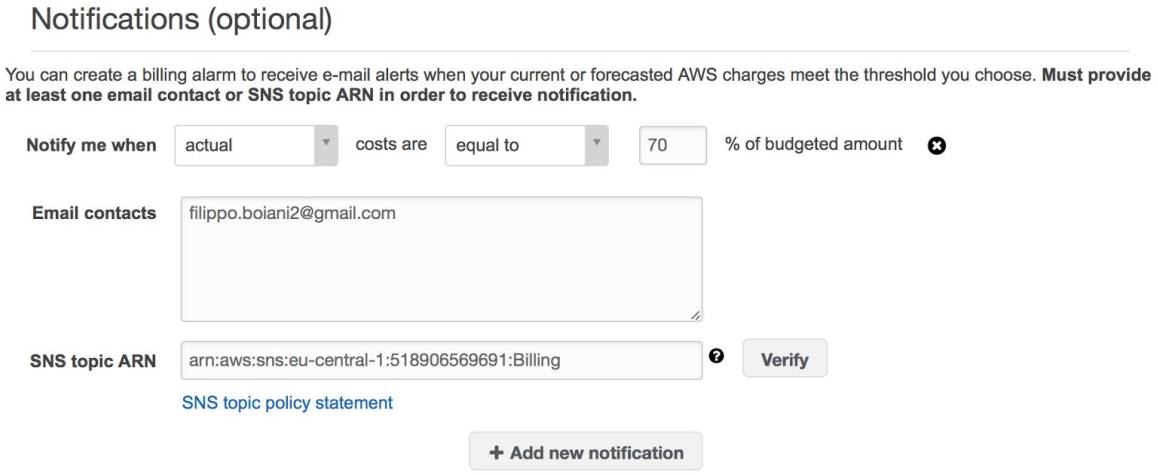
Cloud Computing SS2017 - Assignment 1

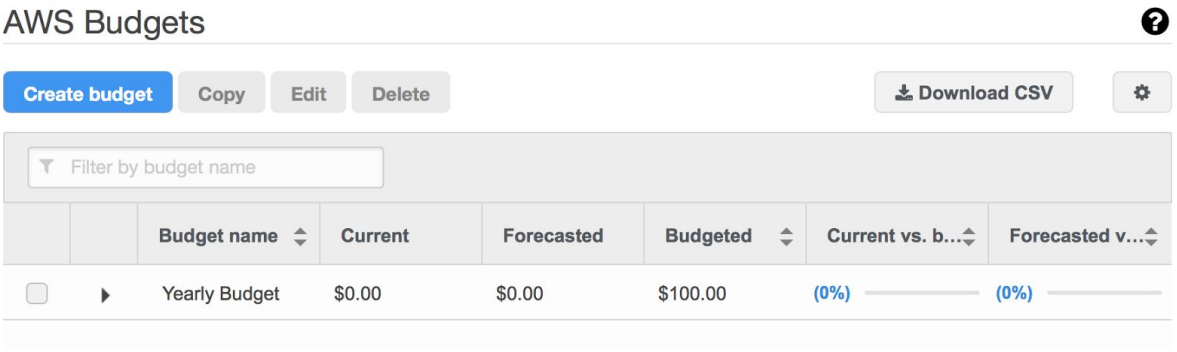
Group 32

- Filippo Boiani - 387680
- Piotr Mrówczyński - 387521
- Riccardo Sibani - 382708
- Stefan Stojkovski - 387529
- Gabriel Vilén - 387555

AWS Budget



Budget created.



Preparing AWS Instance

Installation of AWS CLI:

```
$ curl "https://s3.amazonaws.com/aws-cli/awscli-bundle.zip" -o
"awscli-bundle.zip"
```

```
$ unzip awscli-bundle.zip
$ sudo ./awscli-bundle/install -i /usr/local/aws -b /usr/local/bin/aws
```

In order to check the installation is possible to run:

```
$ aws --version
```

Set up AWS Environment

Run AWS Configure and set up the user credentials, the region where to run the instances and the output format (in this case I prefer Json)

```
$ aws configure
> aws access key Your access key
> aws secret key Your secret key
> default region name: eu-central-1
> Default output format [None]: json
```

Create a key pair from EC2 and we are able to have an asymmetric key.

```
$ aws ec2 create-key-pair --key-name group32-key --query 'KeyMaterial' --output
text > group32-key.pem
```

And create a security group within the network of TU Berlin

```
$ aws ec2 create-security-group --group-name group32-securityGroup --description
"SSH only from TU Berlin VPN"
>
{
    "GroupId": "new_group_id"
}
```

Authorize the security group and open port 22, so it is possible to reach the instance via SSH

```
$ aws ec2 authorize-security-group-ingress --group-name group32-securityGroup
--cidr 130.149.0.0/16 --protocol tcp --port 22
```

The instance the assignment suggest to run is “ami-f603d399”, however, in case we want to see other instances with the requirements specified in the assignment:

```
$ aws ec2 describe-images --owners amazon \
    --filters "Name=architecture, Values=x86_64" "Name=virtualization-type,
Values=paravirtual" "Name=root-device-type, Values=instance-store"
```

And then run the selected image

```
$ aws ec2 run-instances --image-id ami-f52bfa9a --count 1 --instance-type
m3.medium --key-name group32-key --security-groups group32-securityGroup
```

From the output that will come out, it is useful to annotate the InstanceId, however, is possible to retrieve it with

```
$ aws ec2 describe-instances --filter "Name=instance-state-name,Values=running"
```

And check out the public ip

```
$ aws ec2 describe-instances --instance-id i-02438514d865477f5
```

which is under the name of “PublicIp”
And connect via SSH

```
$ sudo ssh -i group32-key.pem ec2-user@public_ip
```

To terminate the instance

```
$ aws ec2 terminate-instances --instance-ids IntanceId
```

Preparing Openstack Instance

In order to prepare the instance we followed the process listed beneath (after having installed openstack cli and loaded the variables inside the environment).

Security group list

```
$ openstack security group list
$ openstack security group rule list default
```

Add a rule to the default security group: enable SSH and ICMP access.
SSH is restricted to the TU VPN (CIDR 130.149.0.0/16)

```
$ openstack security group rule create default --protocol tcp --dst-port
22:22 --remote-ip 130.149.0.0/16
$ openstack security group rule create --protocol icmp default
```

Key pair generation on local machine and upload the public key with the name `group32-key`

```
$ ssh-keygen -t rsa -f /.ssh/cloud.key
$ openstack keypair create --public-key ~/.ssh/cloud.key.pub group32-key
```

List the available images

```
$ openstack image list
$ openstack image show ubuntu-16.04
```

Create an instance with name `group32-instance`

```
$ openstack server create --flavor 604de11c-3222-4902-8523-11cc61b5b485
--image ubuntu-16.04 --nic net-id=cc17-net --security-group default
--key-name group32-key group32-instance

+-----------------------------+------------------------------------------------------------+
| Field                       | Value                                                      |
+-----------------------------+------------------------------------------------------------+
| OS-DCF:diskConfig           | MANUAL                                                     |
| OS-EXT-AZ:availability_zone |                                                            |
| OS-EXT-STS:power_state      | NOSTATE                                                    |
| OS-EXT-STS:task_state       | scheduling                                                 |
| OS-EXT-STS:vm_state         | building                                                   |
| OS-SRV-USG:launched_at      | None                                                       |
| OS-SRV-USG:terminated_at    | None                                                       |
| accessIPv4                  |                                                            |
| accessIPv6                  |                                                            |
| addresses                   |                                                            |
| adminPass                   | m5Szeik7XeVT                                               |
| config_drive                |                                                            |
| created                     | 2017-06-08T17:40:52Z                                       |
| flavor                      | Cloud Computing (604de11c-3222-4902-8523-11cc61b5b485)     |
| hostId                      |                                                            |
| id                          | 0d5de66b-7d29-4006-9cf8-c46d7f78ceae                       |
| image                       | ubuntu-16.04 (11f6b8aa-31df-4b66-8b42-5ee9760c47ba)        |
| key_name                    | group32-key                                                |
| name                        | group32-instance                                           |
| progress                    | 0                                                          |
| project_id                  | 8ddfeafe5eba40508d0ba9bff4aabdb9                           |
| properties                  |                                                            |
| security_groups             | name='default'                                             |
| status                      | BUILD                                                      |
| updated                     | 2017-06-08T17:40:53Z                                       |
| user_id                     | b7a28df7727b409d9b8f9696f9c87025                           |
| volumes_attached            |                                                            |
+-----------------------------+------------------------------------------------------------+
```

(the alpha-numeric code is the id of "`Cloud Computing`" flavor while the network the instance should be attached to is `cc17-net`)

Show and assign floating IPs to an instance

```
$ openstack floating ip list
$ openstack server list
$ openstack server add floating ip group32 10.200.1.199
```

Start and show the instance

```
$ openstack server start group32-instance
$ openstack server show group32-instance
```

connecting via SSH

```
$ ssh -i ~/.ssh/cloud.key ubuntu@10.200.1.199
```

Benchmarking - Introduction

For benchmarking and plotting we used python script (Appendix 2), which on local machine have been first transporting via SCP required scripts to remote VMs, and then executing iteratively via SHH all tests on local, aws and openstack VMs.

Model	**vCPU/CPU**	**Mem**	**Storage**
AWS m3.medium	**1**	**4GB**	**instance-store**
Openstack	**1**	**500MB**	**HDD**
Intel Core i7-4510U @ 2.00GHz	**2/4**	**8GB**	**HDD**

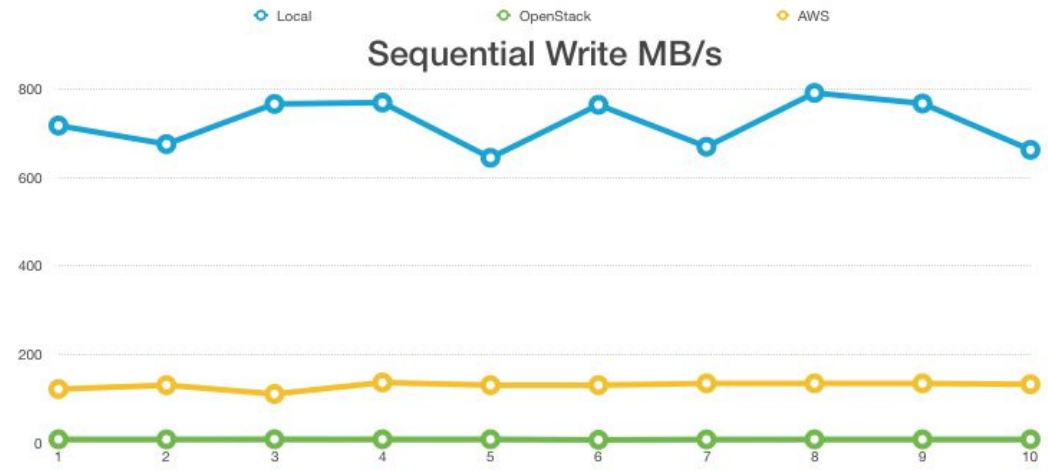
Disk Benchmarking

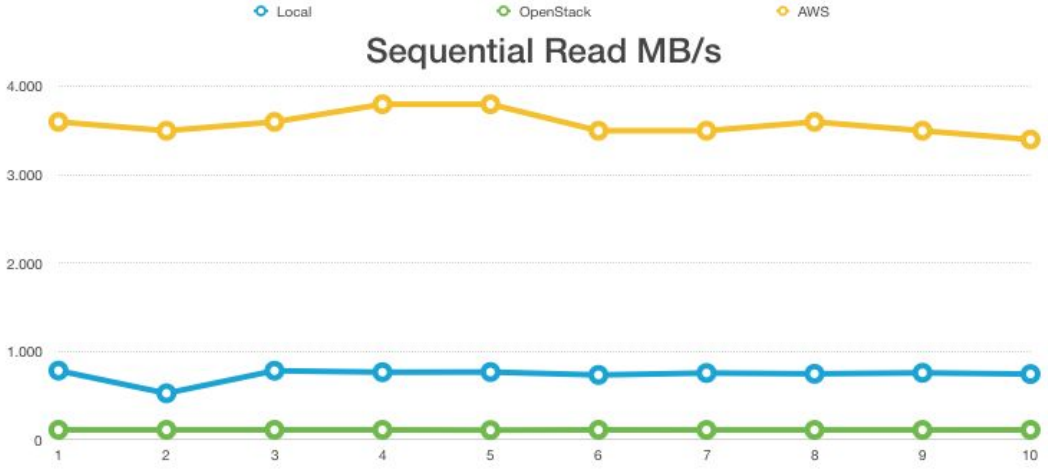
Approach

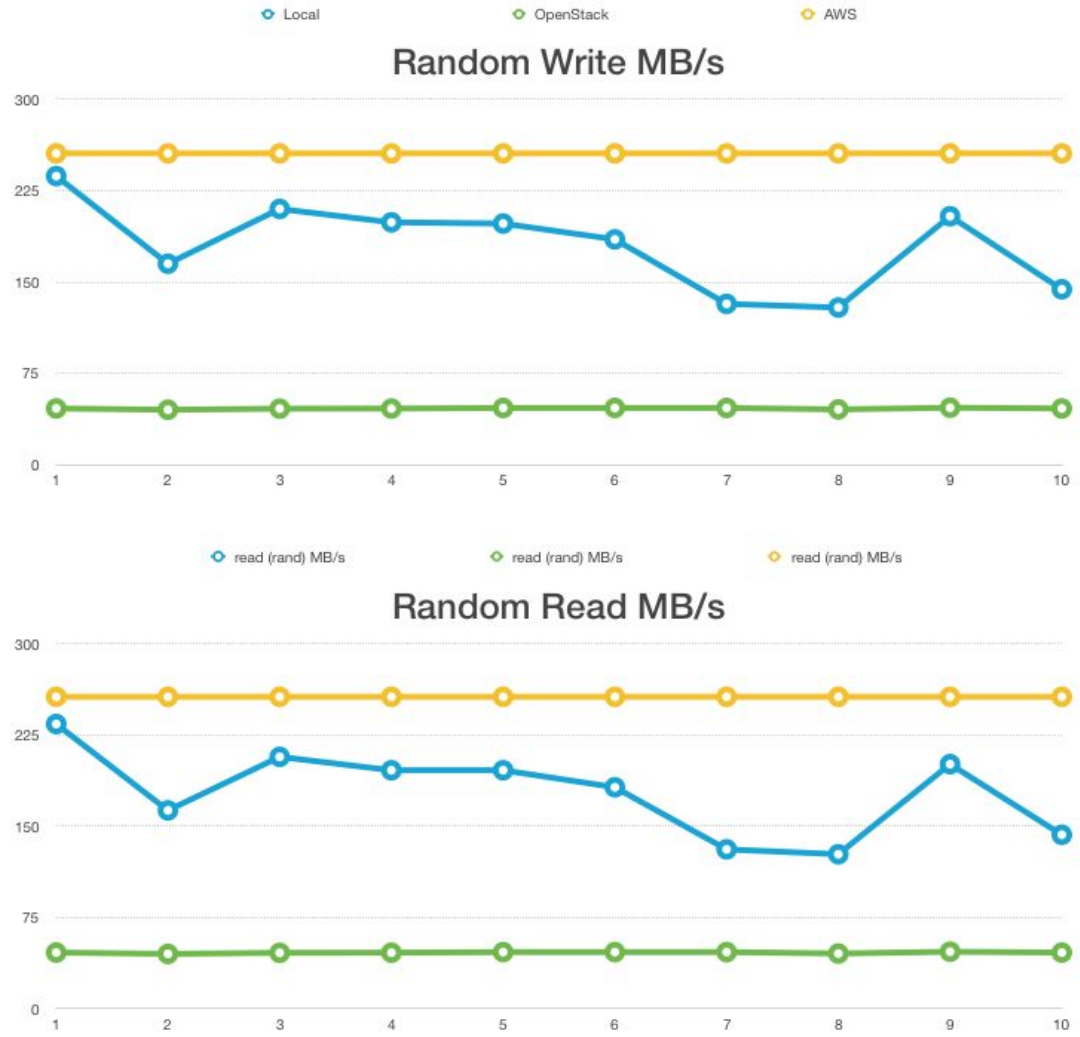
We created a .sh file running `fio` and `dd` (see Appendix 1) which was also writing on two different output file (Appendix 1 outputs). Both fio and dd commands read from and write to a 512MB file. The script along with .fio config file was uploaded on Dropbox and downloaded on the instance. The results (dd_tests.txt and fio_tests.txt) were downloaded with SCP. We ran the test 10 times to avoid possible outliers and we considered the average of the 10 measurements.

Results

Participant	write (seq) MB/s	Read (SEQ) MB/s	Write (rand) MB/s	read (rand) MB/s
Local	723	736	180	178
OpenStack	7	111	46	46
Amazon Web Services	129	3.580	256	256


Local
OpenStack
AWS
Sequential Write MB/s
800
600
400
200
0
1
2
3
4
5
6
7
8
9
10


Local
OpenStack
AWS
Sequential Read MB/s
4.000
3.000
2.000
1.000
0
1
2
3
4
5
6
7
8
9
10



Answers

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

At the beginning, sequential I/O performances were incredibly high due to the fact that the test file was written on and read from memory and not on/from disk. As we can see in the figure below, the writing speed was around 2.4 GB/s on Openstack (and up to 5.8 GB/s on the local machine) after the first test file creation.

```
$
Write
dd: error writing 'tstfile': No space left on device
239+0 records in
238+0 records out
250540032 bytes (251 MB, 239 MiB) copied, 0.445149 s, 563 MB/s
Command exited with non-zero status 1
0.00user 0.46system 0:00.47elapsed 97%CPU (0avgtext+0avgdata 3120maxresident)k
144inputs+0outputs (1major+338minor)pagefaults 0swaps
$
Read
238+1 records in
238+1 records out
250540032 bytes (251 MB, 239 MiB) copied, 0.104897 s, 2.4 GB/s
0.00user 0.10system 0:00.10elapsed 97%CPU (0avgtext+0avgdata 3112maxresident)k
0inputs+0outputs (0major+339minor)pagefaults 0swaps
```

Then we introduced first command to flush the cache, then a sync command and the tests run as expected.

```
dd bs=1M count=256 if=/dev/zero of=test conv=fdatasync
```

This tells dd to require a complete “sync” once, right before it exits. So it commits the whole 256 MB of data, then tells the operating system: “OK, now ensure this is completely on disk”, only then measures the total time it took to do all that and calculates the benchmark result.

Apparently the cache and memory flush command wasn’t working on the AWS instance since we kept getting 450MB/s in writing (SSD-like speed) and around 3.5GB in read (comparable to a read operation from memory)

2 - Based on the comparison with the measurements on your local hard drive, what kind of storage solutions do you think the two clouds use?
The machine used for local testing is a Macbook Pro late 2013 with 512GB SSD and the performances are in line with Third-party tests using the Blackmagic benchmark reportign write speeds of 673.5 MB/s and read speeds of 731 MB/s. (source http://www.everymac.com).

Due to the low performances in I/O and the great difference between sequential read and write operation we assume that Openstack instance uses an HDD.

Due to the low performances in I/O and the great difference between sequential read and write operation we assume that AWS instance uses an SSD. However the high speed in reading of the instance might suggest some caching system.

CPU Benchmarking

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

LINPACK benchmark executes linpack algorithm (various systems of linear equations) repeating it twice more times with each iteration until the limit of 10s is reached.

Internally, it executes:
- MATGEN which generates random matrix
- DGEFA which factors a REAL matrix (partial decomposition with real pivoting)
- DGESL which solves general linear system A * X = B.

2. Find out what the LINPACK benchmark measures (try Google).

LINPACK Measurement shows how much percent of total time is spent in DGEFA decomposition, percentage of time in DGESL solving of equations and measurement overhead for preparation.

$$\frac{1\ \text{operation}}{1\ \text{cycle}} * 750\text{MHz} = 750\text{Mflop/s}.$$

Performance is measured in terms of KiloFlops (KFLOPS), number of floating point operations per second (which usualy include both additions and multiplications in the count), that can be completed in a period of time, usually the cycle time of the machine.

```
Memory required:  7824K.


LINPACK benchmark, Double precision.
Machine precision:  15 digits.
Array size 1000 X 1000.
Average rolled and unrolled performance:

    Reps Time(s) DGEFA   DGESL  OVERHEAD    KFLOPS
----------------------------------------------------
       8   0.52  96.05%   0.59%   3.36%  2688937.268
      16   1.04  95.96%   0.65%   3.39%  2667048.431
      32   2.08  96.04%   0.60%   3.36%  2674525.411
      64   4.16  96.05%   0.60%   3.35%  2671682.711
     128   8.31  96.06%   0.59%   3.35%  2673709.739
     256  16.67  96.04%   0.61%   3.35%  2664447.011
```

Exemplary, for the Intel Core i7-4510U @ 2.00GHz with Turbo mode of 3.00GHz, we received 2.67GFlops

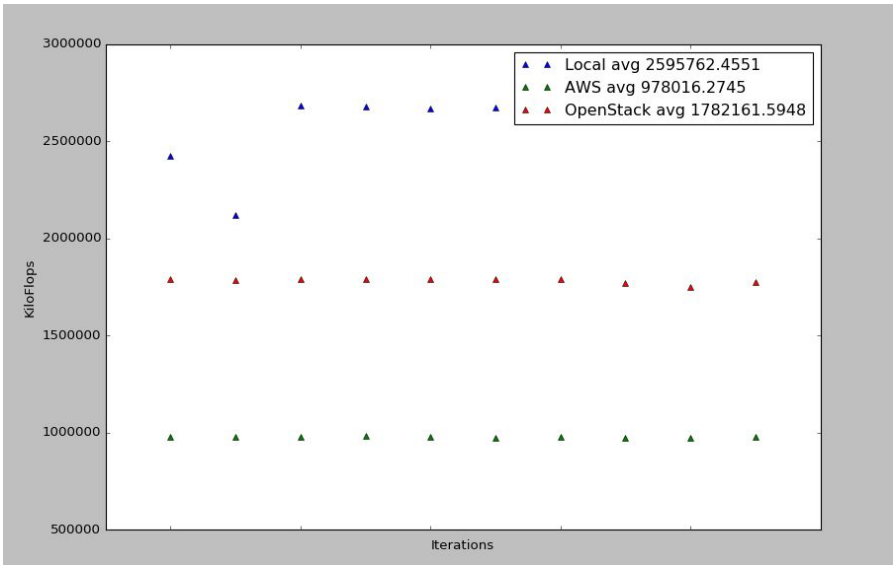
3. Would you expect paravirtualization to affect the LINPACK benchmark? Why?

Paravirtualization involves modifying the OS kernel to replace nonvirtualizable instructions with hypercalls that communicate directly with the virtualization layer hypervisor, thus if LINPACK benchmark contains privileged instructions it might experience the overhead, however smaller one compared to virtualization.

4. Look at your LINPACK measurements. Are they consistent with your expectations? If not, what could be the reason?

Paravirtualization involves modifying the OS kernel to replace nonvirtualizable instructions with hypercalls that communicate directly with the virtualization layer hypervisor and provides hypercall interfaces for some critical kernel operations (memory management, interrupt handling and time keeping), thus it should be faster than full-virtualization which involves emulation overhead (binary transation), however slower than native local hardware, due to overhead of communication with virtualization layer hypervisor and sharing resources with other users. Performance between full and para virtualization also depends on the workload.

Test executed on Friday 16th June at 13:00



Benchmark shows, that On-Premise OpenStack is performing more floating points operations per second then Public AWS Cloud. It might be expected, since less people work on on-premise cloud, thus sharing resources with less users, and offerring better performance.

Memory Benchmarking

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?

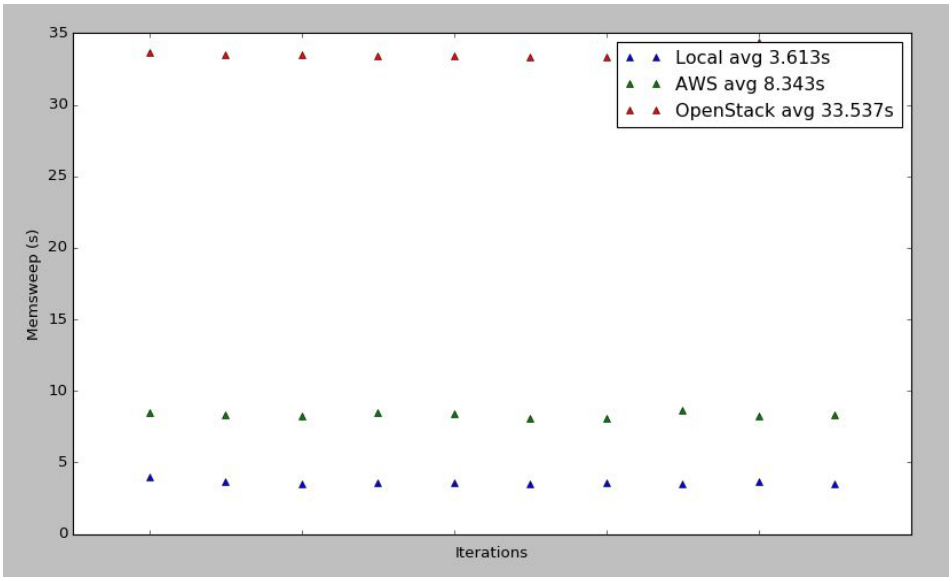
Since AWS uses XEN virtualization and Openstack also uses para-virtualization, we do not expect virtualization to affect the memsweep benchmark for the read part because XEN lets guests maintain their own page tables and guest page tables are visible to the MMU because the guest OS is aware of the virtualization.

However since memsweep benchmark also does writes, for this XEN intervention is required because XEN must validate write requests to ensure isolation of the fractions of physical memory allocated to each VM.

array = (char*) malloc(sizeof(char)*ARR_SIZE);
This command would need hypervisor intervention since it allocates 8096 4k pages. But since XEN uses command batching, it may not affect the performance drastically as full virtualization would have.

In conclusion, since memsweep benchmark requires page table updates, we expect this to affect the performance in a negative manner.

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



The results we got are consistent with our expectations, even though OpenStack performance was significantly worse than AWS. We suspect that this could be due to two reasons. One reason may be that the hardware used by openstack is old and outdated, hence the performance difference. The other possible reason may be that AWS uses more advanced techniques to lower the number of hypervisor interventions needed, better command batching and newer, better performing hardware as well.

Appendix 1 - Disk Benchmark

disk_test.sh

```
#disk_test.sh
# run it inside the src folder
# dd tests
touch dd_stats.txt
touch fio_stats.txt
# clean the cache (also the purge command does the job)
echo "=========== Test time: $(date) ==============" >> dd_stats.txt
echo $'\n\tWrite' >> dd_stats.txt
(dd if=/dev/zero bs=512k of=tstfile count=1024) 2>> dd_stats.txt; sync
# uncomment fot linux systems
# (dd if=/dev/zero bs=512k of=tstfile count=1024 oflag=dsync) 2>> dd_stats.txt; sync
# clean the cache (also the purge command does the job)
echo $'\n\tRead' >> dd_stats.txt
(dd if=tstfile bs=512k of=/dev/null count=1024) 2>> dd_stats.txt; sync
# uncomment fot linux systems
# (dd if=tstfile bs=512k of=/dev/null count=1024 oflag=dsync) 2>> dd_stats.txt; sync
echo $'\n' >> dd_stats.txt
rm -f tstfile

# fio tests
echo "=========== Test time: $(date) ==============" >> fio_stats.txt
(fio trivial.fio) >> fio_stats.txt
echo $'\n' >> fio_stats.txt
```

trivial.fio

```
[global]
ioengine=posixaio
rw=readwrite
size=1g
directory=${HOME}
thread=1

[trivial-readwrite-1g]

```

dd output example

```
========== Test time: Sun 18 Jun 2017 11:23:08 CEST ==============

        Write
1024+0 records in
1024+0 records out
536870912 bytes transferred in 0.793705 secs (676411162 bytes/sec)

        Read
1024+0 records in
1024+0 records out
536870912 bytes transferred in 1.020353 secs (526162020 bytes/sec)
```

fio output example

```
========== Test time: Sun 18 Jun 2017 11:23:10 CEST ==============
trivial-readwrite-1g: (g=0): rw=rw, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B,
ioengine=posixaio, iodepth=1
fio-2.20
Starting 1 thread

trivial-readwrite-1g: (groupid=0, jobs=1): err= 0: pid=3075: Sun Jun 18 11:23:12 2017
   read: IOPS=39.9k, BW=156MiB/s (163MB/s)(249MiB/1596msec)
    slat (usec): min=0, max=850, avg= 3.36, stdev= 7.87
    clat (usec): min=0, max=3798, avg= 7.83, stdev=27.22
     lat (usec): min=4, max=3799, avg=11.19, stdev=28.59
    clat percentiles (usec):
     |  1.00th=[    1],  5.00th=[    1], 10.00th=[    1], 20.00th=[    3],
     | 30.00th=[    5], 40.00th=[    5], 50.00th=[    5], 60.00th=[    6],
     | 70.00th=[    6], 80.00th=[    8], 90.00th=[   12], 95.00th=[   20],
     | 99.00th=[   49], 99.50th=[   76], 99.90th=[  270], 99.95th=[  474],
     | 99.99th=[ 1112]
   bw (  KiB/s): min=92820, max=231493, per=0.09%, avg=154896.33, stdev=70467.58
  write: IOPS=40.3k, BW=158MiB/s (165MB/s)(251MiB/1596msec)
    slat (usec): min=0, max=2130, avg= 3.57, stdev=12.30
    clat (usec): min=1, max=4056, avg= 8.46, stdev=29.90
     lat (usec): min=4, max=4064, avg=12.03, stdev=32.63
    clat percentiles (usec):
     |  1.00th=[    1],  5.00th=[    1], 10.00th=[    2], 20.00th=[    3],
     | 30.00th=[    5], 40.00th=[    6], 50.00th=[    6], 60.00th=[    6],
     | 70.00th=[    7], 80.00th=[    8], 90.00th=[   13], 95.00th=[   21],
     | 99.00th=[   59], 99.50th=[   76], 99.90th=[  266], 99.95th=[  378],
     | 99.99th=[ 1400]
   bw (  KiB/s): min=93027, max=236610, per=0.09%, avg=156759.00, stdev=73136.08
    lat (usec) : 2=10.34%, 4=11.58%, 10=62.89%, 20=9.86%, 50=4.20%
    lat (usec) : 100=0.83%, 250=0.19%, 500=0.07%, 750=0.02%, 1000=0.01%
    lat (msec) : 2=0.01%, 4=0.01%, 10=0.01%
  cpu          : usr=20.75%, sys=32.60%, ctx=139649, majf=2, minf=0
  IO depths    : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%
     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=0.0%, >=64=0.0%
     issued rwt: total=63625,64375,0, short=0,0,0, dropped=0,0,0
     latency   : target=0, window=0, percentile=100.00%, depth=1

Run status group 0 (all jobs):
   READ: bw=156MiB/s (163MB/s), 156MiB/s-156MiB/s (163MB/s-163MB/s), io=249MiB (261MB),
run=1596-1596msec
  WRITE: bw=158MiB/s (165MB/s), 158MiB/s-158MiB/s (165MB/s-165MB/s), io=251MiB (264MB),
run=1596-1596msec
```

Results from measurements

Local

Measures	*write (seq) MB/s*	*Read (SEQ) MB/s*	*Write (rand) MB/s*	*read (rand) MB/s*
1	*718*	*783*	*237*	*234*
2	*676*	*526*	*165*	*163*
3	*767*	*782*	*210*	*207*
4	*770*	*765*	*199*	*196*
5	*645*	*767*	*198*	*196*
6	*765*	*734*	*185*	*182*
7	*670*	*756*	*132*	*131*
8	*792*	*747*	*129*	*127*
9	*768*	*758*	*204*	*201*
10	*663*	*744*	*144*	*143*
Average	*723*	*736*	*180*	*178*

OpenStack

Measures	write (seq) MB/s	Read (SEQ) MB/s	Write (rand) MB/s	read (rand) MB/s
1	7,3	112,0	46,1	46,2
2	7,2	112,0	45,0	45,1
3	7,5	112,0	45,9	46,0
4	7,4	112,0	46,0	46,1
5	7,4	109,0	46,5	46,6
6	6,2	112,0	46,5	46,6
7	7,1	111,0	46,5	46,6
8	7,2	111,0	45,3	45,4
9	7,2	111,0	46,7	46,9
10	7,2	112,0	46,1	46,2
Average	7,2	111,4	46,1	46,2

AWS

Measures	write (seq) MB/s	Read (SEQ) MB/s	Write (rand) MB/s	read (rand) MB/s
1	121,0	3.600,0	255,7	256,3
2	130,0	3.500,0	255,7	256,3
3	110,0	3.600,0	255,7	256,3
4	136,0	3.800,0	255,7	256,3
5	130,0	3.800,0	255,7	256,3
6	130,0	3.500,0	255,7	256,3
7	134,0	3.500,0	255,7	256,3
8	134,0	3.600,0	255,7	256,3
9	134,0	3.500,0	255,7	256,3
10	132,0	3.400,0	255,7	256,3
Average	129,1	3.580,0	255,7	256,3

Appendix 2 - Mem and CPU bench script and plotting

```
#!/usr/bin/env python3
# -*- python -*-
#

import configparser
import sys
import os
import subprocess
import json, time
import numpy as np

filepath = os.path.dirname(os.path.realpath(__file__))
configfile = os.path.join(filepath, 'config')
configParser = configparser.RawConfigParser()
configFilePath = configfile
configParser.read(configFilePath)

aws_key_id = configParser.get('aws', 'awsKeyId')
aws_secret = configParser.get('aws', 'awsSecret')
pemfile = configParser.get('aws', 'pemLocation')
keyLocation = configParser.get('openstack', 'keyLocation')
op_ip = configParser.get('openstack', 'opIP')

def print_help():
  print('benchmark.py <num_of_iterations>')
  print('Example ./benchmark.py 1')
  print('** Remember about filling your config file! **')

## AWS HELP FUNCTIONS

def init_aws():
  result = subprocess.run(['aws', 'ec2', 'run-instances', '--image-id', 'ami-f52bfa9a', '--count', '1', '--instance-type',
'm3.medium', '--key-name', 'group32-key', '--security-groups', 'group32-securityGroup'], stdout=subprocess.PIPE)
  data = json.loads(result.stdout.decode('utf-8'))
  instance_id = data['Instances'][0]['InstanceId']
  print('** Initializing AWS Instance...')
  while(not is_aws_running(instance_id)):
    time.sleep(2)
  print('** AWS Instance initialized')
  return instance_id

def terminate_aws(instance_id):
  # aws ec2 terminate-instances --instance-ids i-001be48dce290e70e
  result = subprocess.run(['aws', 'ec2', 'terminate-instances', '--instance-ids', instance_id], stdout=subprocess.PIPE)
  print(result.stdout.decode('utf-8'))
  pass

def is_aws_running(instance_id):
  #aws ec2 describe-instance-status --instance-id i-02438514d865477f5
  result = subprocess.run(['aws', 'ec2', 'describe-instance-status', '--instance-ids', instance_id],
stdout=subprocess.PIPE)
  data = json.loads(result.stdout.decode('utf-8'))
  if(len(data['InstanceStatuses']) > 0 and data['InstanceStatuses'][0]['InstanceState']['Code'] == 16):
    print (data['InstanceStatuses'][0])
    return True
  return False

def get_aws_ip(instance_id):
  #aws ec2 describe-instance-status --instance-id i-02438514d865477f5
  result = subprocess.run(['aws', 'ec2', 'describe-instances', '--instance-ids', instance_id], stdout=subprocess.PIPE)
```

```
    data = json.loads(result.stdout.decode('utf-8'))
    instance_ip = data['Reservations'][0]['Instances'][0]['PublicIpAddress']
    print(instance_ip)
    return instance_ip

## MEMORY BENCH

def run_mem_benchmark_local(iteration):
    print('** Running MEM Benchmark LOCAL iteration:', iteration)
    filepath = os.path.dirname(os.path.realpath(__file__))
    target = os.path.join(filepath, 'memory_benchmark', 'memsweep.sh')
    result = subprocess.run([target], stdout=subprocess.PIPE, cwd=os.path.join(filepath, 'memory_benchmark'))
    output = result.stdout.decode('utf-8')

    if "Memsweep time in seconds: " not in output:
        print(result)

    start = output.find('Memsweep time in seconds: ') + 26
    bench = float(output[start:])
    print(bench)
    return bench

def run_mem_benchmark_aws(iteration, ip):
    print('** Running MEM Benchmark AWS iteration:', iteration)
    result = subprocess.run("ssh -i %s ec2-user@%s 'cd memory_benchmark && ./memsweep.sh'"%(pemfile, ip),
shell=True, stdout=subprocess.PIPE)
    output = result.stdout.decode('utf-8')

    if "Memsweep time in seconds: " not in output:
        print(result)

    start = output.find('Memsweep time in seconds: ') + 26
    bench = float(output[start:])
    print(bench)
    return bench

def run_mem_benchmark_openstack(iteration):
    print('** Running MEM Benchmark OPENSTACK iteration:', iteration)

    result = subprocess.run("ssh -i %s ubuntu@%s 'cd memory_benchmark && ./memsweep.sh'"%(keyLocation,
op_ip), shell=True, stdout=subprocess.PIPE)
    output = result.stdout.decode('utf-8')

    if "Memsweep time in seconds: " not in output:
        print(result)

    start = output.find('Memsweep time in seconds: ') + 26
    bench = float(output[start:])
    print(bench)
    return bench

## CPU BENCH

def run_cpu_benchmark_local(iteration):
    print('** Running CPU Benchmark LOCAL iteration:', iteration)
    filepath = os.path.dirname(os.path.realpath(__file__))
    target = os.path.join(filepath, 'cpu_benchmark', 'linpack.sh')
    result = subprocess.run([target], stdout=subprocess.PIPE, cwd=os.path.join(filepath, 'cpu_benchmark'))
    output = result.stdout.decode('utf-8')

    if "Benchmark result:" not in output:
        print(result)

    start = output.find('Benchmark result: ') + 18
    end = output.find(' KFLOPS', start)
    bench  = float(output[start:end])
```

```
    print(bench)
    return bench

def run_cpu_benchmark_aws(iteration, ip):
    print('** Running CPU Benchmark AWS iteration:', iteration)
    result = subprocess.run("ssh -i %s ec2-user@%s 'cd cpu_benchmark && ./linpack.sh'"%(pemfile, ip), shell=True,
stdout=subprocess.PIPE)
    output = result.stdout.decode('utf-8')

    if "Benchmark result:" not in output:
        print(result)

    start = output.find('Benchmark result: ') + 18
    end = output.find(' KFLOPS', start)
    bench  = float(output[start:end])
    print(bench)
    return bench

def run_cpu_benchmark_openstack(iteration):
    print('** Running CPU Benchmark Openstack iteration:', iteration)
    result = subprocess.run("ssh -i %s ubuntu@%s 'cd cpu_benchmark && ./linpack.sh'"%(keyLocation, op_ip),
shell=True, stdout=subprocess.PIPE)
    output = result.stdout.decode('utf-8')

    if "Benchmark result:" not in output:
        print(result)

    start = output.find('Benchmark result: ') + 18
    end = output.find(' KFLOPS', start)
    bench  = float(output[start:end])
    print(bench)
    return bench

def run_disk_benchmark_aws(iteration, ip):
    print('** Running DISK Benchmark AWS iteration:', iteration)
    result = subprocess.run("ssh -i %s ec2-user@%s 'cd disk_benchmark && ./disk_test.sh'"%(pemfile, ip),
shell=True, stdout=subprocess.PIPE)
    output = result.stdout.decode('utf-8')

    print(output)
    #return bench

## DISK BENCH

def prepare_tar():
    print('** Prepare tar to be send')
    import tarfile
    tar = tarfile.open(os.path.join(filepath,"benchmark.tar.gz"), "w:gz")
    for name in ["cpu_benchmark", "memory_benchmark", "disk_benchmark"]:
        tar.add(name)
    tar.close()

def send_tar_aws(ip):
    print('** Initialize connection with AWS, install scp and send tar')
    time.sleep(25)
    tarfile = os.path.join(filepath,"benchmark.tar.gz")
    subprocess.call("ssh -i %s ec2-user@%s 'sudo yum install -y openssh-clients'"%(pemfile, ip), shell=True)
    subprocess.call("scp -i %s %s ec2-user@%s:/home/ec2-user/"%(pemfile, tarfile, ip), shell=True)

    print('** Unpack tar')
    subprocess.call("ssh -i %s ec2-user@%s 'tar -zxvf benchmark.tar.gz'"%(pemfile, ip), shell=True)

def send_tar_openstack():
    print('** Initialize connection with OpenStack and send tar')
    tarfile = os.path.join(filepath,"benchmark.tar.gz")
    subprocess.call("scp -i %s %s ubuntu@%s:/home/ubuntu/"%(keyLocation, tarfile, op_ip), shell=True)
```

```
    print('** Unpack tar')
    subprocess.call("ssh -i %s ubuntu@%s 'tar -zxvf benchmark.tar.gz'"%(keyLocation, op_ip), shell=True)

def main(argv):
    print('** Remember to execute all prerequisites from awsinfo.txt before running this script!! **')
    if len(argv) != 1:
        print_help()
        sys.exit(2)

    num_iterations = int(argv[0])

    # Initialize AWS
    instance_id = init_aws()

    # Get AWS IP
    ip = get_aws_ip(instance_id)

    # Prepare TAR with scripts
    prepare_tar()
    send_tar_aws(ip)
    send_tar_openstack()

    cpu_results_local = []
    cpu_results_aws = []
    cpu_results_openstack = []
    mem_results_local = []
    mem_results_aws = []
    mem_results_openstack = []
    disk_results_local = []
    disk_results_aws = []
    disk_results_openstack = []
    for i in range (0, num_iterations):
        cpu_results_local.append(run_cpu_benchmark_local(i+1))
        cpu_results_openstack.append(run_cpu_benchmark_openstack(i+1))
        cpu_results_aws.append(run_cpu_benchmark_aws(i+1, ip))
        mem_results_local.append(run_mem_benchmark_local(i+1))
        mem_results_openstack.append(run_mem_benchmark_openstack(i+1))
        mem_results_aws.append(run_mem_benchmark_aws(i+1, ip))
        #run_disk_benchmark_aws(i+1, ip)

    import matplotlib.pyplot as plt
    fig, ax = plt.subplots(1)
    local_mem, = ax.plot(mem_results_local, 'b^', label="Local avg %ss"%(np.average(mem_results_local)))
    aws_mem, = ax.plot(mem_results_aws, 'g^', label="AWS avg %ss"%(np.average(mem_results_aws)))
    ops_mem, = ax.plot(mem_results_openstack, 'r^', label="OpenStack avg
%ss"%(np.average(mem_results_openstack)))
    plt.legend(handles=[local_mem, aws_mem, ops_mem])
    plt.ylabel('Memsweep (s)')
    plt.xlabel('Iterations')
    plt.xlim([-1, num_iterations])
    ax.set_xticklabels([])

    fig, ax = plt.subplots(1)
    local_cpu, = ax.plot(cpu_results_local, 'b^', label="Local avg %s"%(np.average(cpu_results_local)))
    aws_cpu, = ax.plot(cpu_results_aws, 'g^', label="AWS avg %s"%(np.average(cpu_results_aws)))
    ops_cpu, = ax.plot(cpu_results_openstack, 'r^', label="OpenStack avg %s"%(np.average(cpu_results_openstack)))
    plt.legend(handles=[local_cpu, aws_cpu, ops_cpu])
    plt.ylabel('KiloFlops')
    plt.xlabel('Iterations')
    plt.xlim([-1, num_iterations])
    ax.set_xticklabels([])

    plt.show()

    #Terminate AWS after all
```

```
    terminate_aws(instance_id)
if __name__ == '__main__':
    main(sys.argv[1:])
```

```
** Running CPU Benchmark LOCAL iteration: 1
2426103.904
** Running CPU Benchmark Openstack iteration: 1
1789806.935
** Running CPU Benchmark AWS iteration: 1
980865.326
** Running MEM Benchmark LOCAL iteration: 1
3.972
** Running MEM Benchmark OPENSTACK iteration: 1
33.625
** Running MEM Benchmark AWS iteration: 1
8.48
** Running CPU Benchmark LOCAL iteration: 2
2119847.018
** Running CPU Benchmark Openstack iteration: 2
1785963.295
** Running CPU Benchmark AWS iteration: 2
980865.326
** Running MEM Benchmark LOCAL iteration: 2
3.654
** Running MEM Benchmark OPENSTACK iteration: 2
33.522
** Running MEM Benchmark AWS iteration: 2
8.34
** Running CPU Benchmark LOCAL iteration: 3
2682255.276
** Running CPU Benchmark Openstack iteration: 3
1789719.62
** Running CPU Benchmark AWS iteration: 3
979075.426
** Running MEM Benchmark LOCAL iteration: 3
3.512
** Running MEM Benchmark OPENSTACK iteration: 3
33.481
** Running MEM Benchmark AWS iteration: 3
8.23
** Running CPU Benchmark LOCAL iteration: 4
2678145.455
** Running CPU Benchmark Openstack iteration: 4
1790831.623
** Running CPU Benchmark AWS iteration: 4
983562.481
** Running MEM Benchmark LOCAL iteration: 4
3.566
** Running MEM Benchmark OPENSTACK iteration: 4
33.435
** Running MEM Benchmark AWS iteration: 4
8.5
** Running CPU Benchmark LOCAL iteration: 5
2670388.554
** Running CPU Benchmark Openstack iteration: 5
1790585.238
** Running CPU Benchmark AWS iteration: 5
```

977292.046
** Running MEM Benchmark LOCAL iteration: 5
3.567
** Running MEM Benchmark OPENSTACK iteration: 5
33.432
** Running MEM Benchmark AWS iteration: 5
8.43
** Running CPU Benchmark LOCAL iteration: 6
2673149.586
** Running CPU Benchmark Openstack iteration: 6
1790445.118
** Running CPU Benchmark AWS iteration: 6
975515.152
** Running MEM Benchmark LOCAL iteration: 6
3.535
** Running MEM Benchmark OPENSTACK iteration: 6
33.323
** Running MEM Benchmark AWS iteration: 6
8.09
** Running CPU Benchmark LOCAL iteration: 7
2680377.624
** Running CPU Benchmark Openstack iteration: 7
1789791.859
** Running CPU Benchmark AWS iteration: 7
979969.559
** Running MEM Benchmark LOCAL iteration: 7
3.598
** Running MEM Benchmark OPENSTACK iteration: 7
33.376
** Running MEM Benchmark AWS iteration: 7
8.1
** Running CPU Benchmark LOCAL iteration: 8
2674022.055
** Running CPU Benchmark Openstack iteration: 8
1769896.519
** Running CPU Benchmark AWS iteration: 8
971980.676
** Running MEM Benchmark LOCAL iteration: 8
3.484
** Running MEM Benchmark OPENSTACK iteration: 8
33.547
** Running MEM Benchmark AWS iteration: 8
8.63
** Running CPU Benchmark LOCAL iteration: 9
2678584.504
** Running CPU Benchmark Openstack iteration: 9
1751089.617
** Running CPU Benchmark AWS iteration: 9
973744.707
** Running MEM Benchmark LOCAL iteration: 9
3.702
** Running MEM Benchmark OPENSTACK iteration: 9
34.368
** Running MEM Benchmark AWS iteration: 9
8.28
** Running CPU Benchmark LOCAL iteration: 10
2674750.575
** Running CPU Benchmark Openstack iteration: 10

1773486.124
** Running CPU Benchmark AWS iteration: 10
977292.046
** Running MEM Benchmark LOCAL iteration: 10
3.54
** Running MEM Benchmark OPENSTACK iteration: 10
33.261
** Running MEM Benchmark AWS iteration: 10
8.35