Selecting VM instances in the Cloud through benchmarking

Alaleh Hamidi, Masoumeh Nourollahi, Soude Ghari

***Abstract*—. Cloud application architects can opt from an extensive selection of VM instances to obtain high levels of performance. Selecting the proper instance for the right job would not be a striaghtforward task. Although cloud providers categorized their instances to aid consumers to choose appropriate one, benchmarking instances has been still the best apporoach to choose instances for a cloud application. In this paper, a benchmarking analysis, which is presented on 6 various instances available on Amazon Elastic Compute Cloud (EC2) is done in order to find the best instances for a particular cloud application architecture.**

Keywords:Amazon Elastic Compute Cloud, Benchmark, Cloud Application,

I. Introduction

Cloud providers offer numerous services such as platforms, infrastructure and software as a service. Each of these categories comprise of myriad products users that can choose from in order to build their cloud applications. Amazon Elastic Compute Cloud (Amazon EC2) is an Infrastructure as a service (IaaS) where developers can opt from a large variety of server instances, each instance have their set of technical particularities [1].

Since all cloud applications meet various requirements, having the possibility to opt from plenty of server instances, all optimized for specific job types, is tricky. Done right, choosing the right instances for every module of a cloud application can increase the product performance, availability and scalability while reducing infrastructure costs. Nonetheless, since no cloud application is the same, there is no obvious instruction on how to determine the correct instance. Benchmarking instances to compare the performance of various instances is a great approach to find the propoer instances for every module of a cloud application.

In this paper, we apply benchmarking on multiple Amazon EC2 instances to find the best qualified for a cloud application with heavy data operations, storage and heavy computation necessities. Section II presents the methodology used to find the best instances for the application. Section III propose the benchmarking results and analysis and section IV introduces case study which is the selection of the application of these results to choose the right instances for the cloud application. Finally section V presents the challenges tackled to benchmarking.

II. Methodology

Finding the outstanding instances for a particular cloud application, is of utmost important to understand the architecture of intended applications as well as the necessity of the various modules. Then, server metrics must be taken into account to employ as comparison tools,which the main task of these tools is to to collect those metrics. Finally, benchmarking the instances must be performed as a repeatable process.

We have developed a number of Python scripts as well as Bash script to test the capabilities of each instance.

A. Metrics of Comparison

In order to compare instances performances in respect of the application needs, metrics on these instance characteristics were chosen : IO, CPU, IOPS, Memory, Disk and Network throughput.

1) IO - Throughput: I/O (Input/Output) is the operation of taking information from a destination A and sending it to a destination B. The metric used to make the benchmarks is the throughput, for example, the quantity of data transferred over the time it took to do the task. The utility tool applied to benchmark the I/O is dd. dd [2] is used to benchmark the file system IO performance.

```
dd if=/dev/zero of=sb-io-test bs=2M count=3k conv=fdatasync .
```

- if: Option showing the name of the input file that *dd* will read from it.
- of: name of the output file that *dd* will write the input file to.
- bs: Size of the block.
- count: Number of blocks *dd* will read.
- conv=fdatasync: Option to force the process to write the entire file to disk before completing.
- /dev/zero: Kind of virtual file composed only of zero.

In this benchmark, we just have varied two parameters bs and count in order to experiment our instances. The appropriate values for these parameters is identified 2M and 3k correspondingly.

2) CPU-Computation Time: CPU of a variety of instances can be compared using the speed at which they are able to do a computation task. In this paper, we use the time it takes to compute prime numbers as the metric of comparison for CPU. The benchmarking tool **Sysbench**[3] is used to record the time needed to compute all the prime numbers up to a certain value. This maximum prime number is passed as an argument and must be the same for every instance to have a comparison basis of CPU performance.

Table I: Characteristic of Instannces

Instance Type	Family Type	vcpu	Memory (GiB)	Storage	Network Speed(Gb/s)	Cost (per hour)
m4.large	General propose	2	8	EBS only	Moderate	0.10$
c4.xlarge	Compute optimised	4	7.5	EBS only	High	0.199$
c5.xlarge	Compute optimised	4	8	EBS only	Up to 10	0.17$
m5.xlarge	General propose	4	16	EBS only	Up to 10	0.192$
t2.2xlarge	General propose	8	32	EBS only	–	0.3712
t2.xlarge	General propose	4	16	EBS only	–	0.1856

sysbench –test=cpu –cpu-max-prime=1000000 –events=44000 – num-threads=2000 run

- –test: Name of the test mode to run.
- –cpu-max-prime: Max prime until which we search for prime numbers.
- –events: Number of requests.
- –num-threads: Number of worker threads to create.

As increasing the –cpu-max-prime merely increases the duration of an event and since the –events a are just limiting the time of the benchmark execution, we primarily focused on varying the –num - thread to increase the use of the CPU.

3) IOPS: Bonnie++ is a benchmark suit to test hard drive and file system performance. It benchmarks three following performance metrics:

- Data read and write speed
- Maximum number of seeks per second
- Maximum number of file metadata operations per second (such as creation, deletion and so on) [4]

There are too many parameters that we can pass to Bonnie++. But for our work, we only use the following three ones:

- -r: The amount of RAM the system has installed.
- -s: The dataset size to use for the IO test.
- -u: The root user that the test should run as.

Bonnie++ doesn't allow you to specify the dataset size less than twice RAM size to avoid having a cache biased test. So, we executed the following command:

*Bonnie++ -r (Rsize) -s (2*Rsize) -u ubuntu*

Bonnie++ will execute some reading, writing, creating and deleting processes during two separate consecutive and random scenarios to obtain more detailed results.

In order to tailor the Bonnie command to all instances, we ran the mentioned command several times to obtain the minimum and maximum value of its parameters for each instance. We found that there is no minimum value for r parameter of Bonnie++ and all of instances work properly for all values. So we considered 1MB as the minimum value of all instances. The maximum values of r for all instances were 3.4GB except for c4xlarge which was 2.9GB. So, we identify the value of Rsize 2GB which is appropriate value for all instances.

4) Memory: Using stress-ng benchmark, we evaluated the memory of each instance. Stress-ng is a Linux stress tool that is designed for stressing a computer in different ways to test the instance performance. In this work we focus on testing memory using stress-ng. So, a lot of memory will be consumed under this test aggressively to know how efficient the instance would be under a large workload. The control options we used to run the stress-ng command, are showed in the commend below:

stress-ng –cpu 4 –io 2 –vm 7000 –vm-bytes 15G –timeout 60s –metrics-brief

- –cpu 4: Run 4 instances of the CPU stressor.
- –io 2: Run 2 instances of the io stressor to commit buffer cache to disk.
- –vm 7000: Run 7000 virtual memory stressors (workers) to call mmap(2)/munmap(2) and write to the allocated memory.
- –vm-bytes 15G: Malloc 15GB per vm worker (default is 256MB).
- –timeout 60s:Stop stress test after 60 seconds.
- –metrics-brief: enable metrics and only output metrics that aren't zero.

We ran cpu and io stressors to see the effects of them on the memory. In order to tailor the stress-ng command to all instances, we ran this command several times but with different values for control options to obtain the minimum and maximum value of the parameters for each instance. We kept the values of variables cpu and io constant in all cases.For all instances, the minimum value of vm is 1 and maximum is 8192.The minimum value of all instances for vm-bytes is 4k, but as table 2 shows, the maximum value of this variable for each instance is different from the others.

m4.large	c4.xlarge	c5.xlarge	m5.xlarge	t2.2xlarge	t2.xlarge
59GB	29GB	31GB	66GB	90GB	80GB

5) Disk: Disk throughput is measured by read speed of storage devices. This read speed can be measured by considered both disk and cache read speeds. In this paper we

used hdparm tool to benchmark disk throughput. hdparm is a tool which is used to handle disk devices and hard disks. With the help of this command we can get statistics on hard disk settings and throughput. Also it can set drive cache, sleep mode, power management, acoustic management and DMA settings. We have used hdparm to measure disk and cache throughput, which is an indication of cache, memory and also processor throughput. This tool measures disk and buffered cache throughput in terms of read speed megabyte per second[5].

```
hdparm -Tt /dev/xvda
```

- T: Cache read performance speed for benchmark and comparison purposes. This option displays speed of reading directly from the linux buffer cache without disk access.
- t: Test device read performance speed (-t for timing buffered disk reads) of the first hard drive for benchmark and comparison purposes. This displays the speed of reading through the buffer cache to the disk without any prior caching of data.
- i: provides information about the disk.

This measurement is and indication of the throughput of the processor, cache and memory of the system under test. For meaningful results, this operation should be repeated 2-3 times on an otherwise inactive system (with no other processes) with at least a few megabytes free memory.

6) Network Throughput: The goal is to benchmark network's bidirectional speed (in terms of upload, download and ping speed). We have used speedtest tool to perform this benchmark. Speedtest is a tool written in python, which measure internet speed based on distance in km. The method of speed measurement of this tool is that it first downloads a file which is in a nearby to the mentioned distance, speedtest.net server. Also the upload test is performed in the same manner. This tool return upload, download and ping speed in Mbit/s. Also speedtest can be used to measure latency and packet loss natively, and without relying on a web browser. This tool provides both GUI and Cli interfaces for test. No parameters in required for this test.

```
speedtest-cli –simple
```

- simple: This option just limits the output of the speedtest command to upload, download and ping speed.

III. Regression Analysis

Choosing the right parameters for the benchmarking tests explained in the previous section has been performed by applying regression analysis. Regression analysis intends to find the variables having impact on a set of data [6]. This work uses regression analysis to obtain the limits of each parameter of every benchmarking test. The aim is to find values for every parameter that will remain constant across instance tests and will provide meaningful data and hopefully distinguishable variation between instances for every parameter.

IV. Instances Types

Amazon EC2 catalog comprise myriad server types. This paper focuses on Ubuntu machines running the last version of the OS (18.04). A total of 6 virtual machines will be compared. 4 of them is categorized as a general purpose instance, two of them as compute optimized. Here is a table showing the chosen instances and their characteristics, as provided by Amazon.

General Purpose:"General purpose instances provide a balance of compute, memory, and networking resources, and can be used for a variety of workloads"[1]. **Compute Optimised**: Compute-optimised instances propose the lowest price per vCPU in the Amazon EC2 family and are perfect for running advanced compute-intensive workloads [1].

V. Benchmarking and Results Analysis

Benchmarking instances gives important data regarding instances performances in categories. Moreover, running the test sequence 5 times gives information on how instances perform under different levels of pressure over time. Following are the results of the benchmarks done on the 6 instances.

A. IO

As presented in previous sections, IO throughput was computed by recording the time needed for an instance. The average time needed by each instance to complete the data transfer over 5 executions was depicted in Figure 1.

We found that all the instances, except for m4.large, possess a good efficiency. Especially, the c5.xlarge machine has extremely good results.

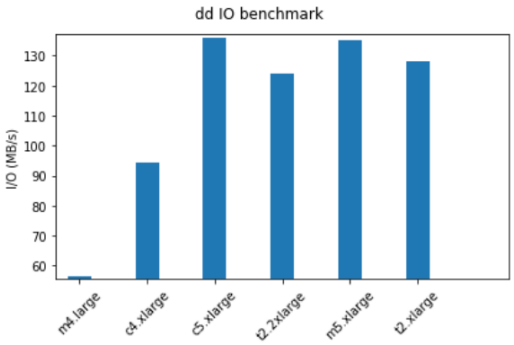

Figure 1: Avarage Value of time over 5 iterations

B. CPU

When testing parameters on the instances, we identified the upper bound to be 8000. The lower bound, which is the upper bound of m4.large instance 8000 threads. As the computation time was extremely long for those values we opted to run the command with 2000 threads. We also identify cpu-max-prime as 10000,100000,1000000 for benchmarking and see m4.large crash for value higher than 1000000. Thus, we identify cpu-max prime as 1000000. As it can be observed from Figure 2 the instance m5.xlarge has the lowest latency, so is the best of the six instances. Next comes t2.2xlarge, then c5.xlarge. The instance m4.large is the worst and should not be used for tasks needing loads of CPU.Figure 3 represents total time of cpu required.

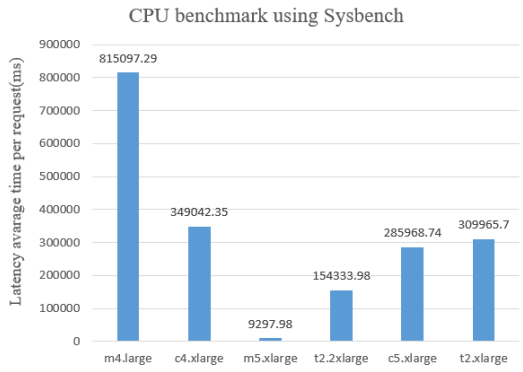

Figure 2: Average latency the time of the Sysbench benchmark with 2000 threads

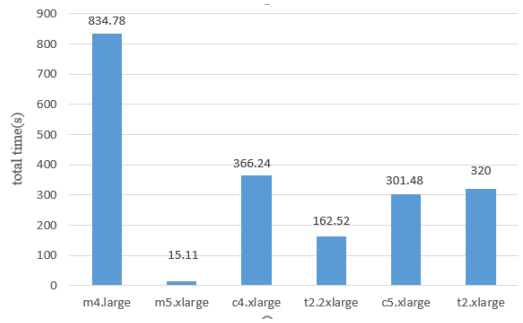

Figure 3: Average latency the time of the Sysbench benchmark with 2000 threads

C. IOPS

To evaluate IOPS, we consider sequential output block latency, sequential input block latency and random seeks per second. Sequential output block is the output rate per second when writing the 4GB file using efficient block writes. as Figure 4 shows, c5.xlarge and m5.xlarge have the lowest latency (best performance) and t2.2xlarge has the worst performance.

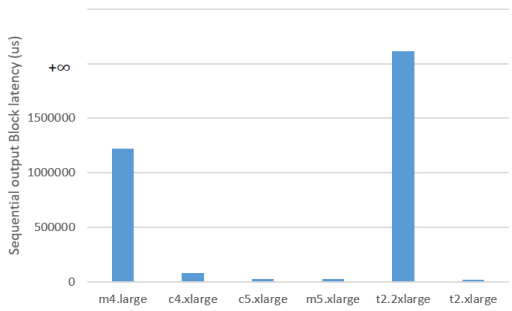

Figure 4: Sequential Output Block Latency for each instance

Sequential input block is the input rate per second while reading the file using efficient block reads. As we can see the Figure 5, again c5.xlarge and m5.xlarge are the best instances based on their lowest latency.

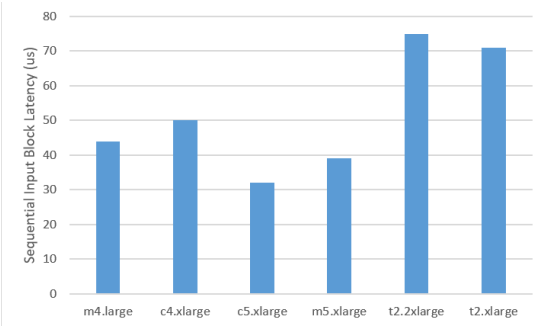

Figure 5: Sequential input Block Latency for each instance

In Figure 6, c5.xlarge and m5.xlarge are again shown as the best ones regarding random seeks per second. So, we can conclude that these two instances are better than the other 4 instances regarding IOPS.

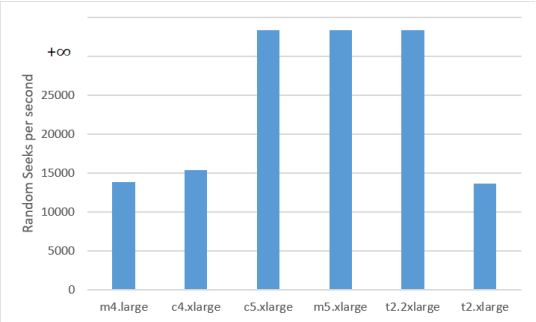

Figure 6: Random Seeks per second for each instance

D. Memory

To compare the memory performance of instances, Bogo operations per second (user+sys time) is considered. As we can see in the Figure 7 the memory performance of t2.xlarge and m4.large is better than the other's, but t2.xlarge is the best one. While t2.2xlarge obtained the greatest maximum value for vm-bytes, it is the worst instance based on Bogo operations per second.

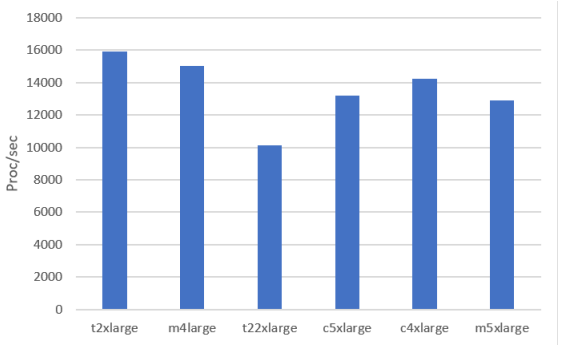

Figure 7: Bogo operations per second (user+sys time) for each instance

E. Disk

As mentioned in previous section, to measure disk performance two parameters were measured. Following we will present the results of our benchmarks on disk read. Please take notice that this results are average of 5 times execution of the same test, with the same conditions, to make sure that results are reliable.

- Timing buffered disk reads (MB/s): IT seems that C5.xlarge and M5.xlarge, by throughput of 170.494 and 170.492 MB/s respectively have the best performance among all instances.
- Timing cache reads (MB/s): As it is shown in the diagram, c4.xlarge, with throughput of 10430.398 MB/s have the performance of cache reads among all instances in this experiment.

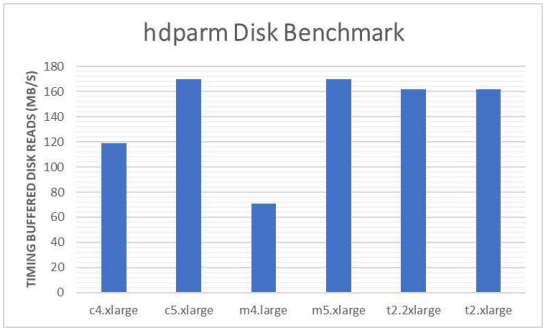

Figure 8: Disk buffered read throughput

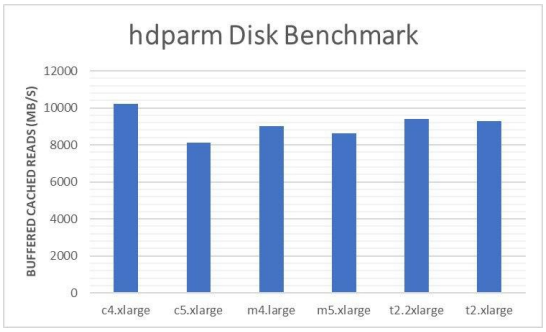

Figure 9: cache read throughput

F. Network

As mentioned in previous section, to measure network performance there parameters were measured. Following we will present the results of our benchmarks on network speed. Please take notice that this results are average of 5 times execution of the same test,after warm-up, with the same conditions to make sure that results are reliable.

- Download (MBit/s): IT seems that C5.xlarge and M5.xlarge, by speed of 2631 and 2610 MBit/s respectively have the best speed among all instances.
- Upload (MB/s): As it is shown in the diagram all instances have the same upload speed of 4.17 MBit/s.
- Latency (MBit/s): As per results, c5.xlarge and c4.xlarge, with 0.82 and 0.89 MBit/s respectively, have the least latency and perform the best among all instances. Other instances do not have very significant latency.

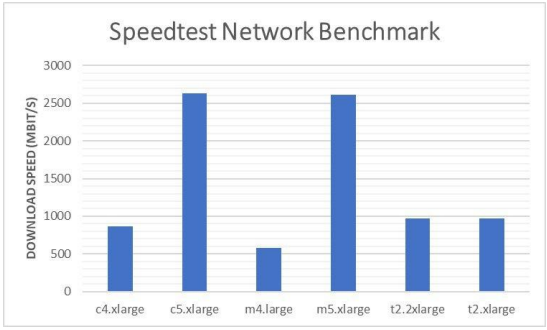

Figure 10: Speedtest network benchmark- Download speed

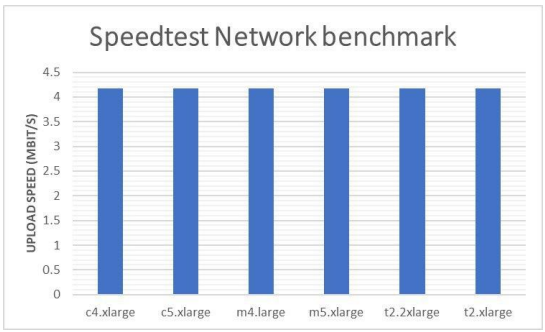

Figure 11: Speedtest network benchmark- Upload speed

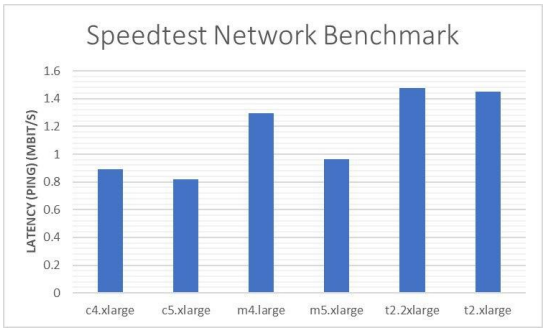

Figure 12: Speedtest network benchmark- Ping speed

VI. Case Study

Benchmarking the instances gave substantial results. Results demonstrated that instances do not always perform as expected from their classification. Applying these results, it is now possible to assign the right instance for every module of the cloud application subject to this paper. First of all, the cloud application needs 2 instances for storage intensive work, 3 instances for heavy input/output operations and data extraction and finally, it needs 2 instances for heavy data computation. Figure shows the way these instances should interact with each other.

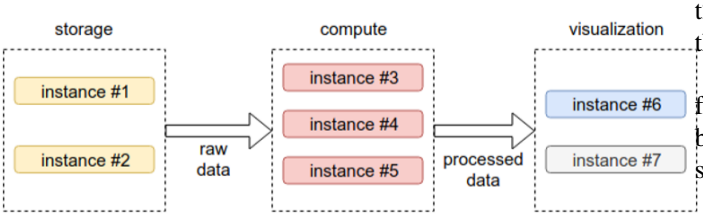

Figure 13: Design Architecture Elements

Storage Module

The 2 instances in the storage module require high storage performance. Storage optimized instances can be known by their high rate of read and write accesses with low latency. Following is an overview of characteristics used to evaluate whether instances would fit in the storage module.

IO Throughput: Instances that performed well and achieved stability for IO throughput were c5.xlarge, m5.xlarge, t2.xlarge.

IOPS: c5.xlarge and m5.xlarge have better IOPS performance than the others.

Disk Read Throughput: Instances that performed well and achieved stability for Disk read throughput were c5.xlarge, m5.xlarge, t2.xlarge, t2.2xlarge.

Disk Cache Read Throughput: Most instances were approximately the same, however c5.xlarge performed better than the others.

Network Upload and latency: all instances have nearly the same upload speed. Instance c5.xlarge has the least latency among all instances.

Using these results, it is recommended to use c5.xlarge and m5.xlarge for this part of the design, by considering the price and performance of the instances. Also, if there is the possibility of using the same kind of instances for both instances, it is recommended to use c5.xlarge, which it's disk performs the best, and has the least network latency.

Compute Module

The 3 instances in the compute module require to be able to download a huge data over the network. They also need to be able to do a lot of input/output operations. They should have great memory performance. We present an overview of characteristics of our instance to use to assess if instances would has good qualification in the compute module or not.

CPU Instance m5.xlarge was the instance that performed the best in the CPU benchmarks, followed by instances t2.2xlarge and c5.xlarge. All instances achieved good performance stability during the benchmarks.

IOPS: We selected c5.xlarge and m5.xlarge as the instances that performed better than other instances. We choose these instances due to their cost and their performance.

Memory: t2.xlarge and m4large are ideal instances with regards to Memory.

Cache Reading Speed: All instances had similar performance in cache reading benchmark, however, c4.xlarge performed slightly better than the others. m4.xlarge, t2.xlarge and t2.2xlarge approximately have the same performance and are the best after c4.xlarge in terms of cache reading speed.

Network Download: Instances m5.xlarge and c5.xlarge performed the best among all instances in the network download benchmark. All the other instances have much less download speed, but they are approximately the same.

Network Upload: All instances have the same network upload speed.

Applying these results, we selected c5.xlarge, m5.xlarge and t2.xlarge as the best instances for computation part.

Visualisation Module

The visualization module needs 2 instances with satisfied CPU and memory performance as they would require to make heavy computations on processed data. We presents an overview of characteristics applied to evaluate whether instances would qualify in the visualization module or not. *CPU* Instance m5.xlarge was the instance that performed the best in the CPU benchmarks, followed by instances t2.2xlarge and c5.xlarge. All instances achieved good performance stability during the benchmarks.

Memory: t2.xlarge and m4.xlarge have the better reluits regarding memory aspect than the other 4 instances.

Cache Reading Speed: All instances had similar performance in cache reading benchmark, however, c4.xlarge performed slightly better than the others. m4.xlarge, t2.xlarge and t2.2xlarge approximately have the same performance and are the best after c4.xlarge in terms of cache reading speed.

Network Download: Instances m5.xlarge and c5.xlarge performed the best among all instances in the network download benchmark. All the other instances have much less download speed, but they are approximately the same.

VII. Issue Encountered

We faced some problems regarding the upper- and lower-bounds we found. First, it was tricky to realise the upper and lower band. *Need to warm up:* Sometimes commands crashes when running the determined values immediately. So we understood that the instances need to be warmed up by issuing random values for the command. *Disk read issues:* The results for disk read with hdparm was not stable at first. we understood that it is required to clean the cache before running this test. Also, the naming of disks are not the same in all instances, in some of them it is xvda, and in other nvme0n1,

which we needed to check for each instance and check the proper disk.

Network speedtest issues: One of the instances produced very high performance in upload, which seemed abnormally good. We did launched another instance, and replaced it with the first one and the issue was resolved.

By taking into consideration all the above results, we opt t2.xlarge and c5.xlarge as the ideal selection for visualis

conclusion

REFERENCES

[1] " What Is Amazon EC2?" http://https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html/ /, 2020.

[2] K. Shibata, "Test I/O Performance of Linux using DD," Available: (https://medium.com/@kenichishibata/test-i-o-performance-of-linux-using-dda5074f1de9ce, Tech. Rep., 2014.

[3] A. Kopytov, "Sysbench manual," *MySQL AB*, pp. 2–3, 2012.

[4]

[5] "hdparm System Manager's Manual," http://man7.org/linux/man-pages/man8/hdparm.8.html/ /, 2018.

[6] B. Foley, "Regression Analysis," Available: https://www.surveygizmo.com/resources/blog/regression-analysis//, Standard, 2019.