Report

1) Present main steps to enable a QEMU VM. In addition, please present the detailed QEMU commands, and VM configurations

1. Install qemu
   brew install qemu

2. Create an QEMU image with 10G memory
   qemu-img create ubuntu.img 10G -f qcow2

3. Install VM
   qemu-system-x86_64 -hda ubuntu.img -boot d -cdrom ./ubuntu-20.04.4-live-server-amd64.iso -m 2046 -boot strict=on

4. Run VM (1 CPU & 2G Memory)
   qemu-system-x86_64 -hda ubuntu.img -boot d -m 2046 -boot strict=on

Additional config when run VM:

a. (2 CPU & 2G Memory & accel) qemu-system-x86_64 -hda ubuntu.img -boot d -m 2046 -smp 2 --accel tcg -boot strict=on

b. (2 CPU & 4G Memory & accel) qemu-system-x86_64 -hda ubuntu.img -boot d -m 4G -smp 2 --accel tcg -boot strict=on

2) Present main steps to enable a Docker container

First I tried to install Docker native using brew, but it didn't work when I tried to run as following screenshot. It also stated on Piazza that I need Docker-Machine + VirtualBox to run on mac. So I installed Docker Desktop (for Mac).

```
(base) zhihaolin@Zhihaos-MBP ~ % brew install docker
Running `brew update --preinstall`...
==> Auto-updated Homebrew!
Updated 1 tap (homebrew/core).
==> Updated Formulae
Updated 55 formulae.

Warning: Treating docker as a formula. For the cask, use homebrew/cask/docker
==> Downloading https://ghcr.io/v2/homebrew/core/docker/manifests/20.10.14
######################################################################## 100.0%
==> Downloading https://ghcr.io/v2/homebrew/core/docker/blobs/sha256:fabf58e97ab
==> Downloading from https://pkg-containers.githubusercontent.com/ghcr1/blobs/sh
######################################################################## 100.0%
==> Pouring docker--20.10.14.monterey.bottle.tar.gz
==> Caveats
zsh completions have been installed to:
  /usr/local/share/zsh/site-functions
==> Summary
🍺  /usr/local/Cellar/docker/20.10.14: 12 files, 55.9MB
==> Running `brew cleanup docker`...
Disable this behaviour by setting HOMEBREW_NO_INSTALL_CLEANUP.
Hide these hints with HOMEBREW_NO_ENV_HINTS (see `man brew`).
(base) zhihaolin@Zhihaos-MBP ~ % brew docker start
Error: Unknown command: docker
```

Then I run “docker run --rm zyclonite/sysbench --test=cpu --cpu-max-prime=100 --time=20 run” which installed the sysbench first if it had not been installed. Then it execute the cpu test. Shown as following screen shot:

```
(base) zhihaolin@Zhihaos-MBP ~ % docker run --rm zyclonite/sysbench --test=cpu --cpu-max-prime=100 --time=20 run
Unable to find image 'zyclonite/sysbench:latest' locally
latest: Pulling from zyclonite/sysbench
1c03554ad6ac: Pull complete
b03501e82efa: Pull complete
Digest: sha256:016020c3b53c7e65cdb58e7d4a98afd14f8a3e2f5781cf4c368596b2e448602b
Status: Downloaded newer image for zyclonite/sysbench:latest
sysbench 1.0.20-6ef8a4d4d7 (using bundled LuaJIT 2.1.0-beta2)

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


Prime numbers limit: 100

Initializing worker threads...

WARNING: the --test option is deprecated. You can pass a script name or path on the command line without any options.
Threads started!

CPU speed:
    events per second: 452920.62

General statistics:
    total time:                          20.0001s
    total number of events:              9059061

Latency (ms):
         min:                                    0.00
         avg:                                    0.00
         max:                                    0.65
         95th percentile:                        0.00
         sum:                                19069.76

Threads fairness:
    events (avg/stddev):           9059061.0000/0.00
    execution time (avg/stddev):   19.0698/0.00
```

Some important operations for docker are:

1. Check currently running process and get information: $ docker ps

```
(base) zhihaolin@Zhihaos-MBP ~ % docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

2. Transfer file from local to container for test automation:
   $ sudo docker cp <file on local> <container id>:<file location at container>
3. --cpuset-cpus flags to specify number of CPU when run
4. --memory flags to specify number of memory when run

3) Proof of experiment

First I tried running the CPU test on QEMU with --cpu-max-prime=20000 the result is shown as following:

```
zlin2@zlin2:~$ sysbench --test=cpu --cpu-max-prime=20000 run
WARNING: the --test option is deprecated. You can pass a script name or path on the command line without any options.
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: 20000

Initializing worker threads...

Threads started!

CPU speed:
    events per second:   170.99

General statistics:
    total time:                          10.0028s
    total number of events:              1711

Latency (ms):
         min:                                    5.49
         avg:                                    5.83
         max:                                   10.15
         95th percentile:                        6.21
         sum:                                 9981.41

Threads fairness:
    events (avg/stddev):           1711.0000/0.00
    execution time (avg/stddev):   9.9814/0.00
```

Then I tried with --cpu-max-prime=30000, which produced the same total time=10s. I also tried --cpu-max-prime=50000 which also had total time=10s. So I used –time=8 for my experiments.

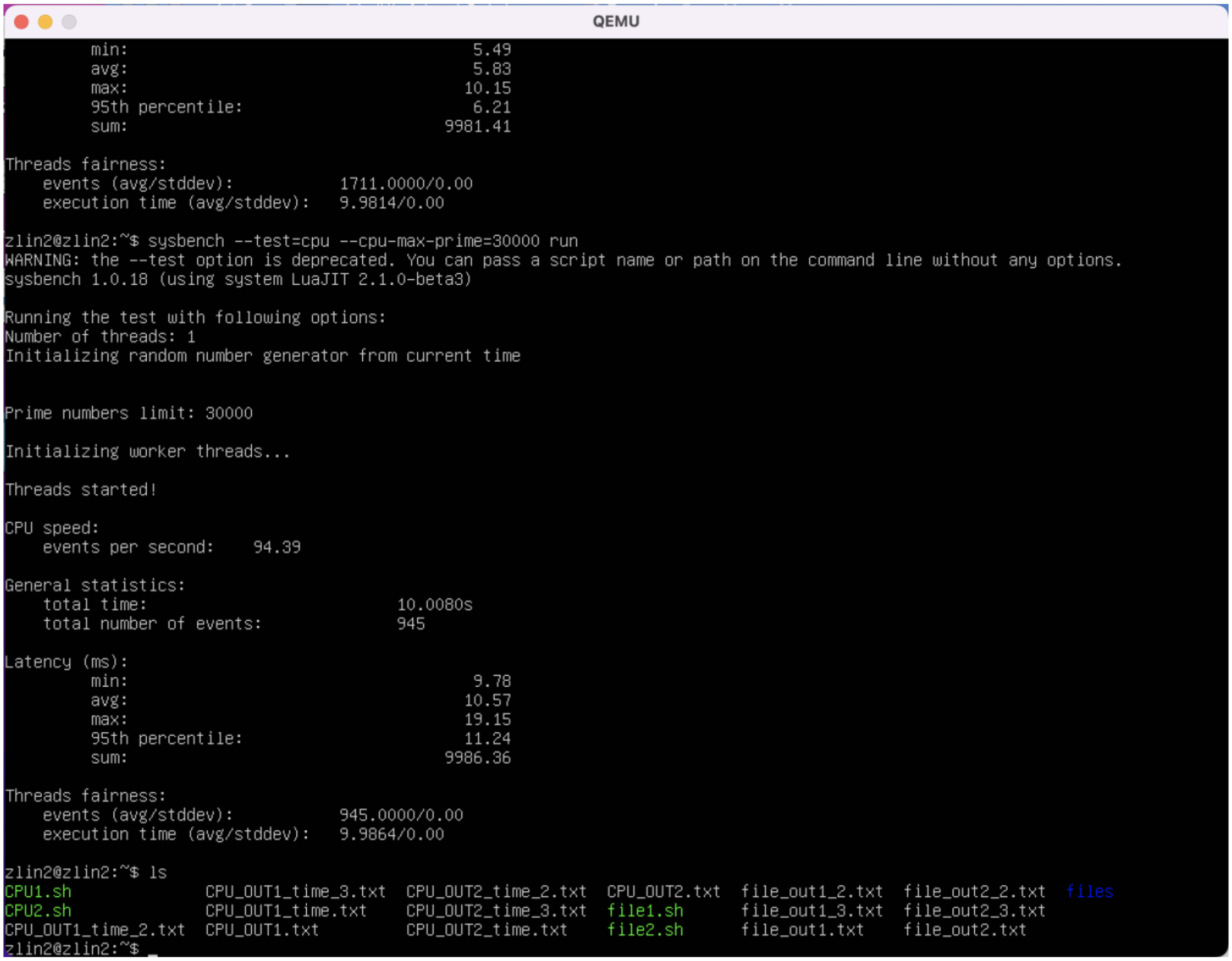
```
zlin2@zlin2:~$ sysbench --test=cpu --cpu-max-prime=30000 run
WARNING: the --test option is deprecated. You can pass a script name or path on the command line without any options.
sysbench 1.0.18 (using system LuaJIT 2.1.0-beta3)

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


Prime numbers limit: 30000

Initializing worker threads...

Threads started!

CPU speed:
    events per second:    94.39

General statistics:
    total time:                          10.0080s
    total number of events:              945

Latency (ms):
         min:                                    9.78
         avg:                                   10.57
         max:                                   19.15
         95th percentile:                       11.24
         sum:                                 9986.36

Threads fairness:
    events (avg/stddev):           945.0000/0.00
    execution time (avg/stddev):   9.9864/0.00
```

Check CPU and memory on QEMU:

```
zlin2@zlin2:~$ nproc
1
zlin2@zlin2:~$ free -m
              total        used        free      shared  buff/cache   available
Mem:           1981         182        1498           0         300        1650
Swap:          1739           0        1739
zlin2@zlin2:~$
```

Present how you use performance tools to collect performance data. For CPU utilization, you should at least divide them into two parts including user-level and kernel-level. For I/O, you should present I/O throughput, latency, and disk utilization: 10 points

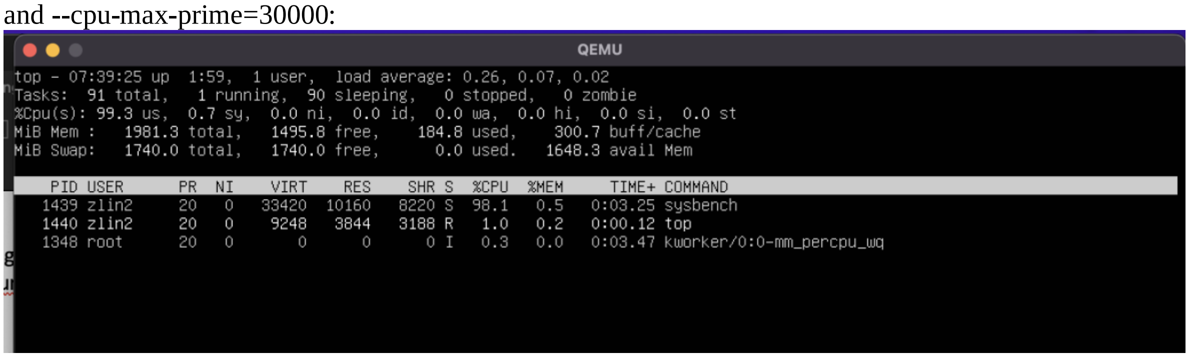
All the experiments were automated by using shell scripts and result was stored into .txt files in order for better excess. Check the following screenshot for reference:



```
QEMU
        min:                                    5.49
        avg:                                    5.83
        max:                                   10.15
        95th percentile:                        6.21
        sum:                                 9981.41

Threads fairness:
    events (avg/stddev):           1711.0000/0.00
    execution time (avg/stddev):   9.9814/0.00

zlin2@zlin2:~$ sysbench --test=cpu --cpu-max-prime=30000 run
WARNING: the --test option is deprecated. You can pass a script name or path on the command line without any options.
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: 30000

Initializing worker threads...

Threads started!

CPU speed:
    events per second:    94.39

General statistics:
    total time:                          10.0080s
    total number of events:              945

Latency (ms):
        min:                                    9.78
        avg:                                   10.57
        max:                                   19.15
        95th percentile:                       11.24
        sum:                                 9986.36

Threads fairness:
    events (avg/stddev):           945.0000/0.00
    execution time (avg/stddev):   9.9864/0.00

zlin2@zlin2:~$ ls
CPU1.sh              CPU_OUT1_time_3.txt  CPU_OUT2_time_2.txt  CPU_OUT2.txt  file_out1_2.txt  file_out2_2.txt  files
CPU2.sh              CPU_OUT1_time.txt    CPU_OUT2_time_3.txt  file1.sh      file_out1_3.txt  file_out2_3.txt
CPU_OUT1_time_2.txt  CPU_OUT1.txt         CPU_OUT2_time.txt    file2.sh      file_out1.txt    file_out2.txt
zlin2@zlin2:~$ _
```

You can read through the output files as a proof.

For CPU utilization, I used “top -i” commands to access the information:
-The following is a screenshot of CPU utilization during the test on QEMU with CPU=1 & Memory=2G and --cpu-max-prime=30000:



```
QEMU
top - 07:39:25 up  1:59,  1 user,  load average: 0.26, 0.07, 0.02
Tasks:  91 total,   1 running,  90 sleeping,   0 stopped,   0 zombie
%Cpu(s): 99.3 us,  0.7 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   1981.3 total,   1495.8 free,    184.8 used,    300.7 buff/cache
MiB Swap:   1740.0 total,   1740.0 free,      0.0 used.   1648.3 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   1439 zlin2     20   0   33420  10160   8220 S  98.1   0.5   0:03.25 sysbench
   1440 zlin2     20   0    9248   3844   3188 R   1.0   0.2   0:00.12 top
   1348 root      20   0       0      0      0 I   0.3   0.0   0:03.47 kworker/0:0-mm_percpu_wq
```

4) Three different scenarios for each virtualization technology

I conducts three scenarios by specifying different number of CPU and memory. The three scenarios are listed below:

1. CPU=1 & Memory=2G
2. CPU=2 & Memory=2G
3. CPU=2 & Memory=4G

Detail commands about how I specifying those scenarios when running the experiments.

QEMU:
1. qemu-system-x86_64 -hda ubuntu.img -boot d -m 2046 -boot strict=on
2. qemu-system-x86_64 -hda ubuntu.img -boot d -m 2046 -smp 2 --accel tcg -boot strict=on
3. qemu-system-x86_64 -hda ubuntu.img -boot d -m 4G -smp 2 --accel tcg -boot strict=on

Docker:
--cpuset-cpus specify # of CPU
--memory or -m specify # of memory

For CPU test:
Prime=20000
1. docker run --rm -m="2g" --cpuset-cpus="0" zyclonite/sysbench --test=cpu --cpu-max-prime=20000 --time=8 run
2. docker run --rm -m="2g" --cpuset-cpus="0-1" zyclonite/sysbench --test=cpu --cpu-max-prime=20000 --time=8 run
3. docker run --rm -m="4g" --cpuset-cpus="0-1" zyclonite/sysbench --test=cpu --cpu-max-prime=20000 --time=8 run

Prime=30000
1. docker run --rm -m="2g" --cpuset-cpus="0" zyclonite/sysbench --test=cpu --cpu-max-prime=20000 --time=8 run
2. docker run --rm -m="2g" --cpuset-cpus="0-1" zyclonite/sysbench --test=cpu --cpu-max-prime=20000 --time=8 run
3. docker run --rm -m="4g" --cpuset-cpus="0-1" zyclonite/sysbench --test=cpu --cpu-max-prime=20000 --time=8 run

For fileIO test:
1. Docker run --rm -it -m="2g" -- cpuset-cpus="0" --entrypoint /bin/sh zyclonite/sysbench
2. Docker run --rm -it -m="2g" -- cpuset-cpus="0-1" --entrypoint /bin/sh zyclonite/sysbench
3. Docker run --rm -it -m="4g" -- cpuset-cpus="0-1" --entrypoint /bin/sh zyclonite/sysbench

5) Presentation and analysis of the performance data

All data are extracted using the average, you can see details with min, max and std in the excel file.

The data is present in the order of three system scenarios:
1) CPU=1 & Memory=2G
2) CPU=2 & Memory=2G
3) CPU=2 & Memory=4G

For CPU testing:
Docker runs faster than QEMU VM in all CPU test cases.

Testcase 1: --cpu-max-prime=20000:
The value represents the average of events per second. The value seems to be increased as increase the number of CPU then increase the memory.

QEMU:
1) 160.4

2) 161.968
3) 171.268

Docker:
1) 463.862
2) 462.664
3) 465.214

Testcase 2: --cpu-max-prime=30000:
QEMU:
1) 96.864
2) 98.944
3) 96.342

Docker:
1) 270.838
2) 270.972
3) 270.846

For fileIO testing:

As I increased the Memory for system. The Through was increased and the latency was decreased. Moreover, Docker had more throughput and less Latency comparing to QEMU VM

Testcase 1: FileIO 1, Threads=16, size=3G

QEMU:
1. Throughput:
   read, MiB/s: 20.37
   written, MiB/s: 13.58
   Latency (ms): 3.12

2. Throughput:
   read, MiB/s: 21.02
   written, MiB/s: 14.01
   Latency (ms): 3.01

3. Throughput:
   read, MiB/s: 24.61
   written, MiB/s: 16.41
   Latency (ms): 2.57

Docker:
1. Throughput:
   read, MiB/s: 163.38
   written, MiB/s: 108.90
   Latency (ms): 0.40

2. Throughput:
   read, MiB/s: 165.08
   written, MiB/s: 110.04
   Latency (ms): 0.40

3. Throughput:
   read, MiB/s: 227.91
   written, MiB/s: 151.94
   Latency (ms): 0.29

Testcase 2: Threads=16, size=4G

QEMU:
Throughput:
read, MiB/s: 18.79
written, MiB/s: 12.52
Latency (ms): 3.38

Throughput:
read, MiB/s: 20.35
written, MiB/s: 13.57
Latency (ms): 3.12

Throughput:
read, MiB/s: 22.88
written, MiB/s: 15.25
Latency (ms): 2.77

Docker:

1. Throughput:
   read, MiB/s: 136.25
   written, MiB/s: 90.82
   Latency (ms): 0.48

2. Throughput:
   read, MiB/s: 178.56
   written, MiB/s: 119.02
   Latency (ms): 0.37

3. Throughput:
   read, MiB/s: 150.67
   written, MiB/s: 100.44
   Latency (ms): 0.44