TECHNISCHE UNIVERSITAT BERLIN

Assignment 1

Cloud Computing

By

Group 18

Hafiz Muhammad Hamza
(M.Sc. Technische Informatik)

Waleed Butt
(M.Sc. Technische Informatik)

Muhammad Ahsan Shahid
(M.Sc. Technische Informatik)

Atique-ur-Rehman Shahid
(M.Sc. Technische Informatik)

Raafay Alam
(M.Sc. Technische Informatik)

AWS Setup

1. **Install AWS Client**

pip install --upgrade --user awscli

2. **Get Root Access Key**

We require “AWS Access Key” and “AWS Secret Access Key”.

i. Open the AWS Console
ii. Go to: Services > IAM > Manage Security Credential > Continue to Security Credential > Access Keys (Access Key ID and Secret Access Key)
iii. Click "Create New Access Key"
iv. Click "Download Key File"

3. **Configure AWS**

aws configure

AWS Access Key ID: <access-key-id>
AWS Secret Access Key : <secret-access-key>
Default region name: eu-central-1
Default output format: json

4. **Configure AWS Profile**

aws configure --profile <user-name>

5. **Create Group**

aws iam create-group --group-name <group-name>

Output:

```
"Groups": [
    {
      "Path": "/",
      "CreateDate": "2017-12-17T00:57:18Z",
      "GroupId": "<group-id>",
      "Arn": "arn:aws:iam::872800132220:group/<group-name>",
      "GroupName": "<group-name>"
    }
  ]
```

6. **Create User**

aws iam create-user --user-name <user-name>

Output:

```
"User": {
      "UserName": <name>,
      "Path": "/",
      "CreateDate": "2017-12-17T00:59:31.415Z",
      "UserId": "<user-id>",
      "Arn": "arn:aws:iam::872800132220:user/<user-name>"
   }
```

7. **Create Login Profile**

aws iam create-login-profile --generate-cli-skeleton > create-login-profile.json

8. **Create Access Key**

aws iam create-access-key--user-name <user-name>

Output:

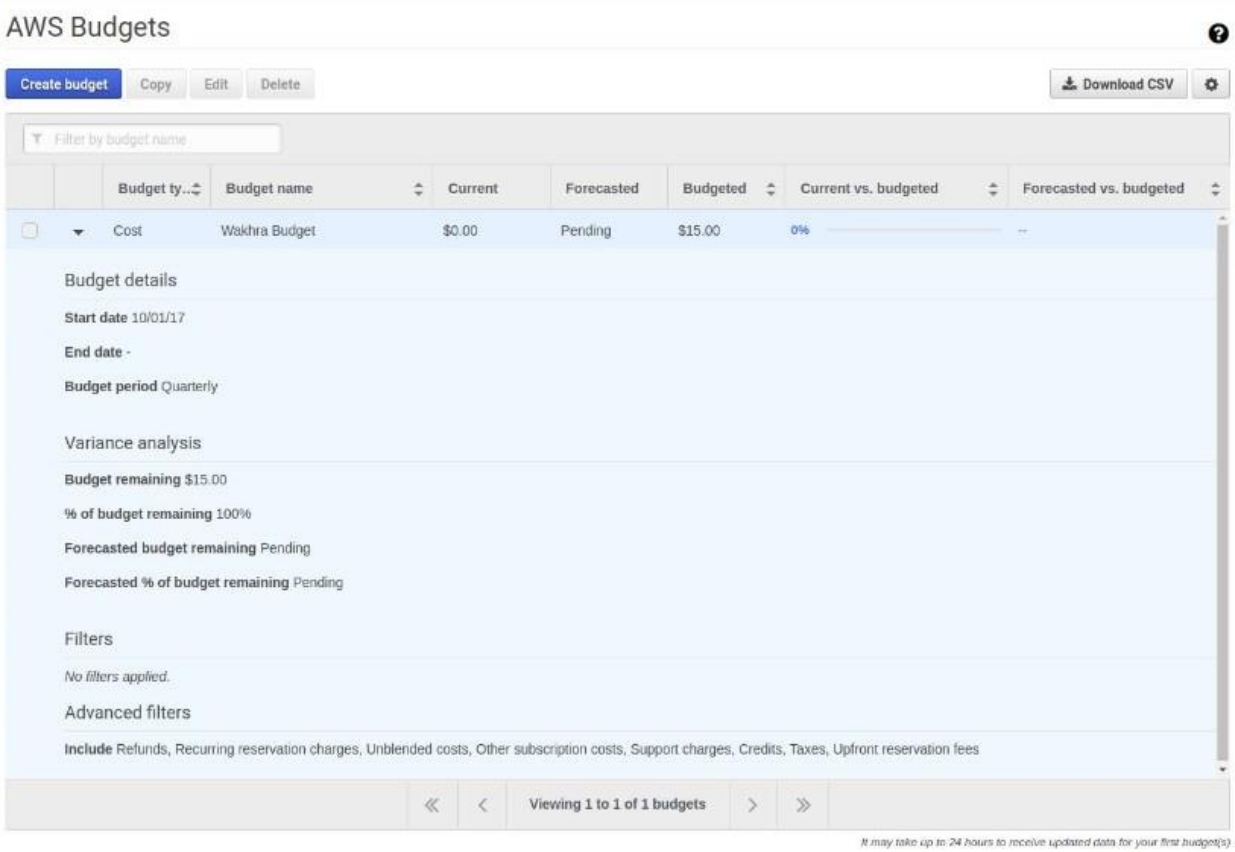
```
  "AccessKey": {
     "UserName": <name>,
     "Status": "Active",
     "CreateDate": "2017-12-17T01:02:17.742Z",
     "SecretAccessKey": "<secret-access-key>",
     "AccessKeyId": "<access-key-id>"
   }
```

9. **Add User to Group**

aws iam add-user-to-group --user-name <user-name> --group-name <group-name>

10. **Create Budget**

We created budget manually on the web interface console



11. Describe Budget

aws budgets describe-notifications-for-budget --account-id <aws accounted> --budget-name "Wakhra Budget"

Output:

```
"Notifications": [
    {
      "Threshold": 10.0,
      "ComparisonOperator": "GREATER_THAN",
      "ThresholdType": "ABSOLUTE_VALUE",
      "NotificationType": "ACTUAL"
    }
  ]
```

12. Create Security Group

aws ec2 create-security-group --group-name <group-name> --description "Security group for CC_Group18"

Output:

"GroupId": "*<group-id>*"

13. Authorize Security Group

aws ec2 authorize-security-group-ingress --group-name <group-name> --ip-permissions "IpProtocol"="tcp","FromPort"=22,"ToPort"=22,"UserIdGroupPairs"=[],"IpRanges"=[{"CidrIp"="0.0.0.0/0"}],"Ipv6Ranges"=[],"PrefixListIds"=[]

14. Create Key-Pair

aws ec2 create-key-pair --key-name <key-pair-name> --output text > <key-pair-file-name>.pem

15. Create AWS Instance

aws ec2 run-instances --image-id ami-f52bfa9a --count 1 --instance-type m3.medium --key-name <key-pair-name> --security-group-ids <security-group-id>

Note: Put *<security-group-id>* from Step# 12

Output:


```
"Instances": [
    {
      "Monitoring": {
        "State": "disabled"
      },
      "StateReason": {
        "Message": "pending",
        "Code": "pending"
      },
      "PublicDnsName": "",
      "RootDeviceType": "instance-store",
      "State": {
        "Code": 0,
        "Name": "pending"
      },
      "EbsOptimized": false,
      "LaunchTime": "2017-12-17T09:40:22.000Z",
      "PrivateIpAddress": "172.31.42.115",
      "ProductCodes": [],
      "VpcId": "vpc-d6f606bd",
      "StateTransitionReason": "",
      "InstanceId": "i-07f9bfba6cbb343d8",
      "ImageId": "ami-f52bfa9a",
      "PrivateDnsName": "ip-172-31-42-115.eu-central-1.compute.internal",
      "KeyName": "<key-name>",
      "SecurityGroups": [
        {
          "GroupName": "<group-name>",
          "GroupId": "<group-id>"
```

```
      }
    ],
    "ClientToken": "",
    "SubnetId": "subnet-7fd82102",
    "InstanceType": "m3.medium",
    "NetworkInterfaces": [
      {
        "Status": "in-use",
        "MacAddress": "06:eb:4e:3c:84:a4",
        "SourceDestCheck": true,
        "VpcId": "vpc-d6f606bd",
        "Description": "",
        "NetworkInterfaceId": "eni-c53783ee",
        "PrivateIpAddresses": [
          {
            "PrivateDnsName": "ip-172-31-42-115.eu-central-1.compute.internal",
            "Primary": true,
            "PrivateIpAddress": "172.31.42.115"
          }
        ],
        "PrivateDnsName": "ip-172-31-42-115.eu-central-1.compute.internal",
        "Attachment": {
          "Status": "attaching",
          "DeviceIndex": 0,
          "DeleteOnTermination": true,
          "AttachmentId": "eni-attach-e7d0a10c",
          "AttachTime": "2017-12-17T09:40:22.000Z"
        },
        "Groups": [
          {
            "GroupName": "<group-name>",
            "GroupId": "<group-id>"
          }
        ],
        "Ipv6Addresses": [],
        "OwnerId": "<owner-id>",
        "SubnetId": "subnet-7fd82102",
        "PrivateIpAddress":  "172.31.42.115"
      }
    ],
    "SourceDestCheck": true,
    "Placement": {
      "Tenancy": "default",
      "GroupName": "",
      "AvailabilityZone": "eu-central-1b"
    },
    "Hypervisor": "xen",
    "BlockDeviceMappings": [],
    "Architecture": "x86_64",
    "KernelId": "aki-931fe3fc",
    "VirtualizationType": "paravirtual",
    "AmiLaunchIndex": 0
}
```

```
],
"ReservationId": "r-07bb0b27af0e7f7bb",
"Groups": [],
"OwnerId": "<owner-id>"
```

16. Unfortunately, we can't proceed further with uploading the files and connecting to the instance because we faced some errors like Public key error and some problems with AWS account

OpenStack Setup

17. Install OpenStack
sudo apt install python-openstackclient

18. Create *openrc.sh* File
touch cc1718-group18-openrc.sh

19. Define OpenStack Credentials (copy the commands below to the openrc.sh file)
export OS_USERNAME="<username>"
export OS_TENANT_NAME="<tenant-name>"
export OS_PASSWORD="<password>"
export OS_AUTH_URL=http://cloud.cit.tu-berlin.de:5000/v2.0

20. Run the openrc.sh File
source ./ cc1718-group18-openrc.sh.sh

21. Create Key
ssh-keygen -t rsa -f <key-pair-file-name> .key

22. Upload Public Key
openstack keypair create --public-key ./ *<key-pair-file-name>* .key.pub *<key-pair-name>*

23. Create Security Group
openstack security group create <security-group-name> --description "Security group for CC Group-18"

24. List Default Security Group Rules
openstack security group rule list <security-group-name>

25. Define Security Group Rules

openstack security group rule create <security-group-name> \--protocol tcp --dst-port 22:22 --remote-ip 0.0.0.0/0

26. Create Security Group Rules

openstack security group rule create --protocol icmp \<security-group-name>

27. List Existing Flavors

openstack flavor list

Output: 44b9e099-91d8-4106-ac42-316ef94c3c55

28. Find OpenStack Image List

Openstack image list

Output: 11f6b8aa-31df-4b66-8b42-5ee9760c47ba

29. Launch OpenStack Instance

openstack server create --flavor <flavor-id> --image <image-id> --security-group <security-group-name> --key-name <key-name> <openstack-instance-name>

Note: Put *<flavor-id>* from Step# 11 and *<image-id>* from Step# 12

30. Find a Floating IP

openstack floating ip list

31. Add floating IP

openstack server add floating ip <openstack-instance-name> <tuberlin-floating-ip>

Note: In Our case, *<tuberlin-floating-ip>* was 10.200.2.134

32. Copy Files to OpenStack Instance

scp -i <key-pair-file-name>.key linpack.c ubuntu@ <tuberlin-floating-ip> :~/
scp -i <key-pair-file-name>.key linpack.sh ubuntu@ <tuberlin-floating-ip> :~/
scp -i <key-pair-file-name>.key memsweep.c ubuntu@ <tuberlin-floating-ip> :~/
scp -i <key-pair-file-name>.key memsweep.sh ubuntu@ <tuberlin-floating-ip> :~/

33. Connect to the Instance

ssh -i <key-pair-file-name>.key ubuntu@ <tuberlin-floating-ip>

34. Install Tools

sudo apt-get install gcc
sudo apt-get install fio

Doing Benchmarks (commands)

- **CPU Benchmark**

source linpack.sh

- **Memory Benchmark**

source memsweep.sh

- **Disk Benchmark**

i. **Sequential**

Write Benchmark: time sh -c "dd if=/dev/zero of=test.tmp bs=4k count=2000000 && sync"

Clear Cache: dd if=/dev/zero of=clearcache.tmp bs=4k count=500000

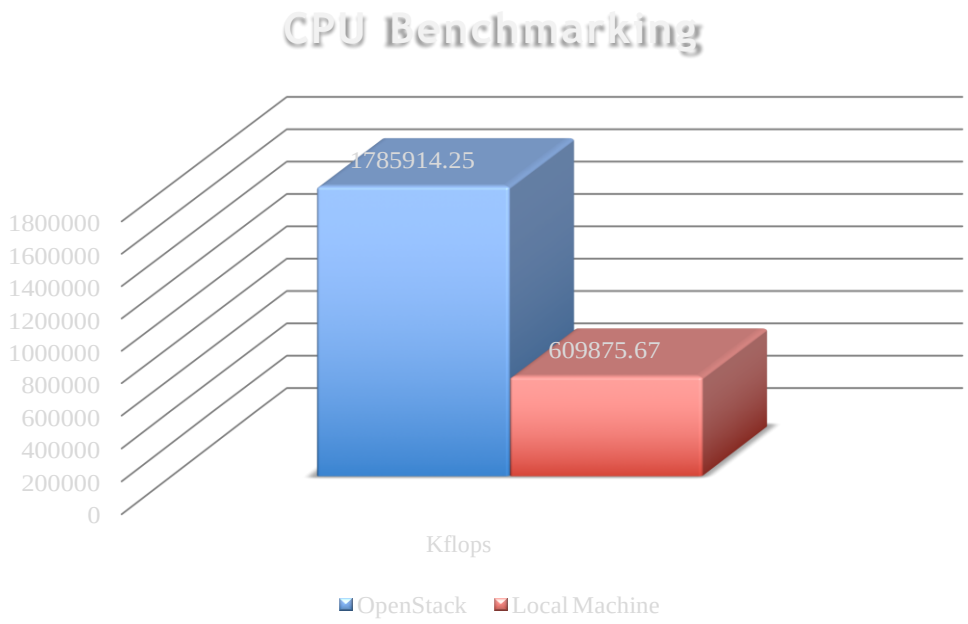
Read Benchmark: time sh -c "dd if=test.tmp of=/dev/null bs=4k"

ii. **Random**

fio --randrepeat=1 --ioengine=libaio --direct=1 --gtod_reduce=1 --name=test --filename=test --bs=4k --iodepth=64 --size=256M --readwrite=randrw –rwmixread=75

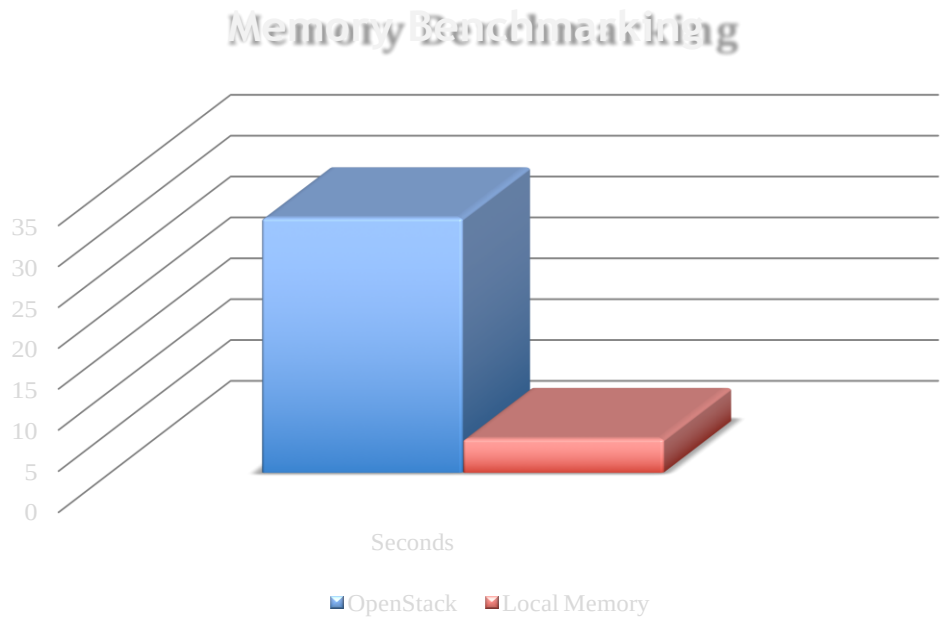
Benchmark Results

- **CPU Results (linpack.sh)**



OpenStack: 1785914.250 KFLOPS
Local Machine (with SSD): 609875.670 KFLOPS

- **Memory Results (memsweep.sh)**

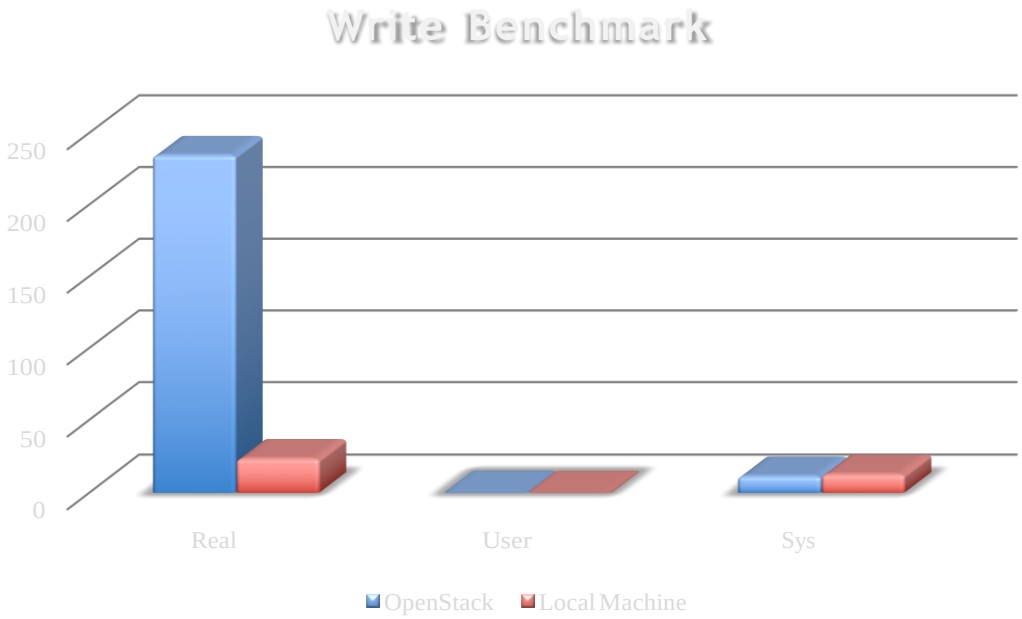


OpenStack: Memsweep time in seconds: 31.125
Local Machine (with SSD): Memsweep time in seconds: 4.131

- **Disk Results**

i. **Sequential**

Write Benchmark



OpenStack

2000000+0 records in
2000000+0 records out
8192000000 bytes (8.2 GB, 7.6 GiB) copied, 231.997 s, 35.3 MB/s

real 3m54.358s
user 0m0.424s
sys 0m11.136s

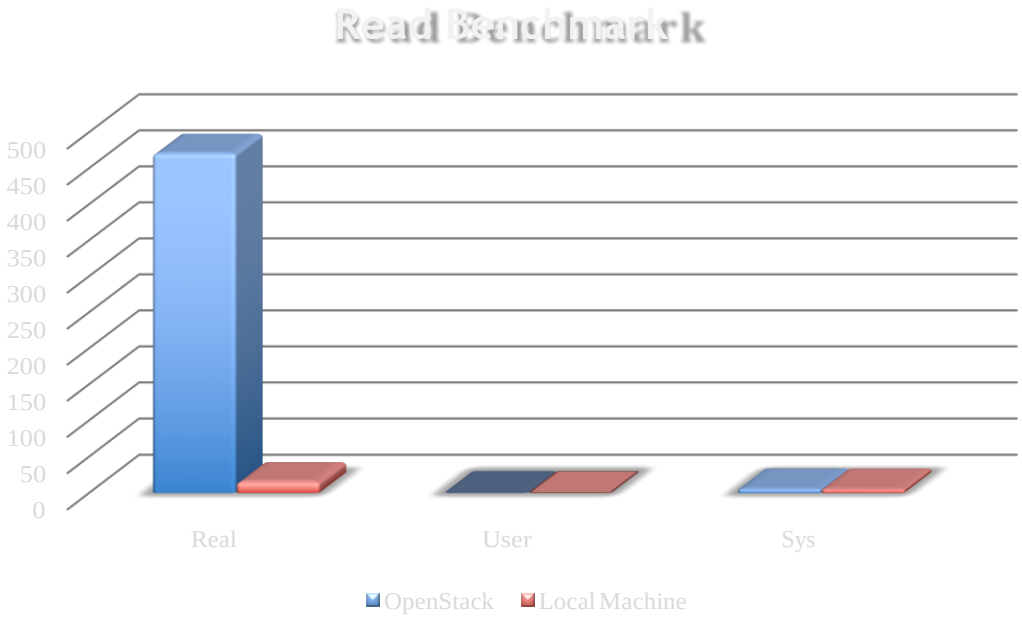
Local Machine (with SSD)

2000000+0 records in
2000000+0 records out
8192000000 bytes (8,2 GB, 7,6 GiB) copied, 22,795 s, 359 MB/s

real 0m23.339s

user 0m0.336s
sys 0m12.600s

Read Benchmark



OpenStack

2000000+0 records in
2000000+0 records out
8192000000 bytes (8.2 GB, 7.6 GiB) copied, 470.583 s, 17.4 MB/s

real 7m50.627s
user 0m0.256s
sys 0m5.692s

Local Machine (with SSD)

2000000+0 records in
2000000+0 records out
8192000000 bytes (8,2 GB, 7,6 GiB) copied, 15,3753 s, 533 MB/s

real 0m15.383s

```
user    0m0.312s
sys     0m5.444s
```

ii. **Random**

OpenStack

```
Jobs: 1 (f=1): [m(1)] [100.0% done] [1520KB/424KB/0KB /s] [380/106/0 iops] [eta 00m:00s]
test: (groupid=0, jobs=1): err= 0: pid=15981: Sat Dec 16 18:13:20 2017
  read : io=196444KB, bw=567827B/s, iops=138, runt=354260msec
  write: io=65700KB, bw=189907B/s, iops=46, runt=354260msec
  cpu        : usr=0.10%, sys=0.49%, ctx=61226, majf=0, minf=9
  IO depths  : 1=0.1%, 2=0.1%, 4=0.1%, 8=0.1%, 16=0.1%, 32=0.1%, >=64=99.9%
     submit  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     complete : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.1%, >=64=0.0%
     issued  : total=r=49111/w=16425/d=0, short=r=0/w=0/d=0, drop=r=0/w=0/d=0
     latency  : target=0, window=0, percentile=100.00%, depth=64

Run status group 0 (all jobs):
   READ: io=196444KB, aggrb=554KB/s, minb=554KB/s, maxb=554KB/s, mint=354260msec,
maxt=354260msec
  WRITE: io=65700KB, aggrb=185KB/s, minb=185KB/s, maxb=185KB/s, mint=354260msec,
maxt=354260msec

Disk stats (read/write):
  vda: ios=49079/16555, merge=0/73, ticks=16510992/5506596, in_queue=22020976,
util=100.00%
```

Local Machine (with SSD)

```
test: (g=0): rw=randrw, bs=4K-4K/4K-4K/4K-4K, ioengine=libaio, iodepth=64
fio-2.2.10
Starting 1 process
test: Laying out IO file(s) (1 file(s) / 256MB)
```

```
test: (groupid=0, jobs=1): err= 0: pid=3353: Sat Dec 16 18:06:34 2017
  read : io=196444KB, bw=270957KB/s, iops=67739, runt=  725msec
  write: io=65700KB, bw=90621KB/s, iops=22655, runt= 725msec
  cpu        : usr=19.34%, sys=38.12%, ctx=54789, majf=0, minf=9
  IO depths  : 1=0.1%, 2=0.1%, 4=0.1%, 8=0.1%, 16=0.1%, 32=0.1%, >=64=99.9%
     submit  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     complete : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.1%, >=64=0.0%
     issued  : total=r=49111/w=16425/d=0, short=r=0/w=0/d=0, drop=r=0/w=0/d=0
     latency  : target=0, window=0, percentile=100.00%, depth=64

Run status group 0 (all jobs):
   READ: io=196444KB, aggrb=270957KB/s, minb=270957KB/s, maxb=270957KB/s,
mint=725msec, maxt=725msec
  WRITE: io=65700KB, aggrb=90620KB/s, minb=90620KB/s, maxb=90620KB/s, mint=725msec,
maxt=725msec

Disk stats (read/write):
  sda: ios=41955/14131, merge=185/16, ticks=29980/8180, in_queue=38152, util=86.38%
```

Disk Benchmark Questions

1. **Look at the disk measurements. Are they consistent with your expectation? If not, what could be the reason?**

   We were expecting the slower results for IO-Access or for data read and write cycles from paravirtualized machine due to the overhead of every write access validation. So sequential read and write test was exploited for open stack in case of random access. This will be because of the different data storage methods that support sequential access.

2. **Compare the results for the two operations (sequential, random). What are reasons for the differences?**

CPU Benchmark Questions

1. **Look at linpack.sh and linpack.c and shortly describe how the benchmark works?**

- The LINPACK Benchmark measure the system floating point computing power. The LINPACK benchmark performance can provide a good correction over the peak performance. So this measures that how fast a computer solves linear algebra calculations, LU decomposition and solving a system of linear equations. The result of this computations are returned in floating point operations per second (FLOPS).
- The LINPACK benchmark features solving a system of linear equation. For this it uses a matrix problem. Here programs take a matrix of large number of rows and columns.
- Software main two functions are as bellow:
  - DGEFA: Performs the decomposition with partial pivoting.
  - DGESL: Uses that decomposition to solve the given system of linear equations.
- Most of the computations cycles is for DGEFA in decomposition of matrix consisting of problem, DGESL is used to find the solution this requires floating-point operations.
- DGEFA and DGESL call other routines DAXPY, IDAMAX and DSCAL.
- 90% of execution is for subroutine DAXPY.
- DAXPY is used to multiply a scalar, times a vector, and add the results to another vector.
- Thus, the benchmark requires roughly 2/3 million floating-point operations for the order of 100 X 100 order matrix.
- The operations also include a few one-dimensional index operations and storage references. The LINPACK routines in general have been organized to access two-dimensional arrays by column.
- In DGEFA, the call to DAXPY passes an address into the two-dimensional array A, which is then treated as a one-dimensional reference within DAXPY. Since the indexing is down a column of the two-dimensional array, the references to the one-dimensional array are sequential with unit stride.

2. **Find out what the LINPACK benchmark measures (try Google). Would you expect paravirtualization to affect the LINPACK benchmark? Why?**

LINPACK benchmark measures floating point computation. All commands of LINPACK are unprivileged instructions to solve linear equation problem so paravirtualization will not affect the overall working of LINPACK benchmarking. And also in paravirtualization environment host is aware of this fact that he is running in virtualized environment. So in short these tasks involve only unprivileged instructions and are basically all executed on the CPU so paravirtualization will not affect the benchmark results as long as the virtualized machine has dedicated access to the host's resources.

3. **Look at the LINPACK measurements. Are they consistent with your expectation? If not, what could be the reason?**

And there are basic 3 types of instances small, medium, large and in medium sort of instance single core processor is available to every host for computation so that's whys the result is consistent so far. And in paravirtualized environment there is a performance increase as less overhead by hypervisor. The reason for this result could be that the benchmark only runs on a single core. This would explain the consistent results when computing resources kept constant.

So open stack are consuming more KFLOPS and is leading in this benchmark and results are consistent as configuration and computational resources are not changed and also there were not such reasons that were responsible of exploiting results.

Memory Benchmark Questions

1. **Find out how the memsweep benchmark works by looking at the shell script and the C code. Would you expect virtualization to affect the memsweep benchmark? Why?**

This measures the required time to access heap memory at different locations. The step is chosen that caches miss occurs and the data has to be reloaded from the memory. Because the hypervisor still needs to validate write requests so an overhead to memory calls is expected here in case of write cycle where hypervisor has to validate the request in case of ache miss. Moreover, differences between the paravirtualized machines should be the result of different memory and a different clock speed of the actual CPU.

2. **Look at your memsweep measurements. Are they consistent with your expectations? If not, what could be the reason?**

The sweep time of the local machine is considerably slower than the results of the paravirtualized machines. This is somehow surprising and implies that the Amazon and open stack hosts have a much faster memory than the local machine. The main reason can be because of different hardware configurations.
Memory benchmark was not so much consistent in case of local machine but is very consistent in paravirtualized environment. And this will be because of the smart management of hypervisor in managing cache miss and in controlling unprivileged calls on domain level.