

Cloud Computing Assignment 2

VM performance and cost-efficiency comparisons

This report documents the steps followed to compare different VMs in terms of their performance and cost efficiency. I firstly did short research about Phoronix Test Suite. I found that it is a testing and benchmarking platform in which tests can be carried out in a fully automated manner. In their website it is written that all these tests are easily reproducible and easy-to-use as well.

Then, I created an account on <http://openbenchmarking.org>. The links to my profile and tests results are as follows:

<https://openbenchmarking.org/user/meteharun>

<https://openbenchmarking.org/result/2409186-METE-240918096>.

After creating the account, I looked for which instance types to choose. I decided to choose the following instances that can be seen in Table 1 below.

Instance Size	vCPU	Memory (GiB)	Network Bandwidth (Gbps)***
c7g.medium	1	2	Up to 12.5
m7g.medium	1	4	Up to 12.5
t4g.medium	2	4	Up to 5

Table 1. Chosen instances

I then launched a VM using c7g.medium as instance type and Ubuntu 24.04 as image. I used the same key pair that I used in the first assignment. Then, I used the command **sudo apt-get update** to check whether there are updates. Since the task I was going to implement required html, xml and unzip dependencies, I installed them using following commands:

- **sudo apt-get install php**
- **sudo apt-get install php-xml php7.4-xml**
- **sudo apt-get install unzip**

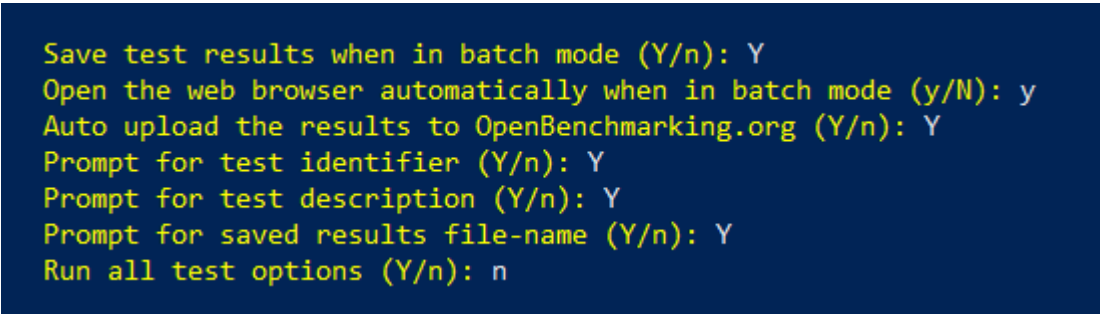
After that, I used the following commands to download Phoronix Test Suite and the required packages:

- **Wget http://phoronix-test-suite.com/releases/repo/pts.debian/files/phoronix-test-suite_10.2.2_all.deb**
- **sudo dpkg -i phoronix-test-suite_10.2.2_all.deb**

Since there were unmet dependencies, I used the command **sudo apt --fix-broken install** to fix them. Now, it was time to install and run the tests that were requested in the pdf. Thus, I used the following command to install all the test suites and tests:

- **phoronix-test-suite install pts/openssl pts/stream pts/encode-mp3 pts/apache pts/network-loopback pts/john-the-ripper**

However, I encountered an error stating that I needed to configure the batch mode first. So, I used the command **phoronix-test-suite batch-setup** and I answered the questions that can be seen in Figure 1 below.



```
Save test results when in batch mode (Y/n): Y
Open the web browser automatically when in batch mode (y/N): y
Auto upload the results to OpenBenchmarking.org (Y/n): Y
Prompt for test identifier (Y/n): Y
Prompt for test description (Y/n): Y
Prompt for saved results file-name (Y/n): Y
Run all test options (Y/n): n
```

Figure 1. Batch-setup configuration

After I re-ran the command to install the tests, I used the following command to login to my openbenchmarking.org account:

- **phoronix-test-suite openbenchmarking-login**

I entered my credentials and logged in. Then I used the following command to execute the benchmarks:

**phoronix-test-suite batch-run pts/openssl pts/stream pts/encode-mp3 pts/apache
pts/network-loopback pts/john-the-ripper**

Before the process started, I was asked to select some parameters to be tested, which can be seen in Figure 2 and Figure 3. For the process test configuration, I choose only RSA4096 as it was mentioned in the pdf. Similarly, I chose only the blowfish and Add options. But I tested all the options for the system test configuration since none of the options were specifically mentioned in the pdf.

```
pts/openssl-3.3.0
Processor Test Configuration
 1: RSA4096
 2: SHA256
 3: SHA512
 4: AES-128-GCM
 5: AES-256-GCM
 6: ChaCha20
 7: ChaCha20-Poly1305
 8: Test All Options
** Multiple items can be selected, delimit by a comma. **
Algorithm: 1

[8192] trim(): Passing null to parameter #1 ($string) of type s

Stream 2013-01-17:
pts/stream-1.3.4
Memory Test Configuration
 1: Copy
 2: Scale
 3: Add
 4: Triad
 5: Test All Options
** Multiple items can be selected, delimit by a comma. **
Type: 5
```

Figure 2. Processor and Memory Test Configuration

```
Apache HTTP Server 2.4.56:
pts/apache-3.0.0
System Test Configuration
 1: 4
 2: 20
 3: 100
 4: 200
 5: 500
 6: 1000
 7: Test All Options
** Multiple items can be selected, delimit by a comma. **
Concurrent Requests: 7

[8192] trim(): Passing null to parameter #1 ($string) of type s
[8192] trim(): Passing null to parameter #1 ($string) of type s
[8192] trim(): Passing null to parameter #1 ($string) of type s

John The Ripper 2023.03.14:
pts/john-the-ripper-1.8.0
Processor Test Configuration
 1: MD5
 2: Blowfish
 3: HMAC-SHA512
 4: bcrypt
 5: WPA PSK
 6: Test All Options
** Multiple items can be selected, delimit by a comma. **
Test: 2
```

Figure 3. System and Processor Test Configuration

Then, I was asked to enter a name for the result file, test run and a description, which I entered as follows:

- Enter a name for the result file: **CC24AutAssignment2**
- Enter a unique name to describe this test run / configuration: **t4g.medium**
- New Description: **amazon testing on Ubuntu 24.04 via the Phoronix Test Suite.**

After finishing the configuration, the testing process began, which was estimated to last an hour. The whole process which was consisting of 12 tests in total was finished in 63 minutes. I was asked two questions, which can be seen below along with my answers:

- Would you like to upload the results to OpenBenchmarking.org (Y/n): **Y**
- Would you like to attach the system logs (lspci, dmesg, etc) to the test result (Y/n): **Y**

Now it was the time to execute the benchmarks for my other instances. In order not to do the installation steps once again, I created an image from this VM before terminating it. In the AWS console, I right clicked my instance and clicked “create image”. Now, I was able to see my image under “Images → AMIs”. By right clicking my image, I chose the option “Launch instance from AMI” and chose c7g.medium as my instance type. Since I did not want to run the tests one by one once again, I ran the following command I found in the openbenchmarking.org to execute the exact same list of tests:

- **phoronix-test-suite benchmark 2409140-HA-2409147HA36**

This time the whole process lasted 76 minutes. Afterwards, I terminated my instance and launch my last instance, which was m7g.medium. This was the fastest one and it took 62 minutes in total to execute all of the benchmarks.

Questions

Report on the selected instance types, and briefly explain why you chose them.

For this assignment, I selected three different instance types: c7g.medium, m7g.medium, and t4g.medium. These instances were chosen with a balance of performance, memory, and network characteristics in mind, while keeping certain parameters constant across them to facilitate a fair comparison. I firstly checked all the instances along with their properties to find some common characteristics. If I had selected totally different instances, my comparion would not have been informative at all. Therefore, all of the instances are medium in terms of their instance size. For the other parameters, I wanted to have two instances having same values to compare these instances based on other parameters, then compare them to my third instance based on the selected parameter.

CPU Architecture was one of the parameters that I took into account. The c7g and m7g instances both utilize AWS’s ARM-based Graviton3 processors, which are designed to deliver better performance-per-watt compared to traditional x86 instances. By choosing both c7g and m7g, I wanted to assess how well the Graviton3 processors handle different workloads. The t4g instance on the other hand uses the Graviton2 processor, which offers a slightly lower performance and efficiency compared to Graviton3. This selection allows me to compare the

improvements in AWS's ARM architecture between Graviton2 and Graviton3, especially for more cost-effective instances like the t4g.medium.

Similarly, both m7g.medium and t4g.medium instances offer 4 GiB of RAM, which keeps the memory capacity constant for certain tests, helping me isolate other factors such as vCPU and network performance. Likewise, I specifically chose the c7g.medium and m7g.medium instances because they both have 1 vCPU, allowing for a direct comparison of Graviton3 processors under similar conditions. On the other hand, t4g.medium offers 2 vCPUs, making it an interesting case for examining whether additional virtual CPUs in a previous generation ARM-based instance can compensate for lower single-core performance in certain workloads.

Lastly, I wanted to explore the impact of network bandwidth. While c7g.medium and m7g.medium instances offer up to 12.5 Gbps of network bandwidth, The t4g.medium instance has up to 5 Gbps, which again, may give important information about whether network performance is significantly affected by bandwidth differences in tasks that rely on data transfer, such as the Loopback TCP Network Performance benchmark. Finally, the reason I chose one instance from each family was to compare the cost-efficiency of different instance types for general-purpose, memory-intensive, and compute-heavy tasks.

Report the obtained results and comment on them

Table 2 shows the results for each test for each instance. Before going into detail, we can say that m7g.medium and c7g.medium have almost the same values for every test. T4g.medium, on the other hand is better than the others in some tests such as memory performance, Apache HTTP server and LAME MP3 encoding, and worse in terms of CPU-intensive workloads and network performance.

Figure 4 and Figure 5 may also provide valuable insights about the comparison of the instances. Among twelve tests, t4g.medium has the best performance in seven of them, while m7g.medium has the first place of three of the results, and c7g.medium has the remaining two. Even though t4g.medium seems to be the most successful instance so far, it is worth noting that it has the last place finish for four times out of the twelve tests.

CC24AutAssignment2			
ptsli	t4g.medium	m7g.medium	c7g.medium
stream: Add	46973.3	48672.8	
network-loopback: Time To Transfer 10GB Via Loopback	19.623	18.009	17.946
john-the-ripper: Blowfish	1400	795	795
encode-mp3: WAV To MP3	11.540	8.234	8.222
openssl: RSA4096	81.8	158.7	158.6
openssl: RSA4096	6684.2	11006.1	11001.7
apache: 4	7477.01	6929.28	6514.93
apache: 20	8891.21	8550.43	7155.32
apache: 100	9230.29	8578.10	7384.19
apache: 200	9019.12	8123.15	7113.44
apache: 500	8277.45	7229.83	6638.92
apache: 1000	8092.64	7398.53	6700.89
OpenBenchmarking.org			

Table 2. Benchmark Test Results

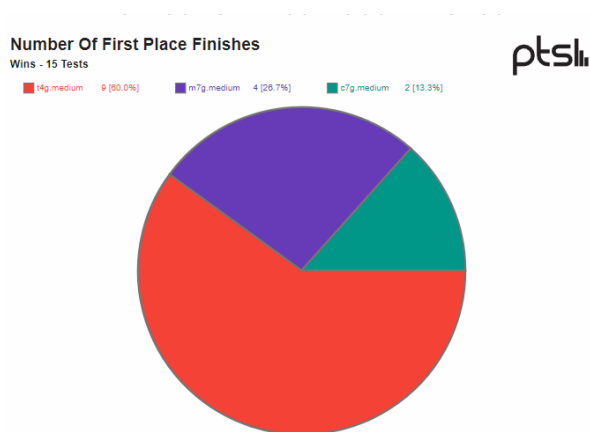


Figure 4. Number of First Place Finishes

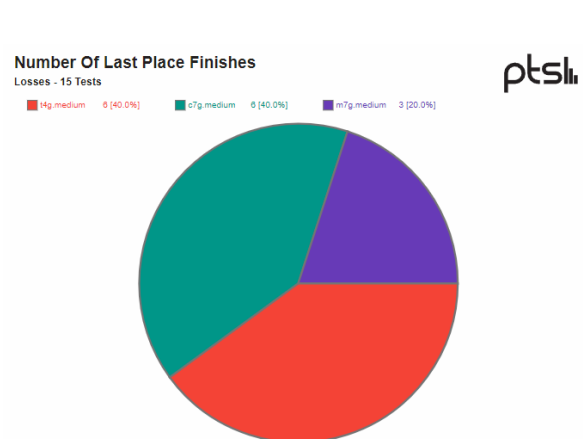


Figure 5. Number of Last Place Finishes

After skimming through the overall results, I tried to analyze each benchmark in detail. Table 3 below shows the execution time, amount of money spent and amount of money spent per hour for each instance. In order to find how much money was spent during the executions, I checked Amazon Console → Billing and Cost Management → Cost Explorer → Costs and usage graph. I then applied filter to see the cost of each instance I used. Finally, I calculated the dollar / hour values by dividing the total cost by execution time for each instance.

Instance	Execution time	Dollar	Dollar / Hour
t4g.medium	1 Hour, 3 Minutes	\$0.08	\$0.076
m7g.medium	1 Hour, 2 Minutes	\$0.11	\$0.106
c7g.medium	1 Hour, 19 Minutes	\$0.13	\$0.098

Table 3. Time and Cost Data for Instances

The first benchmark Stream Add measures how quickly a system can perform a vector operation, and higher scores suggest that the instance can handle memory-intensive applications more effectively. Unfortunately, I failed to test this benchmark with c7g.medium, even though I tried all of the older versions as it was suggested in the assignment pdf. However, since all the other values of m7g.medium and c7g.medium are similar, I would say c7g.medium would have the value that is very close to 48.673 MB/s. As it can be seen, t4g.medium has a slightly higher score than m7g.medium in this benchmark. Since both t4g and m7g are equipped with 4 GiB of RAM, the difference in Stream Add may be attributed to the Graviton3 processor in the m7g.medium instance, which may handle memory operations better than the Graviton2 processor in t4g.medium. However, if we consider the cost efficiency, m7g.medium seems to be significantly better than t4g.medium, which can be seen in Figure 6 below. Since OpenBenchmarking provides results for performance / cost as well, I will be more focusing on these results as the purpose of this study is to compare instances based on cost-efficiency. Finally, I will provide my own evaluation to decide which instance is the best. Based on the data provided, we can conclude that t4g.medium is a better choice for memory-bound workloads if cost is the primary concern.

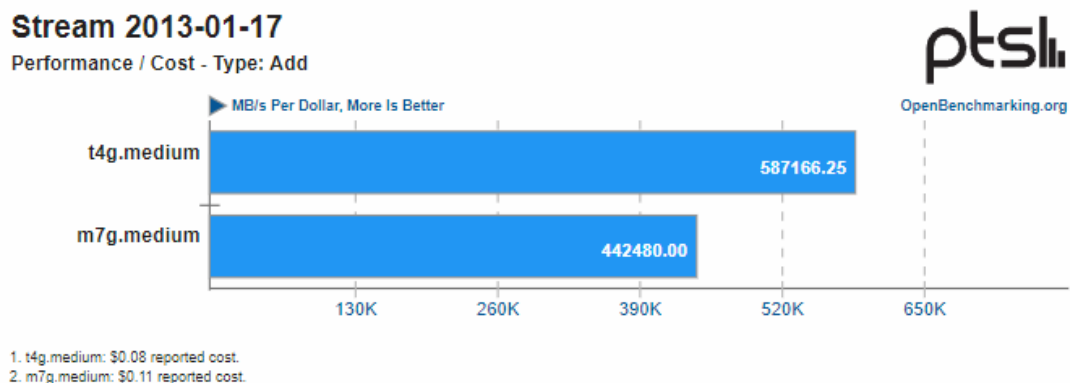


Figure 6. Performance / Cost in Stream Add

Another interesting finding of the test that I conducted is the difference between the Loopback TCP Network Performance for transferring 10GB of data via loopback. The performance metric here is seconds x dollar, where lower values are better. As it can be seen in Figure 7, t4g.medium performs the best in terms of cost-efficiency, taking only 1.57 seconds per dollar to transfer 10GB of data via loopback. Although m7g and c7g have almost the same values for the time it takes to transfer 10 GB via loopback (18 seconds), we can say that m7g is more cost-effective since it is cheaper.

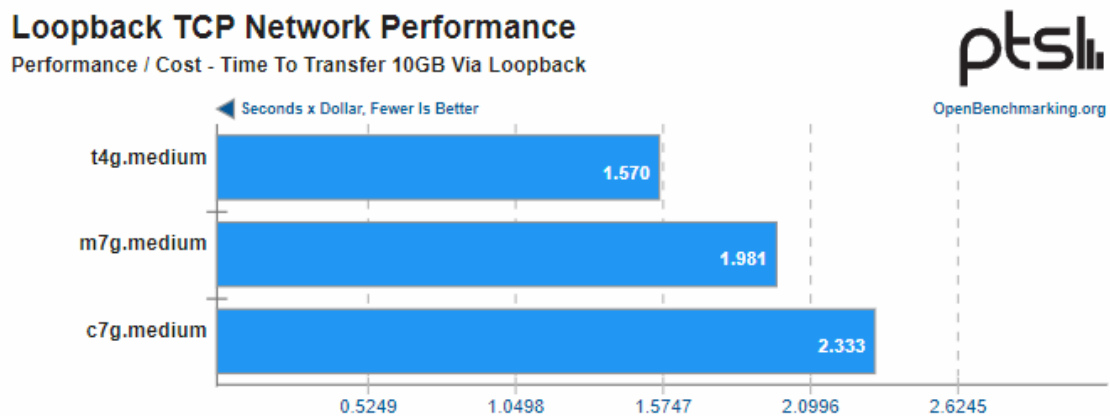


Figure 7. Performance / Cost in Loopback TCP Network Performance

John The Ripper (Blowfish) is another benchmark which t4g is the clear winner in terms of efficiency, which can easily be seen in Figure 8. Whereas m7g and c7g can achieve 7227 and 6115 crypts / seconds per dollar, t4g can achieve more than double the cost-efficiency compared to the other instances with 17500 crypts / second per dollar. The reason behind this may be the difference in vCPU numbers. The t4g.medium instance is equipped with 2 vCPUs, which gives it an advantage in CPU-bound tasks like John The Ripper that benefit from parallel processing. On the other hand, both m7g.medium and c7g.medium have only 1 vCPU, which limits their ability to handle heavily parallelized tasks.

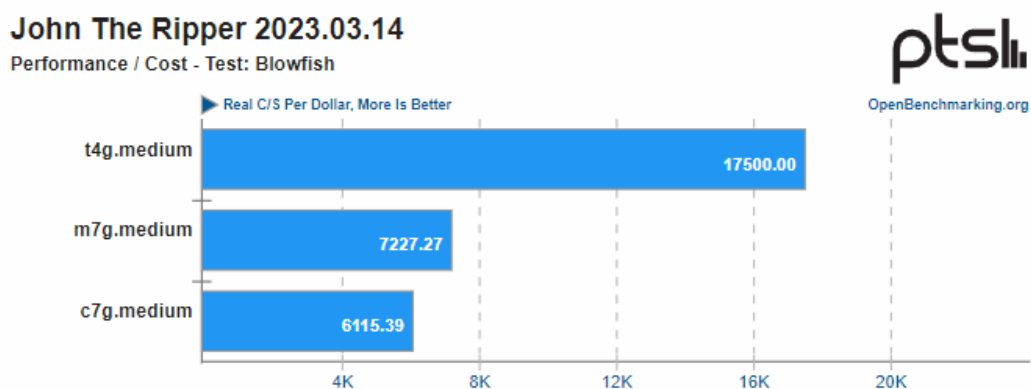


Figure 8. Performance / Cost in Blowfish

The values of instances for LAME MP3 Encoding were similar; therefore, it is difficult to draw a meaningful comparison. The only explanation I can make is that this task does not heavily rely on unique architectural features or specific optimizations of the instances.

The results for OpenSSL 3.3 benchmark on the other hand allow us to make a good comparison. Figure 9 and 10 show the RSA 4096 encryption performance based on two metrics, signutes per second and veridications for second, respectively.

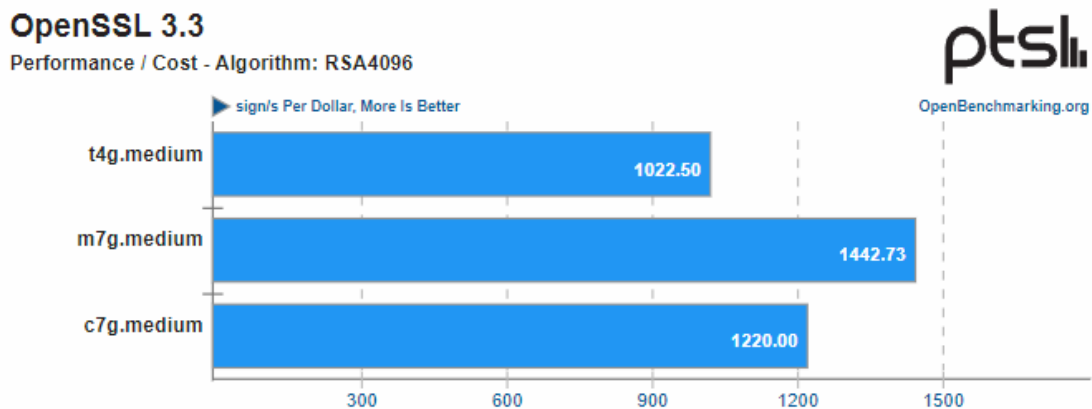


Figure 9. Performance / Cost in RSA 4096 based on sign/s per dollar

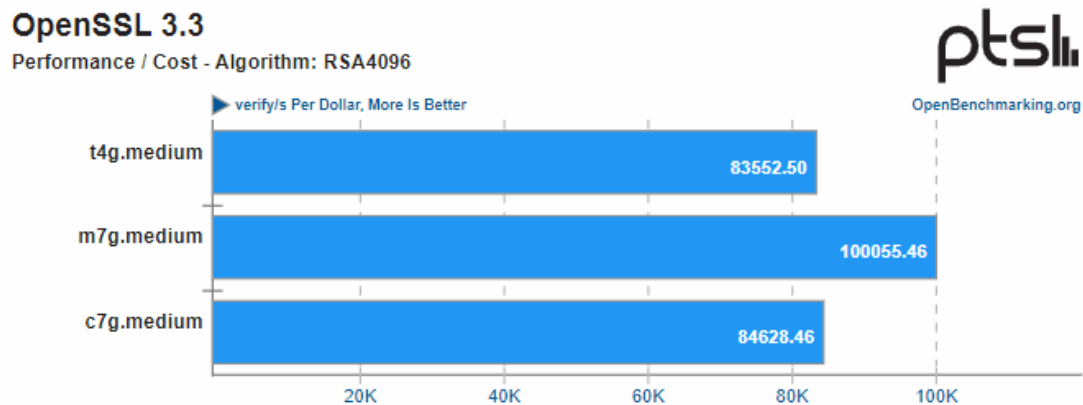


Figure 10. Performance / Cost in RSA 4096 based on verify/s per dollar

We can see that the order of the instances based on their performance on signature and verification is the same, as m7g is the most cost-efficient instance and t4g is the worst. The reason why t4g performed bad compared to other instances can be explained by the fact that it uses Graviton2, while other instances uses Graviton3.

The last benchmark was Apache HTTP Server 2.4.56 benchmark, focusing on concurrent HTTP requests. The metric used is Requests Per Second Per Dollar, where higher values indicate better cost-efficiency in handling HTTP requests. The tests were run with six configurations: 4, 20, 100, 200, 500 and 1000 concurrent requests. Since the order of the instances remained same in all configurations, I wanted to add only one graph to this assignment paper, which is given in Figure 11 below.

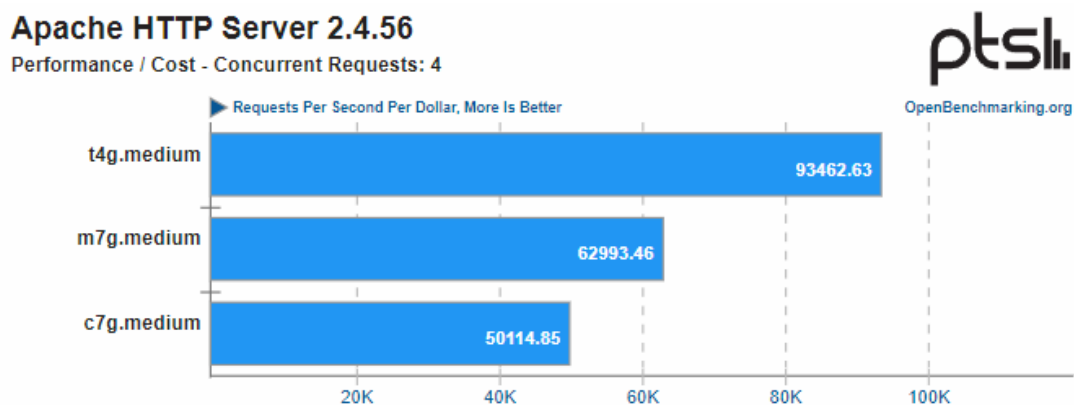


Figure 11. Performance / Cost in Apache HTTP Server 2.4.56 with four concurrent requests

It can be seen that t4g.medium is the most cost-efficient instance for handling 4 concurrent requests, providing nearly 94K requests per second per dollar, which is significantly higher than the other instances. On the other hand, m7g.medium offers 62.9K requests per second per dollar, making it moderately efficient but still far behind t4g.medium. Finally, c7g.medium delivers the lowest cost-efficiency in this test, with 50.1K requests per second per dollar.

Provide your own evaluation and analysis on the obtained relative cost efficiency of the tested VMs.

If we consider the rankings of the instances for benchmarks, t4g.medium is the winner since it has the first place in 8 of the 12 tests, while both m7g.medium and c7g.medium instances have two first places. However, some of the results were really close between the instances. Because of this, simply counting how many tests each instance won doesn't fully capture the bigger picture. To address this, I've decided to use a point-based system for a fairer comparison. The idea is to give each instance points based on how close its performance was to the best result in each test. For each test, the instance with the best performance receives the maximum number of points (10). Other instances get points proportional to their performance relative to the best instance. The formula is as follows:

$$\text{Points for an instance} = \frac{\text{instance's result}}{\text{Best result}} \times 10$$

For example, the points for the RSA 4096 test, based on sign/s per dollar, were calculated as follows:

T4g.medium: $(1022.50 / 1442.73) \times 10 = 7.1$ points

M7g.medium: $(1442.73 / 1442.73) \times 10 = 10$ points

C7g.medium: $(1220 / 1442.73) \times 10 = 8.4$ points

The points for each test and the overall score is given in Table 4.

Category	Test	T4g	M7g	C7g
Stream	Add (MB/s per \$)	10	7.5	7.5
Loopback	Loopback (s x \$)	10	7.9	6.7
John the Ripper	Blowfish (C/S per \$)	10	4.1	3.4
LAME MP3	WAV to MP3 (s x \$)	10	9.8	8.5
OpenSSL	RSA4096 sign (sign/s per \$)	7.1	10	8.5
	RSA4096 verify (verify/s per \$)	8.4	10	8.5
	Average	7.8	10	8.5
Apache HTTP	4 (requests/s per \$)	10	6.7	5.4
	20 (requests/s per \$)	10	7	5
	100 (requests/s per \$)	10	6.8	4.9
	200 (requests/s per \$)	10	6.5	4.9
	500 (requests/s per \$)	10	6.4	4.9
	1000 (requests/s per \$)	10	6.6	5.1
	Average	10	6.7	5
Total Score		115.5	89.3	72.3
Balanced Score		57.8	46	39.6

Table 4. Instance Performance Scores

Since the Apache HTTP tests are overrepresented in terms of the number of measurements, directly summing the points for this category would give it disproportionate weight in the overall comparison. To ensure a fair and balanced evaluation, I have decided to average the points from the six Apache HTTP Server tests and two OpenSSL tests, rather than simply adding them together. Finally, I calculated the balanced overall scores of the instances. The final ranking is as follows: t4g.medium ranks first with a score of 57.8, demonstrating superior overall performance across the tests. m7g.medium takes second place with a score of 46, while c7g.medium closely follows in third with a score of 39.6.

This ranking indicates that t4g.medium emerged as the top performer, offering the best balance of cost-efficiency and performance. Although m7g.medium performed well, particularly in memory-intensive and cryptographic workloads, its higher cost affected its overall score. c7g.medium, despite its compute optimization, ranked lower due to its relatively lower cost-efficiency in the specific tests conducted.

Reflection on what you learned during this exercise

In this exercise, I learned how different AWS instance families are optimized for specific workloads. While I was familiar with general-purpose instances, this exercise gave me a deeper understanding of the strengths and weaknesses of the T, M, and C families. Specifically, I gained a clearer understanding of the trade-offs between cost-efficiency and performance for each instance type and how the architecture of Graviton2 and Graviton3 processors can significantly affect workload performance. The website OpenBenchmarking.com was also new to me, I did not know there was a website to compare and analyse performances of VMs.

One of the things that I found surprising was the fact that most of the test scores were similar between m7g.medium and c7g.medium although they are from different families.

At the end, I was satisfied with my analysis. However, I wish I had more time for this assignment so that I could add more instances and make a wider, more comprehensive analysis, specifically by comparing instances from the same family to see how the other parameters impact the cost-effectiveness of the instances.