to get the average value.

The command for CPU utilization is as below –
$sysbench –test=cpu-max-prime=27000 run

Also, I performed the iostat test simultaneously and the output is saved in the log files.
The command to obtain the user level performance data is as below –
iostat -k 2 20 > [output_file name]
where
iostat – iostat is the command used to get user level performance data

-k – is the flag used, similarly we do have [-c], [-d], [-k] and many more flags.

The next parameters 2 and 20, 2 is the time period gap after which the output should calculated and 20 is the number of observations to be found.

Output_file name – output file name is where the output/ performance data should be stored and to store it '>' operator is used.

The output of first test mode is as below –

```
root@prj-1:/home/radhikardotihal# docker run -it csminpp/ubuntu-sysbench /bin/bash
root@3e16a99c394c:/# sysbench --test=cpu --cpu-max-prime=27000 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: 27000


Test execution summary:
    total time:                          41.0938s
    total number of events:              10000
    total time taken by event execution: 41.0918
    per-request statistics:
         min:                                  4.05ms
         avg:                                  4.11ms
         max:                                  8.94ms
         approx.  95 percentile:               4.20ms

Threads fairness:
    events (avg/stddev):           10000.0000/0.00
    execution time (avg/stddev):   41.0918/0.00

root@3e16a99c394c:/#
```

```
root@prj-1:/home/radhikardotihal# docker run -it csminpp/ubuntu-sysbench /bin/bash
root@32e91415f598:/# sysbench --test=cpu --cpu-max-prime=27000 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: 27000


Test execution summary:
    total time:                          41.1102s
    total number of events:              10000
    total time taken by event execution: 41.1077
    per-request statistics:
         min:                                  4.05ms
         avg:                                  4.11ms
         max:                                  6.82ms
         approx.  95 percentile:               4.20ms

Threads fairness:
    events (avg/stddev):           10000.0000/0.00
    execution time (avg/stddev):   41.1077/0.00

root@32e91415f598:/#
```

```
root@prj-1:/home/radhikardotihal# docker run -it csminpp/ubuntu-sysbench /bin/bash
root@cd6bcbdf1e5d:/# sysbench --test=cpu --cpu-max-prime=27000 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: 27000


Test execution summary:
    total time:                          41.1384s
    total number of events:              10000
    total time taken by event execution: 41.1364
    per-request statistics:
         min:                                  4.05ms
         avg:                                  4.11ms
         max:                                  7.27ms
         approx.  95 percentile:               4.21ms

Threads fairness:
    events (avg/stddev):           10000.0000/0.00
    execution time (avg/stddev):   41.1364/0.00

root@cd6bcbdf1e5d:/#
```

User level performance data is calculated by using the command –

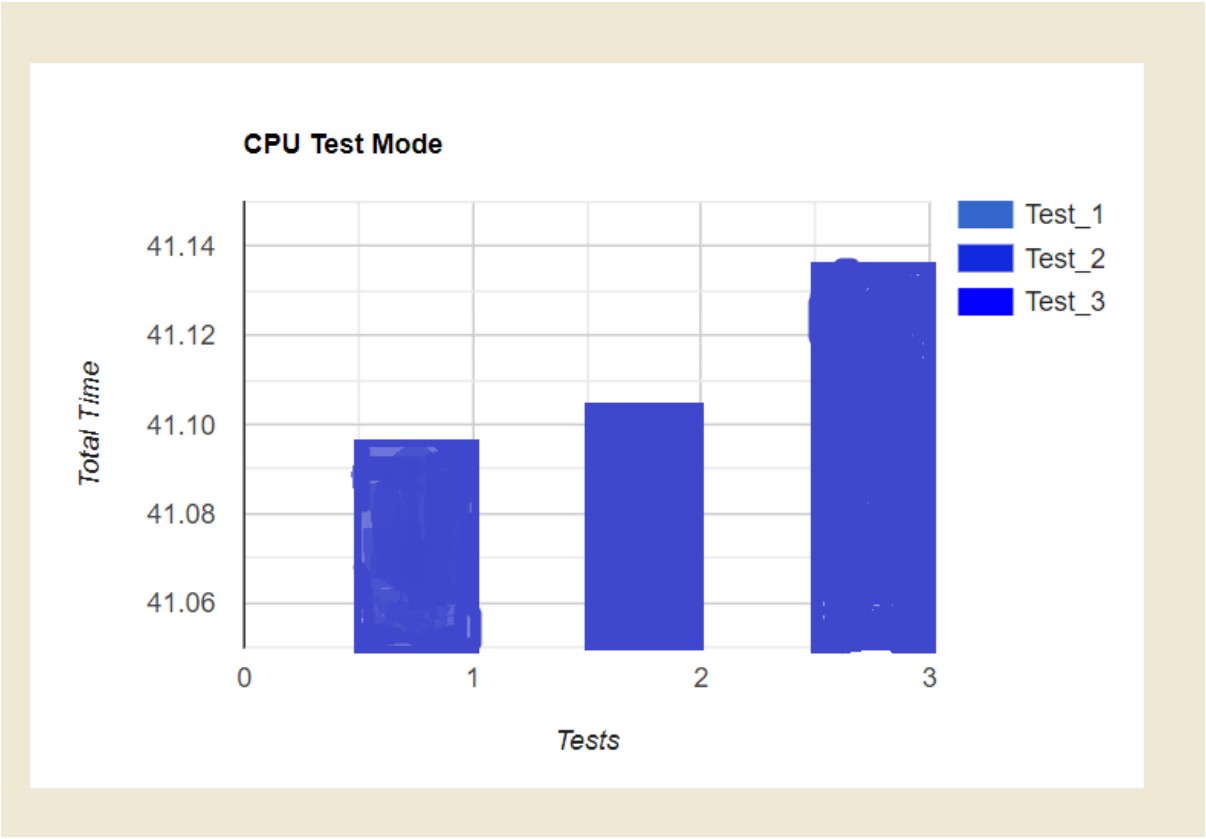
iostat -k 2 20 > output file_name

$docker pull – to pull an image form the docker registry on to the system.
$docker run –creates and runs the container
$docker ps -a –to list all the containers that we ran along with their status

To come out of the docker container and to the native OS terminal I simply used the command
$exit

Analysis of the CPU Test Mode data is as below –

CPU Test Mode	Total Time Taken
Test 1	41.0938 sec
Test 2	41.1102 sec
Test 3	41.1384 sec
Average	41.1141 sec



Above is the bar graph representing total time taken to run the CPU test mode. From the graph, it is visible that the total time taken is increasing exponentially.

2) **File I/O**

The File I/O test mode is used to generate various kinds of file I/O workloads. There are three stages of this test mode – prepare, run and cleanup.

A) Prepare stage –

In this stage, sysbench creates a specified number of files with a specified total size.

Below is the command for the prepare stage –

sysbench –test=fileio –file-total-size=1G prepare

B) Run stage –

In this stage, each thread performs specified Input/Output operations on the set of files mentioned in prepare stage.

Below is the syntax/command for the run stage –

sysbench -test=fileio -file-total-size=1G –file-test-mode=rndrw –max-time=60 –max-requests=0 run

C) Cleanup stage –

In this stage, the files used for the test are removed.

For example –

sysbench --num-threads=12 --test=fileio --file-total-size=3G --file-test-mode=rndrw prepare

This command creates 128 files with the total size of 3 GB in the current directory.

The parameters are as below –
Num-threads -> Number of threads/files to be created
File-total-size -> Size of each file to be created
File-test-mode -> test mode specifys the I/O operations to be supported.

Below are the different I/O operations supported –
1) **seqwr** – sequential write
2) **seqrewr** – sequential rewrite
3) **seqrd** – sequential read
4) **rndrd** – random read
5) **rndwr** – random write
6) **rndrw** – combined random read/write

$ sysbench --num-threads=12 --test=fileio --file-total-size=3G --file-test-mode=rndrw run
This command will run the actual benchmark and displays the results upon completion.

$ sysbench --num-threads=12 --test=fileio --file-total-size=3G --file-test-mode=rndrw cleanup
This command will remove the files used for the test.

After installation of docker, I performed the file I/O test inside docker using the below commands –
sysbench –--test=fileio –file-total-size=1G prepare
where,
file-total-size -> is size of which files should be created

sysbench - -test=fileio –file-total-size=1G - -file-test-mode=rndrw - -max-time=60 - -max-requests=0 run
where,
file-total-size -> is size of which files should be created
file-test-mode -> the type of work load to be created, here it is rndrw which is random read and write.

max-time -> Maximum time for which the test should be running, here it is 60 seconds.

Max-requests -> limit the total number of requests.

sysbench - -test=fileio –file-total-size=1G - -file-test-mode=rndrw - -max-time=60 - -max-requests=0 cleanup

After performing, these 3 commands, I came outside the docker container using command exit.
In the native environment, I logged in as the root user and cleared the cache using the below command –

echo 3 > /proc/sys/vm/drop_caches

```
5]+  Stopped                 vim fileio_2
oot@project-1:/home/radhikardotihal# echo 3 > /proc/sys/vm/drop_caches
oot@project-1:/home/radhikardotihal#
```

While executing the file I/O test mode, I have collected the CPU performance (user level and kernel level), disk utilization and I/O performance of native environment using iostat tool.

Below are the commands for the same –
$iostat ALL -t 90 20

This command displays the cpu utilization by kernel level (i.e CPU usage of operating system), user level (i.e CPU usage of user), nice (i.e amount of time a low priority process have used the cpu) and cpu idle time.

$iostat -xd -t 90 20

This command displays detailed information like tps (transfers per second), read & write per second, I/O latency (await time), rrqm/sec, wrqm/s (read and write request queued for the I/O scheduler per second) and much more.

$df -h

This command displays the disk utilization of file system.

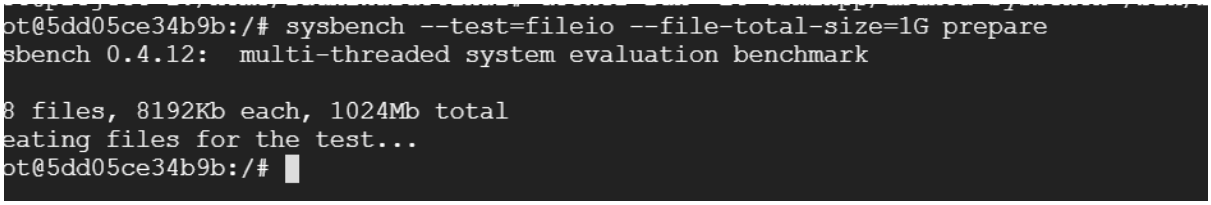
Following are the results of File I/O test mode –

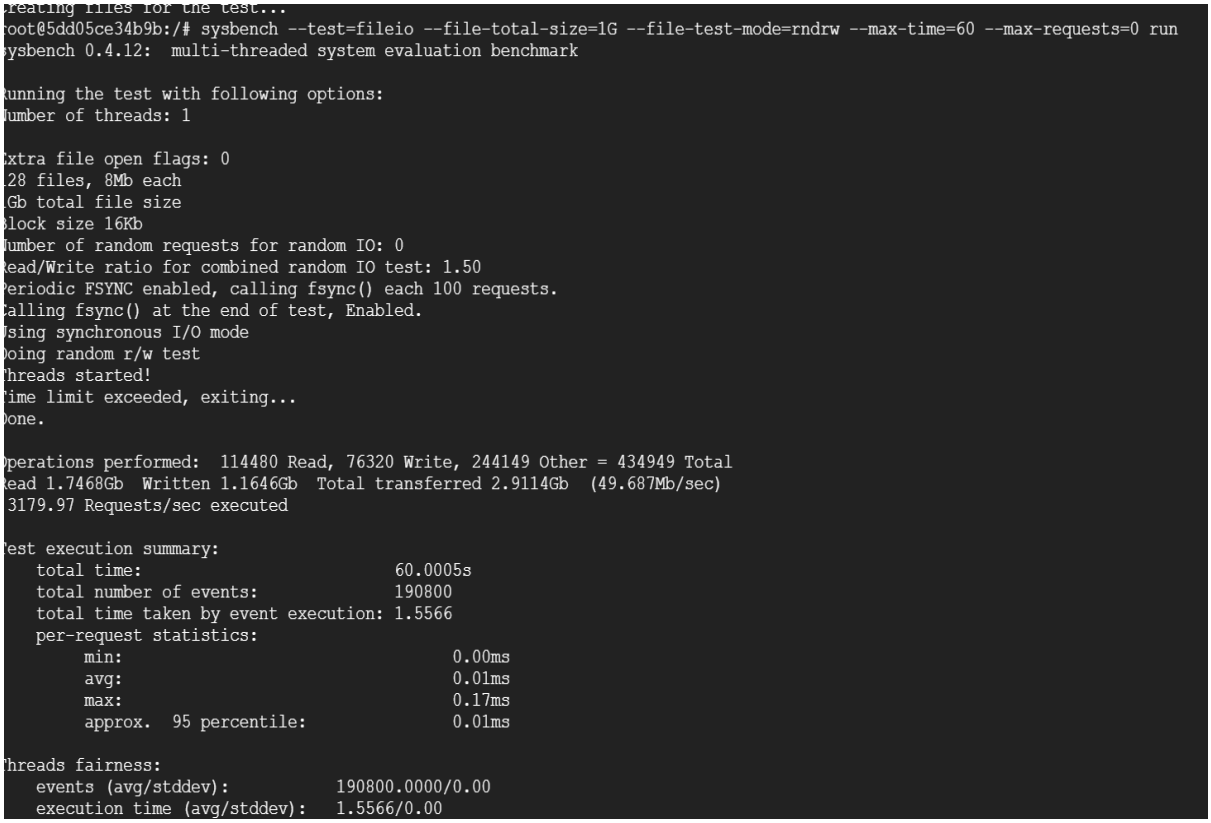
```
[3]+  Stopped                 vim cpu_3
root@project-1:/home/radhikardotihal# docker run -it csminpp/ubuntu-sysbench /bin/bash
root@3905e79f6f98:/# sysbench --test=fileio --file-total-size=1G prepare
sysbench 0.4.12:  multi-threaded system evaluation benchmark

128 files, 8192Kb each, 1024Mb total
Creating files for the test...
root@3905e79f6f98:/#
```

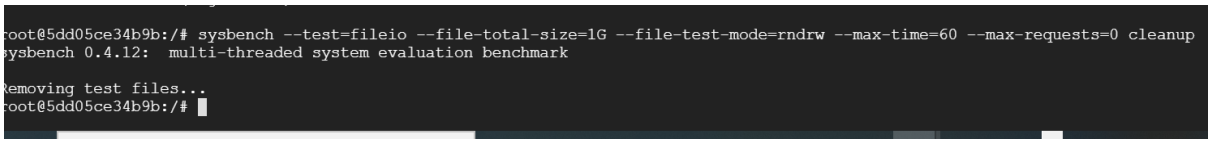
```
Creating files for the test...
root@3905e79f6f98:/# sysbench --test=fileio --file-total-size=1G --file-test-mode=rndrw --max-time=60 --max-requests=0 run
sysbench 0.4.12:  multi-threaded system evaluation benchmark

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

Extra file open flags: 0
128 files, 8Mb each
1Gb total file size
Block size 16Kb
Number of random requests for random IO: 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
Threads started!
Time limit exceeded, exiting...
Done.

Operations performed:  112560 Read, 75040 Write, 240115 Other = 427715 Total
Read 1.7175Gb  Written 1.145Gb  Total transferred 2.8625Gb  (48.853Mb/sec)
 3126.61 Requests/sec executed

Test execution summary:
    total time:                          60.0010s
    total number of events:              187600
    total time taken by event execution: 1.5752
    per-request statistics:
         min:                                  0.00ms
         avg:                                  0.01ms
         max:                                  0.13ms
         approx.  95 percentile:               0.01ms

Threads fairness:
    events (avg/stddev):           187600.0000/0.00
    execution time (avg/stddev):   1.5752/0.00

root@3905e79f6f98:/#
```

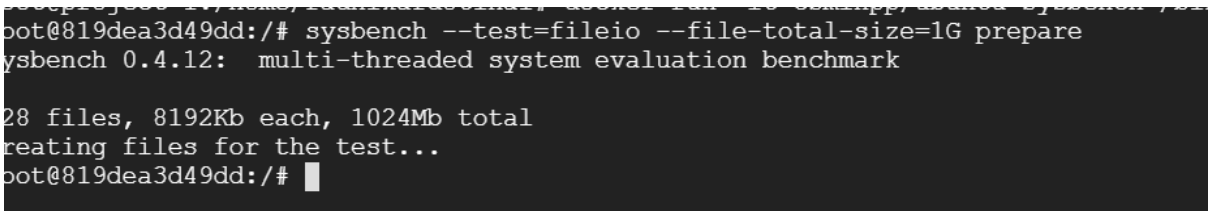
```
Threads fairness:
    events (avg/stddev):           187600.0000/0.00
    execution time (avg/stddev):   1.5752/0.00

root@3905e79f6f98:/# sysbench --test=fileio --file-total-size=1G --file-test-mode=rndrw --max-time=60 --max-requests=0 cleanup
sysbench 0.4.12:  multi-threaded system evaluation benchmark

Removing test files...
root@3905e79f6f98:/#
```

Similarly, I performed this step two more times. The commands and their result are as following –







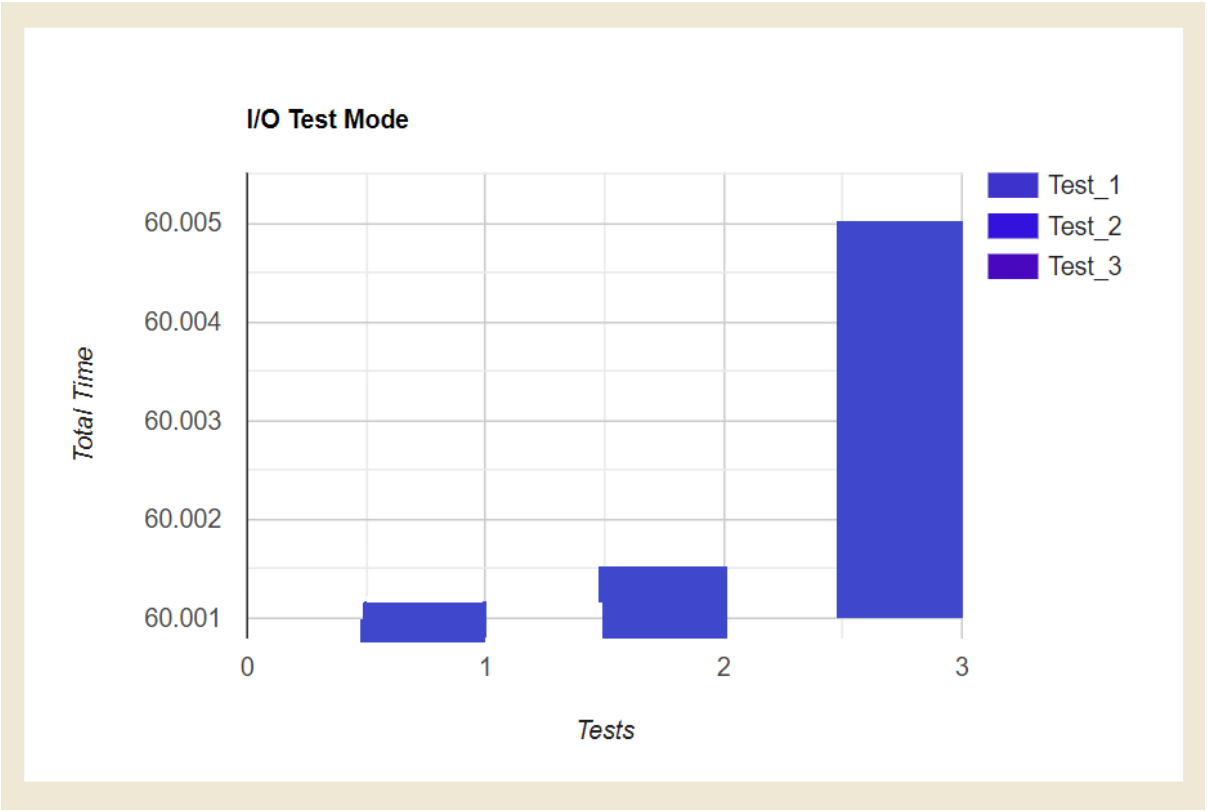


```
root@819dea3d49dd:/# sysbench --test=fileio --file-total-size=1G --file-test-mode=rndrw --max-time=60 --max-requests=0 run
sysbench 0.4.12:  multi-threaded system evaluation benchmark

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

Extra file open flags: 0
128 files, 8Mb each
1Gb total file size
Block size 16Kb
Number of random requests for random IO: 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
Threads started!
Time limit exceeded, exiting...
Done.

Operations performed:  115380 Read, 76920 Write, 246104 Other = 438404 Total
Read 1.7606Gb  Written 1.1737Gb  Total transferred 2.9343Gb  (50.077Mb/sec)
 3204.94 Requests/sec executed

Test execution summary:
    total time:                          60.0011s
    total number of events:              192300
    total time taken by event execution: 1.6306
    per-request statistics:
         min:                                  0.00ms
         avg:                                  0.01ms
         max:                                  3.75ms
         approx.  95 percentile:               0.01ms

Threads fairness:
    events (avg/stddev):           192300.0000/0.00
    execution time (avg/stddev):   1.6306/0.00
```

```
root@819dea3d49dd:/# sysbench --test=fileio --file-total-size=1G --file-test-mode=rndrw --max-time=60 --max-requests=0 cleanup
sysbench 0.4.12:  multi-threaded system evaluation benchmark

Removing test files...
root@819dea3d49dd:/#
```

Analysis of data is as below –

File I/O Test Mode	Total Time Taken
Test 1	60.0010 sec
Test 2	60.0011 sec
Test 3	60.0050 sec
Average	60.0023



The total time taken by the sysbench benchmark to run the I/O test mode is plotted as a bar graph. As we can see, the total time taken to run the benchmark is increasing exponentially.

QEMU Based VM

I installed QEMU as per the steps given in the document and reference. The complete installation took 2 to 3 hours.

Below are the reasons why QEMU based VM so slow to install and execute.

1) QEMU based VM requires various number of libraries to be installed for the completion of the installation. Thus, installation of QEMU based VM is very time consuming.

2) QEMU ignores the presence of hardware virtualization capabilities. Depending on the target architecture, “tcg” is made available where tcg is Tiny Code Generator. Tiny Code Generator slowly emulates the guest CPU in software.

3) QEMU based VM needs a GUI and it occupies a large amount of disk space. So it is slower to install.

4) There can a network connectivity issue as we are installing the VM on Google Cloud Platform via a local server.

5) During the booting process, QEMU based VM has to load all the libraries & dependencies and also GUI.

6) QEMU is very complex and thus has very slow functioning.

Below is the screenshot of QEMU based VM installed –