COEN241
Poonam Kanani
1620239
Assignment 1

System vs OS virtualization

Experiments were performed to understand system virtualization and OS virtualization using QEMU and Docker respectively. This report gives an overview of experiments performed.

Environment

Host environment

I have run this experiment on personal laptop with configuration shown in below screenshot:

Before this I tried running on EC2 instance but since GUI was not working, I was having trouble with getting the VM working.

QEMU environment

I tried running QEMU with Ubuntu 20.04, but after lots of effort I couldn't get it running. Finally I ran the experiments on Ubuntu 16.04.

Commands to setup QEMU

1. To install QEMU command
   `brew install qemu`

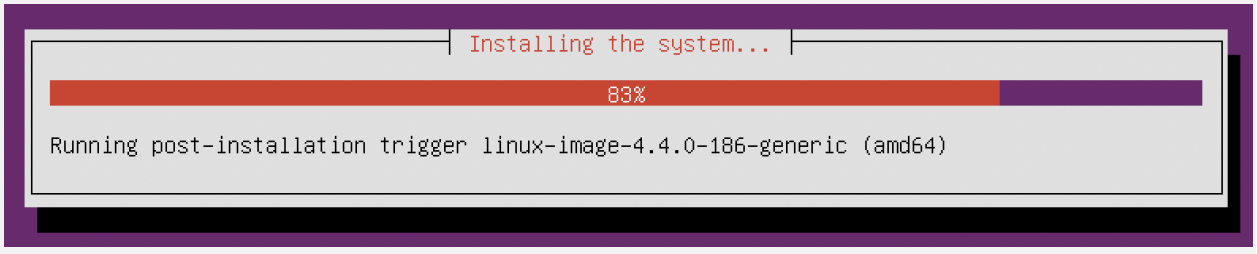
```
[MyMac:Downloads rd$ brew install qemu
==> Downloading https://ghcr.io/v2/homebrew/core/libffi/manifests/3.4.2
######################################################################## 100.0%
==> Downloading https://ghcr.io/v2/homebrew/core/libffi/blobs/sha256:a461f6ad21a23a725691385dbbec3eff9
==> Downloading from https://pkg-containers.githubusercontent.com/ghcr1/blobs/sha256:a461f6ad21a23a725
######################################################################## 100.0%
==> Downloading https://ghcr.io/v2/homebrew/core/pcre/manifests/8.45
######################################################################## 100.0%
==> Downloading https://ghcr.io/v2/homebrew/core/pcre/blobs/sha256:fb2fefbe1232706a603a6b385fc37253e5a
==> Downloading from https://pkg-containers.githubusercontent.com/ghcr1/blobs/sha256:fb2fefbe1232706a6
######################################################################## 100.0%
```

2. Downloading ubuntu server image
   `curl -o ubuntu-20.04.4-live-server-amd64.iso https://releases.ubuntu.com/16.04/ubuntu-16.04.7-server-amd64.iso`

3. Create QEMU image
   `qemu-img create -f qcow2 ubuntu.qcow 10G`

```
[MyMac:Downloads rd$ qemu-img create ubuntu.img 10G -f qcow2
Formatting 'ubuntu.img', fmt=qcow2 cluster_size=65536 extended_l2=off compression_type=zlib size=10737418240 lazy_refcounts=off refcount_bits=16
```

4. Install OA in the QEMU image
   `qemu-system-x86_64 -hda ubuntu.img -boot d -cdrom ubuntu-16.04.7-server-amd64.iso -m 2046 -boot strict=on`



5. Running QEMU after installation
   `qemu-system-x86_64 -hda ubuntu.img -m 2046`

CPU configurations

```
pkanani@ubuntu:~$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                1
On-line CPU(s) list:   0
Thread(s) per core:    1
Core(s) per socket:    1
Socket(s):             1
NUMA node(s):          1
Vendor ID:             AuthenticAMD
CPU family:            15
Model:                 107
Model name:            QEMU Virtual CPU version 2.5+
Stepping:              1
CPU MHz:               2299.853
BogoMIPS:              4599.70
Virtualization:        AMD-V
L1d cache:             64K
L1i cache:             64K
L2 cache:              512K
L3 cache:              16384K
NUMA node0 CPU(s):     0
Flags:                 fpu de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush m
mx fxsr sse sse2 syscall nx lm rep_good nopl extd_apicid pni cx16 hypervisor lahf_lm svm 3dnowprefet
ch vmmcall
```

Docker environment

I tried zyclonite/sysbench but for uniformity switched to csminpp/ubuntu-sysbench image which has ubuntu.

1. Installing docker
   Followed instructions according to https://docs.docker.com/desktop/mac/install/

2. Download required docker image
   `Docker pull csminpp/ubuntu-sysbench`

```
latest: Pulling from csminpp/ubuntu-sysbench
Image docker.io/csminpp/ubuntu-sysbench:latest uses outdated schema1 manifest format. Please upgrade to a schema2 image
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
```

Useful arguments for QEMU

Below are some useful arguments I have experimented in QEMU
-m :
Using -m we can define how much memory do we want to assign to QEMU
Since I have given 2046 as initial value system has 2046MB memory as can be seen in screenshot below:

```
pkanani@ubuntu:~$ cat /proc/meminfo | head -1
MemTotal:        2046088 kB
```

-smp :
Using -smp we can define how many CPU cores we want in our VM.
Initial value of cpu cores can be seen as 1 in below screenshot.

```
pkanani@ubuntu:~$ cat /proc/cpuinfo
processor       : 0
vendor_id       : AuthenticAMD
cpu family      : 15
model           : 107
model name      : QEMU Virtual CPU version 2.5+
stepping        : 1
microcode       : 0x1000065
cpu MHz         : 2303.972
cache size      : 512 KB
physical id     : 0
siblings        : 1
core id         : 0
cpu cores       : 1
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fx
sr sse sse2 syscall nx lm rep_good nopl extd_apicid pni cx16 hypervisor lahf_lm svm 3dnowprefetch vm
mcall
bugs            : apic_c1e fxsave_leak sysret_ss_attrs spectre_v1 spectre_v2
bogomips        : 4607.94
TLB size        : 1024 4K pages
clflush size    : 64
cache_alignment : 64
address sizes   : 40 bits physical, 48 bits virtual
power management:
```

After running QEMU with value of -smp 2, the value of CPU cores has now changed from 1 to 2.

```
cpu cores       : 2
```

-accel :
Using the accel option we can enable the accelerator. It helps with the speed of VM.

-cpu :
Using cpu option we can emulate different CPU models. In initial screen shot we can see that default CPU model is “QEMU Virtual CPU version 2.5+”
After running it with option cpu and value “Broadwell-v1” cpu model have now changed to “Intel Core Processor (Broadwell)”

```
pkanani@ubuntu:~$ cat /proc/cpuinfo | head
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 61
model name      : Intel Core Processor (Broadwell)
stepping        : 2
microcode       : 0x1
cpu MHz         : 2300.437
cache size      : 16384 KB
physical id     : 0
pkanani@ubuntu:~$
```

-boot
Using boot option we can pass values which will be used during booting process

-hda
Supplies hard disk to VM

-cdrom
Supplies cdrom to VM

Useful Docker commands

`docker version`
This command gives information about version of the docker running on the machine.
It's output looks similar to the screenshot below:

```
MyMac:Downloads rd$ docker version
 Client:
  Cloud integration: 1.0.17
  Version:           20.10.8
  API version:       1.41
  Go version:        go1.16.6
  Git commit:        3967b7d
  Built:             Fri Jul 30 19:55:20 2021
  OS/Arch:           darwin/amd64
  Context:           default
  Experimental:      true

 Server: Docker Engine - Community
  Engine:
   Version:          20.10.8
   API version:      1.41 (minimum version 1.12)
   Go version:       go1.16.6
   Git commit:       75249d8
   Built:            Fri Jul 30 19:52:10 2021
   OS/Arch:          linux/amd64
   Experimental:     false
  containerd:
   Version:          1.4.9
   GitCommit:        e25210fe30a0a703442421b0f60afac609f950a3
  runc:
   Version:          1.0.1
   GitCommit:        v1.0.1-0-g4144b63
  docker-init:
   Version:          0.19.0
   GitCommit:        de40ad0
```

`docker images`

Lists all available images in a docker.

Output should be similar to screenshot below:

```
MyMac:Downloads rd$ docker images
REPOSITORY                TAG       IMAGE ID       CREATED         SIZE
zyclonite/sysbench        latest    31638b096d0e   4 months ago    9.75MB
csminpp/ubuntu-sysbench   latest    2787c5e16909   6 years ago     336MB
```

`docker images –help`

Putting –help at the end of any command provides information related to that command, like syntax and available options.

Example output:

```
MyMac:Downloads rd$ docker images --help

Usage:  docker images [OPTIONS] [REPOSITORY[:TAG]]

List images

Options:
  -a, --all             Show all images (default hides intermediate images)
      --digests         Show digests
  -f, --filter filter   Filter output based on conditions provided
      --format string   Pretty-print images using a Go template
      --no-trunc        Don't truncate output
  -q, --quiet           Only show image IDs
```

`docker ps`
Gives list of containers that are running currently. It also provides option to get list of containers in other status like stopped.

`docker pull image`
To fetch docker image

`docker container run imageName`
Creates new container and starts it

`docker container stop containerName`
Stops container with name containerName

`docker container rm containerName`
Deletes docker container with name containerName

`docker image rm imageName`
Deletes docker image with name imageName

CPU test

I have run 3 CPU tests. For each test a different value of max-prime was used. I have run a CPU test using max-prime values of 10000, 20000 and 50000. Steps and test results for each platform are described in this section.

QEMU

qemu_cpu_test.sh contains the actual command and runs it 5 times. qemu_cpu_test_main.sh is the main script to perform cpu tests. It executes qemu_cpu_test.sh with parameters 10000, 20000, and 50000. These parameters will be passed to actual command which as below to check different test cases:

```
sysbench --test=cpu --cpu-max-prime=$prime run
```

Test was run using following command
```
sh qemu_cpu_test_main.sh > qemu_cpu_test.txt
```

Hence test results for QEMU test are in qemu_cpu_test.txt file

Docker

docker_cpu_test.sh contains the command to run the CPU test on docker and runs it five times for given parameters. docker_cpu_test_main.sh is main scripts which executes docker_cpu_test.sh with parameters same as QEMU, i.e., 10000, 20000, and 50000. Command for docker is as below:

```
docker run csminpp/ubuntu-sysbench sysbench --test=cpu --cpu-max-prime=$prime run
```

Docker test can be run using following command:
```
sh docker_cpu_test_main.sh > docker_cpu_test.txt
```

Results

Below we can see test results for both QEMU and docker using different parameters.
For each case performance of docker is better than QEMU. This could be partially because of better CPU but even with same CPU, OS virtualization is usually faster than system virtualization.

Also we can see that with increase in value of max-prime, timing has increased for both systems. This is simply because of increase in amount of work.

Test 1

max-prime = 10000		
	QEMU	Docker
Test-num	Total time (in s)	Total time (in s)
1	22.0822	9.1144
2	21.2081	9.685
3	32.413	10.0548
4	49.0274	9.0191

5	41.5024	9.0951
Min	21.2081	9.0191
Max	49.0274	10.0548
Avg	33.24662	9.39368
StdDev	12.11875	0.45536

Test 2

max-prime = 20000		
	QEMU	Docker
Test-num	Total time (in s)	Total time (in s)
1	70.0395	22.6574
2	105.1749	22.5828
3	100.0249	22.45
4	57.4144	23.0697
5	75.3548	23.0789
Min	57.4144	22.45
Max	105.1749	23.0789
Avg	81.6017	22.76776
StdDev	20.32759	0.28954

Test 3

max-prime = 50000		
max-prime = 50000	QEMU	Docker
Test-num	Total time (in s)	Total time (in s)
1	168.1985	76.7185
2	167.8651	78.192
3	168.7101	78.5719
4	169.3358	79.4365
5	171.7146	77.4922
Min	167.8651	76.7185
Max	171.7146	79.4365
Avg	169.16482	78.08222

StdDev	1.52974	1.03607

In the below screenshot we can see that OS vs kernel space usage of CPU is almost similar for the QEMU.

I wasn't able to get a screenshots from my host OS. because it was probably triggering some background activity, causing unnatural spike in test results. However, I generally observed that during usage of QEMU, user CPU usage spiked up. However during docker there was just small change in system CPU utilization.

File I/O test

I have run 4 file I/O tests. For each test a different value of num-threads and file-test-mode was used. Parameters that I have used for file I/O tests are as below:

- 8 threads seqwr mode (sequential write)
- 16 threads seqwr mode (sequential write)
- 8 threads rndwr mode (Random write)
- 16 threads rndwr mode (Random write)

Steps and test results for each platform are described in this section.

QEMU

qemu_fileio_test.sh contains the actual command and runs it 5 times for given set of parameters. qemu_fileio_test_main.sh is the main script to perform fileio tests. It executes qemu_fileio_test.sh with parameters described earlier. These parameters will be passed to set of commands which as below to check different test cases:

To prepare files for test

```
sysbench --num-threads=$threads --test=fileio --file-total-size=1G --file-test-mode=$mode prepare
```

To run the actual test

```
sysbench --num-threads=$threads --test=fileio --file-total-size=1G --file-test-mode=$mode run
```

To cleanup afterwards

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

Test was run using following command

```
sh qemu_fileio_test_main.sh > qemu_fileio_test.txt
```

Hence test results for QEMU test are in qemu_fileio_test.txt file

Docker

docker_fileio_test.sh contains the command to run the File I/O test on docker and runs it five times for given parameters. docker_fileio_test_main.sh is main scripts which executes docker_file_test.sh with parameters described earlier. Command for docker is as below:

To prepare files for test

```
docker run --rm csminpp/ubuntu-sysbench sysbench
--num-threads=$threads --test=fileio --file-total-size=1G
--file-test-mode=$mode prepare
```

To run actual test

```
docker run --rm csminpp/ubuntu-sysbench sysbench
--num-threads=$threads --test=fileio --file-total-size=1G
--file-test-mode=$mode --max-time=300 run
```

To do cleanup after test

```
docker run --rm csminpp/ubuntu-sysbench sysbench
--num-threads=$threads --test=fileio --file-total-size=1G
--file-test-mode=$mode cleanup
```

Docker test can be run using following command:

```
sh docker_fileio_test_main.sh > docker_fileio_test.txt
```

Results

Below we can see test results for both QEMU and docker using different parameters.

For each case performance of docker is better than QEMU. so performance of OS virtualization is better in terms of I/O as well.

Also with the increase in number of threads performance improved for random write in both systems. In sequential write performance did improve with change but the difference isn't significant.

Test 1

num-threads=8 and file-test-mode = seqwr				
	QEMU		Docker	
Test-num	Transfer rate(in Mb/s)	Request/sec	Transfer rate(in Mb/s)	Request/sec
1	32.523	2081.46	396.77	25393.38
2	33.235	2127.04	235.19	15051.89
3	32.652	2089.76	435.57	27876.6
4	34.2	2188.83	303.26	19408.66
5	34.38	2200.33	414.68	26539.35
Min	32.523	2081.46	235.19	15051.89
Max	34.38	2200.33	435.57	27876.6
Avg	33.398	2137.484	357.094	22853.976
StdDev	0.85969	55.02585	84.86386	5431.3915

Test 2

num-threads=16 and file-test-mode = seqwr				
	QEMU		Docker	
Test-num	Transfer rate(in Mb/s)	Request/sec	Transfer rate(in Mb/s)	Request/sec
1	33.889	2168.89	385.05	24643.2
2	32.803	2099.37	371.88	23800.21
3	33.053	2115.37	461.64	29545.13
4	34.081	2181.21	386.58	24741.3
5	33.063	2116	427.98	27390.62
Min	32.803	2099.37	371.88	23800.21
Max	34.081	2181.21	461.64	29545.13
Avg	33.3778	2136.168	406.626	26024.092
StdDev	0.56807	36.37619	37.27326	2385.53854

Test 3

num-threads=8 and file-test-mode = rndwr				
	QEMU		Docker	
Test-num	Transfer rate(in Mb/s)	Request/sec	Transfer rate(in Mb/s)	Request/sec
1	14.018	897.13	65.39	4184.96
2	13.978	894.58	63.598	4070.26
3	13.968	893.98	59.213	3789.62
4	13.724	878.32	51.868	3319.55
5	13.023	833.48	61.956	3965.21
Min	13.023	833.48	51.868	3319.55
Max	14.018	897.13	65.39	4184.96
Avg	13.7422	879.498	60.405	3865.92
StdDev	0.4184	26.77278	5.28565	338.28306

Test 4

num-threads=16 and file-test-mode = rndwr				
	QEMU		Docker	
Test-num	Transfer rate(in Mb/s)	Request/sec	Transfer rate(in Mb/s)	Request/sec
1	12.045	770.87	87.443	5596.36
2	13.818	884.37	75.815	4852.15
3	14.07	900.48	81.401	5209.65
4	13.786	882.33	80.995	5183.7
5	14.617	935.49	64.646	4137.33
Min	12.045	770.87	64.646	4137.33
Max	14.617	935.49	87.443	5596.36
Avg	13.6672	874.708	78.06	4995.838
StdDev	0.96602	61.83128	8.55559	547.56657

Conclusion

Performance of OS virtualization is better than system virtualization in general.